

ნ. ბერაია, გ. ჯანელიძე, ლ. ტოკაძე, ი. აფციაური

ობიექტზე ორიენტირებული დაპროგრამება C# ენის ბაზაზე



საგამომცემლო სახლი
„ტექნიკური უნივერსიტეტი“

ნინო ბერაია, გულნარა ჯანელიძე, ლალი ტოკაძე, ია აფციაური

ობიექტზე ორიენტირებული დაპროგრამება C# ენის ბაზაზე



საგამომცემლო სახლი
„ტექნიკური უნივერსიტეტი“

საქართველოს ტექნიკური უნივერსიტეტი

ნინო ბერაია, გულნარა ჯანელიძე, ლალი ტოკაძე, ია აფციაური

ობიექტზე ორიენტირებული დაპროგრამება C# ენის ბაზაზე



რეკომენდებულია საქართველოს
ტექნიკური უნივერსიტეტის
სარედაქციო-საგამომცემლო საბჭოს
მიერ. 30.05.2025, ოქმი №2

თბილისი
2025

განხილულია Ms Visual C# პროგრამების შექმნისა და დამუშავების ტექნოლოგია, C# ენის მონაცემთა საბაზო ტიპები, მმართველი ოპერატორები, კოლექციები და მათ დასამუშავებლად გამოყენებული მეთოდები. წარმოდგენილია ობიექტზე ორიენტირებული დაპროგრამების ძირითადი პრინციპები, ფაილურ ნაკადებთან მუშაობის მეთოდები, დელეგატები, თვისებები, განსაკუთრებული სიტუაციების დამუშავების საშუალებები, LINQ ტექნოლოგია და რთული ობიექტების დამუშავების მეთოდები. გარდა თეორიულ-პრაქტიკული მასალისა, ასახულია სასარგებლო ინფორმაცია დამოუკიდებელი დავალებების შესასრულებლად.

დამხმარე სახელმძღვანელო განკუთვნილია ინფორმატიკისა და მართვის სისტემების ფაკულტეტის ბაკალავრიატის საფეხურის სტუდენტებისა და დაპროგრამების C# ენის შესწავლით დაინტერესებული მკითხველისათვის.

რეცენზენტები: საქართველოს ტექნიკური უნივერსიტეტის ინფორმატიკისა და მართვის სისტემების ფაკულტეტის პროფესორი გია სურგულაძე.

ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტის ზუსტი და საბუნებისმეტყველო მეცნიერებების ფაკულტეტის ასოცირებული პროფესორი კომპიუტერული მეცნიერებების მიმართულებით ნათელა არჩვაძე

© საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, 2025

ISBN 978-9941-520-79-20

<http://www.gtu.ge>

ყველა უფლება დაცულია. ამ წიგნის არც ერთი ნაწილის (იქნება ეს ტექსტი, ფოტო, ილუსტრაცია თუ სხვა) გამოყენება არანაირი ფორმით და საშუალებით (იქნება ეს ელექტრონული თუ მექანიკური) არ შეიძლება გამომცემლის წერილობითი ნებართვის გარეშე.

საავტორო უფლებების დარღვევა ისჯება კანონით.

წიგნში მოყვანილი ფაქტების სიზუსტეზე პასუხისმგებელია ავტორი/ავტორები.

ავტორის/ავტორთა პოზიციას შეიძლება არ ემთხვეოდეს საგამომცემლო სახლის პოზიცია.



შინაარსი

1. შესავალი. Microsoft Visual Studio.Net.-ის ვიზუალური კომპონენტები. კომენტარი. C# ენის მონაცემთა საბაზო ტიპები. ცვლადი და მათი ინიციალიზება.	6
2. ლიტერატურა. ოპერატორები. მათემატიკის მეთოდები	20
3. C# ენის მმართველი: ამორჩევის, იტერაციისა და გადასვლის ოპერატორები.....	34
4. მასივი. Length თვისება. foreach ციკლი. ჩამოთვლები. კლასი System.Array. სია მასივი List <T>.....	69
5. მეთოდები. მეთოდის ცნება. მეთოდის ზოგადი აღწერა. მეთოდის სახეები.	84
6. ობიექტზე ორიენტირებული პროგრამირების პრინციპები: ინკაფსულაცია, აბსტრაქცია, პოლიმორფიზმი და მემკვიდრეობითობა. კლასის გამოცხადება. ობიექტების შექმნა. public, private და protected მოდიფიკატორები. მეთოდის გამოცხადება. მეთოდიდან შედეგის დაბრუნება. პარამეტრების გამოყენება.....	101
7. ობიექტის შექმნა კონსტრუქტორის გამოყენებით. this სიტყვის გამოყენების შემთხვევები	129
8. პოლიმორფიზმი: მეთოდის გადატვირთვა. კონსტრუქტორის გადატვირთვა	135
9. მემკვიდრეობითობის არსი. protected მოდიფიკატორი. კონსტრუქტორები და მემკვიდრეობითობა. base საკვანძო სიტყვა. წინაპარი კლასის კონსტრუქტორის გამოძახება	150
10. სტატიკური კომპონენტები: static მოდიფიკატორი, მუდმივები. შემთხვევითი რიცხვების გენერატორი.....	170
11. ინტერფეისის არსი. ინტერფეისების რეალიზება. სახელების სივრცე (namespace). using დირექტივა. ჩადგმული სივრცე.....	191
12. თვისებების არსი. get და set მეთოდები. მეთოდისთვის ობიექტის გადაცემა. მეთოდის მიერ ობიექტის დაბრუნება.	212
13. შეტანა-გამოტანის ნაკადების კლასები. FileStream კლასი: ფაილის გახსნა, დახურვა, ბაიტების ჩაწერა და წაკითხვა. ფაილში სიმბოლოების შეტანა/გამოტანა: StreamWriter და StreamReader კლასები. ორობითი მონაცემების წაკითხვა და ჩაწერა: BinaryReader და BinaryWriter კლასები. ფაილებთან პირდაპირი მიმართვა: Seek() მეთოდი. ფაილისა და კატალოგის შესახებ ინფორმაციის მიღება. File და FileInfo კლასები და მათი თვისებები. Directory და DirectoryInfo კლასები და მათი თვისებები.....	236
14. დელეგატის არსი. დელეგატების მრავალმისამართიანობა. მოვლენის არსი. ფართოსამაუწყებლო მოვლენა.....	270
15. განსაკუთრებული სიტუაციები. try, catch, throw და finally ოპერატორები. პროგრამის გამართვის საშუალებები. წყვეტის წერტილების შექმნა. ცვლადების მნიშვნელობათა ნახვა. პროგრამის შესრულება ბიჯობრივ რეჟიმში.....	296
16. თარიღთან და დროსთან სამუშაო მეთოდები. სტრიქონებთან სამუშაო მეთოდები. სტრიქონების მასივები. გეომეტრიული ფიგურების ხატვა, გამოსახულებასთან მუშაობა.	319
17. Windows პროგრამირება. მართვის ვიზუალური და არავიზუალური ელემენტები: Button, Label, TextBox, RadioButton, CheckBox, ListBox და ComboBox, TabControl, datePicker, WebBrowser, Timer და სხვა მართვის ელემენტები.....	381
18. ნაკადები. Thread კლასი. Thread კლასის თვისებები. ნაკადების მდგომარეობები.	412
19. LINQ ტექნოლოგია. მეთოდის სინტაქსი და ლამბდა გამოსახულება. დიდი ზომის მონაცემებთან მუშაობა. აგრეგირების ოპერატორები: Average(), Sum(), Max(), Min().	427
20. რთულ ობიექტებთან მუშაობა. მოთხოვნაში ახალი ობიექტის შექმნა. LINQ მეთოდები: Distinct(), All(), Take(), Skip(), First(), Intersect(), Except(), Union(), Joins() მეთოდები.	440

1 შესავალი. Microsoft Visual Studio.Net-ის ვიზუალური კომპონენტები. კომენტარი. C# ენის მონაცემთა საბაზო ტიპები. ცვლადი და მათი ინიციალიზება.

შესავალი. C# ენა არის ზოგადი დანიშნულების, მაღალი დონის, აბსტრაქტული პროგრამირების ენა. იგი შეიქმნა 2001 წელს და მისი ბოლო დასრულებული ვერსია გამოვიდა 2017 წლის მარტში. C# შეიქმნა პროგრამული უზრუნველყოფის ინჟინერის ანდერს ჰეილსბერგის მიერ. ანდერს ჰეილსბერგი წარმოშობით არის დანიიდან.

C# წარმოადგენს ობიექტზე ორიენტირებული პროგრამირების ენას (ოოპ). ცნობილია, რომ კოდირების საუკეთესო ენების უმეტესობა ნაწილობრივ ან მთლიანად ობიექტზეა ორიენტირებული. ობიექტზე ორიენტირებული პროგრამირების (ოოპ) ენა კონცენტრირებულია ობიექტებსა და მონაცემებზე. ეს ნიშნავს, რომ C# ენა არის სწრაფი და ეფექტური, რადგან ის დამოკიდებულია ობიექტებზე და არა დეტალებზე (რაც ენას ადაპტირებულს ხდის). ოოპ საშუალებას გვაძლევს, ხელახლა გამოიყენოთ პროგრამული კოდის გარკვეული ფრაგმენტები მომავალში, რაც ძალიან მნიშვნელოვანია, მსხვილ კორპორატიულ პროექტებზე მუშაობის დროს.

Microsoft Visual Studio.Net. C# ენაზე პროგრამის შექმნა მრავალი ხერხით არის შესაძლებელი. უმარტივესი და საუკეთესო გზაა Microsoft Visual Studio-ს პროგრამული პაკეტის გამოყენება.

Microsoft Visual Studio.Net არის ინტეგრირებული გარემო და წარმოადგენს დაპროგრამების თანამედროვე ტექნოლოგიას. მისი საშუალებით შესაძლებელია, როგორც პროგრამების, ისე ვებაპლიკაციების შექმნა.

Visual Studio-ს აქვს მრავალი უპირატესობა და მისი პროგრამული გარემო მეტისმეტად მარტივი და ადვილად გასაგებია. თქვენ შეგიძლიათ ხელის ერთი მოკიდებით გადაიტანოთ სხვადასხვა ობიექტი ფორმაზე და არც კი დაწეროთ კოდი მათ შესაქმნელად, თუმცა ამავდროულად, შეგიძლიათ ესა თუ ის ობიექტი, პროგრამულად აღწეროთ.

Visual Studio-ს გარემოში თავიდანვე შეგიძლიათ აირჩიოთ აპლიკაციის (დანართის) ფორმა: ვებაპლიკაცია, ვინდოუს აპლიკაცია. პროექტების შესაქმნელად გამოვიყენებთ Windows Forms Application აპლიკაციას (დანართს).

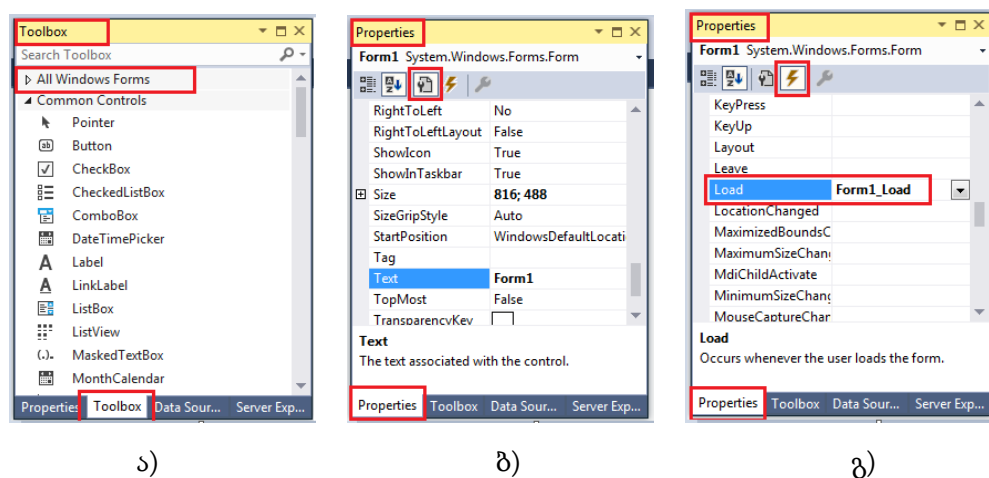
Visual Studio პროგრამულ გარემოში ახალი პროექტის შექმნა ხდება შემდეგი გზით:
Start → Microsoft Visual Studio → New Project → Visual C# → Windows Forms Application →
Name (პროექტს ვარქმევთ ჩვენთვის სასურველ სახელს ან ვტოვებთ არსებულს) →
Location (ვირჩევთ პროექტის შენახვის მისამართს ან ვტოვებთ არსებულს) → OK.

ეკრანზე დავინახავთ სამუშაო ფორმას Form1 და ინსტრუმენტების დაფას
ToolBox, საიდანაც ხდება პროგრამისთვის საჭირო მართვის ელემენტის გადატანა
ფორმაზე. ქვედა მარჯვენა ნაწილში მოთავსებულია ფორმის ელემენტების თვისებათა
(Properties) ფანჯარა. ის ცალსახად აღწერს ფორმის ან მისი მონიშნული ელემენტ(ებ)ის
თვისებებს. პროექტი ასახულია ფანჯარაში Solution Explorer, სადაც განთავსებულია
მისი შემადგენელი კომპონენტები იერარქიული ხის სტრუქტურის სახით: ფორმები,
რესურსები, პროგრამული კოდები (Program.cs) და ა.შ. აქ შესაძლებელია ახალი
ელემენტების დამატება.

ვიზუალური კომპონენტები. პროგრამის შექმნის პროცესი მოიცავს ორ ეტაპს:

1. პროგრამის ფორმის (დიალოგური, ვიზუალური ფანჯარა) შექმნა;
2. მოვლენის დამუშავების ფუნქციები.

გამოყენებითი პროგრამა, რომელიც მუშაობს Windows გარემოში, იქმნება
ფორმაზე კომპონენტების დამატებით და მათი თანამიმდევრული აწყობით. ნებისმიერ
ვიზუალურ კომპონენტს (ზოგჯერ მათ მმართველ ელემენტსაც უწოდებენ) აქვს
თვისებები, მახასიათებლები (Properties) და მოვლენა (Events). მახასიათებლებისა და
მოვლენის საშუალებით განისაზღვრება კომპონენტების სახე, ზომა, ფერი, პროგრამული
კოდი (ნახ. 1.ა.ბ.გ.).



ნახ. 1. ფანჯარა: ა) კომპონენტების; б) თვისებების (მახასიათებლების); в) მოვლენის

ფორმაზე განლაგებული კომპონენტები უზრუნველყოფენ პროგრამასა და მომხმარებელს შორის ინტერფეისს ანუ ურთიერთქმედებას. სწორედ ასეთი სახის კომპონენტებს ეწოდებათ ძირითადი კომპონენტები.

ძირითად კომპონენტებს მიეკუთვნება:

- label - ტექსტის/ინფორმაციის გამოტანის ველი;
- textBox - ტექსტის შეტანის/რედაქტირების ველი;
- button - ბრძანებითი ღილაკი;
- checkBox - მრავალჯერადი ამორჩევის ღილაკი;
- radioButton - ერთჯერადი ამორჩევის ღილაკი;
- listBox - ჩამონათვალის (სიის) ამორჩევის ღილაკი;
- comboBox - კომბინირებული ჩამონათვალის (სიის) ამორჩევის ღილაკი;
- mainMenu – პროგრამის მთავარი მენიუ;
- contextMenu - კონტექსტური მენიუ.

C# ენაზე შედგენილი პროგრამა იწყება გახსნილი ფიგურული ფრჩხილით { და მთავრდება დახუებული ფიგურული ფრჩხილით }.

პროგრამის შესრულება შესაძლებელია ან F5 ფუნქციონალურ ღილაკზე დაჭერით ან Debug მენიუდან Start Debugging ბრძანების არჩევით.

მოვლენის დამამუშავებელზე პროგრამის დაწერა (დამამუშავებლის შექმნა).
პროგრამის მუშაობის დროს ნებისმიერ მართვის ელემენტზე და თავად ფორმაზეც შეიძლება წარმოიქმნას, აღიძრას მოვლენა. მაგალითად, ბრძანებით ღილაკზე შეიძლება წარმოიქმნას მოვლენა „ღილაკზე დაჭერა“, ხოლო ფანჯარაზე, რომლის დაპროექტებასაც ახდენს ფორმა, შეიძლება წარმოიქმნას რიგი მოვლენა: „ფანჯრის შექმნა“, „ფანჯრის ზომის შეცვლა“, „ფანჯარაზე მაუსის დაწკაპუნება/დაჭერა“ და ა.შ. ეს მოვლენები შეგვიძლია დავამუშავოთ კოდში. მოვლენების დასამუშავებლად საჭიროა შეიქმნას მოვლენის დამამუშავებელი - სპეციალური მეთოდი. კოდი ამუშავდება მაშინ, როცა აღიძვრება რომელიმე მოვლენა. არსებობს მოვლენის დამამუშავებლის შექმნის ორი ვარიანტი:

1. ღილაკზე მოვლენის დამამუშავებლის პროგრამის დაწერა;
2. ფორმის ჩატვირთვის მოვლენის დამამუშავებლის პროგრამის დაწერა.

პირველი ვარიანტია მოვლენის დამამუშავებლის შექმნა ჩუმათობის პრინციპის მიხედვით. მაგალითად ბრძანებით button ღილაკისთვის სწორედ ჩუმათობის პრინციპის მიხედვით იქმნება მოვლენის დამამუშავებელი.

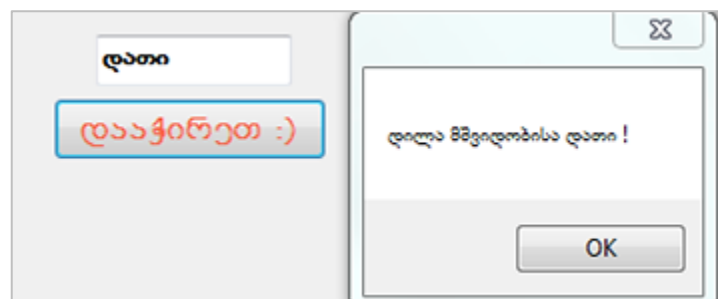
მოდით ახლა დაწვრილებით განვიხილოთ, თუ როგორ ხდება ბრძანებით button ღილაკზე დაჭერის (Click) მოვლენის დამამუშავებლის კოდის დაწერა. ფორმაზე გადავიტანოთ ბრძანებითი button ღილაკი. თვისებების ფანჯრიდან button ღილაკის Text მახასიათებელი შევცვალოთ სიტყვებით: „დილა მშვიდობისა“ ან თქვენთვის სასურველი სიტყვით. დამოუკიდებლად დავარეგულიროთ ბრძანებითი ღილაკის ზომა და ფერი. ამის შემდეგ კი უშუალოდ ბრძანებით button კომპონენტს (ღილაკს) დავაჭიროთ ორჯერ, გამოჩნდება პროგრამული კოდის შემდეგი ფრაგმენტი:

```
private void button1_Click(object sender, EventArgs e)
{ ..... }
```

სწორედ ეს არის ბრძანებით ღილაკზე დაჭერის Click მოვლენის დამამუშავებელი. კოდი ამუშავდება მაშინ, როცა აღიძვრება Click მოვლენა.

პროგრამული კოდი უნდა მივაბათ button1 კომპონენტის Click მოვლენას, როგორც ქვევით არის ასახული. კოდი იწერება ფიგურულ ფხილებს შორის (იხ. ნახ. 2.).

```
private void button1_Click(object sender, EventArgs e) {
    MessageBox.Show("დილა მშვიდობისა " + textBox1.Text + " !"); }
```



ნახ. 2. მარტივი ფორმის შესრულების შედეგი

მეორე ვარიანტია მოვლენის დამამუშავებლის შექმნა ფორმაზე არსებული მონიშნული ელემენტისათვის სასურველი მოვლენის არჩევით. ამ დროს უნდა გამოვიყენოთ თვისებების (Properties) ფანჯარა და მისი ჩანართი მოვლენები  (Events). მოდით ვცადოთ ამ მეთოდის გამოყენება თვითონ პროექტის ფორმისთვის. მაუსით დავაჭიროთ ფორმის ცარიელ (თავისუფალ) არეში. თვისებების ფანჯარაში კი მოვძებნოთ მოვლენა Load. მოვლენა Load-ის გასწვრივ ცარიელ სტრიქონში მაუსით დავაჭიროთ ორჯერ

(მეორენაირად, უფრო მოკლე გზით, ამ მოვლენის გენერირება შეიძლება ფორმის ცარიელ არეში მაუსის ორჯერ დაჭერით). მივიღებთ შემდეგ მეთოდს:

```
private void Form1_Load(object sender, EventArgs e) { }
```

მაგალითად, Load მოვლენის დამამუშავებელი წარმოიქმნება ფორმის ჩატვირთვისას. Load მოვლენის დამამუშავებელი მიუთითებს, იმაზე რომ მართვის ელემენტები სამუშაოდ მზადყოფნაშია.

Load მოვლენის დამამუშავებელში ხდება მართვის ელემენტების თვისებების წაკითხვა და ინიციალიზება, ასევე შესაძლებელია დაკავშირება მონაცემთა ბაზასთან და ა.შ. ხოლო, მაგალითად, **KeyPress** - წარმოიქმნება კლავიატურაზე ღილაკის დაჭერისას. მოვლენა KeyPress-ის პარამეტრი e.KeyChar არის char ტიპის და შეიცავს კლავიატურის ღილაკზე დაჭერის კოდს. კლავიატურის Enter ღილაკს აქვს #13 კოდი, ხოლო Esc ღილაკს - #27 კოდი და ა.შ.

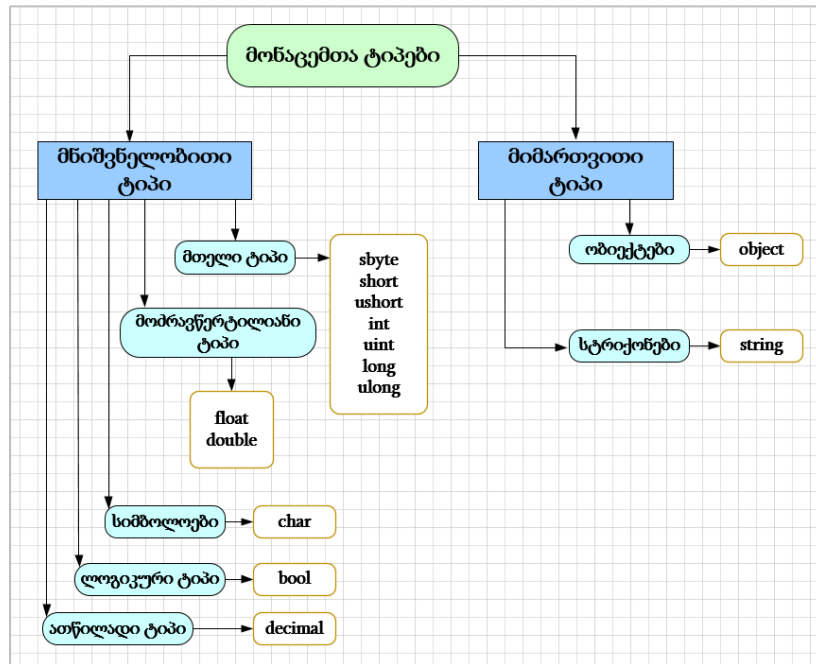
კომენტარები. C# ენაში არსებობს ერთსტრიქონიანი და მრავალსტრიქონიანი კომენტარი. ერთსტრიქონიანი კომენტარი // სიმბოლოებით იწყება. მრავალსტრიქონიანი კომენტარი /* სიმბოლოებით იწყება და */ სიმბოლოებით მთავრდება. კომენტარები შეიცავს პროგრამის სხვადასხვა ნაწილის განმარტებას, ცვლადებისა და მეთოდების დანიშნულებას და ა.შ. კომენტარების გამოყენება აადვილებს პროგრამის გარჩევას. პროგრამის შესრულების დროს ხდება კომენტარების გამოტოვება.

C# ენის მონაცემთა საბაზო ტიპები. C# ენაში მონაცემთა ტიპებს უჭირავს განსაკუთრებული ადგილი, რადგან C, C++ და C# ენები მკაცრად ტიპირებული ენებია. ეს ნიშნავს, რომ კომპილატორი ტიპების თანხვედრასთან მიმართებით ახდენს ყველა ოპერაციის მკაცრ კონტროლს, ამასთან დაუშვებელი ოპერაციები არ კომპილირდება.

ტიპების ასეთი მკაცრი კონტროლი გვადლევეს საშუალებას, თავიდან ავიცილოთ შეცდომა და გავზარდოთ პროგრამის საიმედოობა. ტიპების კონტროლის უზრუნველსაყოფად ყველა ცვლადი, გამოსახულება და მნიშვნელობა უნდა მიეკუთვნოს გარკვეულ ტიპს. ასეთი ცნება, ცვლადი ტიპის გარეშე ანუ „უტიპო“ ცვლადი C# ენაში არ არსებობს, სკრიპტული ენებისაგან განსხვავებით (იხ. ნახ. 3.).

მონაცემთა მთელი ტიპი. C# ენაში განსაზღვრულია მონაცემთა ცხრა მთელი ტიპის მონაცემი: char, byte, sbyte, short, ushort, int, unit, long და ulong. char ტიპის მონაცემს შეუძლია შეინახოს ციფრები, მაგრამ ის ძირითადად სიმბოლოების წარმოდგენისთვის გამოიყენება, ხოლო დანარჩენი რვა ტიპი რიცხვითი

გამოთვლებისთვის. ზოგიერთი მთელი ტიპის მონაცემი ინახავს როგორც დადებით, ისე უარყოფით რიცხვებს, კერძოდ: sbyte, short, int და long, სხვები კი მხოლოდ დადებით რიცხვებს, ესენია: char, byte, ushort, uint და ulong.



ნახ. 3. C# ენის მონაცემთა საბაზო ტიპები

მონაცემთა ნამდვილი (მოდერწერტილიანი) ტიპი. ეს არის ტიპი, რომელიც ინახავს წილად ან ათწილად რიცხვებს. C# ენაში განსაზღვრულია სამი ნამდვილი ტიპის მონაცემი: float, double და decimal. C# ენაში რიცხვები, რომელიც შედის გამოსახულებებში, ჩუმათობის პრინციპის მიხედვით ითვლება მთელი ტიპის მონაცემად.

მაგალითად $\text{double } x = 1/2$; გამოსახულების შემდეგი იქნება 0, რადგან მთელი 1 გაყოფილი მთელ 2-ზე მოგვცემს 0-ს. მსგავს შემთხვევებში, გაყოფის დროს უნდა მივუთითოთ სიმბოლო-მოდერწერტილი, მაგალითად: f სიმბოლო float ტიპისთვის, d კი double ტიპისთვის. ამდენად გამოსახულების სწორი ვარიანტი არის შემდეგი სახის: $\text{double } x = 1d/2d$;

ზოგჯერ პროგრამულ კოდში საჭიროა რიცხვების ჩაწერა ექსპონენციალური ფორმით. ამისათვის მანტისის შემდეგ გამოიყენება სიმბოლო *e* და თანრიგი.

მაგალითად, ნამდვილი რიცხვი $3.52 \cdot 10^{-2}$ შეგვიძლია ჩავწეროთ $3.52e - 2$ სახით.

განმარტება: მანტისა (ლათინ. mantissa, ნამატი, დამატება) - დადებითი რიცხვის ათობითი ლოგარითმის წილობითი (ათწილადი) ნაწილი. მანტისა ეწოდება აგრეთვე ნატურალური ლოგარითმის წილობით ნაწილს. დადებითი რიცხვის ათობითი ლოგარითმის მანტისა არ შეიცვლება, თუ ამ რიცხვს გავამრავლებთ ან გავყოფთ 10^n რიცხვზე ($n \in \mathbb{N}$). **წყარო:** მათემატიკის ენციკლოპედიური ლექსიკონი, ბიბლიოვიკი.

მონაცემთა ტიპი **decimal** გამოიყენება ფულად-საფინანსო გამოთვლებითვის. ასეთი ტიპის მონაცემს მიეწერება სიმბოლო-მოდული M ან m. მაგალითად: **decimal d = 37.15m;**

მონაცემთა სიმბოლური ტიპი. C# ენაში სიმბოლოები წარმოდგენილია არა 8-ბიტიანი კოდით (როგორც პროგრამული ენების უმრავლესობაში), არამედ 16-ბიტიანი კოდით, რომელსაც უნიკოდს (Unicode) უწოდებენ. უნიკოდი სიმბოლოთა საერთაშორისო სტანდარტია, რომელიც კომპიუტერულ ინდუსტრიაში სხვადასხვა დამწერლობისთვის უნივერსალურ კოდებს ადგენს. უნიკოდის ქართული ბლოკი უკვე შეიცავს ასომთავრულ და მხედრულ დამწერლობას (<http://go.on.ge/cfy>).

მონაცემთა ლოგიკური ტიპი. მონაცემთა ლოგიკური ტიპია bool ტიპი. bool ტიპის მონაცემი წარმოადგენს ისეთ ტიპს, რომელიც ინახავს მხოლოდ ორ მნიშვნელობას: ჭეშმარიტი (true, 1) ან მცდარი (false, 0). შესაბამისად ლოგიკური ტიპის მონაცემი ან გამოსახულება იღებს ამ ორი მნიშვნელობიდან ერთ-ერთს: ან ჭეშმარიტს ან მცდარს.

ცვლადი. ყველაზე მეტად გავრცელებული მონაცემებია ცვლადი და მუდმივა (კონსტანტა). ცვლადი არის მეხსიერების უჯრედი, უბანი, რომელსაც მინიჭებული აქვს რაიმე სახელი. სწორედ ეს სახელი გამოიყენება მეხსიერების უბანში არსებულ მონაცემზე (მნიშვნელობაზე) წვდომისთვის, გამოყენებისთვის.

C# ენაში ნებისმიერი ცვლადის აღწერა ანუ გამოცხადება ხდება მონაცემთა ტიპის საშუალებით, რაც გულისხმობს მოცემული ცვლადისათვის ტიპის ყველა შესაძლო მნიშვნელობის დიაპაზონს. ცვლადის აღწერა, გამოცხადება ხდება პროგრამის კოდში, ტანში. სასურველია, რომ ცვლადს გამოცხადებისთანავე მივანიჭოთ საწყისი მნიშვნელობა ანუ განვახორციელოთ ცვლადის ინიციალიზება. მინიჭების ოპერაციის ნიშანი არის = ტოლობა, მაგრამ მას პროგრამირებაში არა ტოლობა, არამედ მინიჭება ეწოდება. თავდაპირველად იწერება ან **ტიპი და შემდეგ ცვლადის სახელი**, ან **ტიპი, ცვლადის სახელი, მინიჭება და ბოლოს მნიშვნელობა**. მაგალითად:

```
//მხოლოდ ცვლადის გამოცხადება; ცვლადის სახელია a; ცვლადი არის მთელი ტიპის;  
int a;  
  
//ცვლადის გამოცხადება და ინიციალიზება; b ცვლადს ენიჭება მნიშვნელობა 9;  
int b = 9;
```


ვთქვათ, თუ გვინდა ცვლადს მივანიჭოთ სიმბოლური ტიპის მნიშვნელობა, მაშინ იგი უნდა ჩავწეროთ ერთმაგ ბრჭყალებში. მაგალითად:

```
char sim;    //მხოლოდ ცვლადის გამოცხადება;  
char simbolo = 'ა'; //ცვლადის გამოცხადება და ინიციალიზება;
```

ცვლადის სახელი შეიძლება შეიცავდეს მაღალი და დაბალი რეგისტრის ლათინურ სიმბოლოებს, ქვედა ტირეს და ციფრებს. თუმცა სახელის ციფრით ან რიცხვით დაწყება არ შეიძლება. ამასთან ცვლადის სახელებს განვასხვავებთ დაბალი და მაღალი რეგისტრის მიხედვით. ეს იმას ნიშნავს, რომ დაბალი რეგისტრის a და მაღალი რეგისტრის A სხვადასხვა ცვლადია. გარდა ამისა, ორი ცვლადის ერთსა და იმავე სახელით აღწერა შეცდომაა.

```
int Age;    //სწორია;  
int age;    //სწორია;  
int 1age;   //არ არის სწორი. სახელის ციფრით დაწყება არ შეიძლება;  
int age1;   //სწორია;  
int age_one; //სწორია;  
int age-one; //არ არის სწორი. სიმბოლო - არ გამოიყენება;  
int age one; //არ არის სწორი. ცარიელი ადგილი არ გამოიყენება.
```

მონაცემთა შეტანა/გამოტანა პროგრამულ გარემოში. განვიხილოთ მონაცემთა შეტანის მართვის ელემენტი (კომპონენტი). მონაცემთა შესატანად უფრო ხშირად გამოიყენება textBox მართვის ელემენტი, რომელიც მიმართავს Text თვისებას.

Text თვისება ინახავს კლავიატურიდან შეტანილ სიმბოლოებს. ამდენად მონაცემები შეიძლება წავიკითხოთ შემდეგი სახით:

```
private void button1_Click(object sender, EventArgs e)  
{  
    string s = textBox1.Text; }  
}
```

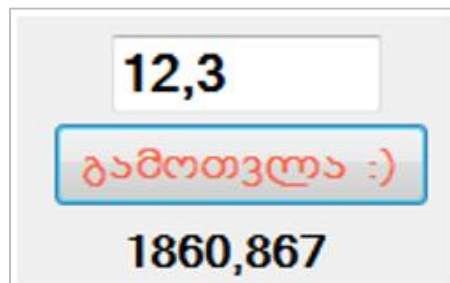
სიმბოლოების სტრიქონთან არითმეტიკული ოპერაციების ჩატარება საკმაოდ რთულია, ამიტომ რიცხვითი მონაცემების შეტანისას შეგვიძლია სტრიქონი გადავიყვანოთ მთელი ან ნამდვილი ტიპის მონაცემად.

ამდენად int და double მონაცემთა ტიპებს გააჩნიათ სტრიქონის რიცხვად გარდაქმნის Parse() მეთოდი. ასეთ რიცხვებთან კი შესაძლებელია უამრავი არითმე-

ტიკული მოქმედებების და ოპერაციების ჩატარება. ამდენად, წინა კოდი შეგვიძლია გადავაკეთოთ შემდეგი სახით:

```
private void button1_Click(object sender, EventArgs e)
{
    string s = textBox1.Text;
    int a = int.Parse(s);
    int b = a * a * a;
}
```

პროგრამირებაში ნამდვილ/წილად რიცხვებთან მიმართებით უფრო ხშირად გამოიყენება წერტილი, მაგალითად, 12.3 ან 11.7 ან 0.0031, მაგრამ C# ენაში სტრიქონის რიცხვად გარდაქმნის მეთოდები (double.Parse() ან Convert.ToFloat()) ითვალისწინებს Windows გარემოს რეგიონალურ პარამეტრებს და ათწილადის გამოყენებას წერტილის ნაცვლად იყენებს მძიმეს, მაგალითად: 12,3 (იხ. ნახ. 4.).



ნახ. 4. შესრულების შედეგი

ამდენად textBox მართვის ელემენტში ათწილადი სახის რიცხვითი მონაცემი უნდა შევიტანოთ მძიმით და არა წერტილით. საწინააღმდეგო შემთხვევაში გარდაქმნა არ მოხდება, პროგრამა გაჩერდება და დააფიქსირებს შეცდომას.

მოდიოთ ზედა კოდი წარმოვადგინოთ ცოტა შეცვლილი სახით:

```
private void button1_Click(object sender, EventArgs e)
{
    double a = double.Parse(textBox1.Text);
    double b = a * a * a;
    label1.Text = b.ToString();
}
```

ფორმაზე **მონაცემთა გამოტანა** კი შესაძლებელია label მართვის ელემენტით (ან ზოგჯერ textBox კომპონენტითაც). სანამ შედეგს გამოვიტანთ ფორმაზე, საჭიროა, რიცხვითი მონაცემი გარდავქმნათ სტრიქონული ტიპის მონაცემად ანუ ახლა მოვიქცეთ პირიქით. ნებისმიერ რიცხვით მნიშვნელობას აქვს მეთოდი ToString(), რომელიც აბრუნებს რიცხვით მნიშვნელობას სტრიქონის სიმბოლოების სახით.

ხშირად საჭიროა **textBox** კომპონენტში კლავიატურიდან შევიტანოთ მხოლოდ ციფრები და გარკვეული აუცილებელი სიმბოლოები. ამისათვის უნდა მოვიდოთ ისეთი მოვლენა, რომლის დახმარებითაც შევძლებთ გავაკონტროლოთ გარკვეული სახის სიმბოლოების შეტანა. ასეთი მოვლენა არის **KeyPress**. ამ საკითხის გადაჭრის რამდენიმე ვარიანტი მოცემულია ცხრილში 1.

ცხრილში 1 აღწერილი პროგრამული კოდი მიბმულია **textBox** კომპონენტის **KeyPress** მოვლენაზე. **დავიმახსოვროთ: textBox კომპონენტში ათწილადი ციფრების შეტანა ხდება მძიმით და არა წერტილით.**

მუდმივა (კონსტანტა) არის იგივე ცვლადი, მხოლოდ მისი მნიშვნელობა პროგრამის შესრულების დროს არ იცვლება. მუდმივაც აღიწერება ისე, როგორც ჩვეულებრივი ცვლადი, მხოლოდ საკვანძო სიტყვა **const**-ის გამოყენებით: **const int A = 11;**

ცხრილი 1. **textBox** კომპონენტში ციფრებისა და გარკვეული სიმბოლოების შეტანა

textBox კომპონენტში მხოლოდ ციფრების და გარკვეული სიმბოლოების შეტანა		
ვარიანტი 1	textBox კომპონენტში მხოლოდ ციფრების შეტანა. ყველაზე სწრაფი ვარიანტი	<pre>char number = e.KeyChar; if (!Char.IsDigit(number)) { e.Handled = true; }</pre>
ვარიანტი 2	textBox კომპონენტში მხოლოდ ციფრების შეტანა ASCII კოდირებით.	<pre>char number = e.KeyChar; if (e.KeyChar <= 47 e.KeyChar >= 58) { e.Handled = true; }</pre>
ვარიანტი 3	textBox კომპონენტში ციფრების, მძიმისა და Backspace ღილაკის გამოყენება.	<pre>char number = e.KeyChar; if ((e.KeyChar <= 47 e.KeyChar >= 58) && number != 8 && number != 44) { e.Handled = true; }</pre> ან შეგვიძლია მეორე სახითაც <pre>char number = e.KeyChar; if (!Char.IsDigit(number) && number != 8 && number != 44) { e.Handled = true; }</pre>
ვარიანტი 4	textBox კომპონენტში ციფრების, მძიმის, ძირითადი მათემატიკური ნიშნების და Backspace ღილაკის გამოყენება.	<pre>char number = e.KeyChar; if ((e.KeyChar <= 47 e.KeyChar >= 58) && number != 8 && (e.KeyChar <= 39 e.KeyChar >= 46) && number != 47 && number != 61) { e.Handled = true; }</pre>

ახლა გადავიდეთ ამოცანების განხილვაზე.

ახალი პროექტის (ფაილის) შექმნა. Visual studio-ს გარემოს გაშვების შემდეგ გამოსულ ფანჯარაში ახალი პროექტის შესაქმნელად ვირჩევთ New Project ბრძანებას, ხოლო შემდეგ ეტაპზე Installed Templates პანელში: Visual C# და Windows forms applicacations (ვიზუალური დაპროგრამების რეჟიმი) რეჟიმს, Name ველში ვუთითებთ ფაილის სახელს, ხოლო Location ველში ფაილის ადგილმდებარეობას.

ამოცანა 1.1. ვიზუალური კომპონენტის თვისების შეცვლა ფორმის დიზაინის დამამუშავებლის გამოყენებით. თვისება შეგვიძლია შევცვალოთ როგორც ფორმის დიზაინის დამამუშავებლის გამოყენებით (**Properties**), ისე პროგრამულადაც.

განვიხილოთ ამ ამოცანის ამოხსნის ორი ვარიანტი:

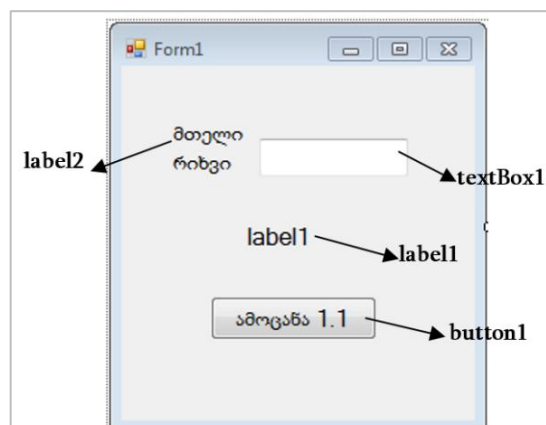
ამოცანა 1.1.-ის პირველი ვარიანტი:

ვიზუალური კომპონენტისათვის **თვისების შეცვლა** განვახორციელოთ **Properties** (ფორმის დიზაინის დამამუშავებლის) გამოყენებით.

ფორმაზე განვათავსოთ ორი **label**, ერთი **textBox** და ერთი **button** ვიზუალური კომპონენტები (იხ. ნახ. 5).

button1 კომპონენტის **Text** მახასიათებელს შევუცვალოთ სახელი და დავარქვათ: „ამოცანა 1.1“.

label2 კომპონენტის **Text** მახასიათებელს დავარქვათ: „მთელი რიცხვი“:



ნახ. 5. ფორმის აგება

პროგრამული კოდი უნდა მივაბათ **button1** კომპონენტის **Click** მოვლენას (**Events**). კოდი ამუშავდება მაშინ, როცა აღიძვრება **Click** მოვლენა. მეორენაირად **Click** მოვლენის გამოძახება ხდება **button1** კომპონენტზე თითის ორჯერ დაჭერით.

```
private void button1_Click(object sender, EventArgs e)
{
    int x;
```

```

        x = int.Parse(textBox1.Text);
        x = x + 7;
        label1.Text = x.ToString();
    }

```

ამოცანა 1.1.-ის მეორე ვარიანტი:

ვიზუალური კომპონენტისათვის **თვისების შეცვლა** განვახორციელოთ **პროგრამულად**.
ფორმაზე განათავსეთ ერთი **label** და ერთი **button** ვიზუალური კომპონენტები.

button1 კომპონენტის **Text** მახასიათებელს შევუცვალოთ სახელი და
დავარქვათ: „**დილა მშვიდობისა, საქართველოვ**“.

განვიხილოთ პროგრამული კოდის ორი ვარიანტი:

პროგრამული კოდის პირველი ვარიანტი:

```

private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "დილა მშვიდობისა, საქართველოვ!";
    label1.Font = new Font("Sylfaen", 16, FontStyle.Bold);
    label1.BackColor = Color.Green;
    label1.ForeColor = Color.Yellow;
    label1.BorderStyle = BorderStyle.Fixed3D;
    label1.TextAlign = ContentAlignment.MiddleCenter;
}

```

პროგრამული კოდის მეორე ვარიანტი:

```

private void button1_Click(object sender, EventArgs e){
    label1.Text = "dila mSvidobisa, saqarTvelov";
    label1.Font = new Font("Zaza", 16, FontStyle.Bold);
    label1.BackColor = Color.Green;
    label1.ForeColor = Color.Yellow;
    label1.BorderStyle = BorderStyle.Fixed3D;
    label1.TextAlign = ContentAlignment.MiddleCenter;
}

```

ამოცანა 1.2. მოცემულია პროგრამული კოდი. დამოუკიდებლად შექმენით
ფორმის დიზაინი და გავუშვათ შესრულებაზე.

ამასთანავე პროგრამას დაუმატოთ კომენტარი-განმარტება.

```

private void button1_Click(object sender, EventArgs e) {
    label1.Text = textBox1.Text;
}

```

ამოცანა 1.3. მოცემული გვაქვს მთელი ტიპის ცვლადი. ცვლადის მნიშვნელობა შევიტანოთ textBox კომპონენტიდან. დავწეროთ პროგრამა, რომელიც რიცხვს გადაიყვანს ორობით, რვაობით და თექვსმეტობით ათვლის სისტემებში.

ფორმაზე განვათავსოთ სამი **label**, ერთი **textBox** და ერთი **button** ვიზუალური კომპონენტები. პროგრამული კოდი დავწეროთ **button1** კომპონენტის **Click** მოვლენაზე (**Events**).

პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e){
    int ricxvix = int.Parse(textBox1.Text);
    string orobiti = Convert.ToString(ricxvix, 2);
    label1.Text = orobiti.ToString();
    string rvaobiti = Convert.ToString(ricxvix, 8);
    label2.Text = rvaobiti.ToString();
    string teqvsmetobiti = Convert.ToString(ricxvix, 16);
    label3.Text = teqvsmetobiti.ToString();
}
```

დამოუკიდებელი სამუშაო

1.1. შეადგინეთ პროგრამა, რომელიც დაბეჭდავს სტუდენტის მონაცემებს (სახელს, გვარს, ჯგუფის ნომერს, უნივერსიტეტის დასახელებას, ფაკულტეტის დასახელებას, მისამართს და ტელეფონს). გამოიყენეთ შვიდი label და ერთი button კომპონენტი.

1.2. მოცემულია პროგრამული კოდის ფრაგმენტი.

იპოვეთ შეცდომები. შესწორებულ კოდს დაურთეთ კომენტარები.

```
int ricxvi1, ricxvi2, jami;
Ricxvi1 = double.Parse(textBox1.text);
Ricxvi2 = double.Parse(textBox2.text);
jami = ricxvi1 + ricxvi2;
label1.Text = Jami.tostring;
```

1.3. მოცემულია პროგრამული კოდის ფრაგმენტი. იპოვეთ შეცდომები. შესწორებულ კოდს დაურთეთ კომენტარები.

```
int a, b, x, y, d;
```

```
a = double.Parse(textbox1.Text)
b = double.Parse(textbox2.Text)
x = double.parse(textbox3.Text)
y = b * Math.pow(Math.Tanx, 2)
d = Math.Sqrt(a) * math.cos(b);
label4.text = " y= " + y.ToString() + " d= " + d.ToString();
```

მოცემულ ნაშრომში პროგრამული კოდი უმეტეს შემთხვევაში დაიწერება button კომპონენტის Click მოვლენაზე (Click მოვლენა მეორენაირად შეგვიძლია გამოვიძახოთ button კომპონენტზე ორჯერ დაჭერით). button კომპონენტის შეგვიძლია ვუწოდოთ ბრძანებითი ღილაკიც.

სასურველია ფორმა გაფორმდეს ისეთი სახით, რომ ვიზუალურად მარტივად კითხვადი და პროგრამულად ადვილად გასაგები იყოს. თუ მოცემულობაში არ არის აღწერილი კომპონენტები, რომლებიც უნდა განვალაგოთ ფორმაზე, მაშინ სტუდენტმა/მკითხველმა დასმული ამოცანა დამოუკიდებლად უნდა გადაწყვიტოს.

2 ლიტერალები. ოპერატორები. მათემატიკის მეთოდები

ლიტერალები. ლიტერალი (literal) არის პროგრამის კოდში ისეთი ჩანაწერი, რომელსაც აქვს ფიქსირებული (მუდმივი) მნიშვნელობა.

არსებობს შემდეგი ტიპის ლიტერალები:

1. ლოგიკური;
2. მთელი;
3. ნამდვილი (წილადი, მოძრავწერტილიანი);
4. სიმბოლური;
5. სტრიქონული;
6. მასივი-ლიტერალი;
7. ცალკე ლიტერალი, რომელსაც წარმოადგენს Null საკვანძო სიტყვა.

ლიტერალების მაგალითებია: **370** - int ტიპის მთელი ლიტერალი; **'G'** - char ტიპის სიმბოლური ლიტერალი; **'@'** - char ტიპის სიმბოლური ლიტერალი; **3.471** - double ტიპის ნამდვილი (წილადი, მოძრავწერტილიანი) ლიტერალი; „თბილისი“-string ტიპის სტრიქონული ლიტერალი; „დილა მშვიდობისა, საქართველო!“ - string ტიპის სტრიქონული ლიტერალი.

ჩვეულებრივი სიმბოლოების გარდა C# ენაში არის ეგრეთ წოდებული სპეციალური მმართველი სიმბოლოები (escape sequence).

უფრო მეტად გავრცელებულია შემდეგი მმართველი სიმბოლოები:

1. **'\'** - ერთმაგი ბრჭყალი (აპოსტროფი);
2. **'\"'** - ორმაგი ბრჭყალი;
3. **'\\'** - ირიბი დახრილი ხაზი;
4. **'\0'** - Null (სიმბოლო, რომლის კოდია 0);
5. **'\a'** - ზარის ხმა;
6. **'\b'** - უკან დაბრუნება (BackSpace);
7. **'\n'** - ახალ სტრიქონზე გადასვლა;
8. **'\r'** - სტრიქონის დასაწყისში გადასვლა;
9. **'\t'** - ჰორიზონტალური ტაბულირება;
10. **'\v'** - ვერტიკალური ტაბულირება.

ოპერატორები. ოპერატორი არის სიმბოლო, რომელიც კომპილატორს ატყობინებს თუ რომელი სპეციფიკური ოპერაცია უნდა შესრულდეს. C# ენაში არსებობს ოპერატორების ოთხი ძირითადი კლასი: **არითმეტიკის; ლოგიკის; შედარების; ბიტობრივი.**

პროგრამაში ნებისმიერი ქმედება სრულდება ოპერაციების ამა თუ იმ სიმრავლის დახმარებით, ოპერაციები ჩაიწერება გამოსახულებების დახმარებით.

გამოსახულება შეიძლება შედგებოდეს ოპერანდებისაგან, ოპერაციის ნიშნებისაგან და მრგვალი ფრჩხილებისაგან.

მაგალითად, $a + b - 10$ გამოსახულებაში სიდიდეები a, b და კონსტანტა 10 არიან ოპერანდები, ხოლო $+$ და $-$ სიმბოლოები ოპერაციის ნიშნები.

გავარკვიოთ თუ რას ნიშნავს უნარული, ბინარული, ტერნარული და n -არული ოპერატორები.

უნარულია ოპერატორი თუ ოპერაციაში მხოლოდ ერთი ოპერანდია: $-a$;

ბინარულია ოპერატორი თუ ოპერაციაში ორი ოპერანდია: $a + b$;

ტერნარული სამი ოპერანდის კომბინაციაა: $?:$ **ჩანაწერი**;

n -არულია ოპერატორი, თუ გამოთვლების ჩასატარებლად საჭიროა სამზე მეტი ოპერანდი.

უნარული ოპერატორებია:

- 1) & მისამართის დაბრუნების ოპერაცია;
- 2) * მისამართზე მიმართვის ოპერაცია;
- 3) – უნარული მინუსი, $-a$;
- 4) + უნარული პლუსი, $+a$;
- 5) ! უარყოფა, $!a$;
- 6) ++ ავტომატური გაზრდა ერთით, $a++$ ან $++a$;
- 7) -- ავტომატური შემცირება ერთით, $a--$ ან $--a$.

ბინარული ოპერატორები:

- 1) + ბინარული პლუსი, $a + b$;
- 2) – ბინარული მინუსი, $a - b$;
- 3) * გამრავლება, $a * b$;
- 4) / გაყოფა, a/b ;
- 5) % ნაშთის გამოყოფა, $a \% b$.

არითმეტიკის ოპერატორები. არითმეტიკის ოპერატორები შეგვიძლია გამოვიყენოთ როგორც მთელი, ისე ნამდვილი, წილადი რიცხვების მიმართ.

ორი მთელი რიცხვის გაყოფისას განაყოფი ყოველთვის იქნება მთელი რიცხვი.

მაგალითად: $9/5$ განაყოფი ყოველთვის 1 იქნება.

ზუსტი განაყოფის მისაღებად აუცილებელია გამოვიყენოთ ან გასაყოფი იყოს float-ის ან double-ის ტიპის.

მაგალითად: $9/5f$ განაყოფი 1.8 იქნება.

სუფიქსი f მიუთითებს, რომ გამოყოფი არის float ტიპის.

ოპერატორი % - პასუხად აბრუნებს ნაშთს.

მაგალითად: $9\%5$ ამ ოპერაციის პასუხი იქნება 4.

ოპერატორი % ხშირად გამოიყენება რიცხვების ლუწობაზე ან კენტობაზე შესამოწმებლად, ვეძებთ 2-ზე გაყოფისას მიღებულ ნაშთს. თუ ნაშთი იქნება 0 რიცხვი ლუწია და თუ ნაშთი იქნება 1 რიცხვი კენტია (იხ. ცხრილი 2).

ინკრემენტისა და დეკრემენტის (უწარულის) ოპერატორები.

++ ინკრემენტაცია გულისხმობს არსებული მნიშვნელობის 1-ით გაზრდას.

-- დეკრემენტაცია გულისხმობს არსებული მნიშვნელობის 1-ით შემცირებას.

ცხრილი 2. არითმეტიკის ოპერატორების ცხრილი

ოპერატორი	მნიშვნელობა	მოქმედება და ჩაწერის ფორმა
+	შეკრება	$a + b$
-	გამოკლება	$a - b$
+	უწარული შეკრება	$+a$ მაგალითად: $+7$
-	უწარული გამოკლება	$-a$ მაგალითად: -3
*	გამრავლება	$a * b$
/	გაყოფა	a / b
%	ნაშთის გამოყოფა	$a \% b$ მაგალითად: $7 \% 3 = 1$
++	ინკრემენტი	$i++$ $++i$
--	დეკრემენტი	$i--$ $--i$
=	მინიჭება	$a = b$
• (წერტილი)	წევრის ან ობიექტის ან კლასის წევრის არჩევა	

ინკრემენტაცია და დეკრემენტაცია შესაძლებელია გამოყენებული იქნას პრეფიქსად და სუფიქსად. პრეფიქსის დროს ოპერატორი დგას წევრის წინ ++a, ხოლო სუფიქსისას იგი მდებარეობს წევრის შემდეგ a++. მაგალითად:

```
int x = 10;  
int y = ++x; // y=11 x=11
```

y-ის მნიშვნელობა გახდება 11-ის ტოლი, რადგან თავიდან x-ის მნიშვნელობა იზრდება 1-ით, ხოლო შემდეგ ენიჭება y-ს.

```
int x = 10;  
int y = x++; //y=10 x=11
```

ამ შემთხვევაში y-ის მნიშვნელობა გახდება 10-ის ტოლი, რადგან თავიდან x-ის მნიშვნელობა ენიჭება y-ს, ხოლო შემდეგ x იზრდება 1-ით. მიაქციეთ ყურადღება იმას, რომ ორივე შემთხვევაში x არის 11-ის ტოლი.

შედარებისა და ლოგიკის ოპერატორები. შედარების ოპერატორები ადარებენ ორ მნიშვნელობას და გასცემენ bool ტიპის მნიშვნელობას: ან ჭეშმარიტს (true) ან მცდარს (false).

ლოგიკის ოპერატორები ასრულებენ ლოგიკურ ოპერაციებს bool ტიპის მნიშვნელობებზე. C# ენაში == და != ოპერატორები შეიძლება გამოვიყენოთ ყველა ობიექტის მიმართ მათი შედარებისთვის ტოლობაზე ან უტოლობაზე.

bool ტიპის მნიშვნელობების შედარება შეიძლება მხოლოდ ტოლობაზე ან უტოლობაზე (იხ. ცხრილი 3).

მინიჭების ოპერატორი. როგორც უკვე პირველ თავში აღვნიშნეთ მინიჭების ოპერატორი გამოიყენება ცვლადებთან, რათა ცვლადს ახალი მნიშვნელობა მივანიჭოთ. მინიჭების ოპერატორის სიმბოლო არის = მათემატიკური ტოლობის ნიშანი.

გამოსახულება. გამოსახულება მიუთითებს მონაცემებზე მოქმედებების შესრულების წესს და შედგება ოპერანდებისაგან (კონსტანტები, ცვლადები, ფუნქციების), მრგვალი ფრჩხილებისა და ოპერაციების ნიშნებისგან,

მაგალითად: $y = a * b - \text{Math.sin}(x) + 10.97$;

მრგვალი ფრჩხილები გამოიყენება პროგრამის კითხვადობის გასაუმჯობესებლად. მაგალითად, მოყვანილი ორი სტრიქონიდან მეორე უკეთესად იკითხება:

```
y = x * 5 + z / 7.2 - w;  
y = ( x * 5 ) + ( z / 7.2 ) - w;
```

მათემატიკის მეთოდები აღწერილია System.Math სტანდარტულ კლასში. მათი უმრავლესობა double ტიპის მნიშვნელობას გასცემს (იხ. ცხრილი 4).

C#-ში ასევე შესაძლებელია სხვადასხვა ოპერაციების შემოკლება:

$a += b;$ //იგივეა, რაც $a = a + b;$

$a -= b;$ //იგივეა, რაც $a = a - b;$

$a *= b;$ //იგივეა, რაც $a = a * b;$

$a /= b;$ //იგივეა, რაც $a = a / b;$

$a \% = b;$ //იგივეა, რაც $a = a \% b;$

ცხრილი 3. შედარების, ლოგიკის ოპერატორების და ჭეშმარიტების ცხრილი

შედარების ოპერატორების ცხრილი		ლოგიკის ოპერატორების ცხრილი																															
==	ტოლია	&	AND (და)																														
!=	არ არის ტოლი		OR (ან)																														
>	მეტია	^	XOR (გამომრიცხავი ან)																														
<	ნაკლებია	&&	AND (სწრაფი „და“ ოპერატორი)																														
>=	მეტია ან ტოლი		OR (სწრაფი „ან“ ოპერატორი)																														
<=	ნაკლებია ან ტოლი	!	NOT (არა)																														
ლოგიკის ოპერატორების ჭეშმარიტების ცხრილი																																	
<table><tr><td>x</td><td>y</td><td>x && y</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>			x	y	x && y	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><td>x</td><td>y</td><td>x y</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	x y	0	0	0	0	1	1	1	0	1	1	1	1
x	y	x && y																															
0	0	0																															
0	1	0																															
1	0	0																															
1	1	1																															
x	y	x y																															
0	0	0																															
0	1	1																															
1	0	1																															
1	1	1																															
			<table><tr><td>x</td><td>!x</td></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	!x	0	1	1	0																								
x	!x																																
0	1																																
1	0																																
			<table><tr><td>x</td><td>y</td><td>x ^ y</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	x ^ y	0	0	0	0	1	1	1	0	1	1	1	0															
x	y	x ^ y																															
0	0	0																															
0	1	1																															
1	0	1																															
1	1	0																															

ცხრილი 4. მათემატიკის მეთოდების ცხრილი. System.Math კლასი.

მეთოდი	ფორმულა	დანიშნულება
Math.Abs(x)	$ x $	აბსოლუტური მნიშვნელობა. გამოიყენება მთელ რიცხვებთან მიმართებით.
Math.Fabs(x)	$ x $	აბსოლუტური მნიშვნელობა. გამოიყენება ნამდვილ რიცხვებთან მიმართებით.
Math.Sin(x)	$\sin x$	სინუსი
Math.Cos(x)	$\cos x$	კოსინუსი

ცხრილი 4-ის გაგრძელება. მათემატიკის მეთოდების ცხრილი. System.Math კლასი.

მეთოდი	ფორმულა	დანიშნულება
Math.Tan(x)	$\tan x$	ტანგენსი
Math.Sqrt(x)	$\sqrt{x}$	კვადრატული ფესვი
Math.Exp(x)	e^x	ექსპონენტა: $\text{Exp}(1) = 2.71$;
Math.Log(x)	$\ln(x)$	x -ის ნატურალური ლოგარითმი, $x > 0$
Math.Log10(x)	$\log x$	x -ის ათობითი ლოგარითმი
Math.Log(x,a)	$\log_a x$	x -ის ლოგარითმი a ფუძით
Math.Max(x, y)		ორ რიცხვს შორის უდიდესი
Math.Min(x, y)		ორ რიცხვს შორის უმცირესი
Math.Pow(x, y)	x^y	ახარისხება: $5^2 = \text{Pow}(5,2) = 25$;
Math.Floor(x)		დამრგვალება ნაკლებობით: $\text{Math.Floor}(1.04) = 1$;
Math.Ceiling(x)		დამრგვალება მეტობით: $\text{Math.Ceiling}(1.04) = 2$;
Math.Round(x)		დამრგვალება: $\text{Math.Round}(3.1) = 3$; $\text{Math.Round}(3.8) = 4$;
Math.Round(x, y)		დამრგვალება მითითებული ფორმატით ორი რიცხვის სიზუსტით: $\text{Math.Round}(3.82456, 2) = 3.82$; სამი რიცხვის სიზუსტით: $\text{Math.Round}(3.01322, 3) = 3.013$;
Math.E		ნატურალური ლოგარითმის ფუძე: $e = 2.718$
Math.PI		კონსტანტა: $\pi = 3.14$;
Math.Sign(x)		x არგუმენტის ნიშანი: $\text{Sign}(-5) = -1$; $\text{Sign}(5) = 1$; $\text{Sign}(x) = \begin{cases} -1 & \text{თუ } x < 0 \\ 0 & \text{თუ } x = 0 \\ +1 & \text{თუ } x > 0 \end{cases}$
Math.Truncate()		ათწილადი რიცხვიდან მთელის გამოყოფა: $\text{Math.Truncate}(3.013) = 3$;

განვიხილოთ რამოდენიმე ამოცანა.

დავალბა 1. მოცემულია ცვლადი A. ჩავწეროთ გამოსახულება, რომელიც გასცემს ჭეშმარიტ მნიშვნელობას მაშინ, თუ A ცვლადი ლუწია.

ამოხსნა: $A \% 2 == 1$

დავალბა 2. მოცემულია ორი ცვლადი A და B. ჩავწეროთ გამოსახულება, რომელიც გასცემს ჭეშმარიტ მნიშვნელობას მაშინ, თუ A და B ცვლადები დადებითია.

ამოხსნა: $A > 0 \ \&\& \ B > 0$

დავალბა 3. მოცემულია ორი ცვლადი A და B. ჩავწეროთ გამოსახულება, რომელიც გასცემს ჭეშმარიტ მნიშვნელობას მაშინ, თუ A და B ცვლადები განსხვავდება.

ამოხსნა: $A != B$

დავალბა 4. მოცემულია სამი მთელი ტიპის რიცხვი A, B, C.

ჩავწეროთ გამოსახულება, რომელიც ადასტურებს შემდეგ მოსაზრებას: „თითოეული A, B და C რიცხვებიდან არის დადებითი“.

ამოხსნა: $(A > 0) \ \&\& \ (B > 0) \ \&\& \ (C > 0)$

გადავიდეთ ამოცანების განხილვაზე.

ამოცანა 2.1. არითმეტიკის ოპერატორების დემონსტრირება

```
private void button1_Click(object sender, EventArgs e){
    int x = 10;
    int z = x + 12; //22
    label1.Text = z.ToString() + '\n';
    z = x - 6; //4
    label1.Text += z.ToString() + '\n';
    z = x * 5; //50
    label1.Text += z.ToString() + '\n';
    z = x / 5; //2
    label1.Text += z.ToString() + '\n';
    double a = 10, b = 3;
    double c = a / b; //3.33333333333333
    label1.Text += c.ToString() + '\n';
    double d = 10 / 4; //2
    label1.Text += d.ToString() + '\n';
}
```

```

        double ddd = 10.0 / 4.0; //2.5
        label1.Text += ddd.ToString() + '\n';
        double aaa = 10.0; //2
        double zzz = aaa % 4.0;
        label1.Text += zzz.ToString() + '\n';
    }

```

ამოცანა 2.2. ინკრემენტის და დეკრემენტის ოპერატორების დემონსტრირება

```

private void button1_Click(object sender, EventArgs e) {
    //ინკრემენტის ოპერატორი
    int x1 = 5;
    int z1 = ++x1; //z1=6 x1=6
    label1.Text = "z1=" + z1.ToString() + " " + "x1=" + x1.ToString() + '\n';
    int x2 = 5;
    int z2 = x2++; //z2=5 x2=6
    label1.Text += "z2=" + z2.ToString() + " " + "x2=" + x2.ToString() + '\n';
    //დეკრემენტის ოპერატორი
    int xx = 5;
    int zz = --xx; //zz=4 xx=4
    label1.Text += "zz=" + zz.ToString() + " " + "xx=" + xx.ToString() + '\n';
    int xxx = 5;
    int zzz = xxx--; //zzz=5 xxx=4
    label1.Text += "zzz=" + zzz.ToString() + " " + "xxx=" + xxx.ToString() + '\n';
}

```

ამოცანა 2.3. შედარების ოპერატორების დემონსტრირება

```

private void button1_Click(object sender, EventArgs e) {
    //შედარების ოპერატორი ==
    int a = 10;
    int b = 4;
    bool c = a == b; //false
    label1.Text = c.ToString() + '\n';
    //შედარების ოპერატორი !=
    int a1 = 10;
    int b1 = 4;
    bool c1 = a1 != b1; //true
}

```

```

bool c2 = a1 != 10; //false
label1.Text += c1.ToString() + " " + c2.ToString() + '\n';
//შედარების ოპერატორი <
int a2 = 10;
int b2 = 4;
bool c12 = a2 < b2; //false
label1.Text += c12.ToString() + '\n';
//შედარების ოპერატორი >
int a3 = 10;
int b3 = 4;
bool c3 = a3 > b3; //true
bool d3 = a3 > 25; //false
label1.Text += c3.ToString() + " " + d3.ToString() + '\n';
// შედარების ოპერატორი <=
int a4 = 10;
int b4 = 4;
bool c4 = a4 <= b4; //false
bool d4 = a4 <= 25; //true
label1.Text += c4.ToString() + " " + d4.ToString() + '\n';
// შედარების ოპერატორი >=
int a5 = 10;
int b5 = 4;
bool c5 = a5 >= b5; //true
bool d5 = a5 >= 25; //false
label1.Text += c5.ToString()+ " " +d5.ToString()+ '\n';
}

```

ამოცანა 2.4. ლოგიკის ოპერატორების დემონსტრირება

```

private void button1_Click(object sender, EventArgs e) {
    //ლოგიკის ოპერატორი | (OR (ან))
    // 5 > 6 false, 4 < 6 true, აბრუნებს true
    bool x1 = (5 > 6) | (4 < 6);
    // 5 > 6 false, 4 > 6 false, აბრუნებს false
    bool x2 = (5 > 6) | (4 > 6);
}

```



```

label1.Text = x1.ToString() + " " + x2.ToString() + '\n';
    //ლოგიკის ოპერატორი & (AND(და))
    // 5 > 6 false, 4 < 6 true, აბრუნებს false
bool x11 = (5 > 6) & (4 < 6);
    // 5 < 6 true, 4 > 6 true, აბრუნებს true
bool x22 = (5 < 6) & (4 < 6);
label1.Text += x11.ToString() + " " + x22.ToString() + '\n';
    //ლოგიკის ოპერატორი || (OR (სწრაფი "ან" ოპერატორი))
    // 5 > 6 false, 4 < 6 true, აბრუნებს true
bool x123 = (5 > 6) || (4 < 6);
    // 5 > 6 false, 4 > 6 false, აბრუნებს false
bool x234 = (5 > 6) || (4 > 6);
label1.Text += x123.ToString() + " " + x234.ToString() + '\n';
    //ლოგიკის ოპერატორი && (AND (სწრაფი "და" ოპერატორი))
    // 5 > 6 false, 4 < 6 true, აბრუნებს false
bool x213 = (5 > 6) && (4 < 6);
    // 5 < 6 true, 4 > 6 true, აბრუნებს true
bool x312 = (5 < 6) && (4 < 6);
label1.Text += x213.ToString() + " " + x312.ToString() + '\n';
    // ლოგიკის ოპერატორი ! (NOT (არა))
bool a = true;
bool b = !a; //false
label1.Text += b.ToString() + '\n';
    //ლოგიკის ოპერატორი ^ (XOR(გამომრიცხავი ან))
    // 5 > 6 false, 4 < 6 true, აბრუნებს true
bool x5 = (5 > 6) ^ (4 < 6);
    // 50 > 6 true, 4/2 < 3 true, აბრუნებს false
bool x6 = (50 > 6) ^ (4/2 < 3);
label1.Text += x5.ToString() + " " + x6.ToString() + '\n'; }

```

ოპერატორები | და || (ასევე & და &&) ასრულებენ ერთსა და იმავე მოქმედებას, მაგრამ სინამდვილეში ისინი განსხვავდებიან ერთმანეთისგან. გამოსახულებაში $z = x | y$; მოხდება ორივე მნიშვნელობის გამოთვლა, გამოითვლება როგორც x , ასევე y ,

ხოლო $z = x \parallel y$; გამოსახულებაში თავიდან გამოითვლება x -ის მნიშვნელობა და თუ იგი არის true, მაშინ y -ის გამოთვლას აზრი არა აქვს, რადგან z ნებისმიერ შემთხვევაში არის true.

y -ის მნიშვნელობა გამოითვლება მხოლოდ იმ შემთხვევაში, თუ x -ის მნიშვნელობა არის false. იგივე შეიძლება ითქვას & და && ოპერატორებზე.

ამოცანა 2.5. შევადგინოთ პროგრამული კოდი, რომელიც ნამდვილი ტიპის რიცხვიდან ამოიღებს კვადრატულ ფესვს და მიღებულ შედეგს დაამრგვალებს ორი თანრიგის სიზუსტით.

ფორმაზე განვალაგოთ ერთი textBox, ორი label და ერთი button კომპონენტები. პროგრამული კოდი განვათავსოთ button1 კომპონენტის Click მოვლენაზე (Events).

Click მოვლენის გამოძახება მეორენაირად ხდება button1 კომპონენტზე ორჯერ დაჭერით.

პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e){
    double ricxvi, shedegi;
    ricxvi = double.Parse(textBox1.Text);
    shedegi = Math.Sqrt(ricxvi);
    label1.Text = shedegi.ToString();
    label2.Text = Math.Round(shedegi,2).ToString();
}
```

ამოცანა 2.6. დავწეროთ პროგრამული კოდი, რომელიც გამოითვლის გამოსახულებების მნიშვნელობას.

მოცემულია ორი: $y = btg^2x - a/(\sin^2(x/a))$, $d = ae^{-\sqrt{a}}\cos(bx/a)$ გამოსახულება, სადაც $a = 3.2$; $b = 17.5$; $x = -4.8$; გამოვითვალოთ გამოსახულებების მნიშვნელობები და მიღებული შედეგი დავამრგვალოთ.

ფორმაზე განვალაგოთ სამი **textBox**, ოთხი **label** და ერთი **button** კომპონენტები. პროგრამული კოდი განვათავსოთ **button1** კომპონენტის **Click** მოვლენაზე.

პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e) {
    double a,b,x,y,d;
    a = double.Parse(textBox1.Text);
    b = double.Parse(textBox2.Text);
```

```

x = double.Parse(textBox3.Text);
y = b * Math.Pow(Math.Tan(x), 2) - a/Math.Pow(Math.Sin(x/2), 2);
d = a * Math.Exp(-Math.Sqrt(a)) * Math.Cos(b * x/a);
label1.Text = "y =" + y.ToString() + " " + "d =" + d.ToString();
}

```

ამოცანა 2.7. დავწეროთ პროგრამული კოდი, რომელიც სამკუთხედის ფართობს გამოითვლის ჰერონის ფორმულის გამოყენებით.

ჰერონის ფორმულას აქვს შემდეგი სახე:

$$S = \sqrt{P(P-a)(P-b)(P-c)}, \text{ სადა } P \text{ ნახევარპერიმეტრია: } P = (a + b + c)/2;$$

ამასთან ვიცით, რომ $a = 8.91$; $b = 12$; $c = 6.0$;

ფორმაზე განვალაგოთ სამი **textBox**, სამი **label** და ერთი **button** კომპონენტები.

პროგრამული კოდი განვათავსოთ **button1** კომპონენტის **Click** მოვლენაზე.

პროგრამული კოდი:

```

private void button1_Click(object sender, EventArgs e) {
    double fartobi, nper;
    double a = double.Parse(textBox1.Text);
    double b = double.Parse(textBox1.Text);
    double c = double.Parse(textBox1.Text);
    nper = (a + b + c) / 2;
    label1.Text = "ნახევარპერიმეტრი = " + nper.ToString();
    fartobi = Math.Sqrt(nper*(nper - a)*(nper - b)*(nper - c));
    label2.Text="ფართობი ჰერონის მიხედვით =" + fartobi.ToString();
    label3.Text="ფართობი დამრგვალებული სამი რიცხვის მიხედვით=" +
        Math.Round(fartobi,3).ToString(); }

```

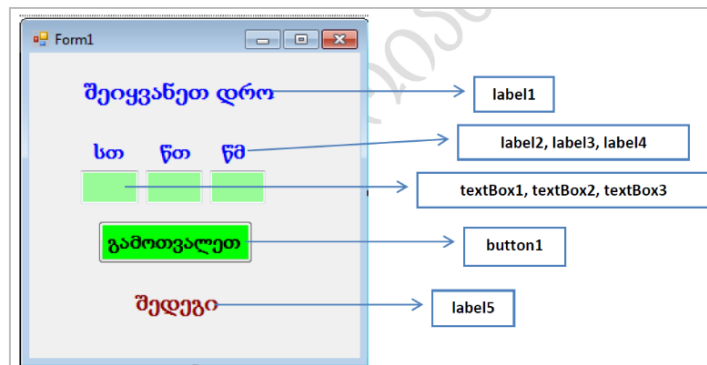
დამოუკიდებელი სამუშაო

2.1. მოცემულია წრეწირი, რომლის რადიუსიც წინასწარ ცნობილია. შეადგინეთ პროგრამა, რომელიც გამოთვლის წრეწირის სიგრძეს $C = 2\pi r$ და ფართობს $S = \pi r^2$. შედეგი ასახეთ შესაბამის კომპონენტში. ვიცით, რომ π რიცხვი მიახლოებით უდრის **3.14159**. ფორმაზე განათავსეთ ორი **textBox**, ორი **label** და ერთი **button** კომპონენტი. **button** კომპონენტის **Click** მოვლენაზე დაწერეთ ძირითადი კოდი. ფორმა გააფორმეთ ისე, რომ ვიზუალურად მარტივად კითხვადი და ადვილად გასაგები იყოს.

2.2. მოცემულია სამკუთხედის გვერდები $a = 16, b = 10, c = 13$. შეადგინეთ ისეთი პროგრამა, რომელიც იპოვოს სამკუთხედის კუთხეების α, β, γ მნიშვნელობებს კოსინუსების თეორემის მიხედვით. კოსინუსების თეორემა გაიხსენეთ ტრიგონომეტრიიდან ☺. ფორმაზე განათავსეთ სამი textBox კომპონენტი a, b და c ცვლადებისათვის, ასევე სამი label კომპონენტი, რომლებშიც აისახება შედეგი, შემდეგ კი განათავსეთ ერთი button კომპონენტი. button კომპონენტის Click მოვლენაზე დაწერეთ ძირითადი კოდი.

2.3. მოცემულია დრო საათის, წუთის და წამის ფორმატით. დაწერეთ ისეთი პროგრამული კოდი, რომელიც მითითებულ დროს გადაიყვანს მხოლოდ წამებში.

ფორმა ააგეთ მე-4 ნახაზზე მოცემული ნიმუშის მიხედვით ან საკუთარი შეხედულებისამებრ უფრო თვალსაჩინოდ (იხ. ნახ. 6).



ნახ. 6. მე-3 დავალების ფორმის ნიმუში

2.4. ფორმაზე შესაბამის კომპონენტში შეიტანეთ მთელი ტიპის რიცხვი, მაგალითად, a ცვლადი. შეადგინეთ პროგრამა, რომელიც გამოთვლის $y = a^3 + \sqrt{a+1} + \frac{1}{a}$ გამოსახულების მნიშვნელობას. მიღებული შედეგი დაამრგვალეთ სამი რიცხვის სიზუსტით. შედეგი გამოიტანეთ შესაბამის კომპონენტში.

2.5. დაწერეთ ისეთი პროგრამული კოდი, რომელიც ლარს გადაიყვანს დოლარში და პირიქით (მაგ: 345.00 ლარი გადაიყვანეთ დოლარში და პირიქით).

2.6. ფორმაზე შესაბამის კომპონენტში შეიყვანეთ მთელი ტიპის რიცხვი, მაგალითად, 1283. დაწერეთ პროგრამული კოდი, რომელიც დაადგენს, თუ რამდენი ერთეული, ათეული, ასეული და ათასეული არის მითითებულ რიცხვში.

2.7. მოცემული გამოსახულება ჩაწერეთ მათემატიკური სახით (კოდის დაწერა არ არის საჭირო):

$$y = (1 + x) * (x - \text{Math.Sqrt}(x - 1) / 4) / (\text{Math.Exp}(x) + 1 / (\text{Math.Pow}(x, 2) + 4))$$

2.8. მოცემულია ლოგიკური გამოსახულებები. კოდის დაწერის გარეშე გაცეით პასუხი:

კოდის ფრაგმენტი	შედეგი დაადგინეთ <i>b</i> ჭეშმარიტია თუ მცდარი?
<pre>bool b; bool x, y; x = true; y = false; b = !x; b = x && y; b = x y; b = y && (!y !x x); b = x (x & !y);</pre>	<pre>b = b = b = b = b = b =</pre>

3 C# ენის მმართველი: ამორჩევის, იტერაციისა და გადასვლის ოპერატორები

არსებობს სამი კატეგორიის მმართველი ოპერატორი:

1. ამორჩევის ორი ოპერატორი:

- იმავე შედარების, შემოწმების, კონტროლის/მართვის if ოპერატორი,
- switch-case ოპერატორი.

2. იტერაციის ოთხი ოპერატორი:

- პარამეტრიანი ციკლის for ოპერატორი,
- წინაპირობიანი ციკლის while ოპერატორი,
- პოსტპირობიანი ციკლის do-while ოპერატორი,
- ციკლის foreach ოპერატორი.

3. გადასვლის break, continue, goto და return ოპერატორი.

ამორჩევის იმავე შედარების, შემოწმების, კონტროლის/მართვის if ოპერატორი. შემოწმების if ოპერატორის ზოგადი სახე ასეთია:

```
if (პირობა იგივე ლოგიკური_გამოსახულება)
```

```
{
```

```
    // კოდი, რომელიც შესრულდება იმ შემთხვევაში,
```

```
    //თუ პირობა არის ჭეშმარიტი
```

```
}
```

პირობა არის ლოგიკური გამოსახულება, რომელიც იღებს ჭეშმარიტ (true) ან მცდარ (false) მნიშვნელობას. პირობის ადგილზე შეიძლება იყოს ნებისმიერი ლოგიკური ტიპის გამოსახულება.

ლოგიკური ტიპის მარტივი გამოსახულებებია: უდრის (==) და არუდრის (!), ასევე შედარებები ნაკლებობაზე ან მეტობაზე (<, <=, >, >=).

იმ შემთხვევაში, თუ პირობა არის მცდარი, მაშინ მოწმდება დამატებითი განშტოება და if ოპერატორის ზოგადი სახე გახდება ასეთი:

```
if (პირობა იგივე ლოგიკური_გამოსახულება) {
```

```
    // კოდი, რომელიც შესრულდება იმ შემთხვევაში, თუ პირობა არის ჭეშმარიტი
```

```
}
```

```
else {
```

```
// კოდი, რომელიც შესრულდება იმ შემთხვევაში, თუ პირობა არის მცდარი
}
```

ასევე შეგვიძლია ჩავწერთ if ოპერატორების მთელი ჯაჭვი:

```
if (პირობა_1) {
    // შემთხვევა, როცა პირობა_1 არის ჭეშმარიტი,
} else if (პირობა_2) {
    // შემთხვევა, როცა პირობა_1 არის მცდარი, ხოლო პირობა_2 კი ჭეშმარიტი,
} else if (პირობა_3) {
    // შემთხვევა, როცა პირობა_1 და პირობა_2 არის მცდარი, პირობა_3 კი ჭეშმარიტი,
} else
    { // შემთხვევა, როცა პირობა_1, პირობა_2 და პირობა_3 არის
      // მცდარი. }
```

ახლა კი განვიხილოთ მაგალითი.

გამოვითვალოთ $y(x)$ ფუნქციის მნიშვნელობა:
$$y(x) = \begin{cases} \sin(x), & x \leq a \\ \cos(x), & a < x < b, \\ \operatorname{tg}(x) & x \geq b \end{cases}$$

მოცემული გამოსახულება შეგვიძლია დავაპროგრამოთ else ოპერატორის გარეშე:

```
if (x <= a)
    fy = Math.Sin(x);
if ((x > a) && (x < b))
    fy = Math.Cos(x);
if (x >= b)
    fy = Math.Tan(x);
```

ან შეგვიძლია დავაპროგრამოთ else ოპერატორის დახმარებით:

```
if (x <= a)
    fy = Math.Sin(x);
else
    if (x < b)
        fy = Math.Cos(x);
    else
        fy = Math.Tan(x);
```

C-ის მონათესავე ენებში, მათ შორის C# ენაშიც, უდრის შედარების ოპერაცია წარმოდგენილია ორი ტოლობის ნიშნით, მაგალითად: `if (a == b)`.

განვიხილოთ მაგალითი:

```
private void button1_Click(object sender, EventArgs e)
{
    int ricxvi = int.Parse(textBox1.Text);
    if (ricxvi <= 0)
    { label1.Text = ("ნული ან უარყოფითი რიცხვი") + '\n'; }
    else if (ricxvi == 1)
    { label1.Text += ("ერთი") + '\n'; }
    else if (ricxvi == 2)
    { label1.Text += ("ორი") + '\n'; }
    else
    { label1.Text += ("დანარჩენი") + '\n'; }
}
```

ახლა კი განვიხილოთ შედარებით უფრო რთული პირობის ჩაწერის მაგალითები. პირობა შეიძლება ჩამოვაყალიბოთ ლოგიკური ტიპის უფრო რთული &&(და), ||(ან), !(არა) ოპერატორების საშუალებითაც.

მოდით ახლა გავარკვიოთ ოპერატორების პრიორიტეტები. ყველა ოპერატორს აქვს თავისი პრიორიტეტი. მაგალითად, არითმეტიკული ოპერატორებიდან ყველაზე მაღალი პრიორიტეტი აქვთ ინკრემენტის და დეკრემენტის ოპერატორებს. მათ შემდეგ სრულდება გამრავლების, გაყოფისა და ნაშთის ოპერატორები და ბოლოს სრულდება შეკრების ოპერატორები. ამ მიმდევრობის დარღვევა საჭიროებისამებრ შესაძებელია ფრჩხილების დახმარებით, როგორც მათემატიკაში.

მაგალითად, ვითვლით სამი რიცხვის საშუალო არითმეტიკულს. ამ დროს ჯერ შეკრებაა საჭირო და შემდეგ გაყოფა, თუმცა სტანდარტულად ჯერ გაყოფა სრულდება და შემდეგ შეკრება. სწორედ ამ მიზეზის გამო უნდა გამოვიყენოთ ფრჩხილები და დავწეროთ $(a + b + c)/3$, რათა ჯერ შეკრება შესრულდეს და შემდეგ გაყოფა.

შედარების ოპერატორებს აქვთ უფრო დიდი პრიორიტეტი, ვიდრე ლოგიკურ ოპერატორებს, ამიტომ მრგვალი ფრჩხილების გამოყენება ელემენტარული შედარებების დროს საჭირო არ არის. თუმცა პროგრამისტების ნაწილი თვლის, რომ სასურველია ყოველთვის მრგვალი ფრჩხილების გამოყენება შეცდომების მინიმუმამდე დაყვანის თვალსაზრისით. აი, რაც შეეხება უარყოფის ოპერატორს მას მაღალი

პრიორიტეტი აქვს და მასთან მიმართებით აუცილებლად ვიყენებთ მრგვალ ფრჩხილებს (იხ. ცხრილი 5). მე-5 ცხრილიდან გამომდინარე თავდაპირველად დამუშავდება ის ოპერაციები ან ოპერატორები რომელთაც აქვთ უფრო მაღალი პრიორიტეტი.

მაგალითად: $D = x1 + y1 * (z1 - 9)$; გამოსახულებაში:

1. პირველად შესრულდება გამოსახულება მრგვალ ფრჩხილებში $(z1 - 9)$;
2. მეორედ შესრულდება ბინარული მინუსი. მას აქვს ორი ოპერანდი: $z1$ და 9 ;
3. მესამედ შესრულდება გამრავლების * ოპერაცია;
4. მეოთხედ შესრულდება ბინარული პლუსი (შეკრება);
5. მეხუთედ შესრულდება მინიჭების = ოპერატორი. მას აქვს საკმაოდ დაბალი პრიორიტეტი.

ცხრილი 5. ოპერაციების (ოპერატორების) პრიორიტეტი

პრიორიტეტი	ოპერაციის (ოპერატორის) ნიშანი
პირველადი	$() []$. ++(პოსტფიქსური) --(პოსტფიქსური) <i>new typeof sizeof checked</i>
უნარული	+(უნარული) -(უნარული) ! ++(პრეფიქსური) --(პრეფიქსური)
გამრავლება და გაყოფა	* / %
მიმატება და გამოკლება	+ -
ნაკადი (ძვრა)	<< >>
შედარება	< > <= >=
ტოლობა	== !=
ლოგიკური და	&&
ლოგიკური ან	
პირობა	?:
მინიჭება	= *= ≠ % = += -= <<= >>=

განვაგრძოთ მაგალითების განხილვა.

მოცემული გვაქვს x წერტილი. შევამოწმოთ x წერტილი ეკუთვნის თუ არა $[a; b]$ ინტერვალს. ქვემოთ მოცემულია კოდის ფრაგმენტი:

```
int a, b; a = 5; b = 18;
int x = int.Parse(textBox1.Text);
```

```

if (a <= x && b >= x)
{ // x წერტილი ეკუთვნის [a; b] დიაპაზონს
    label1.Text = ("x წერტილი ეკუთვნის [a; b] დიაპაზონს") + '\n'; }
    else
{ // x წერტილი არ ეკუთვნის [a; b] დიაპაზონს
    label1.Text += ("x წერტილი არ ეკუთვნის [a; b] დიაპაზონს") + '\n'; }
    იგივე მოცემულობა შეიძლება მეორენაირად ჩავწეროთ უარყოფის გამოყენებით:
if ( !(x < a || x > b) ) //უარყოფა,ოპერატორი ! ფრჩხილებს გარეთაა
{ // x წერტილი ეკუთვნის [a; b] დიაპაზონს
    label1.Text += ("x წერტილი ეკუთვნის [a; b] დიაპაზონს") + '\n';}
    else
    { // x წერტილი არ ეკუთვნის [a; b] დიაპაზონს
        label1.Text += ("x წერტილი არ ეკუთვნის [a; b] დიაპაზონს") + '\n';}

```

განვიხილოთ კიდეც ერთი მაგალითი. შევადაროთ ერთმანეთს ნამდვილი (წილადი, მოძრავწერტილიანი) ტიპის მონაცემები ანუ შევამოწმოთ $0.1 + 0.2 == 0.3$ გამოსახულება ჭეშმარიტია თუ მცადრი.

```

private void button1_Click(object sender, EventArgs e)
{
    double x = 0.1, y = 0.2;
    if (x + y == 0.3)
    { label1.Text = "ტოლია"; }
    else
    { label1.Text += "არ არის ტოლი"; }
    label1.Text += (x + y).ToString();
}

```

ჩვენ ვფიქრობთ, რომ პროგრამის შესრულების შედეგი იქნება: „ტოლია“, მაგრამ სინამდვილეში კი მივიღეთ: „არ არის ტოლი“. დაისმის კითხვა: რატომ? ასეთი სახის შეცდომები თანამედროვე პროგრამირების ენებში ხშირად გვხვდება და დაკავშირებულია წილადი რიცხვების წარმოდგენის ცდომილებასთან ანუ უზუსტობასთან (ცდომილება არის განსხვავება ნამდვილ სიდიდესა და მის მიახლოებით მნიშვნელობას ან გაზომვით მიღებულ შედეგს შორის). თუ ჩვენ გავზრდით ნამდვილი (წილადი) რიცხვის ათწილადი ნაწილის თანრიგს, მაშინ მივიღებთ

შემდეგ შედეგს $0.1 + 0.2 == 0.30000000000000004$. ამდენად რადგან წილადი რიცხვებისათვის დამახასიათებელია ცდომილება, ამიტომ ასეთი შემთხვევების დროს შედარების $==$ ოპერატორი არაკორექტულია. ამიტომ უფრო სწორი იქნება მონაცემების სხვაობის მოდული შევადაროთ ჩვენთვის დასაშვებ ცდომილებას. დასაშვები ცდომილება აღვწეროთ სახელით: *alpha*.

```
private void button1_Click(object sender, EventArgs e)
{
    double alpha = 0.000001;
    double x = 0.1, y = 0.2;
    double sum = x + y;
    if (Math.Abs(sum - 0.3) < alpha)
        {label1.Text = "ტოლია";}
    else
        {label1.Text += "არ არის ტოლი";}
    label1.Text += sum.ToString();
}
```

ზემოთ მოცემული კოდის შესრულების შედეგია: „ტოლია“, რაც შეესაბამება სიმართლეს.

შეიძლება ერთი if ოპერატორი ჩადგმული ანუ მოთავსებული იყოს მეორე if ოპერატორის შიგნით. ჩადგმული if ოპერატორი იმ შემთხვევაში გამოიყენება, როცა მოქმედების შესასრულებლად საჭიროა რამდენიმე პირობის შემოწმება, რომელთა მითითება ერთ if ოპერატორში შეუძლებელია.

ჩადგმულ if ოპერატორში else ნაწილი ყოველთვის ეკუთვნის უახლოეს if ოპერატორს, თუმცა კოდის კითხვადობის გაუმჯობესების და შეცდომების თავიდან აცილების მიზნით კარგი იქნება თუ გამოვიყენებთ { } დამწყებ და დამხურავ ფიგურულ ფრჩხილებს.

განვიხილოთ ჩადგმული if ოპერატორის მაგალითი. დავწეროთ კოდი, რომელიც დაადგენს ორ რიცხვს შორის უდიდესს. თუ ისინი ტოლია, მაშინ ფორმაზე გამოიტანოს შესაბამისი შეტყობინება:

```
private void button1_Click(object sender, EventArgs e)
{
    int pirveli_ricxvi = int.Parse(textBox1.Text);
```

```

int meore_ricxvi = int.Parse(textBox2.Text);
if (pirveli_ricxvi > meore_ricxvi)
    { label3.Text = ("პირველი რიცხვი მეტია მეორე რიცხვზე"); }
    else
        { // ჩადგმული if ოპერატორი
            if (pirveli_ricxvi < meore_ricxvi)
                label3.Text += ("მეორე რიცხვი მეტია პირველ რიცხვზე");
            else
                label3.Text += ("რიცხვები ტოლია");
        }
}

private void Form1_Load(object sender, EventArgs e)
{
    label1.Text = "";
    label1.Text = "შეიტანეთ პირველი რიცხვი";
    label2.Text = "";
    label2.Text = "შეიტანეთ მეორე რიცხვი";
    label3.Text = "";
}

```

ამორჩევის **switch-case** ოპერატორი. C# ენაში switch-case კონსტრუქცია გამოჩენის დღიდანვე გახდა მისი განუყოფელი ნაწილი. იგი გამოიყენება სტრიქონის ან რიცხვითი მნიშვნელობის გადასაცემად (ცვლადი ან გამოსახულება) გარკვეული მუდმივებისათვის (ან მუდმივების თანამიმდევრობათათვის). სხვა სიტყვებით რომ ვთქვათ, ეს არის ამორჩევის (კონტროლის/მართვის) ინსტრუმენტი, რომელიც საშუალებას გაძლევთ, შეამოწმოთ ცვლადი თანატოლობაზე.

ამორჩევის **switch-case** ოპერატორის ზოგადი სახე ასეთია:

```

switch (გამოსახულება) {
    case მუდმივაA: ინსტრუქციები; break;
    case მუდმივაB: ინსტრუქციები; break;
    default: ინსტრუქციები; break;
}

```

საკვანძო სიტყვა switch-ის შემდეგ მყოფი გამოსახულება თანმიმდევრულად ეძებს case ოპერატორით მონიშნულ კონსტანტის მნიშვნელობას და თუ მოხდება მათ შორის თანხვედრა (ტოლობა), პროგრამა გადადის შესაბამის ოპერატორთან და ასრულებს კოდის ბლოკს.

case ინსტრუქციების ყველა ცალკეულ ბლოკს მოჰყვება გადასვლის ოპერატორი: break. მისი გამოყენებისას კონსტრუქციის სხვა ნაწილები/სექციები, რომლებიც აღნიშნულია case საკვანძო სიტყვით, არ შესრულდება. თუ case-ის არც ერთმა განშტოებამ არ იმუშავა, მაშინ პროგრამა გადადის default ოპერატორის შესრულებაზე. default ოპერატორის დახმარებით შეგვიძლია დავამუშაოთ სიტუაცია, როდესაც თანხვედრა არ არის ნაპოვნი.

იმ შემთხვევაში, თუ default ბლოკი არ არსებობს ან თუ არ ხდება თანხვედრა გამოსახულების მნიშვნელობასთან, მაშინ არანაირი მოქმედება არ შესრულდება.

switch-case ოპერატორში გამოსახულებას უნდა გააჩნდეს მთელი ტიპი - char, byte, short ან int ან სტრიქონული ტიპი - string. გამოსახულების მნიშვნელობას არ შეიძლება ჰქონდეს ნამდვილი, წილადი ტიპი (double, float).

switch-case ოპერატორის გამოსახულების ტიპი და case განშტოებების მუდმივების ტიპი უნდა იყოს ერთიდაიგივე ანუ უნდა ჰქონდეთ თანხვედრა.

ჩადგმული switch ოპერატორი. ერთი switch ოპერატორი შეიძლება მოთავსებული იყოს გარე switch ოპერატორში. შიდა switch ოპერატორს ეწოდება ჩადგმული. შიდა და გარე switch ოპერატორების case განშტოებების მუდმივებს შეიძლება ერთნაირი მნიშვნელობები ჰქონდეთ. ეს არ გამოიწვევს კონფლიქტს პროგრამის შესრულების დროს.

დავწეროთ მარტივი კალკულატორის კოდი if-else და switch-case ოპერატორების გამოყენებით და შევადაროთ ერთმანეთს ორივე პროგრამა.

მარტივი კალკულატორის კოდი if-else ოპერატორების გამოყენებით:

```
int pirveli_ricxvi = int.Parse(textBox1.Text);
int meore_ricxvi = int.Parse(textBox3.Text);
char operacia = char.Parse(textBox2.Text);
int shedegi = 0;
if (operacia == '+') { shedegi = pirveli_ricxvi+ meore_ricxvi; }
else
```

```

if (operacia == '-') { shedegi = pirveli_ricxvi-meore_ricxvi; }
else
    if (operacia == '*') { shedegi =
        pirveli_ricxvi*meore_ricxvi;}
    else
        if (operacia == '/' || operacia == ':')
            { shedegi = pirveli_ricxvi / meore_ricxvi; }
            else
                if (operacia == '%')
                    {shedegi = pirveli_ricxvi%meore_ricxvi; }
label1.Text = shedegi.ToString();

```

მარტივი კალკულატორის კოდი switch-case ოპერატორების გამოყენებით:

```

int pirveli_ricxvi= int.Parse(textBox1.Text);
int meore_ricxvi = int.Parse(textBox3.Text);
char operacia = char.Parse(textBox2.Text);
int shedegi;
switch (operacia) {
    case '+': shedegi=pirveli_ricxvi+meore_ricxvi; break;
    case '-': shedegi=pirveli_ricxvi-meore_ricxvi; break;
    case '*': shedegi=pirveli_ricxvi*meore_ricxvi; break;
    case '/':
    case ':': shedegi=pirveli_ricxvi/meore_ricxvi; break;
    case '%': shedegi=pirveli_ricxvi%meore_ricxvi; break;
    default: shedegi = 0; break;
}

label1.Text = shedegi.ToString();

```

პარამეტრიანი ციკლის for ოპერატორი. პარამეტრიანი ციკლის for ოპერატორი განკუთვნილია ციკლური პროცესების შესასრულებლად.

ციკლის for ოპერატორის გამოყენებით შესაძლებელია ნებისმიერი ტიპის ციკლური, მრავალჯერადი, მრავალგანმეორებადი პროცესის აგება, რომელშიც:

ა) ციკლის იტერაციების რაოდენობა წინასწარ ცნობილია;

ბ) ციკლის იტერაციების რაოდენობა უცნობია და განისაზღვრება გარკვეული პირობის შესრულების საფუძველზე.

რა არის იტერაცია? პასუხი: იტერაცია არის ციკლის ტანის ერთხელ შესრულება. პარამეტრიანი ციკლის for ოპერატორის ზოგადი სინტაქსი შემდეგი სახისაა:

for (ცვლადი = გამოსახულება_1; პირობა; გამოსახულება_2)

{ ინსტრუქციები, ოპერატორები }

ცვლადი არის პარამეტრიანი ციკლის for ოპერატორის პარამეტრი, იგივე მთვლელიცვლადი. პარამეტრი შეიძლება იყოს int ან char ტიპისა.

გამოსახულება_1 წარმოადგენს ციკლის პარამეტრის საწყის მნიშვნელობას, იმავე ციკლის პირობის საწყისი მნიშვნელობაა. იგი შესრულებდა მხოლოდ ერთხელ ციკლის დასაწყისში. ჩანაწერს „ცვლადი = გამოსახულება_1;“ ასევე შეგვიძლია ვუწოდოთ პარამეტრის, იმავე მთვლელიცვლადის ინიციალიზება. პარამეტრი, იგივე მთვლელიცვლადი განსაზღვრავს იმ იტერაციების რაოდენობას, რომელიც უნდა შესრულდეს პარამეტრიანი for ციკლის მიერ.

პირობა არის ლოგიკური გამოსახულება, რომელის შემოწმება ხორციელდება პარამეტრიანი ციკლის ყოველი შესაძლო შესრულების წინ. თუ **პირობამ** გასცა **მნიშვნელობა true**, მაშინ სრულდება ციკლის შემდეგი იტერაცია. თუ **პირობამ** გასცა **მნიშვნელობა false**, მაშინ ხდება ციკლის შეწყვეტა და გადასვლა იმ ოპერატორზე, რომელიც აღწერილია for-ის შემდეგ.

გამოსახულება_2 არის გამოსახულება, რომელიც ცვლის პარამეტრის ანუ მთვლელიცვლადის მნიშვნელობას, მას სხვანაირად ბიჯს ან იტერატორსაც უწოდებენ. (ბიჯი არის საზომი ერთეული ძველ საქართველოში. შეადგენდა ორ ნაბიჯს. თავდაპირველად უდრიდა 1.4 მეტრს, შემდგომ 1 ადლს (დაახლოებით 101.15 სმ)). პროგრამირებაში კი ბიჯი წარმოადგენს ერთ იტერაციას ანუ ერთი იტერაციის შესრულება უდრის ერთ ბიჯს.

განვიხილოთ პარამეტრიანი ციკლის for ოპერატორის ჩაწერის ვარიანტები:

```
1. for (int i = 0; i < 10; ++i) // პარამეტრი i იზრდება ერთით 0 1 2 3  
4 5 6 7 8 9
```

```
label1.Text += " i= " + i.ToString()+ '\n';
```

```
2. for (int i = 10; i >= 0; --i) //პარამეტრი i მცირდება ერთით 10 9 8 7 6  
5 4 3 2 1 0
```

```
label1.Text += " i= " + i.ToString() + '\n';
```

```
3. for (int i = 0; i < 10; i += 2) // პარამეტრი i იზრდება 2-ის ბიჯით 0
    2 4 6 8
```

```
    label1.Text += " i= " + i.ToString() + '\n';
```

```
4. for (char i ='a'; i < 'z'; ++i) //ციკლი ლათინური სიმბოლოებით a-
    დან z-მდე
```

```
    label1.Text += " i= " + i.ToString() + '\n';
```

ვთქვათ, მოცემული გვაქვს მთელ ციფრთა მიმდევრობა 1-დან 10-მდე 1-ის ბიჯით. დავწეროთ კოდი, რომელიც დაადგენს ამ მიმდევრობის ნამრავლს:

```
int namravli = 1;
```

```
for (int i = 1; i <= 10; ++i)
```

```
    {    namravli *= i;    }
```

```
label1.Text="1-დან 10-მდე მიმდევრობის ნამრავლია " +
namravli.ToString();
```

პარამეტრიანი ციკლის (for) მუშაობა აღვწეროთ პუნქტებად. როგორც ვხედავთ გვაქვს ხუთი პუნქტი. მაშ ასე, ციკლი მუშაობს შემდეგნაირად:

1. ხდება ციკლის i პარამეტრის, იმავე მთვლელი ცვლადის ინიციალიზება: `int i = 1;`

2. შემდეგ ხდება პირობის შემოწმება `i <= 10;`

- შედარების შედეგი თუ არის true, გადავდივართ შემდეგ ბიჯზე;
- შედარების შედეგი თუ არის არის false, მაშინ გამოვდივართ ციკლიდან;

3. შემდეგ სრულდება ციკლის ტანში (ბლოკში) მოცემული ინსტრუქცია, ჩვენს შემთხვევაში `namravli *= i;` გამოსახულება;

4. შემდეგ ხდება პარამეტრის, მთვლელიცვლადის `++i;` ცვლილება (ერთით გაზრდა);

5. გადავდივართ მე-2 პუნქტზე. ყურადღება მიაქციეთ იმას, რომ პირველი პუნქტის შესრულება ხდება მხოლოდ ერთხელ.

პარამეტრიანი ციკლის for ოპერატორი შეიძლება შეგვხვდეს შემდეგი სახითაც:

პირველი ვარიანტი:

```
// გამოვთვალოთ მიმდევრობის 2+4+8+...+100 ჯამი
```

```
int sum = 0;
```

```
int i = 0;
```


// ციკლში არ ხდება ციკლის მთვლელის (მთვლელი ცვლადის)

ინიციალიზება

```
for (; i <= 100; i += 2)
    sum += i;    // sum = 2550
```

მეორე ვარიანტი:

// გამოვთვალოთ მიმდევრობის $a*(a+1)*\dots*(a+n-1)$ ჯამი

```
int namravli; int i;
```

```
int n, a;
```

```
n = 5; a = 4; namravli = 1;
```

//ციკლში არ არის მითითებული მთვლელი ცვლადი (ანუ იტერატორი)

```
for (i = 0; i < n;)
```

```
{ namravli = namravli * (a + i); // namravli = 6720
```

```
++i; }
```

მოდით ახლა გავარკვიოთ ერთი საკითხი პარამეტრიან ციკლებთან დაკავშირებით. ზოგი პროგრამისტი პარამეტრიან ციკლის მთვლელ ცვლადს წერს შემდეგი სახით:

```
for (int i=1; i <= 10; i++),
```

ზოგი პროგრამისტი კი შემეგი სახით:

```
for (int i=1; i <= 10; ++i).
```

სინამდვილეში ორივე ჩანაწერი სწორია, ორივე ჩანაწერი $++i$ და $i++$ ზრდის პარამეტრის (მთვლელი ცვლადის) მნიშვნელობას ერთით, მაგრამ მუშაობის მხრივ მათ შორის არის განსხვავება და ეს განსხვავება გამოიხატება სისწრაფეში.

ჩანაწერი $++i$ მუშაობს უფრო სწრაფად, ვიდრე ჩანაწერი $i++$.

ჩანაწერი $++i$ -ის შემთხვევაში მონაცემი იზრდება ერთით და ბრუნდება შედეგი, ხოლო $i++$ -ის შემთხვევაში კი ჯერ აიღება მონაცემი, შემდეგ იგი იწახება დროებით ცვლადში, შემდეგ მონაცემი იზრდება ერთით და ბოლოს მონაცემი ბრუნდება დროებითი ცვლადიდან.

მოკლედ რომ ვთქვათ: ოპერაციის პოსტიფიქსური ფორმა აბრუნებს თავდაპირველ, ძველ მნიშვნელობას, ხოლო პრეფიქსური ფორმა კი ახალ მნიშვნელობას. მაგალითად:

/* ინკრემენტის პოსტფიქსური ოპერატორი i -ის ზრდის ერთით

და აბრუნებს საწყისს, ძველ მნიშვნელობას ანუ 0-ს */

```
int i = 0;
```

```
int k = i++; // i = 1, k = 0
```

/* ინკრემენტის პრეფიქსური ოპერატორი i-ის ზრდის ერთით

და აბრუნებს ახალ მნიშვნელობას ანუ 2-ს */

```
int n = ++i; // i = 2, n = 2
```

ამდენად შეიძლება ვთქვათ, რომ ოპერაციის პრეფიქსური ნიშანი შედარებით უფრო ეფექტურია, ვიდრე ოპერაციის პოსტფიქსური ნიშანი, რადგან არ ქმნის ახალ ობიექტს ცვლადის მნიშვნელობის დროებით შესანახად. პარამეტრიანი ციკლის ჩაწერის მაგალითი ოპერაციის პოსტფიქსური ნიშნით:

```
for (int i = 0; i < 7; ++i)
```

```
{
```

```
    label1.Text += i.ToString() + '\n'; }
```

ჩადგმული for ოპერატორი. პარამეტრიანი ციკლის ერთი ოპერატორი შეიძლება იყოს ჩადგმული ციკლის მეორე ოპერატორში. განვიხილოთ ჩადგმული ციკლები კოდის მეშვეობით. დავწეროთ პროგრამული კოდი, რომელიც დაბეჭდავს ციფრებს ქვემოთ მოცემული სქემის სახით:

```
1
```

```
2 2
```

```
3 3 3
```

```
4 4 4 4
```

```
5 5 5 5 5
```

შესაბამისი პროგრამული კოდი:

```
for (int i = 1; i <= 5; ++i)
```

```
{ //for ციკლში ჩადგმულია მეორე for ციკლი
```

```
    for (int j = 1; j <= i; ++j)
```

```
        label1.Text += i.ToString();
```

```
        label1.Text += '\n';    }
```

წინაპირობიანი ციკლის while ოპერატორი. წინაპირობიანი ციკლი while გამოიყენება ერთი ოპერატორის ან ოპერატორების ბლოკის მრავალჯერ შესასრულებლად.

მისი სინტაქსია:

```
while ( პირობა )
```

```

{
    ციკლის კოდი
}

```

პირობა არის ლოგიკური გამოსახულება, რომელიც გასცემს true ან false მნიშვნელობას. თუ ის იღებს false მნიშვნელობას, მაშინ ხდება ციკლიდან გამოსვლა და სრულდება მომდევნო ოპერატორი. თუ ის იღებს true მნიშვნელობას, მაშინ შესრულდება ციკლის კოდი.

პოსტპირობიანი ციკლის do-while ოპერატორი. პოსტპირობიანი ციკლის do-while ოპერატორი გამოიყენება ერთი ოპერატორის ან ოპერატორების ბლოკის მრავალჯერ შესასრულებლად. for და while ციკლებისაგან განსხვავებით, რომლებშიც ჯერ მოწმდება პირობა და შემდეგ სრულდება ციკლის კოდი, do-while ციკლში ჯერ შესრულდება ციკლის კოდი, შემდეგ კი მოწმდება პირობა. მისი სინტაქსია:

```

do
{
    ციკლის კოდი
}
while ( პირობა );

```

თუ ციკლში ერთი ოპერატორი სრულდება, მაშინ ფიგურული ფრჩხილების გამოყენება აუცილებელი არ არის, მაგრამ მათ ხშირად იყენებენ do-while ციკლის კითხვადობის გაუმჯობესების მიზნით, რადგან ის ადვილად შეიძლება აგვერიოს while ციკლში.

მოდით ახლა უფრო დაწვრილებით განვიხილოთ წინაპირობიანი (while) და პოსტპირობიანი (do-while) ციკლები და ჩავატაროთ შედარებითი ანალიზი.

ციკლში while ჯერ მოწმდება პირობა და თუ პირობა არის ჭეშმარიტი, მაშინ სრულდება ციკლის ტანში არსებული ოპერატორები. აუცილებლად შემოწმეთ პირობის ცვლადი, იგი უნდა იცვლებოდეს, საწინააღმდეგო შემთხვევაში ციკლი while გახდება უსასრულო.

```

private void button1_Click(object sender, EventArgs e)
{
    label1.Text = ""; int i = 0; int cifri = 3;
    while (i <= cifri)

```

```

    {
        label1.Text += "while-ის იტერაცია N - " + i.ToString() +
        \n';
        ++i; //პირობის ცვლადის ცვლილება, ერთით გაზრდა
    }
}

```

ციკლი while-ისგან განსხვავებით ციკლში do-while ჯერ სრულდება ციკლის ტანი და შემდეგ ხდება პირობის შემოწმება. ამდენად პირობისაგან დამოუკიდებლად ციკლის ტანი აუცილებლად შესრულდება ერთხელ მაინც:

```

private void button1_Click(object sender, EventArgs e)
{
    label1.Text = ""; int i = 0; int ricxvi = 3;
    do
    {
        label1.Text += "do-while ციკლის იტერაცია N - " + i.ToString() + '\n';
        ++i; //პირობის ცვლადის ცვლილება, ერთით გაზრდა
    } while (i <= ricxvi);
}

```

ციკლები while და do-while მოკლედ შეგვიძლია ასე განვსაზღვროთ:

- while - თავდაპირველად ჯერ შეამოწმე, შემდეგ შეასრულე და ა.შ. ციკლში;
- do-while - თავდაპირველად ჯერ შეასრულე, შემდეგ შეამოწმე და ა.შ. ციკლში.

ჩვენ ყოველთვის შეგვიძლია პროგრამულ კოდში ციკლის ერთი ოპერატორი ჩავანაცვლოთ ციკლის მეორე ოპერატორით, მაგრამ პროგრამირების კარგ სტილად ჩაითვლება თუ გავითვალისწინებთ შემდეგ რეკომენდაციებს: ა) ციკლი for გამოიყენება იმ შემთხვევაში, როდესაც იტერაციების რაოდენობა წინასწარ არის ცნობილი (ვიცით მთვლელიცვლადის იტერაციების რაოდენობა); ბ) ციკლები while და do-while გამოიყენება იმ შემთხვევაში, როდესაც იტერაციების რაოდენობა წინასწარ ცნობილი არ არის. while გამოიყენება იმ შემთხვევაში, როდესაც არსებობს იმის შესაძლებლობა, რომ ციკლი არ შესრულდეს არც ერთხელ, ხოლო do-while უნდა გამოიყენოთ იმ შემთხვევაში, თუ ცნობილია, რომ საჭიროა სულ მცირე ერთი იტერაციის შესრულება.

მოდით უფრო დაწვრილებით განვიხილოთ, თუ რა შემთხვევებში არის უფრო მისაღები, გამოვიყენოთ ესა-თუ ის ციკლის ოპერატორი. მაგალითად:

ა) მოცემული გვაქვს N რიცხვი. გამოვიტანოთ ის ეკრანზე (N განსაზღვრავს იტერაციების რაოდენობას) - მოცემულობა შეგვიძლია ამოვხსნათ პარამეტრიანი ციკლის (for) დახმარებით;

ბ) მოვძებნოთ ორი რიცხვის უდიდესი საერთო გამყოფი ევკლიდეს ალგორითმის გამოყენებით (ერთ-ერთი რიცხვი თუ იქნება 0-ის ტოლი, მაშინ არც ერთი იტერაცია არ გვექნება) - მოცემულობა შეგვიძლია ამოვხსნათ წინაპირობიანი ციკლის (while) დახმარებით;

გ) ჩაფიქრებული რიცხვის გამოიცნობა ანუ თამაშის „გამოიცანი რიცხვი“ პროგრამული რეალიზება (ყოველთვის საჭიროა მინიმუმ ერთი მცდელობა, რადგან ზუსტად არ ვიცით იტერაციების რაოდენობა) - ამ შემთხვევაში მოცემულობა შეგვიძლია ამოვხსნათ პოსტპირობიანი ციკლის (do-while) დახმარებით.

მოდით ახლა ორიოდ სიტყვით განვიხილოთ **უსასრულო ციკლი**. უსასრულო, მუდმივი ციკლის შექმნა საკმაოდ მარტივია.

მაგალითად, პარამეტრიანი ციკლის შემთხვევაში უსასრულო ციკლს ექნება შემდეგი სახე: **for (; ;)**. ასევე თუ **while** და **do-while** ციკლების პირობა თუ არასდროს გახდება **false**, მაშინ ციკლი შესრულდება უსასრულოდ ანუ პროგრამისტების ენაზე „ჩაიციკლება“. ასეთი ხასიათის ციკლებს უწოდებენ უსასრულოს:

```
// უსასრულო for ციკლი
int i = 0;
for ( ; ; )
{ //for ციკლის ტანი }
// უსასრულო while ციკლი
while (true)
{ //while ციკლის ტანი
}
// უსასრულო do-while ციკლი
do
{ //do-while ციკლის ტანი
} while (true);
```

უსასრულო ციკლები საკმაოდ სასრებლოა პროგრამირების სამყაროში. მაგალითად: საჭიროა, რომ ციკლი შესრულდეს იმ დროის განმავლობაში, სანამ პროგრამა მუშაობს. აი მეორე მაგალითი: ვთქვათ, გვაქვს პროგრამა, რომელიც უზრუნველყოფს ნახატების ანიმაციას, ანიმაციამ მანამდე უნდა იტრიალოს ეკრანზე, სანამ ჩვენ თვითონ არ დავაპაუზებთ (ანუ გავაჩერებთ). სწორედ ამ დროს საჭიროა უსასრულო ციკლი, რომ ანიმაცია არ გაჩერდეს.

ახლა განვიხილოთ ჩადგმული ციკლების კიდევ ერთი მაგალითი, სადაც ნაჩვენებია while ციკლში ჩადგმული for ციკლი. მოცემულია მიმდევრობა 1-დან 300-მდე. დავწეროთ პროგრამული კოდი, რომელიც იპოვის ამ მიმდევრობაში ისეთ რიცხვებს, რომლებთაც აქვთ ზუსტად 10 ცალი გამყოფი.

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "";

    /* ვიპოვოთ 1-დან 300-ის ჩათვლით მიმდევრობაში ისეთი
       რიცხვები, რომელთაც აქვთ 10 ცალი გამყოფი*/
    int number;
    int i, k;
    number = 1;
    while (number <= 300)
    {
        k = 0;

        // ციკლი for ჩადგმულია while ციკლში
        for (i = 1; i <= number; ++i)
            if (number % i == 0) ++k;

            if (k == 10)
                label1.Text += "10 ცალი გამყოფი აქვს რიცხვს: " +
                    number.ToString() + '\n';
                ++number;
    }
}
```

გადასვლის break, continue და goto ოპერატორები. break ოპერატორი გვაძლევს ციკლის შესრულების იძულებით შეწყვეტისა და ციკლიდან გამოსვლის

საშუალებას. ამ დროს, ციკლის დანარჩენი ოპერატორები არ შესრულდება. მართვა გადაეცემა ციკლის შემდეგ მოთავსებულ ოპერატორს.

continue ოპერატორი საშუალებას გვაძლევს იძულებით გადავიდეთ მომდევნო იტერაციაზე. **continue** ოპერატორის შემდეგ მოთავსებული ოპერატორები არ შესრულდება.

goto არის უპირობო გადასვლის ოპერატორი. მისი სინტაქსია: **goto ჭდე;**

სადაც, ჭდე არის ამომცნობი (იდენტიფიკატორი), რომელსაც ორი წერტილი (:) მოსდევს. ის მიუთითებს იმ ადგილს, საიდანაც უნდა გაგრძელდეს პროგრამის შესრულება.

ახლა განვიხილოთ რამდენიმე ამოცანა.

ამოცანა 3.1. მოცემულია მთელი ტიპის რიცხვი. **3.1.ა.** დავწეროთ პროგრამული კოდი, რომელიც დაადგენს მთელი რიცხვი კენტია თუ ლუწი. **3.1.ბ.** დავწეროთ პროგრამული კოდი, რომელიც დაადგენს მთელი რიცხვი არის თუ არა 5-ის ჯერადი.

3.1.ა. პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e)
{
    int ricxvi;
    label1.Text = "";
    ricxvi = int.Parse(textBox1.Text);
    if (ricxvi % 2 == 1)
        label1.Text = "რიცხვი კენტია";
    else
        label1.Text = "რიცხვი ლუწია";
}
```

3.1.ბ. პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e)
{
    int ricxvi;
    label1.Text = "";
    ricxvi = int.Parse(textBox1.Text);
    if (ricxvi % 5 == 0)
        label1.Text = "რიცხვი 5-ის ჯერადია";
}
```

```
else
```

```
label1.Text = "რიცხვი 5-ის ჯერადი არ არის";
```

```
}
```

ამოცანა 3.2. მოცემულია სამი მთელი ტიპის რიცხვი. დავწეროთ პროგრამული კოდი, რომელიც დაადგენს სამ რიცხვს შორის მაქსიმალურს.

პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
double ricxvi1, ricxvi2, ricxvi3, maximumi;
```

```
label1.Text = "";
```

```
ricxvi1 = int.Parse(textBox1.Text);
```

```
ricxvi2 = int.Parse(textBox2.Text);
```

```
ricxvi3 = int.Parse(textBox3.Text);
```

```
maximumi = Math.Max(ricxvi1, Math.Max(ricxvi1, ricxvi2));
```

```
label1.Text="სამ რიცხვს შორის უდუდესია რიცხვი "+ maximumi.ToString();}
```

ამოცანა 3.3. მოცემულია მთელი ტიპის x რიცხვი. დავწეროთ პროგრამული კოდი, რომელიც დაადგენს მთელი ტიპის x რიცხვი იმყოფება თუ არა მითითებულ $10 \leq x \leq 19$ დიაპაზონში. პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
int ricxvi;
```

```
label1.Text = "";
```

```
ricxvi = int.Parse(textBox1.Text);
```

```
if ((ricxvi >= 10) && (ricxvi <= 19))
```

```
label1.Text = "რიცხვი იმყოფება მითითებულ დიაპაზონში";
```

```
else label1.Text = "რიცხვი არ იმყოფება მითითებულ დიაპაზონში";}
```

ამოცანა 3.4. მოცემული გვაქვს სამი მთელი რიცხვი (როგორც დადებითი, ასევე უარყოფითი). დავწეროთ პროგრამული კოდი, რომელიც დაადგენს დადებითი და უარყოფითი რიცხვების რაოდენობას. გავითვალისწინოთ, რომ შეიძლება შეგვხვდეს ნულიც. პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
int dadebiTi = 0, uaryofiTi = 0, nuli = 0;
```



```

int A = int.Parse(textBox1.Text);
int B = int.Parse(textBox2.Text);
int C = int.Parse(textBox3.Text);
if (A == 0) ++nuli;
else
    if (A > 0)
        ++dadebiTi;
    else
        ++uaryofiTi;
if (B == 0) ++nuli;
else
    if (B > 0)
        ++dadebiTi;
    else
        ++uaryofiTi;
if (C == 0) ++nuli;
else
    if (C > 0)
        ++dadebiTi;
    else
        ++uaryofiTi;
label1.Text = "დადებითია " + dadebiTi.ToString() + " რიცხვი";
label2.Text = "უარყოფითია " + uaryofiTi.ToString() + " რიცხვი";
label3.Text = "ნულოვანია " + nuli.ToString() + " რიცხვი";
}

```

ამოცანა 3.5. განვიხილოთ **switch-case** ოპერატორის მუშაობა. დაწერეთ პროგრამული კოდი, რომლიც დაადგენს ქართული ანბანის ასო-ბგერა არის თუ არა ხმოვანი. პროგრამული კოდი:

```

private void button1_Click(object sender, EventArgs e)
{
    label1.Text = ""; char simbolo;
    simbolo = textBox1.Text[0];
    switch (simbolo)
    {

```

```

        case 'ა':
        case 'ე':
        case 'ო':
        case 'ლ':
        case 'უ': label1.Text = "ეს არის ხმოვანი ასო-ბგერა";
    break;
    default: label1.Text = "ეს არ არის ხმოვანი ასო-ბგერა";
    break;
}
}

```

ამოცანა 3.6. იმ შემთხვევაში თუ კოდში არ არის default ბლოკი და გამოსახულების მნიშვნელობასთან არ არის case მუდმივას დამთხვევა, მაშინ კოდი არაფერს შეასრულებს, როგორც ქვევით არის ნაჩვენები:

```

private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "";
    int x; x = 7;
    switch (x)
    {
        case 1: label1.Text = "one"; break;
        case 2: label1.Text = "two"; break;
        case 3: label1.Text = "three"; break;
        case 4: label1.Text = "four"; break;
        case 5: label1.Text = "five"; break;
    }
}

```

ამოცანა 3.7. ქვემოთ მოცემული კოდის მოცემულობა დაწერეთ დამოუკიდებლად. ასევე დამოუკიდებლად ააგეთ ფორმის დიზაინი. იგი უნდა იყოს კითხვადი და მარტივი. პროგრამული კოდი:

```

private void button1_Click(object sender, EventArgs e)
{
    int kviris_dge = int.Parse(textBox1.Text);
    int shedegi = 0;
}

```

```

    if ((kviris_dge >= 1) && (kviris_dge <= 5)) shedegi = 1;
    if ((kviris_dge >= 6) && (kviris_dge <= 7)) shedegi = 2;
    switch (shedegi)
    {
        case 1: label2.Text = "სამუშაო დღეა"; break;
        case 2: label2.Text = "დასვენების დღეა"; break;
        default:
            label2.Text = "არასწორი მონაცემი!!!! ";
            label2.ForeColor = Color.Red; break;
    }
}

private void Form1_Load(object sender, EventArgs e)
{
    label1.Text = "შეიტანეთ კვირის დღის დასახელების რიცხვი:" + '\n';
    label1.Text += "1 - ორშაბათი" + '\n';
    label1.Text += "2 - სამშაბათი" + '\n';
    label1.Text += "3 - ოთხშაბათი" + '\n';
    label1.Text += "4 - ხუთშაბათი" + '\n';
    label1.Text += "5 - პარასკევი" + '\n';
    label1.Text += "6 - შაბათი" + '\n';
    label1.Text += "7 - კვირა" + '\n';
}

```

ამოცანა 3.8. ქვემოთ მოცემული კოდის მოცემულობა დაწერეთ დამოუკიდებლად. ასევე დამოუკიდებლად ააგეთ ფორმის დიზაინი. იგი უნდა იყოს კითხვადი და მარტივი. პროგრამული კოდი:

```

private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "";
    string day;
    day = textBox1.Text;
    switch (day)
    {
        case "ორშაბათი":

```

```

        label1.Text = "Numberofday = 1"; break;
    case "სამშაბათი":
        label1.Text = "Numberofday = 2"; break;
    case "ოთხშაბათი":
        label1.Text = "Numberofday = 3"; break;
    case "ხუთშაბათი":
        label1.Text = "Numberofday = 4"; break;
    case "პარასკევი":
        label1.Text = "Numberofday = 5"; break;
    case "შაბათი":
        label1.Text = "Numberofday = 6"; break;
    case "კვირა":
        label1.Text = "Numberofday= 7"; break;
    default:
        label1.Text = "არასწორი დასახელება!! შეიტანეთ თავიდან";
        label1.ForeColor = Color.Red;
        Font MediumFont = new Font("Arial", 10);
        label1.Font = MediumFont; break;
    }
}

```

ამოცანა 3.10. დავწეროთ პროგრამული კოდი, რომელიც თვის ნომრის მიხედვით განსაზღვრავს წელიწადის დროს. პროგრამული კოდი:

```

private void button1_Click(object sender, EventArgs e)
{
    int M = int.Parse(textBox1.Text);
    switch (M)
    {
        case 12:
        case 1:
        case 2:
            label2.Text = "ზამთარი";
            label2.ForeColor = Color.Blue;
            break;
    }
}

```

```

case 3:
case 4:
case 5:
    label2.Text = "გაზაფხული";
    label2.ForeColor = Color.Green;
    break;
case 6:
case 7:
case 8:
    label2.Text = "ზაფხული";
    label2.ForeColor = Color.GreenYellow;
    break;
case 9:
case 10:
case 11:
    label2.Text = "შემოდგომა";
    label2.ForeColor = Color.Gold;
    break;
    default: label2.Text = "არასწორია!! თვის რიცხვითი
მნიშვნელობა არის [1;12] დიაპაზონში";
    label2.ForeColor = Color.Red;
    Font MediumFont = new Font("Arial", 14);
    label2.Font = MediumFont; break;
    }
}

private void Form1_Load(object sender, EventArgs e)
{
    label2.Text = "";
    Font MediumFont1 = new Font("Arial", 18);
    label2.Font = MediumFont1;
    label1.Text = "შეიტანეთ მიმდინარე თვის რიგითი ნომერი:";
    Font MediumFont = new Font("Arial", 12);
    label1.Font = MediumFont;
}

```

ამოცანა 3.10-ის შესრულების ერთ-ერთი ვარიანტი (იხ. ნახ. 7):

შეიტანეთ მიმდინარე თვის რიგითი ნომერი:

ამოცანა 3.10

არასწორია!! თვის რიცხვითი მნიშვნელობა განეკუთვნება [1;12] დიაპაზონს

ნახ. 7. ამოცანა 3.10-ის შესრულების ერთ-ერთი შედეგი

ამოცანა 3.11. განვიხილოთ პარამეტრიანი ციკლის for ოპერატორის მუშაობა.

3.11.ა. მოცემულია მთელ რიცხვთა მიმდევრობა 1-დან 10-ის ჩათვლით. დავწეროთ პროგრამული კოდი, რომელიც დაბეჭდავს მიმდევრობის კვადრატულ მნიშვნელობებს, ბიჯი უდრის ერთს. პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e)
{
    double kvadrati;
    label1.Text = "";
    for (int ricxvi = 1; ricxvi <= 10; ++ricxvi)
    {
        kvadrati = Math.Pow(ricxvi, 2);
        label1.Text += ricxvi.ToString() + " " +
            kvadrati.ToString() + '\n';
    }
}
```

3.11.ბ. მოცემულია მთელ რიცხვთა მიმდევრობა 100-დან 1-მდე. დავწეროთ პროგრამული კოდი, რომელიც დაბეჭდავს მიმდევრობის კვადრატულ მნიშვნელობებს პარამეტრიანი ციკლის დახმარებით, ბიჯი უდრის ათს. პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e){
    double kvadrati;
    label1.Text = "";
    for (int ricxvi = 100; ricxvi > 1; ricxvi -= 10)
    {
        kvadrati = Math.Pow(ricxvi, 2);
        label1.Text += ricxvi.ToString() + " " + kvadrati.ToString()
            + '\n';
    }
}
```

ამოცანა 3.12. პარამეტრიანი ციკლის for ოპერატორის განხილვა იტერაციების მიხედვით:

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "";
    for (int i = 1; i <= 5; ++i)
    {
        label1.Text += "for ციკლი C# ენაში - იტერაცია = " + i.ToString() + '\n';
    }
}
```

ამოცანა 3.12-ის შესრულების შედეგი (იხ. ნახ. 8):

```
for ციკლი C# ენაში - იტერაცია = 1
for ციკლი C# ენაში - იტერაცია = 2
for ციკლი C# ენაში - იტერაცია = 3
for ციკლი C# ენაში - იტერაცია = 4
for ციკლი C# ენაში - იტერაცია = 5
```

ნახ. 8. ამოცანა 3.12-ის შესრულების შედეგი

მოდიოთ ავხსნათ ციკლის მუშაობა ამოცანა 3.12.-ის პროგრამულ კოდში:

- პარამეტრი, იგივე მთვლელიცვლადი არის `int i = 1`;
- პირობა არის `i <= 5`;
- ბიჯი, იტერატორი არის `++ i`;

პროგრამის შესრულებაზე გაშვებისას მიმდინარეობს შემდეგი პროცესი:

- თავდაპირველად ხდება `i` ცვლადის გამოცხადება და ინიციალიზება. ამ შემთხვევაში `i` ცვლადი ასრულებს მთვლელიცვლადის როლს. მას მიენიჭა მნიშვნელობა 1.
- შემდეგ ხდება პირობის შემოწმება `i <= 5`.
- თუ პირობის შემოწმება აბრუნებს მნიშვნელობას `true`, მაშინ სრულდება ციკლის ტანი ან ციკლის ბლოკი და დაიბეჭდება "for ციკლი C# ენაში - იტერაცია = 1".
- შემდეგ განისაზღვრება იტერატორის `++ i` მნიშვნელობა, იზრდება ერთით და იგი გახდება 2-ის ტოლი.
- შემდეგ ისევ ხდება პირობის `i <= 5` შემოწმება და კვლავ მოხდება იტერატორის გაზრდა ერთით. პირველი ხუთი იტერატორი ფასდება მნიშვნელობით `true`.

- როგორც კი მთვლეელი ცვლადის i მნიშვნელობა გახდება 6-ის ტოლი, პირობა დააბრუნებს false მნიშვნელობას. შესაბამისად ციკლის მუშაობაც დასრულდება.

ამოცანა 3.13. დავწეროთ პროგრამული კოდი, რომელიც იპოვის n ცალი ნატურალური რიცხვების ჯამს. გამოვიყენოთ პარამეტრიანი ციკლის for ოპერატორი.

```
private void button1_Click(object sender, EventArgs e) {
    label1.Text = ""; int n = 5, sum = 0;
    for (int i = 1; i <= n; ++i)
    {
        sum += i; //sum = sum + i;
    }
    label1.Text = " პირველი " + n.ToString() +
        " ნატურალური რიცხვის ჯამი = " + sum.ToString() + '\n'; }
```

ამოცანა 3.13.-ის კოდის შესრულების შედეგი განვიხილოთ დაწვრილებით (იხ. ცხრილი 6):

ცხრილი 6. ამოცანა 3.13-ის განხილვა

იტერაცია	პარამეტრი, მთვლეელი ცვლადი i	$i \leq 5$	sum ცვლადის მნიშვნელობა
1	1	true	$0 + 1 = 1$
2	2	true	$1 + 2 = 3$
3	3	true	$3 + 3 = 6$
4	4	true	$6 + 4 = 10$
5	5	true	$10 + 5 = 15$
6	6	false	ციკლის დასრულება

ამოცანა 3.14. მოცემული გვაქვს n ნატურალური რიცხვი. ვიპოვოთ რიცხვის ციფრებს შორის მაქსიმალური ციფრი. გამოვიყენოთ for ოპერატორის ის ვარიანტი, რომელშიც არ იქნება არც ინიციალიზება და არც მთვლეელი ცვლადი.

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = " ";
    int i, d, max;
    int n = int.Parse(textBox1.Text);
```



```

i = n;
max = 0;
for ( ; i > 0; )//ციკლი არ შეიცავს ინიციალიზებას და მთვლელცვლადს
{
    d = i % 10;
    if (max < d) max = d;
    i = i / 10;
}
label1.Text = " რიცხვში " + n.ToString() +
    " მაქსიმალური ციფრი = " + max.ToString() + '\n'; }

```

ამოცანა 3.15. დავწეროთ პროგრამული კოდი, რომელიც დაბეჭდავს მთელი ტიპის რიცხვების მიმდევრობის ჯამს და რაოდენობას მითითებულ $[a; b]$ დიაპაზონში, ამასთანავე ვიცით, რომ $b \geq a$. დიაპაზონის საწყისი a და საბოლოო b მნიშვნელობები შევიყვანოთ textbox1 და textbox2 კომპონენტებში. შედეგი კი ავსახოთ label2, label3 კომპონენტებში. პროგრამული კოდი:

```

private void button1_Click(object sender, EventArgs e){
    int sum = 0, count = 0;
    int a = int.Parse(textBox1.Text);
    int b = int.Parse(textBox2.Text);
    if (a > b)
    {
        label1.Text = "შეცდომა!!!! a > b";
        label1.ForeColor = Color.Red;
    }
    else
    {
        for (int i = a; i <= b; ++i)
        {
            sum += i; count++;
        }
        label2.Text = "ჯამი მითითებულ დიაპაზონში =" + sum.ToString();
        label3.Text = "რაოდენობა მითითებულ დიაპაზონში =" + count.ToString();
    }
}

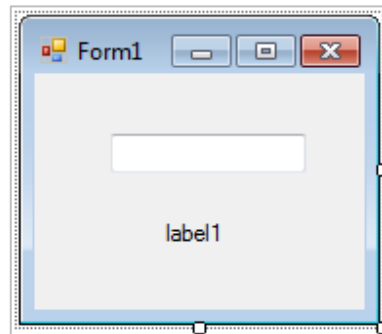
```

ამოცანა 3.16. დავწეროთ ისეთი პროგრამული კოდი, რომლის დახმარებითაც მოხდება textBox კომპონენტში მხოლოდ რიცხვების შეტანა (იხ. ნახ. 9.). ფორმაზე განვათავსოთ ერთი textBox და ერთი label კომპონენტი. პროგრამული კოდი

დავწეროთ textBox კომპონენტის **KeyPress** მოვლენაზე. Properties ფანჯარაში ავირჩიოთ Events ჩანართი და დილაკი გამოსახულით:  „ელვა“. მოვლენა **KeyPress**-ის ველზე მაუსით დავაჭიროთ ორჯერ. ქვემოთ წარმოდგენილია დასმული ამოცანის გადაწყვეტის პროგრამული კოდის სამი ვარიანტი (იხ. ნახ. 9):

პირველი ვარიანტი:

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == 13)
    {
        double ricxvi = double.Parse(textBox1.Text);
        textBox1.Text = "";
        label1.Text = ricxvi.ToString();
    }
}
```



ნახ. 9. ამოცანა 3.15-ის ფორმა

მეორე ვარიანტი:

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    char number = e.KeyChar;
    if (!Char.IsDigit(number))
    {
        e.Handled = true;
    }
}
```

თუ შესრულებაზე გავუშვებთ პროექტს, ვნახავთ, რომ textBox კომპონენტში შევძლებთ ჩავწეროთ მხოლოდ ციფრები და ვერ შევძლებთ ჩავწეროთ რაიმე სხვა სიმბოლო.

თავიდან გამოვაცხადეთ სიმბოლური ტიპის ცვლადი სახელით number. პარამეტრის e.KeyChar საშუალებით პროგრამა უზრუნველყოფს, ცვლადში number კლავიატურიდან აკრეფილი სიმბოლოს შეტანას.

თუ დავაჭერთ „+“ ღილაკს, ცვლადში number მოხდება „+“-ის ჩაწერა, თუ დავაჭერთ „დ“ ღილაკს, მაშინ მოხდება სიმბოლო „დ“-ს ჩაწერა და ა.შ.

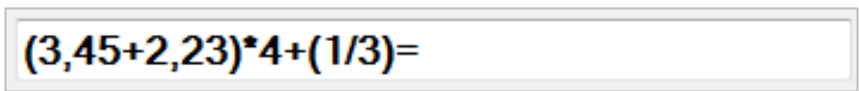
კოდის შემდეგი სტრიქონი არის პირობა !Char.IsDigit(number), რომელიც აიხსნება შემდეგი სახით: „თუ ცვლადში number ჩაწერილი სიმბოლო არ განეკუთვნება ციფრთა და რიცხვთა სისტემას“, ხოლო პირობიდან გამოსვლა e.Handled = true კი აიხსნება ასე: „მაშინ არ დავამუშაოთ შეტანილი სიმბოლო და არ გამოვიტანოთ textBox კომპონენტში“.

მოკლედ ჩვენ ვფილტრავთ და ვეძებთ მომხმარებლის მიერ შეტანილი სიმბოლო განეკუთვნება თუ არა ციფრთა ან რიცხვთა სისტემას, თუ არ განეკუთვნება, მაშინ ვახდენთ მის იგნორირებას, ხოლო თუ განეკუთვნება, ვამუშავებთ და გამოგვაქვს textBox კომპონენტში.

მესამე ვარიანტი:

კომპონენტ textBox-ში ციფრების, მძიმის, მათემატიკური ნიშნების და Backspace ღილაკის გამოყენება ხდება ქვემოთ მოცემულ კოდში. კოდის მესამე ვარიანტი დაგვცხმარება კალკულატორის მუშაობის დაპროგრამებაში. ამ შემთხვევაში ჩვენ ვფილტრავთ არა მხოლოდ ციფრებს და რიცხვებს, არამედ ზოგიერთ მათემატიკურ ნიშანსაც (ნახ. 10.):

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    char number = e.KeyChar;
    if ((e.KeyChar <= 47 || e.KeyChar >= 58) && number != 8 &&
        (e.KeyChar <= 39 || e.KeyChar >= 46) && number != 47 && number
        != 61)
    {
        e.Handled = true;
    }
}
```



ნახ. 10. კომპონენტ textBox-ში ციფრების, მძიმის, მათემატიკური ნიშნების და Backspace ღილაკის გამოყენება

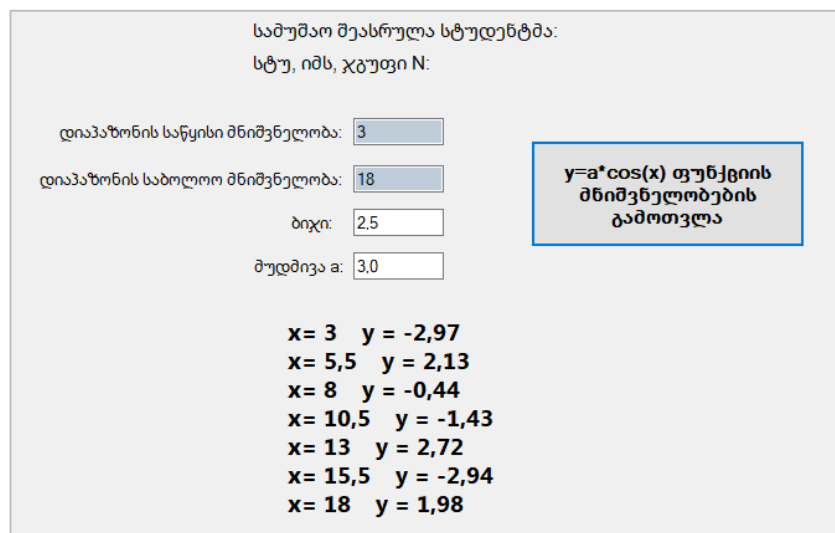
ამოცანა 3.17. მოცემულია ფუნქცია $y = a * \cos(x)$. ფუნქციის x არგუმენტი იცვლება მომხმარებლის მიერ მითითებულ დიაპაზონში გარკვეული ბიჯით, a არის

კონსტანტა. დავწეროთ ისეთი პროგრამული კოდი, რომელიც გამოითვლის და დაბეჭდავს, როგორც არგუმენტის, ისე ფუნქციის მნიშვნელობებს.

პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e)
{
    label4.Text = ""; label5.Text = ""; label11.Text = "";
    double x0 = double.Parse(textBox1.Text);
    double xk = double.Parse(textBox2.Text);
    double dx = double.Parse(textBox3.Text);
    double a = double.Parse(textBox4.Text);
    label5.Text = "სამუშაო შეასრულა სტუდენტმა: " + '\n';
    label11.Text += "სტუ, იმს, ჯგუფი N: ";
    double x = x0;
    while (x <= xk)
    {
        double y = a * Math.Cos(x);
        label4.Text += " x= " + x.ToString() + " y = " +
            Math.Round(y, 2).ToString() + '\n';
        x += dx;
    }
}
```

ამოცანა 3.17.-ის შესრულების შედეგი (იხ. ნახ. 11.):



სამუშაო შეასრულა სტუდენტმა:
სტუ, იმს, ჯგუფი N:

დიაპაზონის საწყისი მნიშვნელობა: 3
დიაპაზონის საბოლოო მნიშვნელობა: 18
ბიჯი: 2,5
მუდმივა a: 3,0

**y=a*cos(x) ფუნქციის
მნიშვნელობების
გამოთვლა**

x= 3 y = -2,97
x= 5,5 y = 2,13
x= 8 y = -0,44
x= 10,5 y = -1,43
x= 13 y = 2,72
x= 15,5 y = -2,94
x= 18 y = 1,98

ნახ. 11. ამოცანა 3.17-ის შესრულების შედეგი

ამოცანა 3.18. მოცემული გვაქვს მთელი ტიპის რიცხვი, მაგალითად, 2045.

წარმოვადგინოთ ეს რიცხვი პირიქით do-while ოპერატორის გამოყენებით.

```
private void button1_Click(object sender, EventArgs e)
{
    int num;
    int nextdigit;
    num = 2045;
    do
    {
        nextdigit = num % 10;
        label1.Text += nextdigit.ToString();
        num = num / 10;
    }
    while (num > 0);
}
```

დამოუკიდებელი სამუშაო

1. მოცემულია პროგრამული კოდის ფრაგმენტები. უპასუხეთ თუ, რომელი ვარიანტია სწორი/მცდარი?

N	პროგრამული კოდი	პასუხი (მცდარი/სწორი)
1	<pre>int x; x = int.Parse(textBox1.Text); if (x / 2 == 0) label1.Text = " x ლუწია"; else label1.Text = "x კენტია";</pre>	
2	<pre>int x; x = int.Parse(textBox1.Text); if (x % 2 == 0) label1.Text=" x ლუწია"; else label1.Text = "x კენტია";</pre>	
3	<pre>int x; x = int.Parse(textBox1.Text); if (x / 2 != 0) label1.Text = " x ლუწია"; else label1.Text = "x კენტია";</pre>	
4	<pre>int x; x = int.Parse(textBox1.Text); if (x % 2 != 0) label1.Text = " x კენტია"; else label1.Text = "x ლუწია";</pre>	

2. ფორმაზე განათავსეთ ერთი **textBox** კომპონენტი. **textBox** კომპონენტში შეიყვანეთ ნებისმიერი რიცხვი. რიცხვი შეიძლება იყოს, როგორც დადებითი, ისე

უარყოფითი. დაწერეთ პროგრამული კოდი, რომელიც დაადგენს რიცხვის ნიშანს. შედეგი გამოიტანეთ **label** კომპონენტში. (პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე).

3. ფორმაზე განათავსეთ ორი **textBox** კომპონენტი ორი მთელი ტიპის რიცხვის შესაყვანად. დაწერეთ პროგრამული კოდი, რომელიც დაადგენს იყოფა თუ არა ერთი რიცხვი მეორეზე უნაშთოდ. შედეგი გამოიტანეთ **label** კომპონენტში. პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე.

4. ფორმაზე განათავსეთ სამი **textBox** კომპონენტი სამკუთხედის a , b და c გვერდების მნიშვნელობების შესაყვანად. დაწერეთ პროგრამული კოდი, რომელიც დაადგენს თუ რომელ ტიპს განეკუთვნება სამკუთხედი: ტოლგვერდას, ტოლფერდას თუ არაწესიერს. შედეგი გამოიტანეთ **label** კომპონენტში. პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე.

5. ფორმაზე განათავსეთ სამი **textBox** კომპონენტი, კვადრატული განტოლების კოეფიციენტების შესაყვანად. მაგ: $a=-7$, $b=2$, $c=9$ ან $a=3$, $b=-17$, $c=10$. დაწერეთ პროგრამული კოდი, რომელიც დაადგენს კვადრატული განტოლების ამონახსნებს. გაითვალისწინეთ დისკრიმინანტის სამივე შესაძლო ვარიანტი: $D>0$, $D=0$, $D<0$ და $a\neq 0$. შედეგი გამოიტანეთ შესაბამის კომპონენტში. პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე.

6. ფორმაზე განათავსეთ ერთი **label** კომპონენტი. წინასწარ მოცემული გვაქვს მთელი რიცხვების მიმდევრობა -11-დან 11-მდე 1-ის ბიჯით. **label** კომპონენტში გამოიტანეთ მიმდევრობის მხოლოდ დადებითი წევრები. პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე.

7. ფორმაზე განათავსეთ ორი **label** კომპონენტი. **label1** კომპონენტში გამოიტანეთ მხოლოდ კენტი რიცხვების მიმდევრობა 5-დან 30-მდე დიაპაზონში 1-ის ბიჯით. **label2** კომპონენტში კი გამოიტანეთ ერთი მნიშვნელობა: კენტი რიცხვების ჯამი იმავე მიმდევრობიდან. პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე.

8. ფორმაზე განათავსეთ ერთ **label** კომპონენტი. კომპონენტში უნდა მოხდეს $y = \sin(x) + 1$ ფუნქციის მნიშვნელობების ასახვა ე. ი. უნდა დაიბეჭდოს y -ის მნიშვნელობები. ამასთან ცნობილია, რომ y ფუნქციის x არგუმენტი იცვლება

დიაპაზონში 0-დან 1-მდე 0.1-ის ბიჯით. პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე.

9. ფორმაზე განთავსეთ ორი **textBox** კომპონენტი. **textBox** კომპონენტებში მოვახდინოთ $M(x,y)$ საკოორდინატო სიბრტყის $x(\text{textBox1})$ და $y(\text{textBox2})$ წერტილის მნიშვნელობების შეყვანა. ცნობილია რომ, საკოორდინატო სიბრტყე დაყოფილია ოთხ მეოთხედად. დაწერეთ პროგრამული კოდი, რომელიც დაადგენს თუ რომელ საკოორდინატო მეოთხედს განეკუთვნება $M(x,y)$ წერტილი. პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე.

10. ფორმაზე განთავსეთ ერთი **textBox** კომპონენტი. კომპონენტში შეიყვანეთ კვირის დღის აღმნიშვნელი მთელი რიცხვი. დაწერეთ პროგრამა, რომელიც კვირის დღის რიგითობის მიხედვით **label** კომპონენტში დაბეჭდავს მის სახელწოდებას. მაგ, 1. ორშაბათი, 2. სამშაბათი, 3. ოთხშაბათი, .. 7. კვირა, ან შეგვატყობინებს „მონაცემი მითითებულია არასწორად“. პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე.

11. ფორმაზე განთავსეთ ერთი **textBox** კომპონენტი. შეადგინეთ პროგრამა, რომელიც მომხმარებელს ეკითხება სატელევიზიო არხის ნომერს (ათამდე არხი) და გამოაქვს შეტყობინება არხის დასახელების შესახებ **label** კომპონენტში. თუ მომხმარებელმა შეცდომით შეიტანა რაიმე სხვა სიმბოლო ან სიტყვა, მაშინ ფორმაზე წითელი ფერით გამოჩნდეს შეტყობილება: „არასწორია! სცადეთ თავიდან“. პროგრამული კოდი დაწერეთ ბრძანებით კომპონენტზე.

12. ფორმაზე განთავსეთ ერთი **textBox** კომპონენტი. შეადგინეთ პროგრამა, რომელიც მომხმარებელს ეკითხება რომელიმე ერთ რიცხვს 1-დან 10-მდე დიაპაზონში, ხოლო შემდეგ **label** კომპონენტში გამოაქვს შეტყობინება: რიცხვი მარტივია თუ შედგენილი. პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე.

განმარტება: მარტივი რიცხვი ეწოდება ისეთ ნატურალურ რიცხვს, რომელსაც მხოლოდ ორი განსხვავებული ნატურალური გამყოფი აქვს: ერთი და თავისი თავი. მაგალითად: მარტივი რიცხვებია: 2, 3, 5, 7. **შედგენილ რიცხვებს** კი ორზე მეტი გამყოფი აქვთ. მაგალითად, შედგენილი რიცხვებია 4, 6, 8, 9, 10.

13. ფორმაზე განთავსეთ ერთი **textBox** კომპონენტი. **textBox** კომპონენტში შეიყვანეთ სტუდენტის რეალური წონა. შეადგინეთ პროგრამა, რომელიც

გამოთვალის ოპტიმალურ წონას. შევადაროთ ოპტიმალური წონა რეალურ წონასთან. გამოვიტანოთ შეტყობინება **label** კომპონენტში წონის მომატების ან დაკლების შესახებ რეკომენდაციის სახით (წონა განისაზღვრება კილოგრამებით). ოპტიმალური წონის გამოთვლა ხდება შემდეგი ფორმულის მიხედვით: სიმაღლე (სმ) - 100. მაგ: თუ თქვენი სიმაღლეა 175 სმ. და თქვენი წონაა 90 კგ. მაშინ, თქვენ უნდა დაიკლოთ 15 კგ. $(90 - (175 - 100))$. პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე.

14. ფორმაზე განთავსეთ ერთი **textBox** კომპონენტი. შეადგინეთ პროგრამა, რომელიც დაბეჭდავს გამრავლების ტაბულას მითითებული **n** მნიშვნელობისათვის. მაგალათად, $n=7$ -სათვის. **label** კომპონენტში მიიღეთ შემდეგი სახის შეტყობინება: $7*1=7$ $7*2=14$ $7*3=21$ $7*4=28$ $7*5=35$ $7*6=42$ $7*7=49$ $7*8=56$ $7*9=63$ $7*10=70$. პროგრამული კოდი დაწერეთ ბრძანებით **button** კომპონენტზე.

15. ამოცანა 3.15 გადააკეთეთ შემდეგი სახით: თუ მომხმარებელმა შეიტანა არასწორი მნიშვნელობა ან არასწორად დააჭირა გამოვიდეს წითელი ფერის შეტყობინებები: 1. თუ დიაპაზონი შეიყვანა არასწორად მაშინ: „არასწორია!! საყისი დიაპაზონი ნაკლებია საბოლოოზე, სცადეთ თავიდან“. 2. თუ პირდაპირ დააჭირა ბრძანებით ღილაკს და არ შეიყვანა დიაპაზონის საწყისი მნიშვნელობა, მაშინ: „ფუნქციის გამოთვლა შეუძლებელია! არ შეგიტანიათ დიაპაზონის საწყისი და საბოლოო მნიშვნელობა“.

4 მასივი. Length თვისება. foreach ციკლი. ჩამოთვლები. კლასი System.Array. სია მასივი List <T>.

მასივი არის ერთსა და იმავე ტიპის მქონე ობიექტების კოლექცია (ერთობლიობა). მასივის გამოცხადების დროს მნიშვნელოვანია მასივის ზომის დაფიქსირება. თუ ზომა არის კონსტანტა, მაშინ ასეთ მასივებს უწოდებენ **სტატიკურ** მასივს. თუ მასივის ზომა ცვალებადია, მაშინ ასეთ მასივებს უწოდებენ **დინამიკურს**. დინამიკური მასივისათვის მეხსიერების გამოყოფა ხდება პროგრამის შესრულების დროს. მასივი გამოიყენება ერთნაირი ტიპის მქონე **რამდენიმე ცვლადის ან ობიექტის** შესანახად.

მასივი შეიძლება იყოს როგორც **ერთგანზომილებიანი**, ისე **მრავალგანზომილებიანი**. მასივში შეგვიძლია შევინახოთ თანამშრომლების გვარები, მათი ხელფასები, ასაკი და ა.შ.

მასივები წარმოადგენენ მიმართვითი ტიპის მქონე ობიექტებს. მასივის ელემენტები დანომრილია, რაც აადვილებს მათთან მუშაობას. ასევე მასივი შეგვიძლია განვიხილოთ როგორც კოლექცია, ხოლო კოლექციის ელემენტების დასამუშავებლად შეგვიძლია გამოვიყენოთ **foreach** ციკლი.

ერთგანზომილებიანი მასივი. ერთგანზომილებიან მასივში კონკრეტული ელემენტის მდებარეობა განისაზღვრება ერთი ინდექსით. ერთგანზომილებიანი მასივის სინტაქსია: **ტიპი [] მასივის_სახელი = new ტიპი [ზომა];**

აქ ტიპი განსაზღვრავს მასივის ტიპს ანუ მისი თითოეული ელემენტის ტიპს. ტიპის მარჯვნივ კვადრატული ფრჩხილები მიუთითებენ ერთგანზომილებიან მასივზე. ზომა არის მასივში ელემენტების რაოდენობა. რადგან მასივები რეალიზებულია ობიექტების სახით, ამიტომ მასივის შექმნის პროცესი ორი ეტაპისაგან შედგება. პირველ ეტაპზე ხდება მასივის სახელის გამოცხადება, რომელიც წარმოადგენს მიმართვითი ტიპის ცვლადის სახელს. მეორე ეტაპზე მასივისთვის მეხსიერება გამოიყოფა და მასივის ცვლადს მიენიჭება მეხსიერების ამ უბანზე მიმართვა (ანუ ამ უბნის მისამართი).

C# ენაში **new** ოპერატორის გამოყენებით მასივს მეხსიერება პროგრამის შესრულების დროს (დინამიურად) გამოიყოფა. ქვემოთ მოყვანილ სტრიქონში იქმნება **int** ტიპის მასივი, რომელიც 10 ელემენტისაგან შედგება და მიმართვითი ტიპის **masivi** ცვლადს ენიჭება მასზე მიმართვა: **int[] masivi = new int[10];**

masivi ცვლადი ინახავს მიმართვას მეხსიერების უბანზე (ანუ ამ უბნის მისამართს), რომელიც **new** ოპერატორის მიერ გამოყოფილი მისთვის. გამოყოფილი მეხსიერების ზომა საკმარისია მასში 10 ცალი **int** ტიპის ელემენტის მოსათავსებლად, რადგან **int** ტიპის ცვლადი 4 ბაიტს (32 ბიტს) იკავებს, ამიტომ **masivi** მასივისათვის მეხსიერებაში სულ $4 * 10 = 40$ ბაიტი გამოიყოფა. მასივის ცალკეულ ელემენტთან მიმართვა ხორციელდება ინდექსის გამოყენებით. მისი საშუალებით ეთითება ელემენტის პოზიცია მასივის ფარგლებში. ნებისმიერი მასივის პირველ ელემენტს ნულოვანი ინდექსი აქვს. რადგან **masivi** მასივი 10 ელემენტისაგან შედგება, ამიტომ ამ მასივის ინდექსის მნიშვნელობები მოთავსებული იქნება დიაპაზონში $0 \div 9$. შედეგად, **masivi** მასივის პირველი ელემენტი აღინიშნება ასე - **masivi[0]**, უკანასკნელი კი - **masivi[9]**.

C# ენაში ყოველ მასივს აქვს **Length** თვისება. **Length** თვისებაში ავტომატურად იწერება მასივის ელემენტების მაქსიმალური რაოდენობა. ის არ ინახავს რეალურად არსებული ელემენტების რაოდენობას. კერძოდ, თუ გვაქვს 10 ელემენტიანი მასივი და შევსებულია მისი პირველი 4 ელემენტი, მაშინ **Length** თვისების მნიშვნელობა იქნება 10 და არა 4.

Length თვისება წარმატებით გამოიყენება მასივის ინდექსების მნიშვნელობების საზღვრებს გარეთ გასვლის თავიდან ასაცილებლად.

მასივის ინიციალიზება. მასივის ინიციალიზება შესაძლებელია მისი შექმნისთანავე. ერთგანზომილებიანი მასივის ინიციალიზაციის სინტაქსია:

ტიპი [] მასივის_სახელი = new ტიპი [] { სიდიდე1, სიდიდე2, ..., სიდიდეN };

აქ საწყისი მნიშვნელობები მოიცემა სიდიდე 1-დან სიდიდე N-მდე სიდიდეებით. მასივის ელემენტებს მნიშვნელობები ენიჭებათ რიგრიგობით მარცხნიდან მარჯვნივ დაწყებული ნულოვანი ინდექსის მქონე ელემენტიდან.

ერთგანზომილებიანი მასივის გამოცხადების ვარიანტები:

```
int[] masivi; // ერთგანზომილებიანი მასივის სახელია masivi
```

```
masivi = new int[3]; // მასივი სახელით masivi შეიცავს 3 ელემენტს
```

```
masivi[0]=-5; masivi[1]=4; masivi[2]=55; // მასივი შეიცავს სამი მთელი ტიპის რიცხვს
```

```
// მასივის ელემენტების მითითება შეიძლება მასივის გამოცხადებისთანავე
```

```
int [] a = new [] {-5, 4, 77, 31};
```

```
int size = Int32.Parse(textBox1.Text); // მასივის ელემენტების რაოდენობა (მასივის ზომა)
```

```
int[] A1 = new int[size]; // დინამიკური მასივის შექმნა
```

ორგანზომილებიანი მასივი. ორგანზომილებიან მასივებში კონკრეტული ელემენტის განლაგება განისაზღვრება ორი ინდექსით. მასივის გამოცხადების დროს პირველი ინდექსი მიუთითებს სტრიქონების რაოდენობას, მეორე კი - სვეტების რაოდენობას. ორგანზომილებიანი მთელრიცხვა **cxrili** მასივი ზომით 5x4 შეგვიძლია შემდეგნაირად გამოვაცხადოთ: `int[,] cxrili = new int[5, 4];`

მასივის გამოცხადებისას ორივე განზომილება ერთმანეთისაგან მძიმით გამოიყოფა. ოპერატორის პირველ ნაწილში ფრჩხილებში მითითებული მძიმე მიუთითებს, რომ იქმნება მიმართვითი ტიპის ცვლადი ორგანზომილებიანი მასივისათვის. **new** ოპერატორი დინამიკურად გამოყოფს მეხსიერებას ორგანზომილებიანი მასივისათვის. ოპერატორის მეორე ნაწილში რიცხვი 5 განსაზღვრავს სტრიქონების რაოდენობას, რიცხვი 4 - კი სვეტების რაოდენობას. მეხსიერება გამოიყოფა $5 * 4 = 20$ რიცხვისათვის.

ორგანზომილებიანი მასივის ინიციალიზება. ორგანზომილებიანი მასივის ინიციალიზაციისათვის თითოეული განზომილების მნიშვნელობების სია უნდა მოვათავსოთ ცალკე ბლოკში, რომელიც, თავის მხრივ, ფიგურულ ფრჩხილებში უნდა იყოს მოთავსებული.

```
მაგალითად: int[,] cxrili = new int[,] { { 1, 1 }, { 2, 4 }, { 3, 9 }, { 4, 16 }, { 5, 25 } };
```

აქ ყოველი შიგა ბლოკი მნიშვნელობას ანიჭებს შესაბამისი სტრიქონის ელემენტებს. სტრიქონის ფარგლებში პირველი მნიშვნელობა მოთავსებული იქნება სტრიქონის ნულოვან ელემენტში, მეორე - პირველ ელემენტში და ა.შ. ბლოკები ერთმანეთისაგან მძიმეებით გამოიყოფა, ხოლო დამხურავი ფიგურული ფრჩხილის შემდეგ მოთავსებულია წერტილ-მძიმე.

მრავალგანზომილებიანი მასივის გამოცხადების გზები:

```
// ორგანზომილებიანი მასივის გამოცხადება:
```

```
// სტრიქონების რაოდენობა არის 2; სვეტების რაოდენობა არის 3;
```

```
int[,] m = new int [2,3];
```

```
// მასივის გამოცხადებისთანავე შეგვიძლია მისი ინიციალიზება:
```

```
int[,] AA = new int [,] { {2,-2},{3,-22},{0,4}};
```

```
// სამგანზომილებიანი მასივის გამოცხადება:
```

```
int[,,] n = new int [7,7,7];
```

foreach ციკლი. **foreach** ციკლი გამოიყენება კოლექციის ელემენტების დასამუშავებლად. კოლექცია არის ობიექტების ჯგუფი. **C#** ენაში განსაზღვრულია კოლექციების რამდენიმე ტიპი, ერთ-ერთი მათგანია მასივი.

foreach ციკლის სინტაქსია:

```
foreach ( ტიპი ცვლადის_სახელი in კოლექცია )  
{  
    ციკლის ტანი  
}
```

აქ ტიპი არის იტერაციული ცვლადის ტიპი და უნდა ემთხვეოდეს კოლექციის ტიპს. ცვლადის_სახელი არის იტერაციული ცვლადის სახელი. იტერაციული ცვლადი რიგ-რიგობით იღებს კოლექციის ელემენტების მნიშვნელობებს. კოლექცია შეიძლება იყოს მასივი, სტრიქონი და ა.შ. იტერაციული ცვლადი მისაწვდომია მხოლოდ წაკითხვის რეჟიმში, ამიტომ, ვერ შევძლებთ მასივის შეცვლას იტერაციული ცვლადისთვის ახალი მნიშვნელობის მინიჭების გზით.

ჩამოთვლები. ჩამოთვლები წარმოადგენენ სახელდებული მთელრიცხვა მუდმივების ნაკრებს, რომლებიც განსაზღვრავენ ცვლადის ყველა დასაშვებ მნიშვნელობას მოცემული ტიპისთვის. ჩამოთვლადი ტიპის განსაზღვრისათვის გამოიყენება **enum** საკვანძო სიტყვა. მისი სინტაქსია:

```
enum ჩამოთვლის_სახელი { მუდმივების_სია };
```

ქვემოთ მოყვანილია **xili** ჩამოთვლის განსაზღვრის მაგალითი:

```
enum xili { ვაშლი, მსხალი, ლეღვი, ნესვი, ყურძენი };
```

ჩამოთვლასთან მუშაობისას მთავარია ის ფაქტი, რომ თითოეულ მუდმივას შეესაბამება მთელრიცხვა მნიშვნელობა. შედეგად, მუდმივების გამოყენების სფერო ემთხვევა მთელრიცხვა მნიშვნელობების (**int** ტიპი) გამოყენების სფეროს. მთელრიცხვა მნიშვნელობა თითოეული მუდმივასთვის მეტია წინა მუდმივას მნიშვნელობაზე. პირველი მუდმივას მნიშვნელობა ნულის ტოლია. ჩამოთვლის ელემენტთან მიმართვისათვის ჯერ უნდა მივუთითოთ ჩამოთვლის სახელი, შემდეგ წერტილი (.) და ელემენტის სახელი.

ჩამოთვლების ინიციალიზება. ინიციალიზების საშუალებით შეგვიძლია მივუთითოთ ერთი ან მეტი მუდმივას მნიშვნელობა. ამისათვის თითოეული მუდმივას შემდეგ უნდა მოვათავსოთ მინიჭების ოპერატორი და მთელრიცხვა

მნიშვნელობა. ინიციალიზატორის მიერ განსაზღვრული მნიშვნელობები უნდა იყოს მეტი ინიციალიზაციის წინა მნიშვნელობაზე.

მაგალითად, ქვემოთ მოყვანილ გამოცხადებაში `nesvi` მუდმივას ენიჭება მნიშვნელობა 55:

`enum xili { ვაშლი, მსხალი, ლეღვი, ნესვი = 55, ყურძენი };` მუდმივებს ექნებათ შემდეგი მნიშვნელობები: ვაშლი - 0 მსხალი - 1 ლეღვი - 2 ნესვი - 55 ყურძენი - 56

კლასი `System.Array`. ნებისმიერი მასივი **C#** ენაში შეგვიძლია განვიხილოთ, როგორც **`System.Array`** კლასის ეგზემპლარი. **`System.Array`** კლასს აქვს მრავალი მეთოდი და თვისება. **`System.Array`** კლასის მეთოდები და თვისებები შეგვიძლია გამოვიყენოთ მხოლოდ ერთგანზომილებიან მასივებთან სამუშაოდ.

`System.Array` კლასის მეთოდები და თვისებები:

- თვისება **`Length`** - გასცემს მასივის სიგრძეს;
- თვისება **`Rank`** - გასცემს მასივის განზომილებას;
- მეთოდი **`BinarySearch()`** - სორტირებულ მასივში ბინარული ძებნა;
- მეთოდი **`Clear()`** - მასივის გასუფთავება;
- მეთოდი **`CopyTo()`** - მთლიანი მასივის ან მასივის ნაწილის გადაწერა მეორე მასივში;
- მეთოდი **`Exists()`** - ამოწმებს, შეიცავს თუ არა მასივი გარკვეულ ელემენტს;
- მეთოდი **`Find()`** - ელემენტის მოძებნა გარკვეული პირობის მიხედვით;
- მეთოდი **`FindAll()`** - ყველა ელემენტის მოძებნა გარკვეული პირობის მიხედვით;
- მეთოდი **`IndexOf()`** - გასცემს მასივის ელემენტის ინდექსს;
- მეთოდი **`Resize()`** - ერთგანზომილებიანი მასივის ზომის შეცვლა;
- მეთოდი **`Reverse()`** - მასივის ელემენტების დაბეჭდვა შებრუნებული სახით;
- მეთოდი **`Sort()`** - მასივის ელემენტების დალაგება ზრდადობის მიხედვით.

სია მასივი `List <T>`. სია მასივი თავისი მნიშვნელობით ძლიერ ჰგავს მასივებს. მათ გააჩნიათ მართვის ფართო სპექტრი. მასივებისაგან განსხვავებით სია მასივში ადვილად დავამატებთ ახალ ელემენტს სიის ბოლოს ან ჩავსვამთ ელემენტს ნებისმიერ ადგილზე, წავშლით სიიდან, შევამცირებთ ან მოვახდენთ დალაგებას მარტივი **`Sort`** მეთოდით.

სია მასივი წარმოადგენს რომელიმე ტიპის ერთიან კოლექციას: რიცხვების, სტრიქონების ან სხვა ობიექტების. მაგალითად, წინადადებაში შემავალი სიტყვების სია, ფაილის სტრიქონების სია, ღონისძიების მონაწილეთა სია და ა.შ.

```
List<int> a = new List<int>(); //მთელი რიცხვების სია მასივის შექმნა
```

```
List<double> b = new List<double>(); //წილადი რიცხვების სია მასივის შექმნა
```

```
List<bool> c = new List<bool>(); //ბულის მნიშვნელობის სია მასივის შექმნა
```

```
List<string> d = new List<string>(); //სტრიქონის სია მასივის შექმნა
```

```
List<string[]> f = new List<string[]>(); //მასივის სია მასივის შექმნა
```

```
List<List<int>> g = new List<List<int>>(); //რიცხვითი ჩამონათვალის სია მასივის შექმნა
```

```
a.Add(16);
```

```
b.Add(25.4);
```

```
c.Add(false);
```

```
d.Add("კლავიატურა");
```

```
f.Add(new string[5]);
```

```
g.Add(new List<int>());
```

მეთოდი Add(). Add() მეთოდის დახმარებით სია მასივში შეგვიძლია დავამატოთ ელემენტები.

მაგალითად: მთელი ტიპის რიცხვები. თითოეულ რიცხვს აქვს თავისი ინდექსი (იხ. ცხრილი 7):

```
List<int> x = new List<int>();
```

```
x.Add(5);
```

```
x.Add(27);
```

```
x.Add(-6);
```

```
x.Add(14);
```

```
x.Add(70);
```

```
x.Add(14);
```

```
x.Add(178);
```

ცხრილი 7. სია მასივში ელემენტების დამატება

ინდექსი:	0	1	2	3	4	5	6
მნიშვნელობა:	5	27	-6	14	70	14	178

არსებობს უფრო მოკლე ვარიანტიც:

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };
```

```
int m = x[0]; // 5
```

```
int n = x[1]; // 27
```

სია მასივის მესამე ელემენტის მნიშვნელობის (ინდექსი 2) შეცვლა შესაძლებელია შემდეგი სახით: $x[2] = 100$;

მეთოდი Remove() სიიდან შლის ყველა მითითებულ ელემენტს. სიიდან წავშალოთ ყველა ის ელემენტი, რომელთა მნიშვნელობაც არის 14.

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };
```

```
x.Remove(14);
```

ამ ჩანაწერის შემდეგ წაიშლება ორი ელემენტი მნიშვნელობით 14, ხოლო ინდექსირება არ შეიცვლება და თანამიმდევრულად გადაინომრება.

მეთოდი RemoveAt() სიიდან შლის იმ ელემენტს, რომლის ინდექსაც მივუთითებთ.

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };
```

```
x.RemoveAt(2); // წაიშლება მესამე ელემენტი
```

თვისება Count() ითვლის სიაში ელემენტის რაოდენობას:

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };
```

```
int n = x.Count; // სია შეიცავს შვიდ ელემენტს
```

```
int m = x.Count - 1; // ბოლო ელემენტის ინდექსი
```

მეთოდი Insert() საშუალებას გვაძლევს, ჩავსვათ ახალი ელემენტი გარკვეულ პოზიციაზე.

მაგალითად, გვინდა მეორე ელემენტის (ინდექსი 1) წინ ჩავსვათ რიცხვი 1000:

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };
```

```
x.Insert(1, 1000);
```

მეთოდი IndexOf() გვაძლევს საშუალებას მოვძებნოთ მითითებული ელემენტი:

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };
```

```
int a = x.IndexOf(5); // პოზიცია 0
```

```
int b = x.IndexOf(-6); // პოზიცია 2
```

```
int k = x.IndexOf(70); // პოზიცია 4
```

```
int q = x.IndexOf(166); // შედეგი არის -1, რადგან არ მოიძებნა მითითებული ელემენტი
```

სხვა მეთოდები მაგალითის სახით:

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };
```

```
x.Reverse(); // შებრუნებული სია 178, 14, 70, 14, -6, 27, 51;
```

```
x.Sort(); // ზრდადობის მიხედვით დალაგებული სია,
```

```
int a = x.Min(); // მინიმალური = -6,
```

```
int b = x.Max(); // მაქსიმალური = 178,
```

```
int c = x.Sum(); // ელემენტების ჯამი = 302,
```

```
int d = x.Average(); // საშუალო არითმეტიკული = 43.14.
```

შენიშვნა: უნდა გვახსოვდეს, რომ ჩვეულებრივი მასივი მუშაობს უფრო სწრაფად, ვიდრე სია მასივი. თუ პროგრამა არ მოითხოვს სისწრაფეს და მონაცემთა რაოდენობაც დიდი არ არის, მაშინ მოხერხებულია სია მასივის გამოყენება, თუ არა და უნდა მივმართოთ ჩვეულებრივ მასივებს.

ახლა გადავიდეთ ამოცანების განხილვაზე.

ამოცანა 4.1. მოცემულია მთელი და ნამდვილი ტიპის ერთგანზომილებიანი მასივი. დაწერეთ პროგრამული კოდი, სადაც მოხდება:

- მთელი და ნამდვილი ტიპის ერთგანზომილებიანი მასივის გამოცხადება;
- ნამდვილი ტიპის მასივის შევსება $i + i * 2$ მნიშვნელობებით;
- მიღებული მასივის მნიშვნელობების დაბეჭდვა label კომპონენტში;

ფორმაზე განვალაგოთ ერთი label და ერთი button კომპონენტები. პროგრამული კოდი განვათავსოთ button1 კომპონენტის Click მოვლენაზე (Events). პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "";
    //მთელი ტიპის მასივი
    int[] AAA = new int[50];
    //ნამდვილი ტიპის მასივი
    float[] BBB = new float[50];
    // BBB მასივის შევსება ფორმულის მიხედვით
    for (int i = 0; i < 10; ++i) BBB[i] = i + i * 2;
```



```

        // BBB მასივის გამოტანა/დაბეჭდვა
        for (int i = 0; i < 10; ++i) label1.Text +=
            BBB[i].ToString() + '\n';
    }

```

ამოცანა 4.2. მოცემულია მთელი ტიპის ინიციალიზებული (წინასწარ განსაზღვრული) ერთგანზომილებიანი მასივი. დავეწეროთ პროგრამული კოდი, სადაც მოხდება:

- მასივის ელემენტების და ინდექსების დაბეჭდვა textBox კომპონენტში;
- ლუწი ელემენტების დაბეჭდვა textBox კომპონენტში;
- ლუწი ელემენტების რაოდენობის დაბეჭდვა textBox კომპონენტში;
- კენტი ინდექსის მქონე ელემენტების დაბეჭდვა textBox კომპონენტში;

ფორმაზე განვალაგოთ ოთხი label, ოთხი textBox და ერთი button კომპონენტები. textBox1 კომპონენტისათვის ავირჩიოთ რეჟიმი MultiLine (იხ. ნახ. 12.).

პროგრამული კოდი განვათავსოთ button1 კომპონენტის Click მოვლენაზე (Events). პროგრამული კოდი:

```

private void button1_Click(object sender, EventArgs e)
{
    int[] A = new int[] {12, 0, -5, 10, 1, 8, 7, 3, -4, 11 };
    for (int i = 0; i < A.Length; ++i)
        textBox1.Text += "A[" + i.ToString() + "]= " +
            A[i].ToString() + " ";

    int count = 0;
    for (int i = 0; i < A.Length; ++i)
        if (A[i] % 2 == 0)
        {
            textBox2.Text += A[i].ToString() + " ";
            ++count;
        }
    textBox3.Text = count.ToString();
    for (int i = 0; i < A.Length; ++i)
        if (i % 2 != 0)
            textBox4.Text += A[i].ToString() + " ";
}

```

მასივი	A[0]=12 A[1]=0 A[2]=-5 A[3]=10 A[4]=1 A[5]=8 A[6]=7 A[7]=3 A[8]=-4 A[9]=11									
ლუნი ელემენტები	12 0 10 8 -4									
ლუნი ელემენტების რაოდენობა	5									
კენტი ინდექსის მქონე ელემენტები	0 10 8 3 11									

ერთგანზომილებიანი მასივი :)

ნახ. 12. ამოცანა 4.2-ის ფორმა და შესრულების შედეგი

ამოცანა 4.3. წინასწარ მოცემული გვაქვს სტრიქონული ტიპის ერთგანზომილებიანი მასივი. მასივი შეიცავს ხილის ასორტის დასახელებებს. დავწეროთ პროგრამული კოდი, რომელიც დაბეჭდავს ხილის დასახელებებს **label** კომპონენტში. ამოცანის გადასწყვეტად გამოვიყენოთ ციკლის ოპერატორი **foreach**. ფორმაზე განვალაგოთ ერთი **button** და ერთი **label** კომპონენტები.

პროგრამული კოდი დავწეროთ **button1** კომპონენტის **click** მოვლენაზე (Events). პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e)
{
    string[] xili = new[] { "ვაშლი", "მსხალი", "ატამი", "გარგარი",
        "ვაშლატამა", "ლეღვატამა", "ყურძენი", "ქლიავი", "თუთა", "მოცვი",
        "მაყვალი", "უნაბი", "ჟოლო", "მარწყვი", "შინდი", "ბროწეული",
        "ალუბალი", "ბალი", "კომში", "ხურმა", "საზამთრო", "ნესვი",
        "მაყვალი", "კივი", "ლეღვი", "მანდარინი", "ფორთოხალი", "ნუში",
        "კაკალი", "თხილი" };
    label1.Text = "";
    foreach (string qartuli_xili in xili)
        label1.Text += qartuli_xili.ToString() + '\n';
}
```

ამოცანა 4.4. მოცემულია ერთგანზომილებიანი მასივი. დავწეროთ პროგრამული კოდი, რომელიც დაადგენს მასივში დადებითი ელემენტების რაოდენობას. გამოვიყენოთ ციკლის ოპერატორი **foreach**. ფორმაზე განვალაგოთ ერთი **button** და ორი **label** კომპონენტები. პროგრამული კოდი დავწეროთ **button1** კომპონენტის **click** მოვლენაზე (Events). პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e){
```

```

label1.Text = "";
int[] ricxvebi=new[] {1,-3,5,78,0 -5,112,41,34,-89,111,60 };
int dadebitebis_raodenoba = 0;
foreach (int ricxvebis_masivi in ricxvebi)
{
    {
        if (ricxvebis_masivi > 0) ++dadebitebis_raodenoba;
    }
    label1.Text += ricxvebis_masivi.ToString() + " ";
}
label2.Text = " დადებითი ელემენტების რაოდენობა = " + dadebitebis_raodenoba; }

```

ამოცანა 4.5. მოცემულია მთელი ტიპის ერთგანზომილებიანი მასივი. დავწეროთ პროგრამული კოდი, რომელიც დაბეჭდავს მასივის ელემენტებს ზრდადობის მიხედვით. გამოვიყენოთ ციკლის ოპერატორი **foreach** და **System.Array** კლასის მეთოდები და თვისებები. ფორმაზე განვალაგოთ ერთი **button** და ორი **textBox** კომპონენტები. პროგრამული კოდი დავწეროთ **button1** კომპონენტის **click** მოვლენაზე (იხ. ნახ. 10). პროგრამული კოდი:

```

private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "";
    int[] intArray = new[] { 1, -3, 5, 78, 0 - 5, 112, 41, 34, -89, 111, 60 };
    Array.Sort(intArray);
    foreach (int zrdadobit_ricxvebi in intArray)
        textBox1.Text += zrdadobit_ricxvebi.ToString() + " ";
    string[] stringArray = new
    string[] {"Csharp","ASP.net","entityframework","ADO.net","WCF"};
    Array.Sort(stringArray);
    foreach (string zrdadobit_stringi in stringArray)
        textBox2.Text += zrdadobit_stringi.ToString() + " ";}

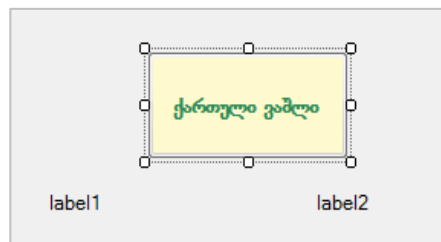
```

int და string ტიპის მასივის დალაგება											
-89	-5	-3	1	5	34	41	60	78	111	112	
ADO.net	ASP.net	Csharp	entityframework	WCF							

ნახ. 10. ამოცანა 4.5-ის შესრულების შედეგი

ამოცანა 4.6. მოცემულია ჩამოთვლადი ტიპის ერთგანზომილებიანი მასივი, სადაც აღწერილია ვაშლის ჯიშის სახეობები. დავწეროთ პროგრამული კოდი, რომელიც დაბეჭდავს, როგორც ვაშლის ჯიშის დასახელებებს, ასევე მათ შესაბამის შეფერილობას. ვაშლის ჯიშის სახეობების შეფერილობები გამოვაცხადოთ, როგორც სტრიქონული ტიპის ერთგანზომილებიანი მასივი.

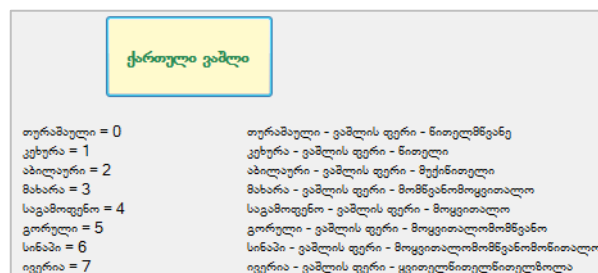
ამოცანის გადასაწყვეტად გამოვიყენოთ ჩამოთვლადი ტიპი **enum**. ფორმაზე განვალაგოთ ერთი **button** და ორი **label** კომპონენტები (იხ. ნახ. 13. და ნახ. 14.). პროგრამული კოდი დავწეროთ **button1** კომპონენტის **click** მოვლენაზე (**Events**).



ნახ. 13. ამოცანა 4.6-ის ფორმა

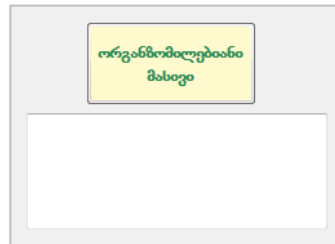
Enum qarTulivashli {თურაშაული, კეხურა, აბილაური, მახარა, საგამოფენო, გორული, სინაპი, ივერია};

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = ""; label2.Text = "";
    string[] color=new string[] {"წითელმწვანე", "წითელი",
        "მუქიწითელი", "მომწვანომოყვითალო", "მოყვითალო",
        "მოყვითალომომწვანო", "მოყვითალომომწვანომოწითალო",
        "ყვითელწითელზოლა"};
    qarTulivashli i;
    for (i = qarTulivashli.თურაშაული; i<=qarTulivashli.ივერია; ++i)
        label1.Text += i + " = " + (int)i + '\n';
    for (i = qarTulivashli.თურაშაული; i <= qarTulivashli.ივერია; ++i)
        label2.Text += i + " - ვაშლის ფერი - " + color[(int)i] + '\n'; }
```



ნახ. 14. ამოცანა 4.6-ის შესრულების შედეგი

ამოცანა 4.7. მოცემულია მთელი ტიპის ორგანზომილებიანი მასივი. ორგანზომილებიანი მასივი გამოცხადებულია წინასწარ. დავწეროთ პროგრამული კოდი, რომელიც ორგანზომილებიანი მასივის ელემენტების მნიშვნელობებს დაბეჭდავს **textBox** კომპონენტში. ფორმაზე განვალაგოთ ერთი **button** და ერთი **textBox** კომპონენტები. **textBox** კომპონენტის თვისება **Multiline**-ში ავირჩიოთ მდგომარეობა **true**; (იხ. ნახ. 15. და ნახ. 16.).

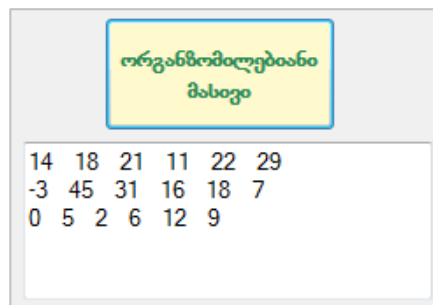


ნახ. 15. ამოცანა 4.7-ის ფორმა

პროგრამული კოდი დავწეროთ **button1** კომპონენტის **click** მოვლენაზე (Events).

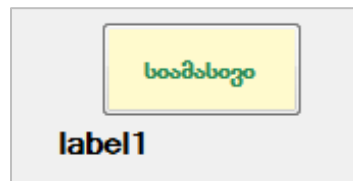
```
private void button1_Click(object sender, EventArgs e)
{
    int[,] myarr = new int[,] { { 14, 18, 21, 11, 22, 29 },
                                { -3, 45, 31, 16, 18, 7 },
                                { 0, 5, 2, 6, 12, 9 } };

    textBox1.Clear();
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 6; ++j)
            textBox1.Text += myarr[i, j].ToString() + " ";
        textBox1.Text += "\r\n"; }
}
```



ნახ. 16. ამოცანა 4.7-ის შესრულების შედეგი

ამოცანა 4.8. სიამასივი **List<t>**. გამოვაცხადოთ მთელი ტიპის სიამასივი და გადავანაცვლოთ სიამასივის ელემენტები შემთხვევითი გენერირების გამოყენებით (იხ. ნახ. 17. და ნახ. 18.).



ნახ. 17. ამოცანა 4.8-ის ფორმა

```
private void button1_Click(object sender, EventArgs e)
{
    Random rnd = new Random();
    List<int> lst = new List<int>(){ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    for (int i = 0; i < lst.Count; i++)
    {
        int tmp = lst[i];
        lst.RemoveAt(i);
        lst.Insert(rnd.Next(lst.Count), tmp);
    }
    foreach(int i in lst) label1.Text += i + " ";
}
```



ნახ. 18. ამოცანა 4.8-ის შესრულების შედეგი. მოხდა კოდის 3-ჯერ გაშვება.

დამოუკიდებელი სამუშაო

1. მოცემულია მთელი ტიპის ერთგანზომილებიანი მასივი. მასივი შეიცავს 25 ელემენტს. მასივის ელემენტები განსაზღვრულია წინასწარ (მასივი ინიციალიზებულია). დაწერეთ პროგრამული კოდი, რომელიც დაბეჭდავს მასივის მხოლოდ იმ ელემენტებს, რომელთა მნიშვნელობაც მეტია 20-ზე და ნაკლებია 50-ზე. მიღებული შედეგი გამოიტანეთ **textBox** კომპონენტში. თუ ასეთი მნიშვნელობები არ არის, მაშინ დაიბეჭდოს შესაბამისი შეტყობინება. პროგრამული კოდი განათავსეთ ბრძანებით **button** კომპონენტზე.

2. მოცემულია მთელი ტიპის ერთგანზომილებიანი მასივი. მასივი შეიცავს 10 ელემენტს. მასივის ელემენტები განსაზღვრულია წინასწარ. დაწერეთ პროგრამული კოდი, სადაც მასივის ელემენტებს შეუცვლით ადგილებს: პირველს და ბოლოს,

მეორეს და ბოლოს წინას და ა.შ. მიღებული მასივი გამოიტანეთ **label** კომპონენტში. პროგრამული კოდი განათავსეთ ბრძანებით **button** კომპონენტზე.

3. მოცემულია მთელი ტიპის ერთგანზომილებიანი მასივი. მასივი შეიცავს 20 ელემენტს. მასივის ელემენტების შეტანა მოახდინეთ **textBox** კომპონენტიდან. დაწერეთ პროგრამული კოდი, რომელიც დაითვლის მასივში ლუწი და კენტი ელემენტების რაოდენობას. მიღებული შედეგი გამოიტანეთ **label** კომპონენტში. პროგრამული კოდი განათავსეთ ბრძანებით **button** კომპონენტზე.

4. მოცემულია მთელი ტიპის ერთგანზომილებიანი მასივი. მასივი შეიცავს 20 ელემენტს, როგორც დადებითს, ისე უარყოფითს. მასივის ელემენტები განსაზღვრულია წინასწარ (მასივი ინიციალიზებულია). დაწერეთ პროგრამული კოდი, რომელიც:

- დაბეჭდავს საწყის მასივს;
- მასივიდან ამოარჩევს 3-ის ჯერად დადებით ელემენტებს და დაბეჭდავს მათ.

მიღებული შედეგი გამოიტანეთ შესაბამისად ორ **textBox** კომპონენტში. პროგრამული კოდი განათავსეთ ბრძანებით **button** კომპონენტზე.

5 მეთოდები. მეთოდის ცნება. მეთოდის ზოგადი აღწერა. მეთოდის სახეები.

ვიცით, რომ ცვლადები ინახავენ გარკვეული ტიპის მნიშვნელობებს, მეთოდები კი ინახავენ ინსტრუქციების ერთობლიობას. ფაქტიურად, მეთოდი არის სახელდარქმეული კოდის ბლოკი, რომელიც ასრულებს გარკვეულ მოქმედებებს.

მეთოდის ზოგადი სინტაქსი ასე გამოიყურება:

წვდომის_მოდIFIკატორი დასაბრუნებელი_ტიპი მეთოდის_სახელი (პარამეტრი)

{

მეთოდის ტანი (მეთოდის კოდი) }

მოდIFIკატორის და პარამეტრების მითითება ყოველთვის არ არის აუცილებელი. თუ წვდომის მოდIFIკატორი არ არის მითითებული, მაშინ იგულისხმება დახურული private მოდIFIკატორი. დახურული private მოდIFIკატორი განსაზღვრავს დახურულ მეთოდს, რომელიც წვდომადია კლასის შიგნით. წვდომის ღია public მოდIFIკატორი გვამძლევს საშუალებას გამოვიძახოთ მეთოდი კოდის ნებისმიერი ფრაგმენტიდან. დასაბრუნებელი_ტიპი განსაზღვრავს თუ რა ტიპის მნიშვნელობის გამოთვალა უნდა განახორციელოს მეთოდმა. ხშირად გამოვიყენებთ ტერმინს: „მეთოდი აბრუნებს მნიშვნელობას“.

მეთოდის განსაზღვრა. მოდით განვსაზღვროთ მხოლოდ ერთი მეთოდი:

```
void Metodi_mshvidoba()
```

```
{
```

```
    MessageBox.Show("დილა მშვიდობისა, საქართველო!"); }
```

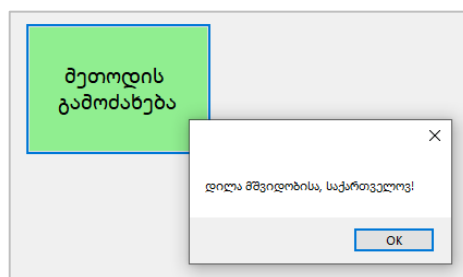
კოდის ფრაგმენტში განსაზღვრულია მეთოდი სახელით Metodi_mshvidoba(), რომელიც გასცემს რაიმე მნიშვნელობას, კერძოდ: "დილა მშვიდობისა საქართველო". სასურველია მეთოდის სახელი დავიწყოთ მაღალი რეგისტრის სიმბოლოთი.

მეთოდის სახელის წინ მითითებული არის დასაბრუნებელი მნიშვნელობის ტიპი. ამ შემთხვევაში ეს არის void ტიპი, რომელიც ფაქტიურად არაფერს აბრუნებს, ის უბრალოდ ახორციელებს გარკვეულ ქმედებებს. მეთოდის სახელის შემდეგ კი მოდის პარამეტრების ჩამონათვალი. პარამეტრების ჩამონათვალი ყოველთვის მოთავსებულია მრგვალ ფრჩხილებში, სწორედ მრგვალი ფრჩხილები მიუთითებს იმაზე, რომ მეთოდთან გვაქვს საქმე და არა ჩვეულებრივ ცვლადთან. ჩვენს შემთხვევაში მრგვალ ფრჩხილებში არაფერია ჩაწერილი, რაც ნიშნავს იმას, რომ

მეთოდი არ იღებს არანაირ პარამეტრს. პარამეტრების შემდეგ კი მოდის ფიგურული ფრჩხილების ბლოკი { }, რომელიც წარმოადგენს მეთოდის მიერ შესასრულებელი ინსტრუქციების ერთობლიობას. თუ ჩვენ ამ პატარა კოდს გავუშვებთ შესრულებაზე არც არაფერი არ მოხდება და ვერც დავინახავთ მეთოდის შესრულების შედეგს, ამდენად მეთოდის აღწერის შემდეგ აუცილებელია გამოვიძახოთ მეთოდი `Metodi_mshvidoba()`. ახლა კი ვნახოთ როგორ შეიძლება მეთოდის გამოძახება.

მეთოდის გამოძახება. იმისათვის, რომ პროგრამულ კოდში გამოვიყენოთ მეთოდი, იგი უნდა გამოვიძახოთ. მეთოდის გამოსაძახებლად უნდა მივუთითოთ მეთოდის სახელი ან მეთოდის სახელი და მეთოდის პარამეტრები, იმ შემთხვევაში თუ მეთოდი იღებს პარამეტრებს. მეთოდის გამოძახება შესაძლებელია არა მარტო ერთხელ, არამედ იმდენჯერ რამდენადაც დაგვჭირდება მისი გამოძახება. ახლა კი გავაერთიანოთ ერთიანი კოდის სახით თვითონ მეთოდის აღწერა და მეთოდის გამოძახება (იხ. ნახ. 19.):

```
void Metodi_mshvidoba()  
{  
    MessageBox.Show(" დილა მშვიდობისა, საქართველო! "); }  
private void button1_Click(object sender, EventArgs e)  
{ //მეთოდის გამოძახება  
    Metodi_mshvidoba();  
    Metodi_mshvidoba(); }
```



ნახ. 19. უპარამეტრო მეთოდის გამოძახება

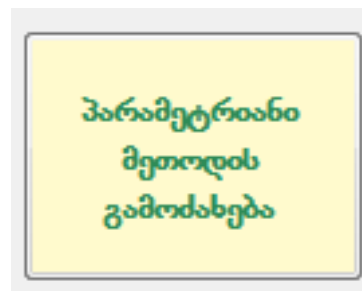
მეთოდის გამოძახება მოხდა ორჯერ, მაგრამ ზემოთ აღწერილი მეთოდის უარყოფითი მხარეა ის, რომ იგი გასცემს ერთსა და იმავე შედეგს. კარგი იქნებოდა, თუ ჩვენ დინამიურად განსაზღვრავდით, თუ რომელი შეტყობინება გამოჩნდეს ეკრანზე. C# ენაში ეს შესაძლებელია პარამეტრების გამოყენებით. პარამეტრები საშუალებას იძლევა, მეთოდს გადაეცეს გარკვეული შესატანი მონაცემები.

პარამეტრები განისაზღვრება მეთოდის სახელწოდების შემდეგ მრგვალი ფრჩხილებს შორის. პარამეტრიანი მეთოდის ზოგადი სახე შემდეგია:

მეთოდის_ტიპი მეთოდის_სახელი (პარამეტრის_ტიპი1 პარამეტრი1,
პარამეტრის_ტიპი2 პარამეტრი2...)

{ მეთოდის ტანი (მეთოდის კოდი) }

პარამეტრი შედგება ორი ნაწილისაგან: პირველი ნაწილი - ჯერ იწერება პარამეტრის ტიპი, მეორე ნაწილი - შემდეგ იწერება პარამეტრის სახელი (იხ. ნახ. 20. და ნახ. 21.).



ნახ. 20. პარამეტრიანი მეთოდის გამოძახების ფორმა/დიზაინი

ასე მაგალითად, განვსაზღვროთ მეთოდი, რომლის სახელია PrintMessage(). ძირითადი კოდიდან მეთოდს გადაეცემა გარკვეული ტიპის მონაცემი/ინფორმაცია.

```
void PrintMessage(string message)
{
    MessageBox.Show(message); }
private void button1_Click(object sender, EventArgs e)
{
    PrintMessage("Hello work");    // Hello work
    PrintMessage("Hello METANIT.COM"); // Hello METANIT.COM
    PrintMessage("Hello C#");      // Hello C#
}
```



ნახ. 21. პარამეტრიანი მეთოდის PrintMessage()-ის შესრულების შედეგი

მეთოდი PrintMessage() არის void ტიპისა. მეთოდს PrintMessage()-ს აქვს ერთი პარამეტრი. პარამეტრის სახელია message, პარამეტრის ტიპია string ე. ი. მეთოდი PrintMessage() არის void ტიპის ერთ პარამეტრიანი (string ტიპის) მეთოდი.

იმისათვის, რომ მეთოდმა იმუშაოს, გამოძახებისას მეთოდის სახელის შემდეგ მრგვალ ფრჩხილებში უნდა ჩავწეროთ პარამეტრისათვის გადასაცემი მნიშვნელობა, მაგ: PrintMessage("Hello work"); პარამეტრს message გადაეცემა სტრიქონი "Hello work". მნიშვნელობები/მონაცემები, რომლებიც გადაეცემა პარამეტრს, იწოდებიან არგუმენტებად. ჩვენს შემთხვევაში პარამეტრისათვის გადაცემული სტრიქონი "Hello work" წარმოადგენს არგუმენტს.

ზოგჯერ შეიძლება შეგვხვდეს შემდეგი სახის ტერმინები: ფორმალური პარამეტრი და ფაქტიური პარამეტრი. ფორმალური პარამეტრი ეს საკუთრივ მეთოდის პარამეტრია (ჩვენს შემთხვევაში ეს არის პარამეტრი message), ფაქტიური კი არის ის პარამეტრი, რომელიც გადაეცემა ფორმალურ პარამეტრს ე.ი. ფაქტიური პარამეტრი არის მეთოდის არგუმენტი.

განვსაზღვროთ მეთოდი, რომელიც შეკრებს მთელი ტიპის ორ რიცხვს:

```
void Sum(int x, int y)
{
    int result = x + y;
    MessageBox.Show("ორი რიცხვის ჯამი = " + result);
}
private void button1_Click(object sender, EventArgs e)
{
    Sum(10, 15); // 10 + 15 = 25
}
```

მეთოდს Sum() გააჩნია ორი პარამეტრი: x და y. ოპივე პარამეტრი არის int ტიპისა. ამიტომ მეთოდის გამოძახების შემდეგ ჩვენ აუცილებლად უნდა გადავცეთ ამ ორი პარამეტრის ნაცვლად, ორი მთელი ტიპის რიცხვი. მეთოდის Sum(10, 15) გამოძახებისას x პარამეტრს გადაეცემა 10, ხოლო y პარამეტრს კი 15. მეთოდს შეიძლება გადაეცეს მონაცემები გამოსახულების სახითაც, მაგალითად:

```
int a = 10, b = 15, c = 6;
Sum(a, b); // 10 + 15 = 25
Sum(3, c); // 3 + 6 = 9
Sum(14, 4 + c); // 14 + 10 = 24
```

თუ ჩვენ უნდა გადავცეთ ცვლადების მნიშვნელობები, მაშინ ეს ცვლადები აუცილებლად უნდა იყოს ინიციალიზებული ანუ თავიდანვე მათ უნდა ჰქონდეთ მინიჭებული რიცხვითი მნიშვნელობები. ქვედა პროგრამული კოდი არ იმუშავებს და გამოგვითვას შეტყობინებას შეცდომის შესახებ:

```
void Sum(int x, int y)
{
    MessageBox.Show("ორი რიცხვის ჯამი = " + (x + y)); }
private void button1_Click(object sender, EventArgs e)
{
    int a;
    int b = 15;
    Sum(a, b); // შეცდომა!!! }
```

განვიხილოთ პარამეტრებისა და არგუმენტების შესაბამისობა მონაცემთა ტიპის მიხედვით. არგუმენტების გადაცემის დროს მნიშვნელოვანია ვიცოდეთ, რომ არგუმენტებსა და პარამეტრებს შორის უნდა არსებობდეს ტიპების შესაბამისობა ანუ თანხვედრა, თუ არა და პროგრამული კოდი გასცემს შეტყობინებას შეცდომის შესახებ, მაგალითად:

```
void adamianis_monacemebi(string name, int age)
{
    MessageBox.Show("სახელი - " + name + ", " + "ასაკი - " + age); }
private void button1_Click(object sender, EventArgs e) {
    adamianis_monacemebi("ლუკა", 21); }
```

მოცემულ კოდში მეთოდის adamianis_monacemebi() პირველი პარამეტრი არის string ტიპისა, ამდენად მეთოდის გამოძახებისას ამ პარამეტრს უნდა გადავცეთ string ტიპის მნიშვნელობა, ანუ სტრიქონი.

მეორე პარამეტრი კი წარმოადგენს მთელ int ტიპს, ამიტომ მას უნდა გადაეცეს მთელი ტიპის მონაცემი, როგორც კოდშია მოცემული:

```
adamianis_monacemebi("ლუკა", 21);
```

სხვა ტიპის მონაცემებს ჩვენ ვერ გადავცემთ პარამეტრებს. მაგალითად, შემდეგი სახის მეთოდის გამოძახება გამოიწვევს შეცდომას:

```
adamianis_monacemebi(45, "ნინო"); //შეცდომა!!!
```

ზემოთ განხილულ მაგალითებში მეთოდი არის void ტიპისა. void ტიპის მეთოდები არ აბრუნებენ არანაირ მნიშვნელობას, ისინი მხოლოდ ასრულებენ რაიმე მოქმედებას, მაგრამ მეთოდმა შეიძლება დააბრუნოს მნიშვნელობა. მნიშვნელობის დასაბრუნებლად გამოიყენება return ოპერატორი.

return ოპერატორის შემდეგ ვწერთ დასაბრუნებელ მნიშვნელობას.

return ოპერატორის ზოგადი სახეა:

return დასაბრუნებელი მნიშვნელობა;

დავწეროთ მეთოდი, რომელიც აბრუნებს string ტიპის მნიშვნელობას, მაგალითად წინადადებას „დილა მშვიდობისა“:

```
string metodi_dila() {  
    return "დილა მშვიდობისა"; }
```

მეთოდი metodi_dila() არის string ტიპისა, ამიტომ მან შესაბამისად უნდა დააბრუნოს string ტიპის მნიშვნელობა. რადგან მეთოდი არის string ტიპისა, (და არა void ტიპისა) ამიტომაც მეთოდის ტანში უნდა გამოვიყენოთ return ოპერატორი, რომლის შემდეგაც იწერება დასაბრუნებელი მნიშვნელობა, ჩვენს შემთხვევაში: „დილა მშვიდობისა“.

მეთოდის დასაბრუნებელ მნიშვნელობასა და მეთოდის ტიპს შორის აუცილებლად უნდა იყოს თანხვედრა. თუ ჩვენ დასაბრუნებელ მნიშვნელობად მივუთითებთ რიცხვით მნიშვნელობას ეს გამოიწვევს შეცდომას, ასე მაგალითად:

```
string metodi_dila() {  
    //მეთოდმა უნდა დააბრუნოს სტრიქონი და არა რიცხვი ან ციფრი  
    return 9; }
```

მეთოდის შედეგი ანუ რეზულტატი შეგვიძლია მივანიჭოთ რაიმე ცვლადს, როგორც ქვედა კოდშია მოცემული:

```
string metodi_dila() {  
    return "დილა მშვიდობისა"; }  
private void button1_Click(object sender, EventArgs e) {  
    //ცვლადისათვის მეთოდის მნიშვნელობის მინიჭება  
    string shedegi = metodi_dila();  
    label1.Text = shedegi; }
```

return ოპერატორის შემდეგ შეგვიძლია გამოვიყენოთ გამოსახულებაც, როგორც შემოთ არის მოცემული:

```
int jami(int x, int y) {  
    return x + y; }  
private void button1_Click(object sender, EventArgs e) {  
    int shedegi = jami(3,6);  
    label1.Text = shedegi.ToString(); }
```

მეთოდს jami() აქვს int ტიპი, შესაბამისად, მან უნდა დააბრუნოს int ტიპის მნიშვნელობა ანუ მთელი რიცხვი ან მთელი ციფრი.

ახლა კი გავარკვიოთ თუ როგორ ხდება მეთოდის გამოძახება. მეთოდის გამოძახებისას სრულდება შემდეგი მოქმედებები:

1. გამოსახულების გამოთვლა, რომელიც არგუმენტის ნაცვლად იმყოფება;
2. პარამეტრებისათვის მუხსიერების გამოყოფა მისი ტიპის მიხედვით;
3. თითოეულ პარამეტრს გადაეცემა შესაბამისი არგუმენტები;
4. სრულდება მეთოდის კოდი;
5. თუ მეთოდი აბრუნებს მნიშვნელობას, იგი გადაეცემა გამოძახების წერტილს; თუ მეთოდი არის void ტიპისა, მაშინ მართვა გადაეცემა გამოძახებული მეთოდის შემდეგ მყოფ ოპერატორს.

გადავიდეთ ამოცანების განხილვაზე.

ამოცანა 5.1. მოცემული გვაქვს სამკუთხედი სამივე გვერდით: a, b და c. გამოვთვალოთ სამკუთხედის ფართობი ჰერონის ფორმულის მიხედვით. ფართობის გამოთვლა დავაკისროთ მეთოდს სახელით Area. მეთოდს Area აქვს სამი პარამეტრი. ძირითად კოდში კი დავადგინოთ შეტანილი გვერდებით სამკუთხედი შეიკვრება თუ არა. თუ სამკუთხედი შეიკვრება, მაშინ დავბეჭდოთ ფართობის მნიშვნელობა, თუ არ შეიკვრება კი გამოვიდეს შეტყობინება “სამკუთხედი არ არსებობს”.

```
double S,a,b,c;  
private double Area(double a, double b, double c) {  
    double p = (a + b + c) / 2;  
    double s = Math.Sqrt(p * (p - a)*(p - b)*(p - c));  
    return s; }  
private void button1_Click(object sender, EventArgs e) {  
    a = double.Parse(textBox1.Text);
```

```

b = double.Parse(textBox2.Text);
c = double.Parse(textBox3.Text);
if ((a < b + c) && (b < a + c) && (c < a + b))
{
    S = Area(a, b, c);
    label1.Text = Convert.ToString(S); }
else
    label1.Text = "სამკუთხედი არ არსებობს"; }

```

ამოცანა 5.2. მოცემულია კვადრატული განტოლება. ვიპოვოთ განტოლების ფესვები. კვადრატული განტოლება არის $ax^2 + bx + c = 0$ სახის მრავალწევრი, სადაც x ცვლადია, ხოლო a, b, c რიცხვებია და $a \neq 0$ -ს. a -ს ეწოდება კვადრატული განტოლების პირველი კოეფიციენტი, b -ს მეორე კოეფიციენტი, ხოლო c -ს თავისუფალი წევრი.

თუ $a = 1$ განტოლებას დაყვანილი სახის განტოლება ეწოდება.

$D = b^2 - 4ac$ გამოსახულების გამოთვლით მიღებულ რიცხვს ეწოდება $ax^2 + bx + c = 0$ მრავალწევრის (კვადრატული განტოლების) დისკრიმინანტი.

დისკრიმინანტი შეიძლება იყოს ან დადებითი ან ნულის ტოლი ან უარყოფითი.

თუ $D > 0$, მაშინ განტოლებას ორი სხვადასხვა x_1 და x_2 ამონახსნი აქვს:

$$x_1 = \frac{-b - \sqrt{D}}{2a}; \quad x_2 = \frac{-b + \sqrt{D}}{2a};$$

თუ $D = 0$, მაშინ ორივე ფესვი არსებითია და ერთმანეთის ტოლია:

$$x_0 = \frac{-b}{2a}$$

თუ $D < 0$, მაშინ განტოლებას ამონახსნი არა აქვს.

პროგრამა დავტესტოთ შემდეგი მონაცემებისათვის (იხ. ნახ. 22.):

ა) (1, 4, 1) - ორი ამონახსნი; ბ) (1, 2, 1) - ერთი ამონახსნი; გ) (1, 1, 1) - არ არის ამონახსნი.

ნახ. 22. ამოცანა 5.2.-ის ფორმა/დიზაინი

```
private void kvadratuligantoleba(){
```

```

double a, b, c, d, x1, x2;
try {
    a = double.Parse(textBox1.Text); }
catch {
    MessageBox.Show("შეცდომა!!! A კოეფიციენტის ველში შეიტანეთ
მხოლოდ რიცხვითი მნიშვნელობა");
    return; }
try {
    b = double.Parse(textBox2.Text); }
catch {
    MessageBox.Show("შეცდომა!!! B კოეფიციენტის ველში შეიტანეთ მხოლოდ
რიცხვითი მნიშვნელობა");

    return; }
try {
    c = double.Parse(textBox3.Text); }
catch{
    MessageBox.Show("შეცდომა!!! C კოეფიციენტის ველში შეიტანეთ
მხოლოდ რიცხვითიმნიშვნელობა");

    return; }
d = b * b - 4 * a * c;
if (d > 0)
{
    x1 = (-b + Math.Sqrt(d)) / (2 * a);
    x2 = (-b - Math.Sqrt(d)) / (2 * a);
    textBox4.Text = Convert.ToString(x1);
    textBox5.Text = Convert.ToString(x2); }
else if (d == 0)
{
    x1 = (-b + Math.Sqrt(d)) / (2 * a);
    textBox4.Text = Convert.ToString(x1);
    textBox5.Text = ""; }
else
{
    textBox4.Text = "ამონახსნი არ აქვს";

```



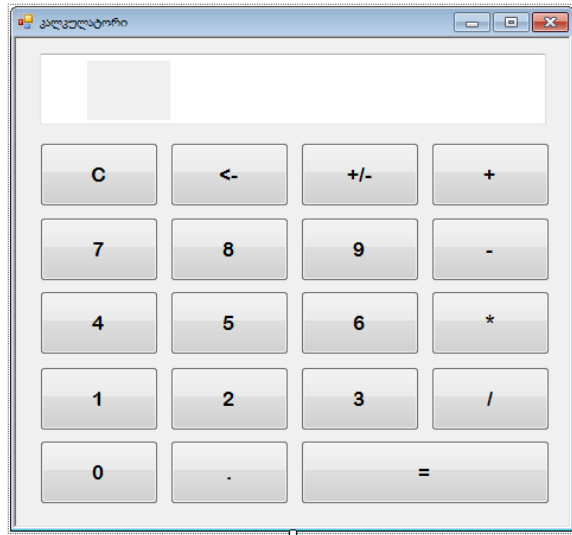
```

        textBox5.Text = ""; }
    }
    private void button1_Click(object sender, EventArgs e) {
        kvadratuliigantoleba(); //მეთოდის გამოძახება }

```

ამოცანა 5.3. კალკულატორის მუშაობის მარტივი კოდის განხილვა.

ფორმაზე კალკულატორი ააწყვეთ ქვემოთ მოცემული ნიმუშის მიხედვით. გამოიყენეთ ცხრამეტი button, ერთი textbox და ერთი label კომპონენტები (იხ. ნახ. 23.).

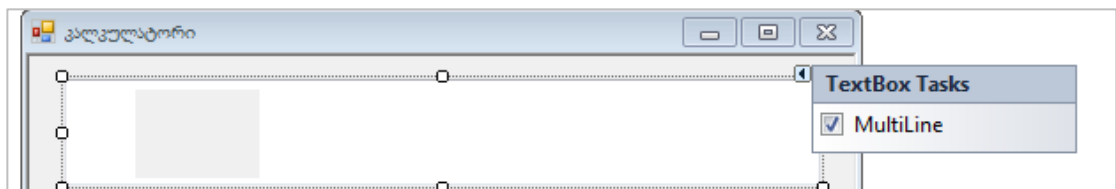


ნახ. 23. კალკულატორის ფორმის აგება-აწყობა

textBox კომპონენტსათვის დააყენეთ MultiLine რეჟიმი ქვემოთ მოცემული ნიმუშის მიხედვით ამასთანავე textBox კომპონენტსათვის Properties მახასიათებელში TextAlign ჩამონათვალში აირჩიეთ Right.

label კომპონენტის Text მახასიათებელში წაშალეთ ტექსტი.

label კომპონენტის MaximumSize და MinimumSize ზომები დააყენეთ ქვემოთ მოცემული ნიმუშის მიხედვით ან აირჩიეთ საკუთარი შეხედულებისამებრ (იხ. ნახ. 24.).



ნახ. 24. MultiLine რეჟიმის და მახასიათებლების არჩევა

პროგრამული კოდი:

პროგრამის დასაწყისში წინასწარ განსაზღვრეთ გლობალური ცვლადები:

```
float a, b; int count; bool nishani = true;
```

button ღილაკებზე, სადაც ციფრებია გამოსახული უნდა დაიწეროს ქვემოთ მოცემული კოდი. მაგალითად აღვწეროთ „0“ ღილაკზე დაწერილი კოდი.

ღილაკზე „0“ დაიწერება შემდეგი კოდი:

```
private void button17_Click(object sender, EventArgs e) {  
    textBox1.Text = textBox1.Text + 0; }  
ასეთივე კოდი დაიწერება დანარჩენ ციფრებზე.
```

ათწილადი ნიშნის „.“ გამომსახველ ღილაკზე დაიწერება შემდეგი კოდი:

```
private void button18_Click(object sender, EventArgs e) {  
    textBox1.Text = textBox1.Text + ","; }  
ღილაკზე „+“ დაიწერება შემდეგი კოდი:
```

```
private void button4_Click(object sender, EventArgs e) {  
    a = float.Parse(textBox1.Text);  
    textBox1.Clear();  
    count = 1;  
    label1.Text = a.ToString() + "+";  
    nishani = true; }  
ღილაკზე „-“ დაიწერება შემდეგი კოდი:
```

```
private void button8_Click(object sender, EventArgs e) {  
    a = float.Parse(textBox1.Text);  
    textBox1.Clear();  
    count = 2;  
    label1.Text = a.ToString() + "-";  
    nishani = true; }  
ღილაკზე „*“ დაიწერება შემდეგი კოდი:
```

```
private void button12_Click(object sender, EventArgs e) {  
    a = float.Parse(textBox1.Text);  
    textBox1.Clear();  
    count = 3;  
    label1.Text = a.ToString() + "*";  
    nishani = true; }  
ღილაკზე „/“ დაიწერება შემდეგი კოდი:
```

```
private void button16_Click(object sender, EventArgs e) {  
    a = float.Parse(textBox1.Text);
```

```

textBox1.Clear();
count = 4;
label1.Text = a.ToString() + "/";
nishani = true; }

```

ლილაკზე „=“ დაიწერება შემდეგი კოდი, აქვე ასევე უნდა აღვწეროთ მეთოდი calculate():

```

private void button19_Click(object sender, EventArgs e){
    calculate(); //მეთოდის პროტოტიპი
    label1.Text = "";
    //მეთოდის აღწერა
private void calculate() {
    switch (count)
    {
        case 1:
            b = a + float.Parse(textBox1.Text);
            textBox1.Text = b.ToString();
            break;
        case 2:
            b = a - float.Parse(textBox1.Text);
            textBox1.Text = b.ToString();
            break;
        case 3:
            b = a * float.Parse(textBox1.Text);
            textBox1.Text = b.ToString();
            break;
        case 4:
            b = a / float.Parse(textBox1.Text);
            textBox1.Text = b.ToString();
            break;
        default:
            break;
    }
}

```

ლილაკზე „C“ დაიწერება შემდეგი მარტივი კოდი:

```

private void button1_Click(object sender, EventArgs e) {

```

```
textBox1.Text = "";  
label1.Text = ""; }
```

დილაკზე „<-“ დაიწერება შემდეგი კოდი (ბოლო ციფრის წაშლა):

```
private void button2_Click(object sender, EventArgs e) {  
    int lenght = textBox1.Text.Length - 1;  
    string text = textBox1.Text;  
    textBox1.Clear();  
    for (int i = 0; i < lenght; ++i)  
    {  
        textBox1.Text = textBox1.Text + text[i];  
    }  
}
```

დილაკზე „+/-“ დაიწერება შემდეგი კოდი:

```
private void button3_Click(object sender, EventArgs e) {  
    if (nishani == true)  
    {  
        textBox1.Text = "-" + textBox1.Text;  
        nishani = false;  
    }  
    else if (nishani == false)  
    {  
        textBox1.Text = textBox1.Text.Replace("-", "");  
        nishani = true;  
    }  
}
```

ქვემოთ კი მოცემულია კალკულატორის პროგრამული კოდი თავიდან
ბოლომდე:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Windows.Forms;
```

```

namespace WindowsFormsApplication45
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        float a, b; int count;
        bool nishani = true;
private void button17_Click(object sender, EventArgs e)
    { textBox1.Text = textBox1.Text + 0; }
private void button13_Click(object sender, EventArgs e)
    { textBox1.Text = textBox1.Text + 1; }
private void button14_Click(object sender, EventArgs e)
    { textBox1.Text = textBox1.Text + 2; }
private void button15_Click(object sender, EventArgs e)
    { textBox1.Text = textBox1.Text + 3; }
private void button9_Click(object sender, EventArgs e)
    { textBox1.Text = textBox1.Text + 4; }
private void button10_Click(object sender, EventArgs e)
    { textBox1.Text = textBox1.Text + 5; }
private void button11_Click(object sender, EventArgs e)
    { textBox1.Text = textBox1.Text + 6; }
private void button5_Click(object sender, EventArgs e)
    { textBox1.Text = textBox1.Text + 7; }
private void button6_Click(object sender, EventArgs e)
    { textBox1.Text = textBox1.Text + 8; }
private void button7_Click(object sender, EventArgs e)
    { textBox1.Text = textBox1.Text + 9; }
private void button18_Click(object sender, EventArgs e)
    { textBox1.Text = textBox1.Text + ","; }
private void button4_Click(object sender, EventArgs e)
    { a = float.Parse(textBox1.Text);
      textBox1.Clear();
      count = 1;
      label1.Text = a.ToString() + "+";
      nishani = true; }
private void button8_Click(object sender, EventArgs e)
    { a = float.Parse(textBox1.Text);
      textBox1.Clear();

```

```

        count = 2;
        label1.Text = a.ToString() + "-";
        nishani = true; }
private void button12_Click(object sender, EventArgs e)
{
    a = float.Parse(textBox1.Text);
    textBox1.Clear();
    count = 3;
    label1.Text = a.ToString() + "*";
    nishani = true;
}
private void button16_Click(object sender, EventArgs e)
{
    a = float.Parse(textBox1.Text);
    textBox1.Clear();
    count = 4;
    label1.Text = a.ToString() + "/";
    nishani = true;
}
private void button19_Click(object sender, EventArgs e)
{
    calculate(); //მეთოდის გამოძახება
    label1.Text = "";
}
private void calculate()
{
    switch (count)
    {
        case 1:
            b = a + float.Parse(textBox1.Text);
            textBox1.Text = b.ToString(); break;
        case 2:
            b = a - float.Parse(textBox1.Text);
            textBox1.Text = b.ToString(); break;
        case 3:
            b = a * float.Parse(textBox1.Text);
            textBox1.Text = b.ToString(); break;
        case 4:
            b = a / float.Parse(textBox1.Text);
            textBox1.Text = b.ToString(); break;
        default: break;
    }
}
private void button1_Click(object sender, EventArgs e)

```

```

        { textBox1.Text = "";
          label1.Text = "";
        }
private void button2_Click(object sender, EventArgs e)
{
    int lenght = textBox1.Text.Length - 1;
    string text = textBox1.Text;
    textBox1.Clear();
    for (int i = 0; i < lenght; ++i)
    {
        textBox1.Text = textBox1.Text + text[i];
    }
}
private void button3_Click(object sender, EventArgs e)
{
    if (nishani == true)
    {
        textBox1.Text = "-" + textBox1.Text;
        nishani = false;
    }
    else if (nishani == false)
    {
        textBox1.Text = textBox1.Text.Replace("-", "");
        nishani = true;
    }
}
}
}

```

დამოუკიდებელი სამუშაო

1. მოცემული გვაქვს გეომეტრიული ფიგურა წრე. ფუნქციის დახმარებით დაწერეთ ისეთი პროგრამული კოდი, რომელიც გამოითვლის წრის დიამეტრს, ფართობს და რადიუსს. ფუნქცია არ ახდენს მიღებული შედეგის დაბრუნებას.

2. მოცემული გვაქვს 1-დან 10-მდე ნატურალური რიცხვების თანამიმდევრობა. ფუნქციის საშუალებით დაწერეთ ისეთი კოდი, რომელიც გამოითვლის ნატურალური რიცხვების კვადრატების მნიშვნელობებს. გაითვალისწინეთ, რომ ფუნქცია ახდენს მიღებული შედეგის დაბრუნებას.

3. მეორე ამოცანაში დაწერილი ფუნქცია გადააკეთეთ შემდეგი სახით: ფუნქციამ არ უნდა მოახდინოს მიღებული შედეგის დაბრუნება; შედეგი დაიბეჭდოს ძირითად პროგრამულ კოდში.

4. ფუნქციის გამოყენებით შევადაროთ რაიმე რიცხვის მნიშვნელობა ნულს. თუ რიცხვი დადებითია მაშინ პროგრამას გადაეცემა 1, თუ რიცხვი უარყოფითია მაშინ პროგრამას გადაეცემა -1, თუ რიცხვი ნულოვანია მაშინ ნული. რიცხვითი მნიშვნელობა შევიტანოთ კლავიატურიდან.

5. ფუნქციის დახმარებით დავწეროთ ისეთი პროგრამა, რომელიც გამოითვლის რიცხვის პროცენტს. მაგალითად, 3-ის 321% ტოლია 9.63-ის. რიცხვითი მნიშვნელობა შევიტანოთ კლავიატურიდან.

6. დავწეროთ a რიცხვის b ხარისხში აყვანის ფუნქცია და ძირითადი კოდი. a და b რიცხვების რიცხვითი მნიშვნელობა შევიტანოთ კლავიატურიდან.

6

ობიექტზე ორიენტირებული პროგრამირების პრინციპები: ინკაფსულაცია, აბსტრაქცია, პოლიმორფიზმი და მემკვიდრეობითობა. კლასის გამოცხადება. ობიექტების შექმნა. public, private და protected მოდიფიკატორები. მეთოდის გამოცხადება. მეთოდიდან შედეგის დაბრუნება. პარამეტრების გამოყენება.

ობიექტზე ორიენტირებული პროგრამირება (ოპზ) - ეს არის პარადიგმა დაფუძნებული ობიექტების სისტემაზე, რომელებიც შეიცავენ გარკვეული ტიპის (სახის) მონაცემებსა და მეთოდებს.

ობიექტზე ორიენტირებული პროგრამირების (ოპზ) მთავარი დანიშნულებაა რთული პროცესების და კოდების რაც შეიძლება მარტივი სახით წარმოდგენა. ამისათვის ხდება პროგრამის დაყოფა დამოუკიდებელ ბლოკებად - ობიექტებად.

ობიექტი არის მონაცემების და ფუნქციების ნაკრები. წარმოვიდგინოთ, რომ პროგრამის რომელიმე ფრაგმენტი გადავიტანეთ რაიმე ყუთში, ყუთს კი დავახურეთ თავსახური. სწორედ ასეთი თავდახურული ყუთი არის ობიექტი.

პროგრამისტები შეთანხმდნენ, რომ ობიექტის შიგნით არსებულ მონაცემებს ეწოდოს თვისებები, ხოლო ფუნქციებს კი მეთოდები ანუ თავისი არსით ეს არის იგივე ცვლადები და ფუნქციები. სამზარეულოში არსებული ნებისმიერი ელექტრო მოწყობილობა შეგვიძლია წარმოვიდგინოთ როგორც ობიექტი.

მაგალითად ჩაიდანნი, ელექტროქურა, ბლენდერი და ა. შ. თითოეულ მათგანს გააჩნია ძრავა, კონტროლერები, ღილაკები, ზამბარები, დამცავები, მაგრამ როდესაც საჭმელს ვამზადებთ არ ვფიქრობთ ასეთ დეტალებზე და სწორედ ელექტრო მოწყობილობაში მათი ერთობლივი მუშაობით ვიმზადებთ საუზმეს ან სადილს. ობიექტები ხასიათდებიან შემდეგი ოთხი პრინციპით: ინკაფსულაცია, აბსტრაქცია, მემკვიდრეობა და პოლიმორფიზმი. მოდით, განვიხილოთ თითოეული მათგანი.

ინკაფსულაცია - არის მონაცემებზე წვდომის და მასზე ცვლილების განხორციელების შეზღუდვა. ინკაფსულაციის დროს ობიექტი არის დამოუკიდებელი - ანუ თითოეული ობიექტი ისეა მოწყობილი, რომ მისთვის საჭირო მონაცემები „ცოცხლობენ“ ობიექტის შიგნით და არა პროგრამის გარეთ. მაგალითად, თუ ჩვენ გავქვს ობიექტი „მომხმარებელი“, მაშინ ჩვენ გვაქვს ამ

ობიექტის შესახებ მონაცემებიც: სახელი, მისამართი და ა.შ. ამავე ობიექტში გვექნება მეთოდები: „მისამართის შემოწმება“ ან „ელ. ფოსტის შემოწმება“ და ა.შ.

ინკასულაციის პრინციპის ქვეშ იგულისხმობა, რომ ობიექტის ყველა თვისება და მეთოდი რჩება დახურული და დაცული სხვა ობიექტებისაგან. თითოეულ ობიექტში შეგვიძლია გვექონდეს როგორც დახურული, ასევე ღია ცვლადები და მეთოდები. დახურული ცვლადების და მეთოდების გამოძახება ან გამოყენება (წვდომა) სხვა ობიექტების მიერ არ შეიძლება, ხოლო ღია ცვლადებისა კი შეიძლება.

მაგალითად, გვყავს კერძო ავტომობილი. ჩვენს ავტომობილს აქვს შემდეგი ატრიბუტები: ფერი, გამოშვების წელი, მოდელი და ა.შ. ეს ატრიბუტები არის დახურული ანუ კერძო მონაცემები. სწორედ ამ კერძო მონაცემების შეცვლა არ შეუძლიათ სხვა ობიექტებს. ინკასულაციის ყველაზე უფრო გავრცელებული მაგალითი - ეს არის წვდომის, ხილვადობის მოდიფიკატორები private, public და ა.შ.

აბსტრაქცია - აბსტრაქცია შეიძლება განვიხილოთ, როგორც ინკასულაციის უფრო განზოგადოებული სახე, ასევე ობიექტისათვის ისეთი მახასიათებლების მინიჭების საშუალება, რომელიც განასხვავებს მას სხვა ობიექტებისაგან. აბსტრაქცია საკმაოდ მარტივი პრინციპია. აბსტრაქცია ნიშნავს ობიექტის ძირითადი, ყველაზე მნიშვნელოვანი მახასიათებლების ამორჩევას და მეორეხარისხოვან, უმნიშვნელო მახასიათებლების უგულველყოფას. აბსტრაქციის პრინციპი ობიექტს განიხილავს როგორც „ინტერფეისს“ - ობიექტს აქვს მეთოდი და თვისება, რომელსაც ჩვენ შეგვიძლია მივმართოთ ამ ობიექტის გარედან. ეს შეგვიძლია შევადაროთ ბლენდერის გარეთა კორპუსზე არსებულ ღილაკს. ბლენდერს აქვს უამრავი შიგა მოწყობილობა, რომლითაც ის მუშაობს, მაგრამ გარეთა პანელზე ვხედავთ მხოლოდ ღილაკს, რომელზე დაჭერაც ამუშავებს ამ შიგა მოწყობილობებს. აი, სწორედ ბლენდერის გარეთა კორპუსზე მყოფი ღილაკი, რომელსაც ხელი დავაჭირეთ ბლენდერის ასამუშავებლად, არის ინტერფეისი.

პროცედურულ პროგრამირებაში შეგვიძლია ვთქვათ, „მომხმარებელი წავშალოთ“. ობიექტ ორიენტირებულ დაპროგრამების ენაზე კი გვექნება: „მომხმარებელი.წავშალოთ()“ ანუ ჩვენ მივმართავთ ობიექტს „მომხმარებელი“ და ვიძახებთ მეთოდს „წავშალოთ“. ოპ-ში არც თუ ისე მნიშვნელოვანია თუ ზუსტად როგორ მოხდება წაშლა ანუ ობიექტ ორიენტირებულ პროგრამირება გვაძლევს

საშუალებას, ობიექტზე („მომხმარებელი“) მიმართვის დროს ჩვენ არ ვიფიქროთ მეთოდ „წაშლაზე“.

მაგალითად: მაღაზიის მუშაობის პროგრამულ უზრუნველყოფაზე მუშაობს ორი პროგრამისტი. ერთი წერს მოდულს „შეკვეთა“, მეორე კი მოდულს „მიწოდება“. პირველ პროგრამისტს აღწერილი აქვს ობიექტი „შეკვეთა“, რომელსაც აქვს მეთოდი „გაუქმება“.

მეორე პროგრამისტს („მიწოდება“) დასჭირდა შეკვეთის გაუქმება და ის თავისუფლად წერს შემდეგ კოდს: „მიწოდება.გაუქმება()“. მეორე პროგრამისტს არც კი აინტერესებს თუ პირველი პროგრამისტი როგორ ახორციელებს გაუქმების პროცესს.

მემკვიდროებითობა - არის ასლის შექმნის უნარი. ობიექტ ორიენტირებული პროგრამირება გვაძლევს საშუალებას, შევქმნათ მრავალი ობიექტი სხვა ობიექტის მსგავსად და სახედ, ამის ხარჯზე ჩვენ არ გვჭირდება კოდი „დავაკოპიროთ“ მრავალჯერ, არამედ ერთხელ გამართულად დავწეროთ და შემდეგ გამოვიყენოთ იმდენჯერ რამდენჯერაც ეს დაგვჭირდება.

როგორც ვიცით, პროგრამული კოდები შეიცავენ ათასობით სტრიქონს. თუ ჩვენ ყოველი ახალი ობიექტისათვის ყოველ ჯერზე მოგვიწია კოდის თავიდან ანუ ნულიდან დაწერა, მაშინ ეს ჩვენ მიგვიყვანს უამრავ შეცდომასთან და წაგვართმევს ძალიან დიდ დროს.

ამ პრობლემის გადაწყვეტა შესაძლებელია მემკვიდროებითობის პრინციპის გამოყენებით. მემკვიდროება არის მექანიზმი, რომელიც საშუალებას გაძლევთ აღწეროთ ახალი კლასი არსებული (შშობელი) კლასზე დაყრდნობით. ახალ (მემკვიდრე) კლასს გადაეცემა ძირითადი (შშობელი) კლასის თვისებები და ფუნქციონირება. რა არის მემკვიდროება და რა უპირატესობა აქვს მას? უპირველესად შესაძლებელია კოდის განმეორებითი გამოყენება, ასევე შშობელ კლასში განსაზღვრული წევრები (ველები) და მეთოდები შეიძლება გამოყენებულ იქნას შვილობილ კლასებში.

ვთქვათ, გვაქვს სრულყოფილი, იდეალური ობიექტი „მომხმარებელი“: მასში აღწერილი გვაქვს ყველაფერი რაც შეიძლება ახასიათებდეს მომხმარებელს, მაგალითად, თვისებები: სახელი, გვარი, ასაკი, მისამართი, პირადი ნომერი და ა.შ. ასევე შეიძლება გვქონდეს მეთოდები: „ფასდაკლება“, „შეკვეთა“, „შეკვეთის მოძებნა“, „აუდიოზარი“ და ა.შ.

ამ იდეალური მომხმარებლის საფუძველზე ჩვენ შეგვიძლია შევექმნათ რეალური „მხატვარი ლუკა“. რეალურ „მხატვარ ლუკას“ გადაეცემა ყველა ის თვისება და მეთოდი, რომელიც გააჩნია იდეალურ მომხმარებელს, მაგრამ ამასთანავე შეიძლება დაიმატოს საკუთარი (რეალური „მხატვარი ლუკას“) თვისებები და მეთოდები. პროგრამისტები კლასებს უწოდებენ იდეალურ ობიექტებს.

პოლიმორფიზმი - არის ურთიერთობის ერთიანი ენა. ოპ-ში მნიშვნელოვანია, რომ ობიექტები ურთიერთქმედებდნენ ერთმანეთთან მათთვის გასაგებ ენაზე. თუ სხვადასხვა ობიექტებს აქვთ მეთოდი „წაშლა“, მაშინ მან უნდა შეასრულოს სწორედ ეს ფუნქცია და არა სხვა მიმსგავსებული ქმედება. დაუშვებელია, რომ ერთ ობიექტთან მუშაობდეს როგორც „წაშლა“, ხოლო მეორესთან - როგორც „გადაშლა“. მაგალითად: საქონლის წაშლის ქვეშ იგულისხმება შეტყობინების ფანჯრის გამოსვლა და მონაცემთა ბაზაში საქონლის, როგორც წაშლილის, მონიშვნა. ხოლო მომხმარებლის წაშლის ქვეშ იგულისხმება შენაძენის გაუქმება, გამოწერების ჩამოწერა და შენაძენების დააარქივება. ვხედავთ, რომ სხვადასხვა მოვლენასთან გვაქვს საქმე, მაგრამ პროგრამისათვის ეს არ არის მნიშვნელოვანი, რადგან მას აქვს მეთოდი „წაშლა()“ და ის ამ მეთოდს უცხადებს ნდობას.

პოლიმორფიზმი საშუალებას გვაძლევს, ობიექტი გამოვიყენოთ ზუსტად ისე, როგორც მშობელი კლასი, მაგრამ ამასთანავე მოხდეს, ობიექტის მიერ, მემკვიდრე კლასის თვისებების შენარჩუნებაც. მაგალითად, სატვირთო მანქანით შესაძლებელია გადაადგილება, მაგრამ ამასთანავე სატვირთო მანქანა წარმოადგენს ტვირთის გადაზიდვის საშუალებას ქალაქის ერთი წერტილიდან მეორე წერტილში. ობიექტზე ორიენტირებული პროგრამირება ხელს უწყობს პროგრამირების ძირითადი პრინციპების პრაქტიკაში განხორციელებას და ეხმარება გადაწყვიტოს კოდის უნიკალურობის და მართვადობის პრობლემები რთულ ინფორმაციულ ტექნოლოგიურ პროექტებში. ზოგადად ეს პრინციპები შეგვიძლია ასე წარმოვიდგინოთ:

```
class A //ინკაპსულაცია
{ private int x; }
class B : A //მემკვიდრეობა
{ ..... }
private void button1_Click(object sender, EventArgs e) {
    A a = new B(); // პოლიმორფიზმი }
```

ობიექტზე ორიენტირებული პროგრამირებას აქვს ბევრი დადებითი მხარე და სწორედ ამიტომ თანამედროვე პროგრამისტების ძალიან დიდი ნაწილი იყენებს სწორედ ამ მიდგომას კოდის აგება/დაწერის დროს.

ობიექტზე ორიენტირებული პროგრამირების (ოოპ) დადებითი მხარეები:

1. ვიზუალურად, პროგრამული კოდი წარმოჩინდება შედარებით მარტივი სახით და შესაბამისად სხვა ენებთან შედარებით ადვილად კითხვადია. ოოპ-ში ყველაფერი დაყოფილია ობიექტებად. ობიექტებს გააჩნიათ მკაფიოდ განსაზღვრული წესები. სწორედ ამ წესებზე დაყრდნობით შეგიძლიათ გავიგოთ, თუ რაზეა პასუხისმგებელი თითოეული ობიექტი და რისგან შედგება იგი.

2. ერთმანეთის მსგავსი ანუ თითქმის ერთნაირი პროგრამული კოდები გვხვდება ნაკლებად. მაგალითად, თუ ჩვეულებრივ პროგრამირებაში ერთი ფუნქცია ითვლის ერთნაირი სიმბოლოების რაოდენობას ერთგანზომილებიან მასივში, ხოლო მეორე - ორგანზომილებიანში, მაშინ მათ კოდის დიდი ნაწილი აქვთ ერთნაირი ანუ კოდი არის დუბლირებული. ოოპ-ში კი ეს პრობლემა მემკვიდრეობის პრინციპის გამოყენებით აღმოფხვრილია.

3. რთული პროგრამული კოდები იწერება შედარებით უფრო მარტივად. თითოეული დიდი პროგრამა შეიძლება დაიშალოს რამდენიმე ბლოკად, თავიდან ისინი შეგვიძლია შევავსოთ მინიმალური მონაცემებით და შემდეგ ერთმანეთის მიყოლებით თითოეულ ბლოკში დაწვრილებით დავწეროთ პროგრამული კოდები.

4. საკმაოდ გაზრდილია პროგრამული კოდის წერის სიჩქარე. დაწყებისას, შეგვიძლია სწრაფად შექმნათ საჭირო კომპონენტები, რათა მივიღოთ მინიმალურ დონეზე გამართული პროტოტიპი. ობიექტზე ორიენტირებული პროგრამირების უარყოფითი მხარეები:

1. ობიექტზე ორიენტირებული პროგრამირება შედარებით უფრო რთულია, ვიდრე ჩვეულებრივი პროცედურული პროგრამირება. საჭიროა თეორიული ნაწილის საკმაოდ კარგი ცოდნა და გააზრება, თუნდაც კოდის რაიმე მარტივი ფრაგმენტის დასაწერად.

2. მოითხოვს საკმაოდ დიდ მეხსიერებას. ობიექტზე ორიენტირებულ პროგრამირებაში ობიექტები შედგება მონაცემებისგან, ინტერფეისებისგან, მეთოდებისგან და ა.შ., რაც გაცილებით მეტ მეხსიერებას იკავებს, ვიდრე უბრალოდ ცვლადი.

3. ზოგჯერ კოდის შესრულების (მწარმოებლურობის) ეფექტურობა საკმაოდ დაბალია. მიდგომების თავისებურებებიდან გამომდინარე, ზოგიერთი რამ შეიძლება

განხორციელდეს უფრო რთულად, ვიდრე ეს შეიძლება წარმოვიდგინოთ. ამიტომ, ხდება, რომ ობიექტზე ორიენტირებული პროგრამული კოდი უფრო ნელა მუშაობს, ვიდრე პროცედურული (თუმცა თანამედროვე პროცესორების სიმძლავრე იმდენად მაღალია, რომ ეს არც იგრძნობა და პროგრამისტების დიდ ნაწილს არც აწუხებთ).

კლასი. კლასის მოკლე აღწერა. ძირითადი ცნებები. კლასი - არის მონაცემთა ძირითადი ტიპი, რომელიც განსაზღვრულია მომხმარებლის მიერ. კლასი რეალური ობიექტის ან პროცესის მოდელია ანუ წარმოადგენს ლოგიკურად დაკავშირებულ სტრუქტურას. კლასი წარმოადგენს კონსტრუქციას, რომელიც აერთიანებს ველებს, თვისებებს და მეთოდებს. კლასის წევრები არიან მონაცემები და მეთოდები (ფუნქციები), რომლებიც ამ მონაცემებს ამუშავებენ.

კლასი არის განზოგადებული ცნება, რომელიც განსაზღვრავს გარკვეული ობიექტების ნაკრების მახასიათებლებს და ქცევას. იმისათვის, რომ იმუშაოთ კლასთან, აუცილებლად უნდა განსაზღვროთ ინდივიდუალური ობიექტები (იგივე კლასის ობიექტები). თითოეული ობიექტისთვის გამოყოფილია მეხსიერების არე, რომელიც აუცილებელია ამ ობიექტის ყველა ველის (მონაცემის) შესანახად. გარდა ამისა, კლასი შეიძლება შეიცავდეს სტატიკურ წევრებს, რომლებიც განისაზღვრება ერთ ეგზემპლარად. სტატიკური მონაცემები - ეს არის უშუალოდ კლასის მონაცემები, ყველა სხვა ველი კი წარმოადგენს კლასის ეგზემპლარის მონაცემებს.

მოკლედ, რომ ვთქვათ, კლასი არის კოდის შაბლონი, ლოგიკური აბსტრაქცია, აღწერა, რომლის მიხედვითაც ხდება ობიექტების შექმნა. კლასი შეგვიძლია წარმოვიდგინოთ როგორც, რომელიმე კერძის რეცეპტი ან ავეჯის აწყობის ინსტრუქცია. თავისთავად კლასი არაფერსაც არ აკეთებს, მაგრამ მისი დახმარებით შესაძლებელია შეიქმნას ახალი ობიექტი, რომელსაც გამოვიყენებთ რეალურ ცხოვრებაში.

თავისი არსით, კლასი წარმოადგენს ახალ ტიპს, რომელიც განსაზღვრულია მომხმარებლის მიერ. კლასი აღწერა იწყება `class` საკვანძო სიტყვის გამოყენებით:

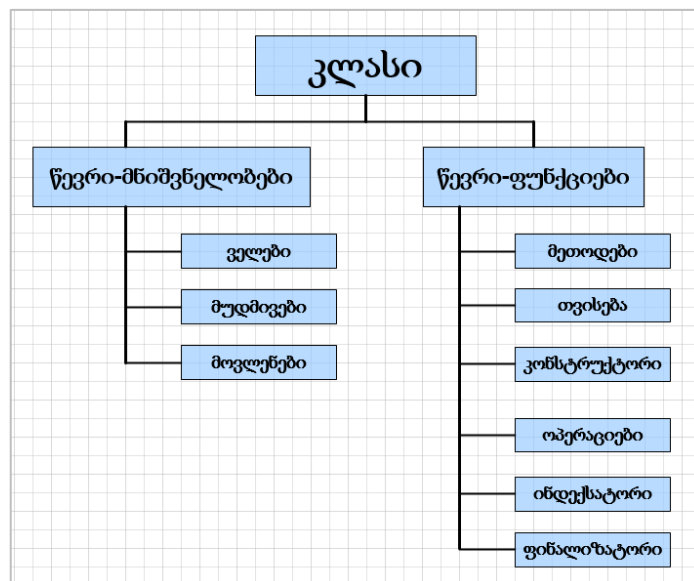
```
class კლასის_სახელი {  
    // კლასის შემცველობა }
```

კლასის ყველა ფუნქციური ელემენტი ინახება ერთ ეგზემპლარად.

კლასში მონაცემებთან სამუშაოდ უნდა განისაზღვროს კლასის მეთოდები.

კლასში შემავალ მეთოდებსა და ცვლადებს (ველებს) კლასის წევრები ეწოდებათ. C# ენაში არსებობს კლასის წევრების რამდენიმე სახესხვაობა: ობიექტის (ეგზემპლარის) ცვლადები, სტატიკური ცვლადები, მუდმივები, მეთოდები, კონსტრუქტორები, დესტრუქტორები, ინდექსატორები, მოვლენები, ოპერატორები და თვისებები.

კლასის განსაზღვრისას ხდება იმ მონაცემების გამოცხადება (აღწერა), რომლებსაც კლასი შეიცავს და ისეთი ძირითადი კოდის დაწერა, რომელიც იყენებს ამ მონაცემებს. მიუხედავად იმისა, რომ უმარტივესი კლასები შეიძლება შეიცავდეს მხოლოდ კოდს ან მხოლოდ მონაცემებს, რეალური კლასების უმეტესობა შეიცავს ორივეს ერთად. უფრო ზუსტად, წევრ-მნიშვნელობებში (მონაცემებში) შედის კლასის მიერ განსაზღვრული/გამოცხადებული მონაცემები, ხოლო წევრ-ფუნქციებში კი შედის პროგრამული კოდები. კლასის წევრების თვალსაჩინოდ წარმოსადგენად დაგვეხმარება ქვემოთ მოცემული სქემა (იხ. ნახ. 25):



ნახ. 25. კლასის წევრების სქემა. C#-ში მოცემული წევრი-მნიშვნელობები და წევრი-ფუნქციები

წევრი-მნიშვნელობები. წევრი-მნიშვნელობები არის კლასის ისეთი წევრები, რომლებიც შეიცავენ შემდეგ მონაცემებს: ველი, მუდმივა, მოვლენა. წევრი-მნიშვნელობები, იგივე მონაცემები შეიძლება იყოს სტატიკური (static).

კლასის წევრი (ობიექტი) არის ეგზემპლარის წევრი, თუ ის არ არის გამოცხადებული როგორც static წევრი. სტატიკური წევრი-მნიშვნელობები (მონაცემები) შეიძლება გამოყენებულ იქნეს როგორც ზოგადი მონაცემები მრავალი ობიექტისათვის.

განვიხილოთ წევრი-მნიშვნელობების სახესხვაობები:

1. ველი (field) - ეს არის ნებისმიერი ცვლადი, რომელიც დაკავშირებულია კლასთან.

2. მუდმივა (const) - მუდმივები შეიძლება დავაკავშიროთ კლასთან იმავე გზით, როგორც ცვლადები. მუდმივას გამოცხადება ხდება საკვანძო სიტყვით const. თუ ველი ან კონსტანტა გამოცხადებულია როგორც public წევრი, მაშინ მათზე წვდომა შესაძლებელი ხდება კლასის გარეთ.

3. მოვლენა (event) - ეს არის კლასის წევრი, რომელიც საშუალებას აძლევს ობიექტს შეატყობინოს/აცნობოს გამოსაძახებელ კოდს, რომ რაღაც მოხდა ანუ მოხდა ისეთი რამ, რაც აღნიშვნის ღირსია. მაგალითად მოვლენაა: ა) კლასის თვისების შეცვლა; ბ) გარკვეული ურთიერთქმედება მომხმარებელთან.

წევრი-ფუნქციები. არის კლასის ისეთი წევრები, რომლებიც უზრუნველყოფენ გარკვეულ ფუნქციონალურობას კლასის მონაცემთა მანიპულირების/მართვის განსახორციელებლად. წევრი-ფუნქციები შეიცავენ შემდეგ მონაცემებს: მეთოდებს, თვისებებს, კონსტრუქტორებს, ოპერაციებსა და ინდექსატორებს. ახლა კი განვიხილოთ წევრი-ფუნქციების სახესხვაობები:

1. მეთოდი (method) - არის ფუნქციები, რომლებიც ასოცირდება კონკრეტულ კლასთან. როგორც წევრი მონაცემები, ისინი ჩვეულებრივ არიან ეგზემპლარის წევრები. ისინი შეიძლება გამოცხადდეს სტატიკურად static მოდიფიკატორის საშუალებით, მაშინ მეთოდი ხელმისაწვდომი ხდება ობიექტის მითითების გარეშე.

2. თვისება (property) - არის ფუნქციების ერთობლიობა (ნაკრები), რომლებზეც კლიენტს აქვს ისეთივე წვდომა, როგორც ღია კლასის ველებთან (წევრებთან). კლასის თვისების წასაკითხად და ჩასწერად C#-ში ენაში შემოტანილია სპეციალური სინტაქსი, ამიტომ არ არის საჭირო ისეთი მეთოდების დაწერა, რომლებიც იწყება Set და Get-ით სახელებით.

3. კონსტრუქტორი (constructor) - ეს არის ისეთი სპეციალური ფუნქცია, რომელის გამოიძახებაც ხდება ავტომატურად ობიექტის ინიციალიზაციისას. კონსტრუქტორის დასახელება ყოველთვის იგივეა, რაც იმ კლასის დასახელება, რომელსაც ის მიეკუთვნება ანუ კონსტრუქტორის სახელი ემთხვევა კლასის სახელს.

კონსტრუქტორს (სპეციალურ ფუნქციას) არ გააჩნია დასაბრუნებელი მნიშვნელობის ტიპი. კონსტრუქტორი ძალიან სასრგებლო და აუცილებელი

ნაწილია პროგრამული კოდის დაწერისას, რადგან ის გამოიყენება კლასის ველების (წევრების) ინიციალიზაციისთვის.

4. ოპერაცია (operator) - ეს არის მარტივი მოქმედებები: + ან -. თუ ორ მთელ რიცხვს ვკრიბავთ, მაშინ ვიყენებთ ოპერაციას +. თუმცა, C# გვადლევს საშუალებას არსებულმა ოპერაციებმა იმუშაონ სამომხმარებლო კლასებთან (ე. წ. ოპერაციის გადატვირთვა).

5. ინდექსატორი (indexer) - საშუალებას იძლევა ობიექტების ინდექსირება განვახორციელოთ ზუსტად ისე, როგორც მასივებთან ან კოლექციებთან მიმართებით. წვდომის ხილვადობის მოდიფიკატორის დახმარებით შეგვიძლია განვსაზღვროთ, თუ კოდის რომელი ნაწილიდან შეგვიძლია მივმართოთ/მივწვდეთ კლასს ან კლასის წევრებს.

C# ენაში ინკაფსულაცია წარმოდგენილია ხილვადობის შემდეგი მოდიფიკატორებით: public, private, protected, internal, protected internal, abstract, sealed, static.

ხილვადობის ღია მოდიფიკატორი public - ძლიერ გახსნილი წვდომა, წვდომა არ არის შეზღუდული. ხილვადობის მოდიფიკატორი public ნებას რთავს კლასს, რომ მისი წევრი ველები და მეთოდები გამოყენებულ იქნეს სხვა მეთოდებსა და ობიექტებში. ნებისმიერი public წევრი შეიძლება გამოვიყენოთ კლასის გარეთ. წვდომა დასაშვებია აპლიკაციაში (დანართში) არსებული კოდის ნებისმიერი ფრაგმენტიდან. ელემენტები რომლებიც განსაზღვრულია როგორც public, წვდომა-ღია/ხილვადია ნებისმიერი კლასის ნებისმიერი მეთოდისათვის.

```
class MyClass {  
    public int mnishvneloba; //public მოდიფიკატორის გამოყენება }  
    private void button1_Click(object sender, EventArgs e) {  
        MyClass obieqti = new MyClass();  
        obieqti.mnishvneloba = 99;  
        label1.Text = "რიცხვი = " + obieqti.mnishvneloba; }  
}
```

ხილვადობის დახურული მოდიფიკატორი private - ძლიერ დახურული, წვდომა აკრძალულია. ხილვადობის მოდიფიკატორი private საშუალებას აძლევს კლასს, რომ დამალოს მისი - მეთოდები, წევრები (ველები), თვისებები - სხვა ობიექტებისაგან.

private ელემენტი წვდომაღია კლასის ხილვადობის არეში, მხოლოდ იმავე კლასის მეთოდებს შეუძლიათ გამოიყენონ ამ კლასის private წევრები, ხოლო კლასის ობიექტზე მიმთითებელსაც არ შეუძლია private წევრებზე წვდომა. ვთქვათ მოცემული

გვაქვს კლასი სახელით AAA. კლასი AAA-ს ის წევრები, რომლებიც აღწერილია როგორც private წევრი, წვდომადია მხოლოდ კლასი AAA-ს მეთოდებისათვის.

ხილვადობის დაცული მოდიფიკატორი protected - ღიაა მხოლოდ მემკვიდრე კლასებისათვის. მხოლოდ მემკვიდრე (შვილობილ) კლასს შეუძლია, რომ მიწვდეს საბაზო კლასის protected წევრებს. ელემენტები წვდომადია თავად კლასისთვის და ამ კლასის მემკვიდრე კლასებისთვის. მემკვიდრეობითობის დროს მეთოდები და ველები ძირითადად არის protected წვდომისა. ვთქვათ, მოცემული გვაქვს საბაზო/მშობელი/წინაპარი კლასი სახელით AAA. კლასი AAA-ს ის წევრები, რომლებიც აღწერილია როგორც **protected** წევრი, წვდომადია საბაზო კლასი AAA-ს მეთოდებისათვის და საბაზო AAA კლასის მემკვიდრე კლასების მეთოდებისათვის.

ხილვადობის შუალედური მოდიფიკატორი internal - ხილვადობის მოდიფიკატორი internal წარმოადგენს შუალედური წვდომის მოდიფიკატორს. ელემენტები ხელმისაწვდომია ერთი Microsoft.NET პროექტის ფარგლებში (ერთი ფაილისათვის ან ერთი .dll ფაილისათვის), სხვა პროექტიდან წვდომა აკრძალულია. ნებისმიერ კლასს და მეთოდს აქვს უფლება გამოიყენოს (ჰქონდეს წვდომა) სხვა კლასის internal ხილვადობის წევრებთან, ოღონდ იმ პირობით, რომ ყოველი ეს მეთოდი და კლასი უნდა იყოს ერთ აპლიკაციაში (დანართში).

ხილვადობის მოდიფიკატორი protected internal - ხილვადობის მოდიფიკატორის protected internal ქვეშ იგულისხმება, რომ ამ სახის ხილვადობის წევრებს შეგვიძლია მივწვდეთ მხოლოდ მემკვიდრე კლასიდან და იმ პირობით, რომ ეს კლასები უნდა იყოს ერთსა და იმავე აპლიკაციაში (პროექტში).

ხილვადობის მოდიფიკატორ internal-ით წარმოდგენილ კლასზე წვდომა აქვს პროექტში არსებულ ყველა სხვა კლასს. კლასები და ელემენტები, რომლებიც წარმოდგენილია ხილვადობის მოდიფიკატორით protected internal კი ხელმისაწვდომია მთელი პროექტისათვის და ასევე მიმდინარე კლასის ყველა მემკვიდრე კლასისათვის.

ხილვადობის მოდიფიკატორი abstract - მიუთითებს იმაზე, რომ კლასი მხოლოდ სხვა კლასების საბაზო კლასად არის გამოყენებული.

ხილვადობის მოდიფიკატორი sealed - მიუთითებს იმაზე, რომ კლასს არ შეიძლება ჰქონდეს მემკვიდრე კლასი. ასეთ კლასებს „გაუხსნელ/დაგმანულ“ კლასებს უწოდებენ.

ხილვადობის მოდიფიკატორი static - აცხადებს წევრს, რომელიც განეკუთვნება ტიპს და არა კონკრეტულ ობიექტს. static ხილვადობის მოდიფიკატორი მიუთითებს, რომ მეთოდი არის კლასის სტატიკური წევრი და არა კონკრეტული ობიექტის (კლასის ეგზემპლარის) წევრი.

თუ კლასს ან კლასის წევრს არ აქვს მითითებული ხილვადობის მოდიფიკატორი, მაშინ ჩუმადობის პრინციპის მიხედვით კლასისთვის იგულისხმება ხილვადობის მოდიფიკატორი internal, ხოლო კლასის წევრებისთვის ხილვადობის მოდიფიკატორი private.

ობიექტი. ობიექტი არის კლასის ეგზემპლარი. სხვანაირად რომ ვთქვათ ობიექტი კლასის განხორციელების რეალური სტრუქტურაა.

ობიექტი, შექმნის შემდეგ, იკავებს ოპერატიულ მეხსიერებას. ობიექტს გააჩნია მდგომარეობა და ქმედება/ქცევა.

მდგომარეობა ეს არის ობიექტის მახასიათებლები, პარამეტრები, ხოლო ქმედება კი არის ობიექტის მიერ შესრულებული რაიმე ქცევა.

მაგალითად: **ობიექტი** - სახლი; **მდგომარეობა** - მისამართი, ფერი, ფართობი, ოთახების რაოდენობა და სხვა; **ქმედება/ქცევა** - ფანჯრის გაღება, კარების დაკეტვა, ფანჯრის გაწმენდა და სხვა.

ობიექტს შეუძლია მოქმედებების შესრულება. მაგალითად: ფეხბურთელს გააქვს გოლი, მეკარე ბურთს იგერიებს.

განვიხილოთ მაგალითი რეალური ცხოვრებიდან (იხ. ცხრილი 8):

ცხრილი 8. კლასის აღწერა

კლასი	ობიექტი	თვისება	მეთოდები	მოვლენა
თევზი	კალმახი ლოქო	დასახელება, ზომა, ასაკი, გარემო პირობები, მტაცებელი	ცურვა, კვება, მიყუჩება/მიმალვა	საშიშროება, ქანგადადის კლება, წყლის დაბინძურება

კიდევ ერთი მაგალითი კლასებთან მიმართებით: ვიცით რომ საქართველოს თითქმის ყველა მთავარ ქალაქში და დედაქალაქში არსებობს ქართველი გენიალური პოეტისა და მოაზროვნის შოთა რუსთაველის სახელობის ქუჩა. თუ ჩვენ ვაგზავნით წერილს ნათესავთან, რომელიც ცხოვრობს შოთა რუსთაველის

სახელობის ქუჩაზე და მივუთითებთ მხოლოდ „შოთა რუსთაველის ქუჩა, N7“, მაშინ წერილი ადრესატამდე ვერ მივა, რადგან მითითებული არ არის ქალაქის დასახელება. სწორედ ასე ხდება დაპროგრამებაშიც.

კიდევ ერთი მაგალითი: ვთქვათ Microsoft-მა შექმნა კლასი სახელით Draw, რომელიც ხატავს სახლს, მაგრამ რომელიმე XXX პროგრამისტიმაც შექმნა ზუსტად იგივე დასახელების კლასი, მაგრამ XXX პროგრამისტის მიერ შექმნილი კლასი კი ხატავს ყვავილებს, კერძოდ, გვირილებს. მაგრამ რა ქნას პროგრამამ? როგორ მიმართოს ერთსა და იმავე დასახელების კლასს ანუ როგორ გაარჩიოს ისინი ერთმანეთისაგან? ასეთი შემთხვევების დროს .NET გარემო გვაძლევს საშუალებას გამოვიყენოთ სხვადასხვა დასახელების სახელების სივრცე: **Microsoft.Draw**, **My.Draw**. კლასების სახელები შენარჩუნებულია, მაგრამ სახელების სივრცეები კლასის სახელების წინ ნათლად მიუთითებს, თუ რომელ კლასთან გვაქვს საქმე.

ობიექტის შესაქმნელად გამოიყენება new ოპერატორი. ოპერატორი new დინამიკურად (პროგრამის შესრულების დროს) გამოყოფს მეხსიერებას ობიექტისათვის და გასცემს მიმართვას მეხსიერების ამ უბანზე. მიმართვა არის მეხსიერების იმ უბნის მისამართი, რომელიც ობიექტს გამოეყო new ოპერატორის მიერ, შემდგომში ეს მიმართვა ინახება ცვლადში. კლასის ობიექტებთან მუშაობა ხორციელდება მიმართვის საშუალებით, ამიტომ კლასებს სხვანაირად მიმართვით ტიპებსაც უწოდებენ. როგორც ვიცით, მთავარი განსხვავება ჩვეულებრივ და მიმართვით ტიპებს შორის მდგომარეობს იმ მონაცემებში, რომლებსაც ისინი ინახავენ. ჩვეულებრივი ტიპის ცვლადი ინახავს მნიშვნელობას, მიმართვითი ტიპის ცვლადი კი - ობიექტის მისამართს. კლასები, ისევე როგორც მათი (აღწერიდან მიღებული) ობიექტები, შედგება კომბინირებული წევრებისაგან (ინკაფსულაცია) - მონაცემთა ველებისა და მეთოდების კოდებისაგან.

მსგავსი სპეციფიკაციის ობიექტების მეთოდებს შეიძლება ჰქონდეთ ერთი და იგივე სახელი და დანიშნულება, მაგრამ შეასრულონ კონკრეტული ობიექტისთვის დამახასიათებელი მოქმედება (პოლიმორფიზმი).

ობიექტთან მიმართვამდე, ობიექტი უნდა შეიქმნას, რადგან კლასი არის მხოლოდ აბსტრაქცია, აღწერა. როგორც აღვნიშნეთ ობიექტის ეგზემპლარი იქმნება new ოპერატორით.

მაგალითად, აღვწეროთ კლასი დასახელებით Student:

```
public class Student { .... }
```

იმისათვის რომ შევქმნათ კლასის ეგზემპლარი ანუ ობიექტი, უნდა გამოვიყენოთ new ოპერატორი:

```
Student gtu_student = new Student(); // gtu_student არის კლასის ეგზემპლარი, ობიექტი
```

ობიექტის ეგზემპლარის ცვლადი არის მიმართვითი ტიპის, ე. ი. ინახავს მიმართვას მეხსიერების იმ უბნის შესახებ, რომელშიც ობიექტი შეიქმნა და არა თვით ობიექტს.

მეთოდი, როგორც კლასის წევრი. მეთოდი - ესაა კლასის ფუნქციონალური ელემენტი, რომელიც უზრუნველყოფს გამოთვლებს ან რაიმე სხვა ქმედებებს. მეთოდები მუშაობენ კლასში განსაზღვრულ მონაცემებთან და უზრუნველყოფენ ამ მონაცემებთან მიმართვას.

მეთოდი (შეგვიძლია ვუწოდოთ ფუნქცია, ქვეპროგრამა) არის კოდის დასრულებული ფრაგმენტი, რომელსაც აქვს სახელი. მეთოდს მიმართავენ სწორედ ამ სახელით. მეთოდი შეგვიძლია გამოვიძახოთ ერთხელ, ორჯერ და იმდენჯერ რამდენჯერაც დაგვჭირდება მისი გამოძახება. მეთოდი გვებმარება კლასის წევრებთან (ველებთან, მონაცემებთან) წვდომაში. მეთოდების საშუალებით ხდება კლასის ურთიერთქმედება პროგრამის სხვა ნაწილებთან ან სხვა კლასებთან. როგორც წესი, თითოეული მეთოდი ასრულებს ერთ ფუნქციას, ერთ კონკრეტულ დავალებას.

დავიმახსოვროთ: მეთოდის გამოძახება ბევრჯერ შეიძლება. მეთოდი შეიძლება შეიცავდეს ერთ ან მეტ ოპერატორს და უნდა წყვეტდეს მხოლოდ ერთ კონკრეტულ ამოცანას. მეთოდის სახელად შეგვიძლია გამოვიყენოთ ნებისმიერი სიტყვა, მაგრამ საკვანძო სიტყვების გამოყენება მეთოდების სახელებად არ შეიძლება. მაგალითად, სახელი Main() უკვე დარეზერვებულია იმ მეთოდისათვის, საიდანაც იწყება პროგრამის შესრულება. მეთოდის ხილვადობის_მოდიფიკატორი იღებს private ან public მნიშვნელობებს. თუ ის არ არის მითითებული, მაშინ იგულისხმება private, ე. ი. მეთოდი მისაწვდომი იქნება მხოლოდ იმ კლასის შიგნით, რომელშიც ის არის აღწერილი. მეთოდის სინტაქსის ზოგადი სახე ასეთია:

ხილვადობის_მოდიფიკატორი ტიპი

მეთოდის_სახელი(მეთოდის_პარამეტრი) {

მეთოდის ტანი }

პარამეტრების გამოყენება. გამოძახებისას მეთოდს შეგვიძლია გადავცეთ ერთი ან მეტი არგუმენტი. მეთოდისთვის გადასაცემ მნიშვნელობას არგუმენტი ეწოდება. ცვლადს მეთოდის შიგნით, რომელიც იღებს არგუმენტის მნიშვნელობას, პარამეტრი ეწოდება. პარამეტრების გამოცხადება ხდება მრგვალი ფრჩხილების შიგნით, რომლებიც მეთოდის სახელს მოსდევს. პარამეტრის გამოცხადების სინტაქსი ცვლადების გამოცხადების სინტაქსის ანალოგიურია. პარამეტრი იმყოფება თავისი მეთოდის ხილვადობის უბანში ანუ ხილულია მხოლოდ ამ მეთოდის შიგნით და მოქმედებს როგორც ლოკალური ცვლადი.

არგუმენტის გადაცემის ხერხები. არსებობს არგუმენტის გადაცემის ორი ხერხი: მნიშვნელობის მიხედვით (Pass-by-value) და მიმართვის მიხედვით (Pass-by-reference, Pass-by-const reference).

პირველი ხერხი: არგუმენტის გადაცემის უმარტივესი ხერხია გადაცემა მნიშვნელობის მიხედვით (Pass-by-value). ამ შემთხვევაში ფუნქციის გამოძახების დროს მის პარამეტრს ენიჭება არგუმენტის მნიშვნელობის ასლი. მეთოდის ოპერატორები მუშაობენ არგუმენტის მნიშვნელობების ასლებთან. მეთოდს არგუმენტების საწყის მნიშვნელობებთან წვდომა არ აქვს და შესაბამისად შეუძლებელია მათი ცვლილებაც.

მეთოდის გამოძახების დროს მის პარამეტრებს და ლოკალურ ცვლადებს გამოეყოფათ ადგილი ოპერატიული მეხსიერების სპეციალურ არეში - სტეკში. როდესაც არგუმენტი გადაეცემა მეთოდს მნიშვნელობით, მეთოდის პარამეტრს ენიჭება ამ არგუმენტის ასლი. ამიტომ, თუ მეთოდის ტანში პარამეტრი იცვლის მნიშვნელობას, შესაბამისი არგუმენტი არ იცვლება.

ხშირად საჭიროა მეთოდში გადაცემული არგუმენტის შეცვლა. მაგალითად, როდესაც:

1. მეთოდიდან ერთზე მეტი მნიშვნელობის დაბრუნება გვჭირდება. მაშინ ერთ მნიშვნელობას დააბრუნებს return შეტყობინება, დანარჩენი მნიშვნელობების დაბრუნება კი უნდა მოხერხდეს პარამეტრების მეშვეობით;
2. პროგრამაში გვჭირდება მეთოდის მიერ არგუმენტის შეცვლილი მნიშვნელობა;
3. მეთოდს დასამუშავებლად გადაეცემა მონაცემთა დიდი ობიექტი.

ასეთი არგუმენტის გადაცემა მნიშვნელობით (მისი კოპირება პარამეტრში) მოითხოვს გარკვეულ დროს, რაც ანელებს პროგრამის შესრულებას. მსგავს შემთხვევებში მეთოდს უნდა გადაეცეს არა არგუმენტის ასლი, არამედ მისი

მისამართი. მაშინ ყველა ცვლილება, რომელსაც მეთოდში განიცდის პარამეტრი, სინამდვილეში ხორციელდება არგუმენტზე. ასეთ პარამეტრებს ეწოდებათ ცვლადი პარამეტრები.

მეორე ხერხი: არგუმენტის გადაცემის მეორე ხერხია გადაცემა მიმართვის (მისამართის) მიხედვით (Pass-by-reference, Pass-by-const reference). მიმართვით გადაცემის დროს მეთოდს გადაეცემა არა არგუმენტის მნიშვნელობა, არამედ არგუმენტის მისამართის ასლი. ფუნქციის გამოძახების დროს პარამეტრს ენიჭება არგუმენტის მისამართის ასლი, შესაბამისად მეთოდს აქვს წვდომა მეხსიერების უბნის ამ მისამართზე და შეუძლია შეცვალოს არგუმენტის საწყისი მნიშვნელობა პარამეტრის შეცვლით. C# ენაში გამოძახებულ და გამოსაძახებელ მეთოდებს შორის მონაცემთა გაცვლის ოთხი სახის პარამეტრი არსებობს:

1. მნიშვნელობა_პარამეტრი - მეთოდის საწყისი მონაცემებისათვის;
2. მიმართვა_პარამეტრი (ref) - არგუმენტების შეცვლისათვის;
3. გამომავალი_პარამეტრი (out) - არგუმენტების ფორმირებისათვის;
4. მასივი_პარამეტრი (params) - ცვალეზადი რაოდენობის არგუმენტებისათვის.

საკვანძო სიტყვა წინ უსწრებს პარამეტრის ტიპის აღწერას. თუ ის გამოტოვებულია, პარამეტრი ითვლება მნიშვნელობა-პარამეტრად. თუ პარამეტრების ჩამონათვალში გვჭირდება მასივი-პარამეტრის შემოტანა, მაშინ ის უნდა განთავსდეს პარამეტრების სიის ბოლოში. ასე მაგალითად:

```
public int Calculate( int a, ref int b, out int c, params int[] d)
```

გადავიდეთ ამოცანების განხილვაზე.

ამოცანა 6.1. შეექმნათ კლასი სახელით სტუდენტი, რომელიც შეიცავს პრივატულ ცვლადებს: გვარი, სახელი და ასაკი; ღია ცვლადებს: უნივერსიტეტის დასახელება და კურსი. კლასის მუშაობის შედეგი ავსახოთ შესაბამის კომპონენტებში:

```
class სტუდენტი {  
    private string gvari;  
    private string saxeli;  
    private int asaki;  
    public string universitetis_dasaxeleba;  
    public int kursi; }
```

```
private void button1_Click(object sender, EventArgs e) {
    სტუდენტი obieqti_studenti = new სტუდენტი();
    obieqti_studenti.universitetis_dasaxeleba = textBox1.Text;
    obieqti_studenti.kursi = int.Parse(textBox2.Text);
    label1.Text = obieqti_studenti.universitetis_dasaxeleba;
    label2.Text = obieqti_studenti.kursi.ToString(); }

```

ამოცანა 6.2. სამკუთხედის პერიმეტრის გამოთვლა კლასის შიგნით გამოცხადებულ მეთოდში. მეთოდის სახელია Perimetri(). ამ ამოცანის გადაწყვეტა შესაძლებელია ორი გზით: პირველი გზა - როდესაც მეთოდი არ აბრუნებს მნიშვნელობას; მეორე გზა - როდესაც მეთოდი აბრუნებს მნიშვნელობას return ოპერატორის გამოყენებით.

ამოცანა 6.2.-ის პირველი ვარიანტი - მეთოდი არ აბრუნებს მნიშვნელობას.

```
class SamkuTxedi {
    public int pirveli_gverdi, meore_gverdi, mesame_gverdi;
    private int perimetri; //ლოკალური ცვლადის გამოცხადება
    //Perimetri() მეთოდის განსაზღვრა, რომელიც არ აბრუნებს მნიშვნელობას
    public void Perimetri() {
        perimetri = pirveli_gverdi + meore_gverdi + mesame_gverdi;
        MessageBox.Show("სამკუთხედის პერიმეტრი = " +
            perimetri.ToString());
    }
}

```

```
private void button1_Click(object sender, EventArgs e) {
    SamkuTxedi obieqti_pirveli_samkuTxedi = new SamkuTxedi();
    SamkuTxedi obieqti_meore_samkuTxedi = new SamkuTxedi();
    obieqti_pirveli_samkuTxedi.pirveli_gverdi =
        int.Parse(textBox1.Text);
    obieqti_pirveli_samkuTxedi.meore_gverdi =
        int.Parse(textBox2.Text);
    obieqti_pirveli_samkuTxedi.mesame_gverdi =
        int.Parse(textBox3.Text);
    //პირველი სამკუთხედის პერიმეტრის გამოთვლა
    obieqti_pirveli_samkuTxedi.Perimetri();
}

```



```

        obieqti_meore_samkuTxedi.pirveli_gverdi =
        int.Parse(textBox4.Text);
        obieqti_meore_samkuTxedi.meore_gverdi =
        int.Parse(textBox5.Text);
        obieqti_meore_samkuTxedi.mesame_gverdi =
        int.Parse(textBox6.Text);
        //მეორე სამკუთხედის პერიმეტრის გამოთვლა
        obieqti_meore_samkuTxedi.Perimetri();
    }

```

ამოცანა 6.2.-ის მეორე ვარიანტი - მეთოდი აბრუნებს მნიშვნელობას **return** ოპერატორით.

```

class SamkuTxedi {
    public int pirveli_gverdi, meore_gverdi, mesame_gverdi;
    /*Perimetri() მეთოდის განსაზღვრა, რომელიც აბრუნებს მნიშვნელობას return
    ოპერატორით*/
    public int Perimetri() {
        return pirveli_gverdi + meore_gverdi + mesame_gverdi; }
}

private void button1_Click(object sender, EventArgs e) {
    SamkuTxedi obieqti_pirveli_samkuTxedi = new SamkuTxedi();
    SamkuTxedi obieqti_meore_samkuTxedi = new SamkuTxedi();
    obieqti_pirveli_samkuTxedi.pirveli_gverdi =
    int.Parse(textBox1.Text);
    obieqti_pirveli_samkuTxedi.meore_gverdi = int.Parse(textBox2.Text);
    obieqti_pirveli_samkuTxedi.mesame_gverdi =
    int.Parse(textBox3.Text);
    //პირველი სამკუთხედის პერიმეტრის გამოთვლა
    int pirvelis_perimetri = obieqti_pirveli_samkuTxedi.Perimetri();
    label1.Text = "პირველი სამკუთხედის პერიმეტრი = " +
                    pirvelis_perimetri.ToString();
    obieqti_meore_samkuTxedi.pirveli_gverdi = int.Parse(textBox4.Text);
    obieqti_meore_samkuTxedi.meore_gverdi = int.Parse(textBox5.Text);
    obieqti_meore_samkuTxedi.mesame_gverdi = int.Parse(textBox6.Text);
    //მეორე სამკუთხედის პერიმეტრის გამოთვლა

```

```
int meores_perimetri = obieqti_meore_samkuTxedi.Perimetri();
label2.Text = "მეორე სამკუთხედის პერიმეტრი = "+
                meores_perimetri.ToString(); }
```

ამოცანა 6.3. დავწეროთ პროგრამული კოდი რომელიც გამოითვლის ოთახის ფართობს და მოცულობას კლასში განსაზღვრული მეთოდების დახმარებით. მეთოდს, რომელითაც განვსაზღვრავთ ფართობს გადავცეთ ორი არგუმენტი (double Otaxis_farTobi(double x, double y)), ხოლო მეთოდს, რომელიც განსაზღვრავს მოცულობას გადავცემა სამი არგუმენტი (double Otaxis_moculoba(double x, double y, double z)), ასევე დავადგინოთ ოთახს გააჩნია თუ არა ფანჯარა. ამის განხორციელებაში დაგვეხმარება checkBox კომპონენტი.

```
class ოთახი {
    public double sigrdze, simagle, sigane;
    // მეთოდის განსაზღვრა, რომელიც აბრუნებს ფართობის მნიშვნელობას;
    public double Otaxis_farTobi(double x, double y)
    {
        return x * y;
    }
    // მეთოდის განსაზღვრა, რომელიც აბრუნებს მოცულობის მნიშვნელობას;
    public double Otaxis_moculoba(double x, double y, double z)
    {
        return x * y * z;
    }
}

private void button1_Click(object sender, EventArgs e) {
    ოთახი chemi_otaxi = new ოთახი();
    chemi_otaxi.sigrdze = double.Parse(textBox1.Text);
    chemi_otaxi.simagle = double.Parse(textBox2.Text);
    chemi_otaxi.sigane = double.Parse(textBox3.Text);
    label1.Text = "ოთახის ფართობი = " +
chemi_otaxi.Otaxis_farTobi(chemi_otaxi.sigrdze, chemi_otaxi.sigane);
    label2.Text = "ოთახის მოცულობა = " +
chemi_otaxi.Otaxis_moculoba(chemi_otaxi.sigrdze, chemi_otaxi.sigane,
                            chemi_otaxi.simagle);
}
```

```

if (checkBox1.Checked)
    label3.Text = "ოთახს გააჩნია ფანჯარა";
else label3.Text = "ოთახს არ გააჩნია ფანჯარა"; }

```

ამოცანა 6.3-ის შესრულების შედეგი იხ. 26-ე ნახაზზე:

ნახ. 26. ამოცანა 6.3-ის შესრულების შედეგი

ამოცანა 6.4. დავეწეროთ პროგრამული კოდი, რომლის დახმარებითაც განვსაზღვრავთ რიცხვი კენტია თუ ლუწი. მეთოდისათვის მნიშვნელობის გადაცემა ხდება პარამეტრით.

```

class ArisLuwi {
// მეთოდის განსაზღვრა, რომელიც ამოწმებს რიცხვი კენტია თუ ლუწი;
public bool luwi_kenti(int cvladi)
{
    if ((cvladi % 2) == 0) return true;
    else return false;
}
}

```

```

private void button1_Click(object sender, EventArgs e) {
    int ricxvi;
    ArisLuwi obieqti = new ArisLuwi();
    ricxvi = int.Parse(textBox1.Text);
    if (obieqti.luwi_kenti(ricxvi))
        label1.Text = " რიცხვი " + ricxvi.ToString() + " ლუწია";
    else label1.Text = " რიცხვი " + ricxvi.ToString() + " კენტია"; }

```

ამოცანა 6.5. დავეწეროთ პროგრამული კოდი, რომლის დახმარებითაც განვსაზღვრავთ ერთი რიცხვი არის თუ არა მეორის გამყოფი.

```

class Gamyofi

```

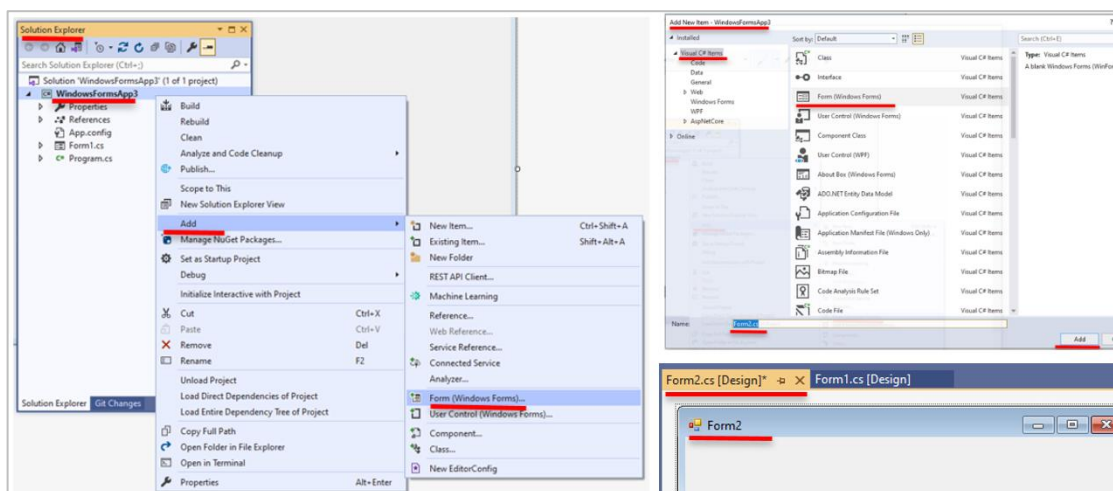
```

/* მეთოდი ამოწმებს პირველი პარამეტრი უნაშთოდ იყოფა თუ არა მეორე პარამეტრზე */;
public bool ArisGamyofi(int par1, int par2) {
    if ((par1 % par2) == 0) return true;
    else return false; }
}

private void button1_Click(object sender, EventArgs e) {
    int ricxvi1, ricxvi2;
    Gamyofi obieqti = new Gamyofi();
    ricxvi1 = int.Parse(textBox1.Text);
    ricxvi2 = int.Parse(textBox2.Text);
    if (obieqti.ArisGamyofi(ricxvi1, ricxvi2))
        label2.Text=ricxvi2.ToString()+" არის "+ricxvi1.ToString()+"-ის გამყოფი";
    else
        label2.Text=ricxvi2.ToString()+"არ არის"+ ricxvi1.ToString()+"-ის გამყოფი";}

```

ამოცანა 6.6. პროექტში ახალი ფორმის დამატება. იმისათვის, რომ პროექტს დავამატოთ ახალი ფორმა, Solution Explorer-ის ფანჯარაში მოცემულ პროექტის სახელს დავაჭიროთ მაუსის მარჯვენა ღილაკი და ავირჩიოთ Add ბრძანება, შემდეგ Windows Form და ახალ ფორმას დავარქვათ ჩვენთვის სასურველი სახელი (იხ. ნახ. 27.). ამდენად პროექტს დავმატა მეორე ფორმა სახელით Form2.cs[Design].

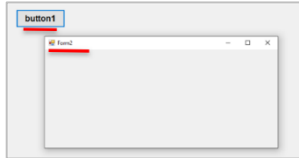


ნახ. 27. პროექტში ახალი ფორმის დამატება

ახლა კი განვიხილოთ თუ როგორ განვახორციელოთ ერთი ფორმიდან მეორე ფორმაზე გადასვლა: კერძოდ პირველი ფორმიდან მეორე ფორმის გამოძახება. პირველ ფორმაზე გადმოვიტანოთ button1 ბრძანებითი ღილაკი და მასზე დავაჭიროთ ორჯერ. შესაბამისად ჩვენ გავგეხსნება კოდის ჩაწერის ფანჯარა

Form1.cs და შესაბამისად ბრძანებით button1 ღილაკზე დაჭერის მოვლენა, რომელიც იქმნება ჩუმათობის პრინციპის მიხედვით როგორც კი ბრძანებით ღილაკს დავაჭერთ ორჯერ (იხ. ნახ. 28.).

```
private void button1_Click_2(object sender, EventArgs e) {  
    Form2 newForm = new Form2();  
    newForm.Show(); }
```



ნახ. 28. ერთი ფორმიდან მეორე ფორმის გამოძახება

ახლა კი დავწეროთ ისეთი კოდი, რომლის მიხედვითაც მეორე ფორმა იმოქმედებს პირველ ფორმაზე შემდეგი სახით: პირველ ფორმას შეუცვლის ფერს. ამისათვის მეორე ფორმის ცარიელ სივრცეში დავაჭიროთ „მაუსის“ მარჯვენა ღილაკს და ავირჩიოთ View Code. დავწეროთ ქვემოთ მოცემული კოდი:

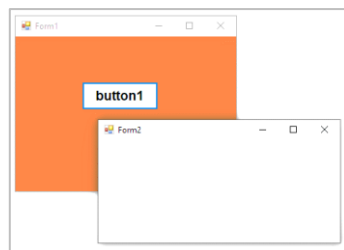
```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
namespace WindowsFormsApp1  
{  
    public partial class Form2 : Form  
    {  
        public Form2()  
        {  
            InitializeComponent();  
        }  
        public Form2(Form1 f) {  
            InitializeComponent();  
            f.BackColor = Color.Chocolate; }  
    }  
}
```

ჩვენ მეორე ფორმას დავამატეთ კონსტრუქტორი `public Form2(Form1 f)`, რომლის დახმარებითაც წვდომა გვაქვს პირველ ფორმასთან და ამავედროულად პირველივე ფორმას შევუცვალეთ ფერი, კოდის მიხედვით პირველი ფორმის ფერი გახდება ყავისფერი. ახლა კი გადავიდეთ პირველი ფორმის კოდზე და ძველი კოდი შევცვალოთ ქვემოთ მოცემული სახით:

```
private void button1_Click_2(object sender, EventArgs e) {
    Form2 newForm = new Form2(this);
    newForm.Show(); }
```

ამ შემთხვევაში `this` საკვანძო სიტყვა წარმოადგენს მიმართვას `Form1` ობიექტთან, ამდენად მეორე ფორმის შექმნისას მას გადაეცემა სწორედ ეს მიმართვა და მისი მეშვეობით განახორციელებს პირველი ფორმის მართავს (გამოძახებას).

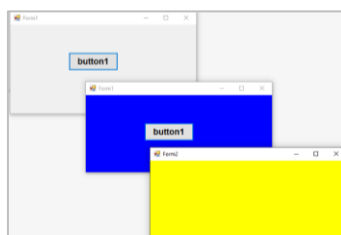
როგორც კი დავაჭერთ ბრძანებით `button1` ღილაკს, შეიქმნება მეორე ფორმა. სწორედ ეს მეორე ფორმა შეუცვლის პირველ ფორმას ფერს (ნახ. 29.):



ნახ. 29. ერთი ფორმის მეორე ფორმასთან ურთიერთქმედება

შეიძლება კოდი გადავაკეთოთ შემდეგი სახით და გავხადოთ კიდევ უფრო ფუნქციონალური (იხ. ნახ. 30.):

```
private void button1_Click_2(object sender, EventArgs e) {
    Form1 newForm1 = new Form1();
    newForm1.BackColor = Color.Blue;
    newForm1.Show();
    Form2 newForm2 = new Form2(newForm1);
    newForm2.BackColor = Color.Yellow;
    newForm2.Show(); }
```



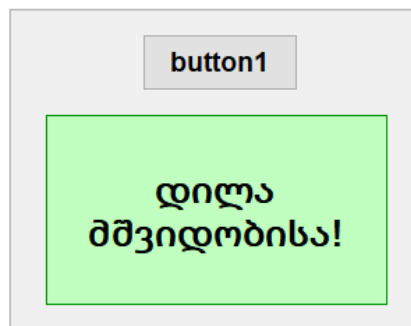
ნახ. 30. ერთი ფორმიდან მეორე ფორმის გამოძახება

რამოდენიმე ფორმასთან მუშაობისას უნდა გავითვალისწინოთ ის, რომ ერთ-ერთი ფორმა აუცილებლად უნდა იყოს მთავარი (ანუ ძირითადი), რომელიც გაიშვება Program.cs პირველ ფაილში. თუ ჩვენ ერთდროულად გახსნილი გვაქვს რამდენიმე ფორმა, ძირითადი ფორმის დახურვისას დაიხურება ყველა დანართი და შესაბამისად სხვა დანარჩენი გახსნილი ფორმებიც.

ამოცანა 6.7. დავწეროთ პროგრამული კოდი რომელიც მოგვცემს საშუალებას, რომ პროგრამულად აღვწეროთ ბრძანებითი ლილაკი, ამასთანავე მისი დაჭერის შემდეგ მივიღოთ ახალი ლილაკი ან ლილაკები. მოცემული ამოცანის გადაჭრა წარმოდგენილია ოთხი სხვადასხვა ვარიანტით. მოდით განვიხილოთ ეს ვარიანტები:

ვარიანტი 6.7.1. კოდი მიბმულია button1 ლილაკის Click მოვლენაზე ანუ button1 ლილაკზე ვაჭერთ ორჯერ და შემდეგ ვწერთ ქვემოთ მოცემულ კოდს (იხ. ნახ. 31.):

```
private void button1_Click(object sender, EventArgs e) {  
    Button testbutton = new Button();  
    testbutton.Text = "დილა მშვიდობისა!";  
    testbutton.Location = new Point(70, 70);  
    testbutton.Size = new Size(200, 200);  
    testbutton.Visible = true;  
    testbutton.BackColor = Color.LightPink;  
    testbutton.Padding = new Padding(6);  
    testbutton.Font = new Font("Sylfaen", 14,  
        FontStyle.Bold, GraphicsUnit.Point);  
    testbutton.BringToFront();  
    this.Controls.Add(testbutton); }  
}
```

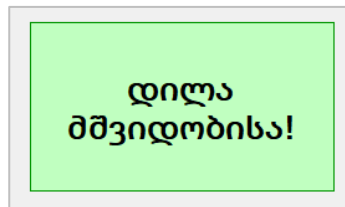


ნახ. 31. ერთი ბრძანებითი ლილაკიდან მეორე ბრძანებითი ლილაკის მიღება

ვარიანტი 6.7.2. კოდი მიბმულია ფორმის Load მოვლენაზე ანუ ფორმაზე ვაჭერთ ორჯერ და შემდეგ ვწერთ ქვემოთ მოცემულ კოდს (იხ. ნახ. 32.):

```
private void Form1_Load(object sender, EventArgs e) {
    Button testbutton = new Button();
    testbutton.Text = "დილა მშვიდობისა!";
    testbutton.Location = new Point(70, 70);
    testbutton.Size = new Size(200, 200);
    testbutton.Visible = true;
    testbutton.BackColor = Color.LightPink;
    testbutton.Padding = new Padding(6);
    testbutton.Font = new Font("Sylfaen", 14,
    FontStyle.Bold, GraphicsUnit.Point);
    testbutton.BringToFront();
    this.Controls.Add(testbutton); }

```



ნახ. 32. ფორმაზე ბრძანებითი ღილაკის შექმნა პროგრამულად

ვარიანტი 6.7.3. დავწეროთ პროგრამული კოდი, რომლის დახმარებითაც შევძლებთ ერთი ბრძანებითი ღილაკიდან რამდენიმე ბრძანებითი ღილაკის მიღებას (იხ. ნახ. 33.):

```
private void Form1_Load(object sender, EventArgs e) {
    Button testbutton = new Button();
    testbutton.Text = "button1";
    testbutton.Location = new Point(70, 70);
    testbutton.Size = new Size(50, 50);
    testbutton.Visible = true;
    testbutton.BringToFront();
    this.Controls.Add(testbutton);

    Button testbutton1 = new Button();
    testbutton1.Text = "button2";
    testbutton1.Location = new Point(130, 70);
    testbutton1.Size = new Size(50, 50);
    testbutton1.Visible = true;
    testbutton1.BringToFront();
    this.Controls.Add(testbutton1);
}

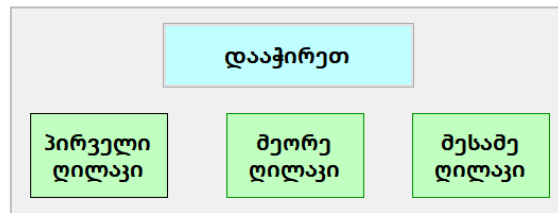
```



```

Button testbutton2 = new Button();
testbutton2.Text = "button3";
testbutton2.Location = new Point(190, 70);
testbutton2.Size = new Size(50, 50);
testbutton2.Visible = true;
testbutton2.BringToFront();
this.Controls.Add(testbutton2); }

```



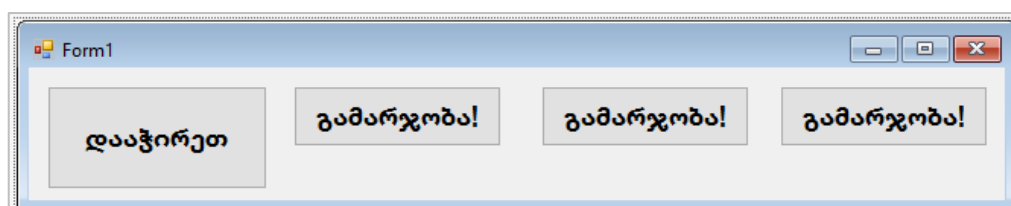
ნახ. 33. ერთი ბრძანებითი ღილაკიდან რამოდენიმე ბრძანებითი ღილაკის მიღება

ვარიანტი 6.7.4. დავწეროთ პროგრამული კოდი, რომლის დახმარებითაც ერთი ბრძანებითი ღილაკიდან მოხდება მეორე ღილაკის მიღება, მეორე ღილაკიდან მესამე ღილაკის მიღება და ა. შ. თანამიმდევრობით (იხ. ნახ. 34.):

```

private void button1_Click(object sender, EventArgs e) {
    //მიმართვა იმ ღილაკზე, რომელსაც დავაჭირეთ;
    Button b = (Button)sender;
    //შევქმნათ ახალი ღილაკი
    Button temp = new Button();
    temp.Text = "გამარჯობა!";
    temp.Width = b.Width;
    //დაჭერის შემდეგ, ახალი ღილაკის 10 პიქსელით მარჯნივ გადაადგილება;
    temp.Location = new Point(b.Location.X + b.Width + 10,b.Location.Y);
    // ახალ ღილაკზე დაჭერის მოვლენის დამატება;
    temp.Click += new EventHandler(button1_Click);
    //ფორმაზე კომპონენტის (ელემენტის) დამატება;
    this.Controls.Add(temp); }

```



ნახ. 34. ერთი ბრძანებითი ღილაკიდან მეორე ღილაკის მიღება, მეორედან მესამის და ა.შ.

ვარიანტი 6.7.5. დავეწეროთ პროგრამული კოდი, რომლის დახმარებითაც ერთი ბრძანებითი ღილაკიდან მოხდება სხვა ბრძანებითი ღილაკების მიღება თანამიმდევრობით და თითოეულ ღილაკზე მიახლოებისას x და y კოორდინატის პოზიციის დაფიქსირება (იხ. ნახატი 35.):

```
namespace WindowsFormsApp11
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

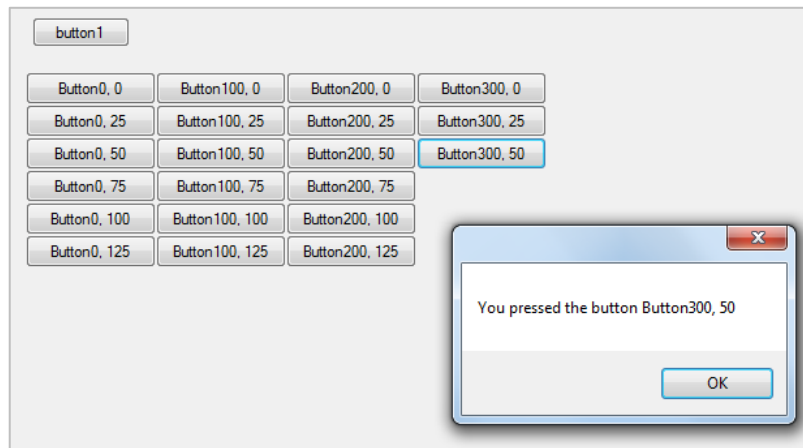
            int _count = 0;
            Point location = new Point(0, 0);
            private void button1_Click_2(object sender, EventArgs e){
                Button firstButton = new Button();
                firstButton.Name = "firstButton" + _count.ToString();
                firstButton.Text = "Button" + location.X.ToString() + ", " +
                                   location.Y.ToString();

                firstButton.Size = new System.Drawing.Size(100, 25);
                firstButton.Left = location.X;
                firstButton.Top = location.Y;
                firstButton.UseVisualStyleBackColor = true;
                firstButton.Click += firstButtons_Click;
                panel1.Controls.Add(firstButton);
                if (_count == 5)
                {
                    _count = 0;
                    location = new Point(location.X + firstButton.Width);
                }
                else
                {
                    _count++;
                    location = new Point(location.X, location.Y + firstButton.Height);
                }
            }
            private void firstButtons_Click(object sender, EventArgs e){
```

```

        Button btn = (Button)sender;
        MessageBox.Show("You pressed the button " + btn.Text);
    }
}
}

```



ნახ. 35. ბრძანებითი ღილაკების მიღება თანამიმდევრობით

დამოუკიდებელი სამუშაო

1. მოცემულია პროგრამული კოდის ფრაგმენტი. იპოვეთ შეცდომა და აღწერეთ იგი.

```

class MyPoint
{
    int x;
    public int y;
}
... ..
MyPoint point_object = new MyPoint();
point_object.x = 3;
point_object.y = 9;

```

2. მოცემული ფრაგმენტის თითოეული სტრიქონი დაწვრილებით აღწერეთ:

1. `class MyPoint`
2. `{`
3. `int x;`
4. `public int y;`
5. `}`
6. `... ..`
7. `MyPoint point_object = new MyPoint();`

8. point_object.x = 31;

9. point_object.y = -9;

3. მოცემულია კოდის ფრაგმენტი. თითოეული სტრიქონი დაწვრილებით აღწერეთ კომენტარების დართვით. დაასრულეთ კოდი და გახადეთ იგი მუშა.

```
public class MyBook
{
    public string Title;
    public string Author;
    public int year;
    public float price;
}
MyBook obieqti_wigni = new MyBook();
obieqti_wigni.Title = "წიგნის დასახელება: ";
obieqti_wigni.Author = "წიგნის ავტორი: ";
obieqti_wigni.year = 1998;
obieqti_wigni.price = 15.80;
```

4. გამოთვალეთ სამი რიცხვის საშუალო არითმეტიკული კლასის შიგნით გამოცხადებულ მეთოდში. გაითვალისწინეთ, რომ მეთოდი არ აბრუნებს მნიშვნელობას.

5. დაადგინეთ რიცხვი არის თუ არა დადებითი. გაითვალისწინეთ, რომ მეთოდი აბრუნებს მნიშვნელობას.

6. გადააკეთეთ ამოცანა 6.3 შემდეგი სახით: კლასს **ოთახი** დაამატეთ ახალი ველი **fasi**, რომელიც აღწერს 1 კვ.მ-ის ღირებულებას. ასევე შექმენით მეთოდი, რომელსაც ექნება ერთი პარამეტრი, კერძოდ, 1 კვ.მ-ის ღირებულება. მოცემულმა მეთოდმა უნდა დააბრუნოს მოცემული ოთახის მთლიანი ღირებულება ფართობის გათვალისწინებით.

7 ობიექტის შექმნა კონსტრუქტორის გამოყენებით. this სიტყვის გამოყენების შემთხვევები

ჩვეულებრივი მეთოდების გარდა კლასში შეგვიძლია გამოვიყენოთ სპეციალური მეთოდი: მეთოდი კონსტრუქტორი. კლასის ახალი ობიექტის შექმნისას ხდება კონსტრუქტორის გამოძახება. კონსტრუქტორი ახდენს კლასის ობიექტის ინიციალიზებას. თუ კლასში არ არის აღწერილი არც ერთი კონსტრუქტორი, მაშინ ასეთი კლასისათვის კონსტრუქტორი ავტომატურად იქმნება ჩუმათობის პრინციპის მიხედვით. ასეთ კონსტრუქტორს არ გააჩნია არც პარამეტრები და არც კოდის ტანი. მისი სინტაქსია:

```
კლასის_სახელი() {  
    კონსტრუქტორის კოდი  
}
```

კონსტრუქტორს ისეთივე სახელი აქვს, როგორც კლასს და მისი სინტაქსი მეთოდის სინტაქსის მსგავსია. მეთოდისგან განსხვავებით კონსტრუქტორში დასაბრუნებელი მნიშვნელობის ტიპი არ ეთითება.

C# ენაში არსებობს საკვანძო სიტყვა `this`, რომელიც უზრუნველყოფს კლასის ობიექტებზე წვდომას. საკვანძო სიტყვა `this` შეგვიძლია გამოვიყენოთ შემდეგ შემთხვევებში:

1. როდესაც კლასის მეთოდის ცვლადის სახელი ემთხვევა თვითონ კლასის რომელიმე წევრის სახელს;
2. თუ კლასს აქვს რამდენიმე კონსტრუქტორი, მაშინ საკვანძო სიტყვა `this` გამოყენება ხდება სხვა კონსტრუქტორებიდან მთავარი კონსტრუქტორის გამოსაძახებლად (მთავარ კონსტრუქტორზე გადამისამართება). ეს საჭიროა კოდის დუბლირების თავიდან ასაცილებლად;
3. როდესაც მეთოდს უნდა გადაეცეს მიმართვა მიმდინარე ობიექტზე. ამ შემთხვევაში, მეთოდის გამოძახების დროს, მიეთითება საკვანძო სიტყვა `this`;
4. იმ შემთხვევაში, თუ კლასში გამოცხადებულია ინდექსატორი.
განვიხილოთ საკვანძო სიტყვა `this` გამოყენების მაგალითები.

მაგალითი 1. ობიექტის წევრთან წვდომა, რომელსაც მაღავს მეთოდის პარამეტრი:

```
public void Metodi_this_1(int age) {  
    this.age = age; } }
```

მეთოდის სახელია `Metodi_this_1()`. მეთოდს `Metodi_this_1()` აქვს ერთი მთელი ტიპის პარამეტრი სახელით `age`. თავად კლასსაც აქვს ცვლადი სახელით `age`. როგორც ვხედავთ მეთოდის პარამეტრის სახელი ემთხვევა კლასის ცვლადის სახელს. სახელების დამთხვევის დროს გამოსავალი არის საკვანძო სიტყვა `this` გამოყენება, რათა თავიდან ავიცილოთ გაურკვევლობა. ზემოთ აღწერილ მეთოდში `this.age` აღნიშნავს კლასის ცვლადს, ხოლო `age` კი აღნიშნავს მეთოდის პარამეტრს.

მაგალითი 2. საკვანძო სიტყვა `this` გადაცემა მიმდინარე ობიექტის სხვა მეთოდს არგუმენტის სახით:

```
public void pirveli_metodi(sxva_Clasi sxva_obieqti) {  
    sxva_obieqti.meore_meTodi(this); }  
}
```

მოცემულ მაგალითში გამოიყენება ორი კლასი: პირველია კლასი, რომელშიც აღწერილია მეთოდი სახელით `pirveli_metodi()`. მეორეა კლასი დასახელებით `sxva_Clasi`, რომელშიც მოცემულია მეთოდი `meore_meTodi()`. მეთოდის `pirveli_metodi()` ტანში ხდება მეთოდის `meore_meTodi()` გამოძახება, რომელსაც გადაეცემა მიმდინარე ობიექტი შემდგომი დამუშავებისთვის.

იმისათვის, რომ კლასის წევრები (ველები) და პარამეტრები განვასხვაოთ ერთმანეთისაგან, კლასის წევრებთან (ველებთან) მიმართვა ხდება `this` საკვანძო სიტყვით. გამოსახულებაში `this.name = name`; კლასის წევრის სახელი და პარამეტრის სახელი ემთხვევა ერთმანეთს. მოცემულ `this.name` ჩანაწერში `name` არის კლასის წევრი და არა პარამეტრი. თუ ჩვენ კლასის წევრის (ველის) სახელი და პარამეტრის სახელი გვექნებოდა განსხვავებული დასახელებებისა, მაშინ არ დაგვჭირდებოდა `this` საკვანძო სიტყვის გამოყენება. ამასთანავე `this` საკვანძო სიტყვის გამოყენებით შეგვიძლია მივმართოთ კლასის ნებისმიერ წევრს (ველს) და მეთოდს.

გადავიდეთ ამოცანების განხილვაზე.

ამოცანა 7.1. შექმენით კლასი „ეკიმი“, რომელიც შეიცავს პრივატულ ცვლადებს: გვარი, სახელი, ასაკი. საერთო წვდომის მეთოდებს: პირველი მეთოდია კონსტრუქტორი, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს შეტყობინების ფანჯარაში გამოაქვს პრივატული ცვლადების მნიშვნელობები (იხ. ნახ. 36.).

```
class Eqimi {  
    private string saxeli;  
    private string gvari;  
}
```

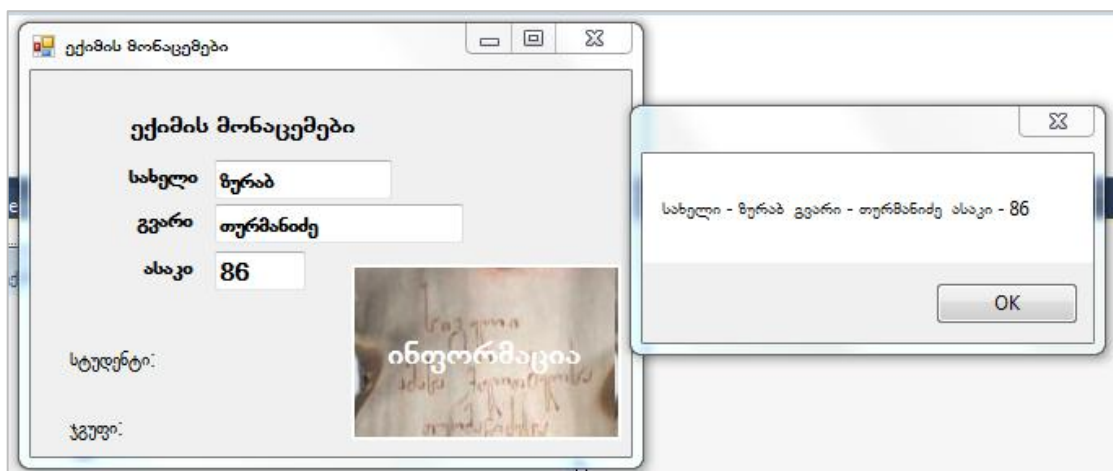
```

private int asaki;
// კონსტრუქტორი
public Eqimi(string a1, string a2, int a3){
    saxeli = a1;
    gvari = a2;
    asaki = a3; }
public void informacia() {
    MessageBox.Show("სახელი - " + saxeli + " გვარი - " + gvari +
                    " ასაკი - " + asaki.ToString()); }

}

private void button1_Click_2(object sender, EventArgs e) {
    /*საქართველოში გვარი თურმანიძე ოდითგანვე ხალხურ მედიცინასთან ასოცირდება.
    თურმანიძეები სამეფო კარის მკურნალ ექიმებად გვევლინებოდნენ. მათ შესახებ ისტორიულ
    წყაროებშიც უამრავი დოკუმენტური ინფორმაციაა შემონახული, რომელთა მიხედვითაც გვარი
    თურმანიძე პირველად XI საუკუნეშია ნახსენები. XX საუკუნის დასაწყისში საბჭოთა მთავრობამ
    სახალხო მკურნალს - მაღაქია თურმანიძეს მთელ საქართველოში სამედიცინო საქმიანობის
    ოფიციალური ნება მიანიჭა (1929 წ.). ზურაბ თურმანიძე (დაბ. თარიღი 1937 წ.) რომელიც
    განაგრძობს გვარის ტრადიციებს, არის ნახევრსაუკუნოვანი უწყვეტი პრაქტიკის მქონე ექიმი-
    ქირურგი.*/
    string Gvari = textBox1.Text;
    string Saxeli = textBox2.Text;
    int Asaki = int.Parse(textBox3.Text);
    Eqimi obieqti_eqimi = new Eqimi(Gvari,Saxeli,Asaki);
    obieqti_eqimi.informacia();
}

```



ნახ. 36. ამოცანა 7.1. -ის შესრულების შედეგი

ამოცანა 7.2. კონსტრუქტორის დემონსტრირების მეორე მაგალითი.

```

class Animal {
    string name;
    int weight;
    int height;
    public Animal(string name1, int weight1, int height1)
    {
        name = name1;
        weight = weight1;
        height = height1;
    }
    public void Info()
    {
        MessageBox.Show("ცხოველის სახელი " + name);
        MessageBox.Show("ცხოველის წონა " + weight);
        MessageBox.Show("ცხოველის სიმაღლე " + height);
    }
    public void Info_dasveneba()
    {
        height = height + 5;
        MessageBox.Show("სიმაღლე ძილის მშემდეგ " + height);
    }
    public void Info_tamashi()
    {
        weight = weight - 5;
        MessageBox.Show("წონა თამაშის შემდეგ " + weight);
    }
    public void Info_kveba()
    {
        weight = weight + 15;
        MessageBox.Show("წონა ჭამის შემდეგ " + weight);
    }
}

private void button1_Click_2(object sender, EventArgs e) {
    Animal obieqti_dzagli = new Animal("ძაღლი", 20, 50);

```



```

obieqti_dzagli.Info();
obieqti_dzagli.Info_dasveneба();
obieqti_dzagli.Info_tamashi();
obieqti_dzagli.Info_kveba();
Animal obieqti_kata = new Animal("კატა", 8, 25);
obieqti_kata.Info();
obieqti_kata.Info_dasveneба();
obieqti_kata.Info_tamashi();
obieqti_kata.Info_kveba();
Animal obieqti_kurdgeli = new Animal("კურდღელი", 10, 30);
obieqti_kurdgeli.Info();
obieqti_kurdgeli.Info_dasveneба();
obieqti_kurdgeli.Info_tamashi();
obieqti_kurdgeli.Info_kveba(); }

```

ამოცანა 7.3. საკვანძო სიტყვის this გამოყენებით მეთოდის ბლოკიდან კლასის წევრებთან (ველებთან) წვდომა. კლასის წევრებთან (ველებთან) წვდომა ხდება მეთოდების დახმარებით.

```

class Programa_this {
    int ricxvi = 123; //კლასის წევრი (ველი) სახელით ricxvi;
    public Programa_this() { } //კონსტრუქტორი პარამეტრების გარეშე;
    public Programa_this(int ricxvi)//კონსტრუქტორი ერთი პარამეტრით;
    {
        //საკვანძო სიტყვის this გამოყენება კლასის წევრთან (ricxvi) წვდომისათვის;
        this.ricxvi = ricxvi; }

    /*ფუნქცია ბეჭდავს პარამეტრის და კლასის წევრის ricxvi სახელს. პარამეტრის
    სახელი ricxvi ემთხვევა კლასის წევრის ricxvi სახელს. კლასის წევრთან წვდომა:
    ხდება კლასის წევრის სახელის გადაფარვა მეთოდის პარამეტრის სახელით*/
    public void PrintInt(int ricxvi)
    { // პარამეტრთან წვდომა
        MessageBox.Show("პარამეტრის გამოტანა " + ricxvi);

        // კლასის წევრთან (ველი ricxvi = 123;) წვდომა this.ricxvi მიმართვით;
        MessageBox.Show("კლასის წევრთან წვდომა პარამეტრით "+this.ricxvi);}

    // ფუნქცია, რომელშიც აღწერილია ლოკალური ცვლადი სახელით ricxvi
    // კლასის წევრთან წვდომა. ხდება კლასის წევრის სახელის გადაფარვა

```

```

// მეთოდის ბლოკში გამოცხადებული ცვლადი სახელით:
    public void Print() {
        int ricxvi = 12345; // ცვლადის სახელი ემთხვევა კლასის წევრის სახელს
        // ცვლადის სახელის გამოტანა
        MessageBox.Show("ცვლადის გამოტანა " + ricxvi);
        // კლასის ველთან წვდომა this.ricxvi მიმართვით;
        MessageBox.Show("კლასის წევრთან წვდომა ცვლადით " + this.ricxvi);
    }
}

private void button1_Click(object sender, EventArgs e){
    // ერთპარამეტრიანი კონსტრუქტორის გამოძახება;
    Programa_this objeqtithis = new Programa_this(100);
    // მეთოდის PrintInt() გამოძახება;
    objeqtithis.PrintInt(150);
    // მეთოდის Print() გამოძახება;
    objeqtithis.Print();
}

```

დამოუკიდებელი სამუშაო

1. ამოცანა 7.3 გადააკეთეთ ისეთი სახით, რომ მონაცემები შეიტანოთ textBox კომპონენტის დახმარებით. ასევე ფორმაზე ინფორმაციის გამოსატანად გამოიყენეთ შესაბამისი კომპონენტები. ააგეთ ფორმა და გაუშვით შესრულებაზე პროექტი. გადმოიტანეთ შესრულების შედეგი და სრული კოდი.

2. აღწერეთ ქვემოთ მოცემული პროგრამის ფრაგმენტის თითოეული სტრიქონი კომენტარების დართვით:

```

class MyBook
{
    private string Title;
    private string Author;
    private int year;
    private float price;
    public MyBook()
    {

```

8

```
Title = "";  
Author = "";  
year = 1900;  
price = 0.00f;  
}  
public MyBook(string t, string a, int y, float p)  
{  
    Title = t;  
    Author = a;  
    year = y;  
    price = p;  
}  
}  
...  
MyBook mb1 = new MyBook();  
MyBook mb2 = new MyBook("Title-1", "Author-1", 2000,  
3.99f);  
}
```

პოლიმორფიზმი: მეთოდის გადატვირთვა. კონსტრუქტორის გადატვირთვა

მეთოდის გადატვირთვა. ობიექტზე ორიენტირებული პროგრამირების ერთ-ერთი მთავარი მახასიათებელი არის პოლიმორფიზმი. პოლიმორფიზმის საშუალებით

ბით ობიექტებს შეუძლიათ მოიქცნენ განსხვავებულად, მიუხედავად იმისა თუ როგორ ხდება მათი გამოყენება. პოლიმორფიზმის ერთ-ერთი შემადგენელი ნაწილია მეთოდის ისეთი უნარი, როდესაც მას შეუძლია სრულიად განსხვავებულად მოიქცეს მასზე გადაცემული პარამეტრების ტიპებისა და პარამეტრების რაოდენობის მიხედვით/შესაბამისად.

ზოგჯერ საჭიროა ერთსა და იმავე დასახელების მეთოდის შექმნა, მაგრამ პარამეტრების განსხვავებული რაოდენობით ან განსხვავებული ტიპებით. ამის მიღწევა შესაძლებელია მეთოდის გადატვირთვით (Method Overloading). მეთოდების გადატვირთვა პროგრამისტს საშუალებას აძლევს განსაზღვროს უამრავი/ზევრი, რამდენიც მას ჭირდება იმდენი მეთოდი ერთსა და იმავე დასახელებით, მაგრამ პარამეტრების და პარამეტრების ტიპების სხვადასხვა რაოდენობით.

პარამეტრებისა და პარამეტრების ტიპების თითოეულ კომბინაციას ეწოდება მეთოდის ხელმოწერა (სიგნატურა, Signature). როდესაც ერთ-ერთი ამ გადატვირთული მეთოდის გამოძახება ხდება, კომპილატორი ავტომატურად განსაზღვრავს თუ რომელი მეთოდი უნდა იყოს გამოყენებული, არგუმენტებისა და ხელმისაწვდომი მეთოდების ხელმოწერების რაოდენობის მიხედვით.

მეთოდის ხელმოწერის შემადგენელი ნაწილებია:

- ა) მეთოდის სახელი;
- ბ) პარამეტრების რაოდენობა;
- გ) პარამეტრების ტიპები;
- დ) პარამეტრების რიგითობა;
- ე) პარამეტრების მოდიფიკატორი.

პარამეტრების დასახელებები კი მეთოდის ხელმოწერაში არ შედის.

კომპილატორი იყენებს მეთოდის ხელმოწერას რათა განასხვავოს კლასში არსებული მეთოდები. ხელმოწერა არ შეიცავს პარამეტრების სახელებს და დასაბრუნებელი მნიშვნელობის ტიპს (იხ. ცხრილი 9).

ცხრილი 9. მეთოდის ხელმოწერის მაგალითები

მეთოდებს აქვს	<code>static int Code() { ... }</code>
განსხვავებული ხელმოწერა:	<code>static int Code(int n) { ... }</code>
	<code>static int Code(int n, int m) { ... }</code>

მეთოდებს აქვს ერთნაირი ხელმოწერა:	<pre>static int Code(int n) { ... } static string Code(int n) { ... } static int Code(int x) { ... }</pre>

მაგალითად მოცემული გვაქვს შემდეგი მეთოდი:

```
public int Sum(int x, int y){
    return x + y; }
```

მეთოდის Sum() ხელმოწერას ექნება შემდეგი სახე: Sum(int, int). როგორც ვიცით, მეთოდმა შეიძლება მიიღოს ერთი პარამეტრი, მაგრამ ჩვენ ასევე შეგვიძლია დავწეროთ ისეთი მეთოდი, რომელიც მიიღებს ორ ან რამდენიმე ან იმდენ პარამეტრს, რამდენიც გვჭირდება. მნიშვნელოვანია ვიცოდეთ, რომ ნებისმიერ პარამეტრს უნდა ჰქონდეს საკუთარი ტიპი, სახელი და ისინი ერთმანეთისაგან აუცილებლად უნდა იყვნენ გამოყოფილი მძიმეებით. მაგალითად, მოცემულია მეთოდი:

```
public void AllNumbers(int a, int b, int c) { //..... }
```

როდესაც ჩვენ ვიძახებთ მეთოდს AllNumbers(), მაშინ მის პარამეტრებს გადავცემთ სხვადასხვა მნიშვნელობას (ანუ არგუმენტებს): AllNumbers(3, 7, 11);

ახლა კი გადავიდეთ მეთოდის გადატვირთვაზე. მეთოდის გადატვირთვა გულისხმობს სხვადასხვა სახის/ვარიანტის ხელმოწერის (სიგნატურის) არსებობას. მეთოდის გადატვირთვისას მეთოდის სახელები ემთხვევა ერთმანეთს, ხოლო დანარჩენი მახასიათებლები არა.

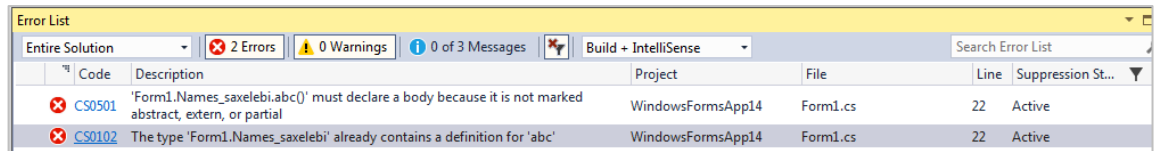
მეთოდის გადატვირთვისას მეთოდები უნდა განსხვავდებოდნენ შემდეგი მახასიათებლების მიხედვით: ა) პარამეტრების რაოდენობა; ბ) პარამეტრების ტიპები; გ) პარამეტრების რიგითობა; დ) პარამეტრების მოდიფიკატორი.

მეთოდის გადატვირთვისას მეთოდის სახელი არ შეიძლება დაემთხვეს კლასის ნებისმიერი სხვა ელემენტის სახელს, მაგრამ შეიძლება დაემთხვეს სხვა მეთოდის სახელს. გადატვირთული მეთოდები არის მეთოდები, რომლებსაც აქვთ ერთიდაიგივე სახელები ერთსა და იმავე კლასში. არ შეიძლება მეთოდის დასახელება დაემთხვეს ან ცვლადის ან მუდმივის ან ჩამონათვალის დასახელებას. მაგალითად:

```
class Names_saxelebi {
    static int abc;
```

```
static void abc(); //შეცდომაა, მეთოდის სახელი ემთხვევა ცვლადის სახელს
}
```

ზემოთ ნაჩვენები კოდის ფრაგმენტის აკრეფის დროს კომპილატორი აჩვენებს შეცდომას (ნახ. 37.):



ნახ. 37. კომპილატორის მიერ შეცდომის დაფიქსირება

განვიხილოთ მაგალითი. ვთქვათ, მოცემული გვაქვს შემდეგი კოდი:

```
class Shekreba {
    public void meTodi_shekreba(int a, int b) {
        int result = a + b; }
    public void meTodi_shekreba(int a, int b, int c) {
        int result = a + b + c; }
    public int meTodi_shekreba(int a, int b, int c, int d) {
        int result = a + b + c + d;
        return result; }
    public void meTodi_shekreba(double a, double b) {
        double result = a + b; }
}
```

კოდში მოცემულია ერთსა და იმავე დასახელების მეთოდის meTodi_shekreba() ოთხი ვარიანტი ანუ განსაზღვრულია მეთოდის meTodi_shekreba() ოთხი ცალი გადატვირთვა. მეთოდების პირველი სამი ვარიანტი განსხვავებულია პარამეტრების რაოდენობების მიხედვით. მეოთხე ვარიანტი კი ემთხვევა პირველ ვარიანტს პარამეტრების რაოდენობის მიხედვით, მაგრამ განსხვავებულია პარამეტრების ტიპების მიხედვით. ამ შემთხვევაში საკმარისია ერთი პარამეტრი მაინც განსხვავებული იყოს ტიპის მიხედვით.

მეთოდების meTodi_shekreba() ხელმოწერა შეგვიძლია წარმოვიდგინოთ შემდეგი სახით:

```
meTodi_shekreba(int, int)
meTodi_shekreba(int, int, int)
meTodi_shekreba(int, int, int, int)
meTodi_shekreba(double, double)
```

მეთოდის გადატვირთული ვარიანტების განსაზღვრის შემდეგ, ჩვენ შეგვიძლია მათი გამოყენება პროგრამაში ქვემოთ მოცემული სახით:

```
Shekreba obieqti_gadatvirTva = new Shekreba();
obieqti_gadatvirTva.meTodi_shekreba(1, 3); // 4
obieqti_gadatvirTva.meTodi_shekreba(1, 2, 3); // 6
obieqti_gadatvirTva.meTodi_shekreba(1, 2, 3, 4); // 10
obieqti_gadatvirTva.meTodi_shekreba(2.4, 2.5); // 4.9
```

გადატვირთული მეთოდების გამოყენება სასარგებლოა მაშინ, თუ არსებობს ერთნაირი დასახელების მეთოდები, რომლებიც განსხვავდება. მაგალითად, მხოლოდ პარამეტრების ჩამონათვალით, როგორც ქვედა კოდშია ნაჩვენები. კოდს დვარქვით „მისალმება“:

```
class misalmeba {
    public static void studenti_misalmeba(){
        MessageBox.Show("დილა მშვიდობასა "); }
    public static void studenti_misalmeba(string Name) {
        MessageBox.Show("დილა მშვიდობასა " + Name);}
}
private void button1_Click(object sender, EventArgs e){
    misalmeba.studenti_misalmeba();
    misalmeba.studenti_misalmeba(" ლუკა"); }
```

მეთოდის გადატვირთვა სასარგებლოა მაშინ როდესაც საჭიროა კოდი გავხადოთ უფრო ფუნქციონალური. მოდიოთ, ზემოთ აღწერილი კოდი „მისალმება“ გავხადოთ შედარებით უფრო ფუნქციური:

```
class misalmeba {
    public enum TimeOfDay { Morning, Afternoon, Evening };
    public static void studenti_misalmeba(){
        MessageBox.Show("დილა მშვიდობასა "); }
    public static void studenti_misalmeba(string Name) {
        MessageBox.Show("დილა მშვიდობასა " + Name); }
    public static void studenti_misalmeba(string Name, TimeOfDay td) {
        string Message = "";
        switch (td)
```

```

{
    case TimeOfDay.Morning:
        Message = "დილა მშვიდობის ";
        break;
    case TimeOfDay.Afternoon:
        Message = "შუა დღე მშვიდობის ";
        break;
    case TimeOfDay.Evening:
        Message = "საღამო მშვიდობის ";
        break;
}
MessageBox.Show(Name + " " + Message ); }
}

private void button1_Click(object sender, EventArgs e){
    misalmeba.studenti_misalmeba();
    misalmeba.studenti_misalmeba(" ლუკა ");
    misalmeba.studenti_misalmeba(" ნინო ",
    misalmeba.TimeOfDay.Afternoon); }

```

დასკვნის სახით:

1. პროექტში შეიძლება გვქონდეს ორი სხვადასხვა მეთოდი ერთსა და იმავე სახელით, მაგრამ არ შეიძლება ორი სხვადასხვა მეთოდი გვქონდეს ერთსა და იმავე მეთოდის ხელმოწერით. როგორც მინიმუმ მეთოდის ხელმოწერის ერთი ნაწილი მაინც განსხვავებული უნდა იყოს. როდესაც ორი ან მეტი მეთოდი არის ერთსა და იმავე დასახელების, მაგრამ განსხვავებული ხელმოწერის, საქმე გვაქვს მეთოდის გადატვირთვასთან.

2. მეთოდის გადატვირთვა ნიშნავს, ორი ან მეტი ერთსა და იმავე სახელის და განსხვავებული პარამეტრების მქონე მეთოდების არსებობას ერთსა და იმავე კლასში.

3. მეთოდები, რომლებსაც აქვთ პარამეტრებში ref და out საკვანძო სიტყვები, არ შეიძლება გადატვირთული იყოს.

4. მეთოდები, რომლებსაც აქვთ ერთნაირი ხელმოწერა (მეთოდის სახელი პარამეტრების სიით), არ შეიძლება გადატვირთული იყოს.

კონსტრუქტორების გადატვირთვა. მეთოდების მსგავსად შესაძლებელია კონსტრუქტორების გადატვირთვაც. ეს საშუალებას გვაძლევს, ობიექტები შევქმნათ სხვადასხვა საშუალებით. კონსტრუქტორების გადატვირთვა იძლევა იმის საშუალებას, რომ ერთი ობიექტის ინიციალიზებისათვის კონსტრუქტორმა მეორე ობიექტი გამოიყენოს.

რიგ შემთხვევაში გადატვირთვადი კონსტრუქტორის დაწერა სასარგებლოა იმ კუთხით, რომ ერთ კონსტრუქტორს აქვს საშუალება გამოიძახოს მეორე კონსტრუქტორი. C# ენაში ეს ხორციელდება `this` სიტყვის გამოყენებით. მისი სინტაქსია: **კონსტრუქტორის_სახელი (პარამეტრების_სია_1) : this ((პარამეტრების_სია_2) {**

// კონსტრუქტორის კოდი, რომელიც შეიძლება იყოს ცარიელი

}

გადატვირთვადი კონსტრუქტორების გამოძახებისათვის `this` სიტყვის გამოყენების ერთ-ერთი მიზეზი არის კოდის დუბლირების თავიდან აცილების შესაძლებლობა.

მოდით ახლა განვიხილოთ გადატვირთული კონსტრუქტორის მაგალითი:

```
class elementebi {  
    public string dasaxeleba;  
    public int fasi;  
    public int vada;
```

//კონსტრუქტორის გადატვირთვის პირველი ვარიანტი;

```
    public elementebi() {  
        this.dasaxeleba = "Item";  
        this.fasi = 123;  
        this.vada = 1; }  
    
```

//კონსტრუქტორის გადატვირთვის მეორე ვარიანტი;

```
    public elementebi(string name){  
        this.dasaxeleba = name;  
        this.fasi = 123;  
        this.vada = 1; }  
    
```

//კონსტრუქტორის გადატვირთვის მესამე ვარიანტი;

```
    public elementebi(string name, int price, int v) {  
        this.dasaxeleba = name;
```

```

        this.fasi = price;
        this.vada = v; }
    }

```

ახლა კი განვიხილოთ კიდეც ერთი მაგალითი, სადაც ხდება კონსტრუქტორის გადატვირთვა:

```

class Adamiani {
    public string name;
    public int age;

    // პირველი კონსტრუქტორი;
    public Adamiani() { name = "უცნობია"; age = 18; }

    // მეორე კონსტრუქტორი;
    public Adamiani(string n) { name = n; age = 21; }

    // მესამე კონსტრუქტორი;
    public Adamiani(string n, int a) { name = n; age = a; }

    public void metodi() //მეთოდი
    {
        MessageBox.Show ("სახელი " + name + " ასაკი " + age);
    }
}

```

კლასში **Adamiani** აღწერილია სამი კონსტრუქტორი. თითოეული მათგანი იღებს პარამეტრების სხვადასხვა რაოდენობას და კლასის წევრებს ანიჭებს მნიშვნელობებს. კონსტრუქტორის გადატვირთვის კიდეც ერთი მაგალითი:

```

private void button1_Click(object sender, EventArgs e) {
    //პირველი კონსტრუქტორის გამოძახება.
    //პირველი კონსტრუქტორი არის პარამეტრების გარეშე;
    Adamiani ana = new Adamiani();

    //მეორე კონსტრუქტორის გამოძახება.
    //მეორე კონსტრუქტორი არის ერთი პარამეტრით;
    Adamiani luka = new Adamiani("ლუკა");

    //მესამე კონსტრუქტორის გამოძახება.
    //მესამე კონსტრუქტორი არის ორი პარამეტრით;
    Adamiani saba = new Adamiani("საბა", 27);
}

```

```

        ana.metodi(); //სახელი: უცნობია ასაკი: 18
        luka.metodi(); //სახელი: ლუკა ასაკი: 21
        saba.metodi(); //სახელი: საბა ასაკი: 27
    }

```

ზემოთ მოცემულ კოდში აღწერილი არის სამი კონსტრუქტორი. სამივე კონსტრუქტორი აღწერს ერთსა და იმავე ტიპის მოქმედებას - კლასის წევრებს (ველებს name და age) ანიჭებს მნიშვნელობებს. ასეთი სახის ქმედებები შეიძლება იყოს საკმაოდ ბევრი. კონსტრუქტორის ფუნქციონალურობის დუბლირების თავიდან ასაცილებლად შეგვიძლია, ერთი კონსტრუქტორიდან მეორე კონსტრუქტორს მივმართოთ this საკვანძო სიტყვით, რომელსაც გადავცემთ პარამეტრების მნიშვნელობებს:

```

class Adamiani {
    public string name;
    public int age;
    public Adamiani() : this("უცნობია") { }
    public Adamiani(string name) : this(name, 18) { }
    public Adamiani(string name, int age)
    { this.name = name; this.age = age; }
    public void metodi() //მეთოდი
    {
        MessageBox.Show ("სახელი: " + name + " ასაკი: " + age);
    }
}

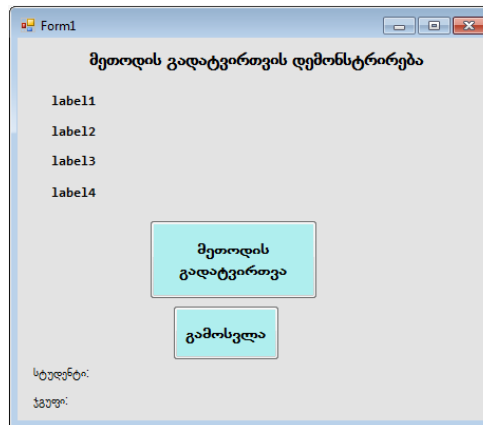
```

განხილულ მაგალითში პირველი კონსტრუქტორი იძახებს მეორეს, მეორე კონსტრუქტორი კი - მესამეს. პარამეტრების ტიპის ან რაოდენობის მიხედვით კომპილატორი ამოიცნობს, თუ რომელი კონსტრუქტორი უნდა იქნას გამოძახებული.

პრაქტიკული ლაბორატორიული

ამოცანა 8.1. მოცემული გვაქვს კლასი დასახელებით Gadatvirtva. კლასში განვახორციელოთ მეთოდის (დასახელებით Metodi()) ოთხჯერ გადატვირთვა. პირველ ვერსიას არ აქვს პარამეტრი, მეორე ვერსიას აქვს ერთ მთელი ტიპის

პარამეტრი, მესამეს - ორი მთელი ტიპის პარამეტრი და მეოთხეს კი - ორი წილადი ტიპის პარამეტრი (იხ. ნახ. 38.).



ნახ. 38. მეთოდის გადატვირთვა

```
class Gadatvirtva {  
    // მეთოდს პარამეტრები არ აქვს და აბრუნებს სტრიქონს;  
    public string Metodi()  
    {  
        return "გამოცხადებული იქნა უპარამეტრებო მეთოდი";  
    }  
    // მეთოდს double ტიპის ერთი პარამეტრი აქვს და აბრუნებს მის  
    კვადრატს  
    public double Metodi(double par1)  
    {  
        return Math.Pow(par1, 2);  
    }  
    // მეთოდს int ტიპის ორი პარამეტრი აქვს და აბრუნებს მათ ჯამს  
    public int Metodi(int par1, int par2)  
    {  
        return par1 + par2;  
    }  
    // მეთოდს double ტიპის ორი პარამეტრი აქვს და აბრუნებს მათ ჯამს  
    public double Metodi(double par1, double par2)  
    {  
        return par1 + par2;  
    }  
}
```

```

    }
    private void button1_Click(object sender, EventArgs e){
        Gadatvirtva obieqti = new Gadatvirtva();
        int ricxvi1;
        double wiladi1, wiladi2;
        string striqoni;
        // Metodi მეთოდის ყველა ვერსიის გამოძახება
        striqoni = obieqti.Metodi();
        label1.Text = striqoni;
        wiladi2 = obieqti.Metodi(5.7);
        label2.Text = "გამოძახებული იქნა ერთპარამეტრიანი მეთოდი. შედეგი = "
            + wiladi2.ToString();
        ricxvi1 = obieqti.Metodi(5, 10);
        label3.Text = "გამოძახებული იქნა ორპარამეტრიანი მეთოდი. შედეგი = "
            + ricxvi1.ToString();
        wiladi1 = obieqti.Metodi(5.5, 10.10);
        label4.Text = " გამოძახებული იქნა ორპარამეტრიანი მეთოდი. შედეგი = "
            + wiladi1.ToString();
    }
    private void button2_Click(object sender, EventArgs e) {
        Close(); // ფორმის დახურვა
        //Application.Exit(); აპლიკაციის დახურვა }
    }

```

ამოცანა 8.2. კონსტრუქტორის გადატვირთვის დემონსტრირების მარტივი ნიმუში (იხ. ნახ. 39).

```

class One {
    int val0;
    int val1;
    int val2;
    int val3;
    //კონსტრუქტორის გადატვირთვის პირველი ვარიანტი
    public One(int arg0, int arg1)
    {

```

```

        val0 = arg0;
        val1 = arg1;
    }
    //კონსტრუქტორის გადატვირთვის მეორე ვარიანტი
    public One(int arg0, int arg1, int arg2)
    {
        val0 = arg0;
        val1 = arg1;
        val2 = arg2;
    }
    //კონსტრუქტორის გადატვირთვის მესამე ვარიანტი
    public One(int arg0, int arg1, int arg2, int arg3){
        val0 = arg0;
        val1 = arg1;
        val2 = arg2;
        val3 = arg3;
    }
    public int Sum() // მეთოდი
    {
        return (val0 + val1 + val2 + val3);
    }
}

private void button1_Click(object sender, EventArgs e) {
    One A = new One(10, 20);
    int aa = A.Sum();
    label1.Text = "გამოძახებული იქნა ორპარამეტრიანი კონსტრუქტორი = " +
        aa.ToString();

    One B = new One(10, 20, 30);
    int bb = B.Sum();
    label2.Text = "გამოძახებული იქნა სამპარამეტრიანი კონსტრუქტორი = " +
        bb.ToString();

    One C = new One(10, 20, 30, 40);
    int cc = C.Sum();
    label3.Text = "გამოძახებული იქნა ოთხპარამეტრიანი კონსტრუქტორი = "

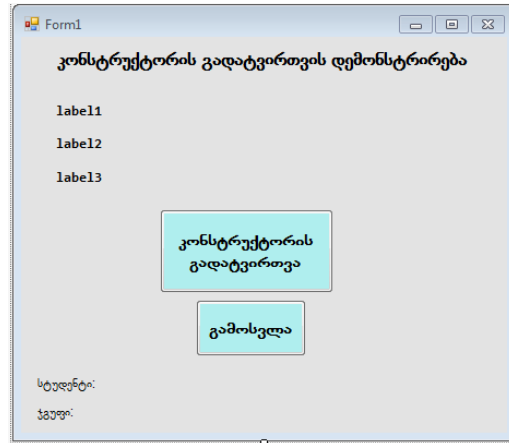
```

```

+ cc.ToString(); }

private void button2_Click(object sender, EventArgs e){
    Close(); // ფორმის დახურვა
    //Application.Exit(); // აპლიკაციის დახურვა }

```



ნახ. 35. კონსტრუქტორის გადატვირთვა

ამოცანა 8.3. კონსტრუქტორის გადატვირთვის დემონსტრირების კიდეც ერთი განსხვავებული ნიმუში.

```

class UserInfo {
    public string Name, Family, abraKa;
    public byte Age;

    // დავწეროთ რამდენიმე გადატვირთული კონსტრუქტორი;
    public UserInfo() {abraKa = "----- მონაცემები უცნობია -----"; }
    public UserInfo(string Name) {
        this.Name = Name; }
    public UserInfo(string Name, string Family) {
        this.Name = Name;
        this.Family = Family; }
    public UserInfo(string Name, string Family, byte Age) {
        this.Name = Name;
        this.Family = Family;
        this.Age = Age;
    }
}

private void button1_Click(object sender, EventArgs e){
    //შევქმნათ UserInfo ტიპის ობიექტები.
    //გამოვიყენოთ გადატვირთული კონსტრუქტორი:

```

```

UserInfo obj1 = new UserInfo();
UserInfo obj2 = new UserInfo(Name: "ლუკა");
UserInfo obj3 = new UserInfo(Name: "ლუკა", Family: "ბერიძე");
UserInfo obj4 = new UserInfo(Name: "ლუკა", Family: "ბერიძე", Age: 18);
label1.Text = obj1.abraka;
label2.Text = obj2.Name;
label3.Text = obj3.Name + " " + obj3.Family;
label4.Text = obj4.Name + " " + obj4.Family + " " + obj4.Age.ToString(); }

```

ამოცანა 8.4. კონსტრუქტორის გამოძახება this საკვანძო სიტყვის დახმარებით გამორიცხავს კოდის ზედმეტ დუბლირებას. ამოცანა 8.3-ში მოყვანილ კოდში არ არის აუცილებელი სამივე კონსტრუქტორში ერთსა და იმავე ინიციალიზაციის თანმიმდევრობის დუბლირება და სწორედ this საკვანძო სიტყვის გამოყენების წყალობით ხდება კოდის დუბლირების თავიდან აცილება. გადატვირთული კონსტრუქტორის კიდეც ერთი უპირატესობაა კონსტრუქტორების შექმნა „ჩუმათობის/ნაგულისხმევი“ არგუმენტებით, როდესაც ეს არგუმენტები არ არის მითითებული. ახლა კი განვიხილოთ ამოცანა 8.3-ის უფრო გაუმჯობესებული ვერსია, სადაც არ ხდება კოდების დუბლირება:

```

class UserInfo {
    public string Name, Family;
    public byte Age;

    // გამოვიყენოთ საკვანძო სიტყვა this კონსტრუქტორების შესაქმნელად:
    public UserInfo() : this("უცნობია ", "უცნობია ", 0) { }
    public UserInfo(UserInfo obj) : this(obj.Name, obj.Family, obj.Age)
    { }
    public UserInfo(string Name, string Family, byte Age) {
        this.Name = Name;
        this.Family = Family;
        this.Age = Age;
    }
}

private void button1_Click(object sender, EventArgs e) {
    UserInfo ui1 = new UserInfo();
    label1.Text = ui1.Name + ui1.Family+ ui1.Age;
}

```



```

        UserInfo ui2 = new UserInfo("ლუკა ", " ბერიძე ", 18);
        label2.Text = ui2.Name + ui2.Family + ui2.Age;
        UserInfo ui3 = new UserInfo(ui2);
        label3.Text = ui3.Name + ui3.Family + ui3.Age;
    }

```

დამოუკიდებელი სამუშაო

1. ქვემოთ მოცემული კოდის მოცემულობა დაწერეთ დამოუკიდებლად. პროგრამას დაურთეთ კომენტარები და აღწერეთ სტრიქონები. ასევე ახსენით თუ რა იქნება კოდის შესრულების შედეგი.

```

class MaxMethods
{
    // .....
    public int Max(int a, int b)
    {
        if (a < b) return b;
        return a;
    }

    public int Max(int a, int b, int c) // .....
    {
        int max = a;
        if (max < b) max = b;
        if (max < c) max = c;
        return max;
    }

    public int Max(int a, int b, int c, int d) // .....
    {
        int max = a;
        if (max < b) max = b;
        if (max < c) max = c;
        if (max < d) max = d;
        return max;
    }
}

private void button1_Click(object sender, EventArgs e)
{

```

```

MaxMethods mm = new MaxMethods();
9 int max;
max = mm.Max(5, 6);
label1.Text = max.ToString(); // .....
max = mm.Max(1, -3, 8);
label2.Text = max.ToString(); // .....
max = mm.Max(2, 3, 5, 1);
label3.Text = max.ToString(); // .....
}

```

2. მოცემულია მეთოდი:

```

int Summa(int a, int b = 0, int c = 0) {
    return a + b + c;
}

```

დაწერეთ აღწერილი მეთოდის ხელმოწერა (სიგნატურა) ანუ ზოგადად როგორი სიგნატურა ექნება მეთოდს?

3. დაწერეთ პროგრამული კოდი რომელიც იპოვოს სამ მთელ რიცხვს შორის მაქსიმალურს. ამ კოდის დასაწერად გამოიყენეთ გადატვირთული მეთოდი ორ რიცხვს შორის მაქსიმუმის დასადგენად. რიცხვების შეტანა განახორციელეთ კლავიატურიდან.

მემკვიდრეობითობის არსი. protected მოდიფიკატორი. კონსტრუქტორები და მემკვიდრეობითობა. base საკვანძო სიტყვა. წინაპარი კლასის კონსტრუქტორის გამოძახება

მემკვიდრეობითობის არსი. protected მოდიფიკატორი. მემკვიდრეობითობა (inheritance) არის ობიექტზე-ორიენტირებული პროგრამირების ერთ-ერთი მთავარი და მნიშვნელოვანი თვისება. მემკვიდრე კლასი მემკვიდრეობით იღებს წინაპარი/მშობელი/საბაზო კლასის მონაცემებს და ფუნქციურობას ანუ სხვა სიტყვებით რომ ვთქვათ ერთ კლასს შეუძლია მემკვიდრეობით მიიღოს რომელიმე მეორე კლასის ფუნქციურობა. მოდით ახლა განვსაზღვროთ, თუ რას ნიშნავს წინაპარი/მშობელი/საბაზო კლასი და მემკვიდრე კლასი? წინაპარი/საბაზო (base class) კლასი არის ისეთი პროგრამული კოდი, რომელსაც გამოიყენებენ მემკვიდრე

(წარმოებული) კლასები. მემკვიდრე/წარმოებული (derived class) კლასი ეს არის ისეთი კლასი, რომელიც იყენებს წინაპარი/საბაზო კლასის პროგრამულ კოდს და ამასთანავე ცვლის (ამატებს, აფართოებს) თავის კოდს საჭიროების მიხედვით. ზოგიერთ პროგრამულ ენებში, მაგალითად Java-ში წინაპარი/საბაზო კლასს უწოდებენ სუპერკლასს, ხოლო მემკვიდრე კლასს კი ქვეკლასს.

მემკვიდრეობითობის დროს წინაპარი/საბაზო კლასის მეთოდებსა და ველებს შეიძლება დავმატოვოთ მემკვიდრე კლასისათვის საჭირო სხვა ველები/წევრები და მეთოდები. თუ შევცვლით წინაპარი/საბაზო კლასს, მაშინ მემკვიდრე კლასი შეწყვეტს მუშაობას. მემკვიდრე კლასს სხვანაირად შეიძლება ვუწოდოთ წინაპარი/საბაზო კლასის გაფართოება. თუ კლასი B არის A კლასის მემკვიდრე, მაშინ კლასს A-ს უწოდებენ წინაპარ/მშობელ კლასს, ხოლო კლასს B-ს კი უწოდებენ მემკვიდრე კლასს.

კლასს, რომელიც მემკვიდრეობით იღებს წინაპარი კლასის არსებულ წევრებს და მათ ახალ წევრებს უმატებს, მემკვიდრე კლასი ეწოდება. ამრიგად, მემკვიდრე კლასი მემკვიდრეობით იღებს წინაპარ კლასში განსაზღვრულ ყველა ცვლადს, მეთოდს, თვისებას, ინდექსატორს, მოვლენას და მათ უმატებს თავის საკუთარ წევრებს. მემკვიდრე კლასი მემკვიდრეობით არ იღებს წინაპარ კლასში განსაზღვრულ კონსტრუქტორს, დესტრუქტორს და სტატიკურ წევრს.

მემკვიდრე კლასის გამოცხადების სინტაქსია:

```
class მემკვიდრე_კლასის_სახელი : წინაპარი_კლასის_სახელი {  
    მემკვიდრე კლასის ტანი  
}
```

მემკვიდრე კლასის განსაზღვრისას მემკვიდრე კლასის სახელის შემდეგ ეთითება წინაპარი კლასის სახელი. მემკვიდრე კლასის სახელი წინაპარი კლასის სახელისაგან გამოყოფილია ორი წერტილით. თითოეული მემკვიდრე კლასისათვის შეგვიძლია მივუთითოთ მხოლოდ ერთი წინაპარი კლასი. ასეთი გზით შესაძლებელია მემკვიდრეობითობის იერარქიის შექმნა, რომელშიც მემკვიდრე კლასს თვითონ შეუძლია გახდეს წინაპარი სხვა კლასისთვის. არც ერთი კლასი არ შეიძლება იყოს თავისი თავის წინაპარი. მაგალითად:

// წინაპარი/საბაზო კლასი:

```
class Base {  
    // წევრები/ველები, კლასის მეთოდი }
```

//კლასი, რომელიც არის Base კლასის მემკვიდრე

```
class Derived : Base {
```

```
// წევრები/ველები, კლასის მეთოდი }
```

მოცემულ მაგალითში Base არის წინაპარი/საბაზო კლასი, ხოლო Derived არის კლასი, რომელიც მემკვიდრეობით იღებს Base კლასის შესაძლებლობებს.

კლასს Derived-ისათვის უშუალოდ ხელმისაწვდომია ყველა ის ელემენტი, რომლებსაც გააჩნია protected, public და internal წვდომის მოდიფიკატორი. თავის მხრივაც კლასი Derived შეიძლება იყოს წინაპარი/საბაზო კლასი სხვა ქვედა დონის კლასისათვის.

მემკვიდრეობითობის უპირატესობა ისაა, რომ წინაპარი/საბაზო კლასის შექმნის შემდეგ, სადაც განსაზღვრულია საერთო მახასიათებლები, ის შეგვიძლია გამოვიყენოთ ნებისმიერი რაოდენობის მემკვიდრე კლასების შესაქმნელად, რომლებიც დამატებით შეიცავს უნიკალურ მახასიათებლებს. შედეგად, მემკვიდრე კლასში აღარ ხდება საჭირო წინაპარ კლასში გამოცხადებული წევრების განმეორებით გამოცხადება.

მემკვიდრეობითობის დროს ძალა აქვს კლასის დახურულ წევრებთან მიმართებისას არსებულ შეზღუდვებს. მემკვიდრე კლასის მეთოდს არ შეუძლია მიმართოს წინაპარი კლასის დახურულ წევრებს. ამ პრობლემის გადაწყვეტის ერთერთი გზაა protected მოდიფიკატორის გამოყენება.

C# ენაში არსებობს კლასის დაცული წევრების შექმნის შესაძლებლობა. ისინი გახსნილია მხოლოდ მემკვიდრე კლასებისათვის და დახურულია სხვა კლასებისათვის. კლასის დაცული წევრი იქმნება protected მოდიფიკატორის საშუალებით.

მემკვიდრე კლასს წვდომა აქვს წინაპარ/საბაზო კლასის ისეთ ელემენტებთან, რომლებსაც აქვს წვდომის შემდეგი მოდიფიკატორები: private, protected, public, internal, protected internal.

წინაპარი/მემკვიდრე კლასის ის ელემენტები, რომელთაც აქვს წვდომის მოდიფიკატორი private, დახურულია და არ ჩანს მემკვიდრე კლასისათვის.

ქვემოთ მოცემულ მაგალითში ნაჩვენებია წვდომის მოდიფიკატორების ქცევა წინაპარ/საბაზო და მემკვიდრე კლასების ელემენტებთან მიმართებით.

```
class Base {
```

```
// წინაპარი/საბაზო კლასი, Base კლასის წევრები/ველები:
```

```
private int private_Item;
```

```
protected int protected_Item;
```

```

        internal int internal_Item;
        protected internal int protIntern_Item;
        public int public_Item; }
// კლასი Derived - არის Base კლასის მემკვიდრე
class Derived : Base {
    // მეთოდი, რომელიც ცვლის Base კლასის წევრებს
    void Method() {
        protected_Item = 10;
        internal_Item = 20;
        protIntern_Item = 30;
        public_Item = 40;
        //private_item = 50; შეცდომაა, ელემენტი მიუწვდომელია }
    }
private void button1_Click(object sender, EventArgs e) {
    // კლასის ეგზემპლარიზაცია წვდომა
    // 1. წინაპარი/საბაზო კლასი Base
        // 1.1. წინაპარი/საბაზო კლასის Base ეგზემპლარის/ობიექტის შექმნა
            Base bs = new Base();
        // 1.2. bs ეგზემპლარის წევრებთან წვდომა
        // bs.private_Item = 10; - წვდომა აკრძალულია
        // bs.protected_Item = 20; - წვდომა აკრძალულია
            bs.internal_Item = 30; // internal წვდომა დაშვებულია
            bs.public_Item = 40; // public - წვდომა დაშვებულია
        // protected internal - წვდომა დაშვებულია
            bs.protIntern_Item = 50;
    // 2. მემკვიდრე კლასი Derived - წინაპარი კლასის Base წევრებთან წვდომა
        // 2.1.მემკვიდრე კლასის Derived ეგზემპლარის/ობიექტის შექმნა
            Derived dr = new Derived();
        // dr.private_Item - private - წვდომა აკრძალულია
        // dr.protected_Item - protected - წვდომა აკრძალულია
        // internal - წვდომა დაშვებულია
            dr.internal_Item = 30;

```

```
// public - წვდომა დაშვებულია
    dr.public_Item = 40;
// protected internal - წვდომა დაშვებულია
    dr.protIntern_Item = 50; }
```

საინტერესოა შემდეგი საკითხიც: რამდენ მემკვიდრე კლასს შეუძლია ერთდროულად გახდეს საბაზო/წინაპარი კლასის მემკვიდრე? დაპროგრამების ენა C# არ უჭერს მხარს მრავალჯერად მემკვიდრეობას (C++-ისგან განსხვავებით). კონკრეტული კლასიდან შესაძლებელია მხოლოდ ერთი კლასის მემკვიდრეობა, მაგრამ, ნებისმიერი რაოდენობის ინტერფეისის მიღება.

ახლა კი განვიხილოთ წვდომის მოდიფიკატორების protected და protected internal-ის გამოყენების შესაძლებლობები. თუ პროგრამული კოდი იყენებს მემკვიდრეობითობას, მაშინ განსაკუთრებული ყურადღება ეთმობა წვდომის შემდეგ მოდიფიკატორებს protected და protected internal. თუ კლასის ელემენტი/წევრი (ველი, მეთოდი, თვისება და ა.შ.) აღწერილია წვდომის მოდიფიკატორით protected, მაშინ მასთან მიმართებით უნდა ვიცოდეთ შემდეგი წესები:

1. ელემენტი/წევრი ხელმისაწვდომია როგორც იმ კლასის ფარგლებში, რომელშიც გამოცხადებულია, ასევე მემკვიდრეობით მიღებულ კლასებშიც;
2. ელემენტი/წევრი მიუწვდომელია კლასის ეგზემპლარიდან/ობიექტიდან.

წვდომის მოდიფიკატორი protected internal აერთიანებს მოდიფიკატორების protected-ის და internal-ის შეზღუდვებს.

protected internal-ის შემთხვევაში განვიხილავთ ორ ვარიანტს:

პირველი ვარიანტია, როდესაც protected internal ელემენტის/წევრის მქონე კლასი და ამ კლასის ეგზემპლარი/ობიექტი ერთ ნაკრებში/აპლიკაციაში იმყოფება. ასეთ შემთხვევაში კლასის ეგზემპლარიდან წვდომა protected internal ელემენტზე დაშვებულია (საკვანძო სიტყვის გაფართოება internal). ასევე დასაშვებია წვდომა მემკვიდრე კლასიდან (საკვანძო სიტყვის გაფართოება protected).

მეორე ვარიანტია, როდესაც კლასი protected internal ელემენტით, გამოცხადებულია ერთ ნაკრებში/აპლიკაციაში, ხოლო ამ კლასის ეგზემპლარი იქმნება სხვა ნაკრებში/აპლიკაციაში. ასეთ შემთხვევაში ეგზემპლარს არ აქვს წვდომა protected internal ელემენტზე (შეზღუდვა internal), მაგრამ შესაძლებელია შევქმნათ

მემკვიდრე კლასი და სწორედ მემკვიდრე კლასიდან განვახორციელოთ წვდომა protected internal ელემენტზე (protected გაფართოება).

განვიხილოთ მაგალითი, სადაც ხდება წვდომის მოდიფიკატორების protected და protected internal გამოყენება სამ კლასთან (A, B და C) მიმართებით:

```
//წინაპარი/საბაზო კლასი A
class A {
//A კლასის დაცული წევრები/ველები
    protected int a;
    protected internal int aa; }

//მემკვიდრე კლასი B
class B : A { //B კლასის დაცული წევრები/ველები
    protected int b;
    protected internal int bb;
    public void Method() {
// წვდომა დაშვებულია A კლასის protected და protected internal წევრებთან
        a = 25;        aa = 30;
    }
}

// მემკვიდრე კლასი C
class C : B {
// C კლასის დაცული წევრები/ველები
    protected int c;
    protected internal int cc;
    public void Method() {
//წვდომა დაშვებულია A და B კლასის,protected და protected internal წევრებთან
        a = 10;
        aa = 20;
        b = 30;
        bb = 40;
    }
}

private void button1_Click(object sender, EventArgs e) {
// კლასის ეგზემპლარებიდან არ არის წვდომა protected ელემენტებზე
```

```

A objA = new A(); // A კლასის ობიექტი/ეგზემპლარი
B objB = new B();
C objC = new C();
// objA.a = 30; - წვდომა აკრძალულია
// objB.a = 20; - წვდომა აკრძალულია
// objB.b = 20; - წვდომა აკრძალულია
// objC.a = 20; - წვდომა აკრძალულია
// objC.b = 20; - წვდომა აკრძალულია
// objC.c = 20; - წვდომა აკრძალულია
//წვდომა შესაძლებელია A,B,C კლასის ეგზემპლარის protected internal წევრებთან
//რადგან A,B,C კლასები და objA, objB, objC ეგზემპლარები აღწერილია ერთ ნაკრებში
    objA.aa = 40;
    objB.aa = 50;
    objB.bb = 50;
    objC.aa = 10;
    objC.bb = 20;
    objC.cc = 30;
}

```

კონსტრუქტორები და მემკვიდრეობითობა. base საკვანძო სიტყვა. განვიხილოთ იერარქიული წყობის კლასების კონსტრუქტორების გამოძახების რიგითობა. კლასის ობიექტის (ეგზემპლარის) შექმნის დროს პირველად გამოიძახება სწორედ ამ კლასის კონსტრუქტორი. თუ კლასი B არის კლასი A-ს მემკვიდრე, მაშინ ჯერ გამოიძახება წინაპარი/საბაზო კლასის A-ს კონსტრუქტორი, ხოლო შემდეგ კი გამოიძახება მემკვიდრე კლასის B-ს კონსტრუქტორი.

თუ კლასები წარმოადგენენ მემკვიდრეობის რამდენიმე დონიან იერარქიას, მაშინ პირველად გამოიძახებული იქნებიან მაღალი/ზედა დონის კლასის კონსტრუქტორები ჯაჭვური ბმის სახით, ქვედა დონის კონსტრუქტორების გამოიძახებამდე (იხ. ცხრილი 10).

მე-10 ცხრილში მოცემულია სამი კლასის (A, B, C) კონსტრუქტორების გამოძახების თანმიმდევრობა მემკვიდრეობითობის არსებობის პირობებში. პირველ რიგში ხდება წინაპარი/საბაზო კლასის A-ს კონსტრუქტორის გამოძახება. მეორე რიგში კი აუცილებლად მოხდება კლასი B-ს კონსტრუქტორის გამოძახება,

რომელიც იერარქიის შუა დონეზე იმყოფება. ბოლოს კი მოხდება კლასი C-ს კონსტრუქტორის გამოძახება, რომელიც იერარქიის ქვედა დონეზეა.

მე-10 ცხრილში ნაჩვენები კონსტრუქტორების გამოძახების რიგითობა ლოგიკურად სწორია. ამის მიზეზი ისაა, რომ მემკვიდრე კლასებს უფრო სპეციფიკური ხასიათი აქვთ, ვიდრე წინაპარ/საბაზო კლასს. სპეციფიკური ნაწილი თითქოს გადაეფარება კოდის საერთო ნაწილს. ამგვარად, მემკვიდრე კლასის კონსტრუქტორს შეუძლია გამოიყენოს საბაზისო კლასის კონსტრუქტორის მემკვიდრეობით მიღებული თვისებები.

ისეთი შემთხვევების დროს, როდესაც კლასები იერარქიას ქმნის, მაშინ მემკვიდრე კლასი მიმართავს საბაზო/წინაპარი კლასის ელემენტს/წევრს საკვანძო სიტყვა base-ის დახმარებით. თუ საბაზო/წინაპარ კლასში არის დამალული ელემენტები (გამოცხადებული private წვდომის მოდიფიკატორით), ამ ელემენტებზე წვდომა შეუძლებელია საკვანძო სიტყვის base-ის გამოყენებით.

საკვანძო სიტყვა base მნიშვნელოვანია იმ შემთხვევებში, როდესაც კლასებს შორის მემკვიდრეობის თანაფარდობა გამოიყენება. ასევე შესაძლებელია შეიქმნას ისეთი სიტუაცია, რომ საბაზო/წინაპარ და მემკვიდრე კლასებში არსებულ მეთოდებს (ელემენტებს) ჰქონდეთ ერთი და იგივე სახელები და ხელმოწერები. ასეთ შემთხვევაში, მემკვიდრე კლასის ელემენტი ფარავს/მაღავს საბაზო/წინაპარი კლასის შესაბამის ელემენტს. იმისათვის, რომ მემკვიდრე კლასიდან წვდომა გვქონდეს წინაპარ/საბაზო კლასის ელემენტთან, უნდა გამოვიყენოთ საკვანძო სიტყვა base.

ცხრილი 10.

კონსტრუქტორების გამოძახების რიგითობა მემკვიდრეობითობასთან მიმართებით

A, B, C კლასების იერარქია	ობიექტის გამოყენება, კონსტრუქტორის გამოძახება
<pre>class A { // A კლასის კონსტრუქტორი public A() { } }</pre>	<pre>// A კლასის ეგზემპლარის/ობიექტის შექმნა // A კლასის კონსტრუქტორის გამოძახება A objA = new A();</pre>
<pre>class B : A { // B კლასის კონსტრუქტორი public B() { } }</pre>	<pre>// B კლასის ეგზემპლარის/ობიექტის შექმნა // კონსტრუქტორების გამოძახების // თანამიმდევრობა A() -> B() B objB = new B();</pre>

<pre>class C : B { // C კლასის კონსტრუქტორი public C() { } }</pre>	<pre>// C კლასის ეგზემპლარის/ობიექტის შექმნა // კონსტრუქტორების გამოძახების // თანამიმდევრობა A() -> B() -> C() C objC = new C();</pre>

საკვანძო სიტყვა `base` გამოიყენება მემკვიდრე კლასებში შესაძლებელია შემდეგი სიტუაციების დროს:

- როდესაც მემკვიდრე კლასის კონსტრუქტორიდან უნდა გამოვიძახოთ წინაპარი/საბაზო კლასის კონსტრუქტორი;
- როდესაც მემკვიდრე კლასის მეთოდიდან საჭიროა გამოვიძახოთ წინაპარი/საბაზო კლასის მეთოდი, თუ ეს მეთოდი განისაზღვრება თავიდან;
- როდესაც მემკვიდრე კლასის მეთოდიდან უნდა განვახორციელოთ წვდომა წინაპარი/საბაზო კლასის ელემენტებზე. ამ შემთხვევაში ელემენტი შეიძლება გამოცხადდეს ნებისმიერი წვდომის მოდიფიკატორით `private`-ის გარდა;
- როდესაც მემკვიდრე კლასის მეთოდიდან უნდა განვახორციელოთ წვდომა წინაპარი/საბაზო კლასის თვისებაზე. თვისება არ შეიძლება გამოცხადებული იყოს წვდომის მოდიფიკატორით `private`;
- თუ საჭიროა ხაზგასმით აღვნიშნოთ, რომ მემკვიდრე კლასის მეთოდში გამოყენებულია ძირითადი კლასის ელემენტები.

იმისათვის რომ მემკვიდრე კლასმა გამოიძახოს საბაზო/წინაპარი კლასის კონსტრუქტორი უნდა გამოვიყენოთ საკვანძო სიტყვა `base`. საბაზო/წინაპარი კლასის გამოძახების კოდი იწერება მემკვიდრე კლასის კონსტრუქტორის დასახელების და ორი წერტილის აკრეფის შემდეგ:

A (პარამეტრების სია) : `base` (წინაპარი კლასის კონსტრუქტორისთვის გადასაცემი

პარამეტრები)

{ `// მემკვიდრე კლასის კონსტრუქტორის ტანი` }

საკვანძო სიტყვა `base` მიმართავს საბაზო/წინაპარ კლასს. თუ მემკვიდრე კლასში კონსტრუქტორი არსებობს ჩუმათობის პრინციპის მიხედვით, მაშინ წინაპარი/საბაზო კლასის კონსტრუქტორის გამოძახებაც ხდება ჩუმათობის პრინციპის მიხედვით ანუ შეგვიძლია იგი არ დავწეროთ. თუ კონსტრუქტორი არ არის დაწერილი ჩუმათობის პრინციპის მიხედვით, მაშინ უნდა მოხდეს საბაზო/წინაპარი კლასის კონსტრუქტორის აშკარად აღწერა:

```

class მომღერალი {
    protected მომღერალი(string name) { }
}
class კომენტარიმომღერალი : მომღერალი {
    //დაფიქსირდება შეცდომა კომპილაციისას
    public კომენტარიმომღერალი(string name) { }
}

```

ზემოთ მოცემულ მაგალითში კომპილაციისას დაფიქსირდება შეცდომა, რადგან მოხდება კონსტრუქტორის გამოძახება ჩუმათობის პრინციპის მიხედვით, მაგრამ იგი არ არის კლასში „მომღერალი“. კოდის სწორი ვარიანტი გამოისახება შემდეგი სახით:

```

class მომღერალი {
    protected მომღერალი(string name) { } }
class კომენტარიმომღერალი : მომღერალი {
    public კომენტარიმომღერალი(string name) : base(name) { }
}

```

თუ წინაპარი/საბაზო კლასის კონსტრუქტორი არის private, მაშინ ვერ შევქმნით მემკვიდრე კლასის კონსტრუქტორს. მაგალითად:

```

class შეუძლებელი {
    private შეუძლებელი() { }
}
class შესაძლებელი : შეუძლებელი {
    //დაფიქსირდა შეცდომა, ვერ ვქმნით კონსტრუქტორს
    public შესაძლებელი() { }
}

```

ახლა კი განვიხილოთ მაგალითი, სადაც აღწერილია წინაპარი კლასის კონსტრუქტორის გამოძახება საკვანძო სიტყვა base-ის გამოყენებით. ვიყენებთ სამი კლასის A-ს, B-ს და C-ს იერარქიას:

```

class A // წინაპარი კლასი A {
    public int a;
    // A კლასის კონსტრუქტორი ერთი პარამეტრით
    public A(int _a) { a = _a;
}

```

```

        MessageBox.Show("A კლასის კონსტრუქტორი ერთი პარამეტრით A(int) = " + a);
    }
}

// B კლასი - A კლასის მემკვიდრე
class B : A {
    public int b;
    // B კლასის კონსტრუქტორი ერთი პარამეტრით
    // არაცხადად იძახებს A კლასის კონსტრუქტორს A()
    public B(int _b) : base(0) { b = _b;
        MessageBox.Show("B კლასის კონსტრუქტორი ერთი პარამეტრით B(int) = " + b);
    }
}

// C კლასი - B კლასის მემკვიდრე
class C : B {
    public int c;
    public C(int _c) : base(0) {
        c = _c;
        MessageBox.Show("C კლასის კონსტრუქტორი ერთი პარამეტრით C(int) = " + c);
    }
}

private void button1_Click(object sender, EventArgs e) {
    // 1.objB ობიექტის შექმნისას კონსტრუქტორის გამოძახების რიგითობა: A(int)->B(int)
    B objB = new B(4);
    //2.objC ობიექტის შექმნისას კონსტრუქტორის გამოძახების რიგითობა: A(int)->B(int)->C(int)
    C objC = new C(5);
}

```

ცვლადების დამალვა მემკვიდრეობითობის დროს. მემკვიდრე კლასში შეგიძლიათ განსაზღვროთ იგივე სახელდების წევრი, როგორი სახელითაც არის იგი განსაზღვრული საბაზო კლასში. ამ შემთხვევაში, საბაზო/წინაპარი კლასის წევრი იმალება მემკვიდრე კლასისათვის. მიუხედავად იმისა, რომ ტექნიკურად ეს არ ითვლება შეცდომად C#-ში, კომპილატორი მაინც გასცემს შეტყობინებას, შეცდომის შესახებ.

თუ გსურთ წინაპარი კლასის წევრი განზრახ დამალოთ, უნდა გამოიყენოთ new საკვანძო სიტყვა, რომ თავიდან აიცილოთ გამაფრთხილებელი შეტყობინების გამოჩენა.

ამასთან, გაითვალისწინეთ, რომ ეს არის new საკვანძო სიტყვის სრულიად განსხვავებული გამოყენება და მას ობიექტის ეგზემპლარის შექმნასთან საერთო არაფერი აქვს.

კლასების იერარქიაში კონსტრუქტორები გამოიძახება იმ რიგითობით, რა რიგითობითაც სრულდება მემკვიდრეობითობა - წინაპარი კლასიდან მემკვიდრე კლასისკენ. უფრო მეტიც, ეს რიგითობა რჩება უცვლელი იმისგან დამოუკიდებლად, გამოიყენება თუ არა base საკვანძო სიტყვა. თუ ეს საკვანძო სიტყვა არ არის მითითებული, მაშინ გამოიძახება თითოეული წინაპარი კლასის ავტომატური კონსტრუქტორი. base საკვანძო სიტყვა ყოველთვის მიმართავს უახლოესი წინაპარი კლასის კონსტრუქტორს.

განვიხილოთ რამდენიმე ამოცანა.

ამოცანა 9.1. მემკვიდრეობითობა. მარტივი მაგალითი.

```
class Book {
    public string Name;
    public string GetInfo()
    {
        return "წიგნის დასახელება: " + Name;
    }
}

class ShowBook : Book {
    public void Show() {
        //GetInfo() მეთოდის გამოძახება Book საბაზო კლასიდან
        string info = GetInfo();
        MessageBox.Show(info); }
}

private void button1_Click(object sender, EventArgs e){
    ShowBook objbook = new ShowBook();
    // კლასის ობიექტის მნიშვნელობის განსაზღვრა
    objbook.Name = "Visual C#.Net";
    // GetInfo() მეთოდის გამოძახება
    string str2 = objbook.GetInfo();
    label1.Text = str2;
    objbook.Show(); // Show() მეთოდის გამოძახება
```

```

    }

private void button2_Click(object sender, EventArgs e) {
    Close(); // ფორმის დახურვა
    //Application.Exit(); // აპლიკაციის დახურვა }

ამოცანა 9.2. მემკვიდრეობითობის დემონსტრირების მეორე მაგალითი.
class TwoDshape {
    public double Width;
    public double Height;
    public void Showdim() {
        MessageBox.Show("სიგანე = " + Width + "\nსიმაღლე = " + Height); }
    }

    class Rectangle : TwoDshape {
        public string Style;
        // ოთხკუთხედის ფართობის გამოთვლა
        public double Area() {
            return Width * Height; }
        // ოთხკუთხედის ტიპის დადგენა
        public void ShowStyle(){
            MessageBox.Show("ოთხკუთხედის ტიპი: " + Style); }
        }

private void button1_Click(object sender, EventArgs e) {
    Rectangle t1 = new Rectangle();
    Rectangle t2 = new Rectangle();
    t1.Width = 4.0;
    t1.Height = 4.0;
    t1.Style = "კვადრატი";
    t1.ShowStyle();
    t1.Showdim();

    label1.Text="ფიგურა: " + t1.Style + ". ფართობი="+ t1.Area().ToString();

    t2.Width = 8.0;
    t2.Height = 12.0;
    t2.Style = "მართკუთხედი";
    t2.ShowStyle();

```

```

        t2.Showdim();
label2.Text="ფიგურა:"+t2.Style+" .ფართობი="+t2.Area().ToString();}
private void button2_Click(object sender, EventArgs e)
    { Close(); // ფორმის დახურვა
      //Application.Exit(); // აპლიკაციის დახურვა }

```

ამოცანა 9.3. protected მოდიფიკატორის გამოყენების დემონსტრირება.

```

class B {
    protected int i, j;
    public void Set(int a, int b) {
        i = a;
        j = b;
    }
    public int Show() {
        return i + j;
    }
}
class D : B{
    int k; //დახურული წევრი
    // i და j წევრები გახსნილია D კლასისათვის
    public void Setk() {
        k = i * j;
    }
    public int Showk() {
        return k;
    }
}
private void button1_Click(object sender, EventArgs e){
    D obj = new D();
    obj.Set(7, 3);
label1.Text="Show() მეთოდის გამოძახება: "+obj.Show().ToString();
    obj.Setk();
label2.Text="Showk() მეთოდის გამოძახება: "+obj.Showk().ToString(); }
private void button2_Click(object sender, EventArgs e)

```

```

        { Close(); // ფორმის დახურვა
          //Application.Exit(); // აპლიკაციის დახურვა }

```

ამოცანა 9.4. კონსტრუქტორის გამოყენება მემკვიდრე კლასში.

```

class Student {
const string ADDR = "http:\\gtu.ge";
    public string level, inlevel;
    public int numberSt;
public void InfoPN() {
    MessageBox .Show( "ვებ-მისამართი: " + ADDR + " \იგანყოფილება: " +
    level + "\ითემა: " + inlevel + " \ისტატიების რაოდენობა: " +
    numberSt.ToString()); }
}
class CSharp : Student {
public string st;
public CSharp(string level, string inlevel, int numberSt, string st)
{
    this .level = level;
    this.inlevel = inlevel;
    this.numberSt = numberSt;
    this.st = st;
}
public void StWrite() {
    MessageBox .Show( " სტატია: " + st) ;
}
}
private void button1_Click(object sender, EventArgs e){
    CSharp obj = new CSharp("C#", "კლასები და ობიექტები", 7, " მემკვიდრეობითობა");
    obj.InfoPN();
    obj.StWrite();
}
private void button2_Click(object sender, EventArgs e)
{ Close(); // ფორმის დახურვა
  //Application.Exit(); // აპლიკაციის დახურვა }

```


ამოცანა 9.5. საკვანძო სიტყვის base-ის გამოყენება. კონსტრუქტორების მემკვიდრეობითობის დემონსტრირება.

```
class MyClass{
    public int x, y, z ;
    public MyClass(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }
}

class ClassA : MyClass {
    int point;
public ClassA (int point, int x, int y, int z) : base (x, y, z){
    this.point = point;
}

    public int Pointer(ClassA obj){
        obj.x *= obj.point;
        obj.y *= obj.point;
        obj.z *= obj.point;
        return (obj.x + obj.y + obj.z);
    }
}

private void button1_Click(object sender, EventArgs e){
    ClassA obj = new ClassA(10, 1, 4, 3);
    label1.Text = "ობიექტის კოორდინატებია: " + obj.y + " " + obj.z;
    int shedegi=obj.Pointer(obj);
    label2.Text = "ობიექტის ახალი კოორდინატებია: " + obj.x + " " +
    obj.y + " " + obj.z;
}

private void button2_Click(object sender, EventArgs e)
{
    Close(); // ფორმის დახურვა
    //Application.Exit(); // აპლიკაციის დახურვა }
}
```

დამოუკიდებელი სამუშაო

1. მოცემულობა დაწერეთ დამოუკიდებლად. პროგრამას დაურთეთ კომენტარები და აღწერეთ სტრიქონები. ასევე ახსენით შესრულების შედეგი.

```
class MaxMethods // .....
{
    public int Max(int a, int b) // .....
    {
        if (a < b) return b;
        return a;
    }
    public int Max(int a, int b, int c) // .....
    {
        int max = a;
        if (max < b) max = b;
        if (max < c) max = c;
        return max;
    }
    public int Max(int a, int b, int c, int d) // .....
    {
        int max = a;
        if (max < b) max = b;
        if (max < c) max = c;
        if (max < d) max = d;
        return max;
    }
}

private void button1_Click(object sender, EventArgs e)
{
    MaxMethods mm = new MaxMethods();
    int max;
    max = mm.Max(5, 6);
    label1.Text = max.ToString(); // .....
    max = mm.Max(1, -3, 8);
    label2.Text = max.ToString(); // .....
```

```

        max = mm.Max(2, 3, 5, 1);
        label3.Text = max.ToString(); // .....
    }

```

2. მოცემულია პროგრამული კოდი. კოდში არის შეცდომა. იპოვეთ იგი და გაასწორეთ. წარმოადგინეთ კოდის შესწორებული ვარიანტი.

```

class A {
    private int a = 5;
    private int retA() { return a; }
    public int geta() { return a; }
    public int usePrivate() { return retA(); }
}

class B : A { }

private void button1_Click(object sender, EventArgs e)
{
    B b = new B();
    int varb = b.geta();
    label1.Text= varb.ToString();
    varb = b.retA();
    varb = b.usePrivate();
    label2.Text= varb.ToString();
}

```

3. მოცემულია პროგრამული კოდის ფრაგმენტი. აღწერეთ თითოეული სტრიქონი.

```

class Rectangle : TwoDShape
{
    public bool IsSquare()
    {
        if (Width == Height) return true;
        return false;
    }
    public double Area()
    {
        return Width * Height;
    }
}

```

```
}
```

4. მოცემულ ფრაგმენტში კლასი `Rectangle`-ის კომპილაცია არ მოხდება. აღწერეთ რატომ? მიუთითეთ კომენტარის სახით, თუ რომელი სტრიქონია შეცდომით აღწერილი.

```
class TwoDShape {
    double Width;
    double Height;
    public void ShowDim(){
        MessageBox.Show("სიგანე= " + Width + "\nსიმაღლე= " + Height); }
}

class Rectangle : TwoDShape {
    public string Style;
    public double Area()
    {
        return Width * Height;
    }
    public void ShowStyle() {
        MessageBox.Show("ოთხკუთხედის ტიპი: " + Style);
    }
}
```

5. ქვემოთ მოცემულია პროგრამული კოდი, რომელიც აღწერს ინჟინერებისათვის დატვირთვის განაწილებას. აღწერეთ კომენტარების ველი და ააგეთ შესაბამისი ფორმა. ასევე უპასუხეთ ქვემოთ მოცემულ შეკითხვებს:

1. რა მოქმედებას ასრულებს მემკვიდრე კლასის `CivilEngineer` კონსტრუქტორი?
2. როგორ ხდება დატვირთვის გამოთვლა ინჟინერებისათვის?
3. შეიძლება თუ არა კონსტრუქტორის მემკვიდრეობით გადაცემა?

```
class Engineer {
    private string name;
    protected float billingRate;
    public Engineer(string name, float billingRate) // .....
    {
        this.name = name;
        this.billingRate = billingRate; }
}
```

```

        public float CalculateCharge(float hours) // .....
        { return(hours * billingRate); }
        public string TypeName() // .....
        { return("Engineer"); }
    }

class CivilEngineer: Engineer {
public CivilEngineer(string name, float billingRate):
                                base(name, billingRate){ }

    public new float CalculateCharge(float hours) // .....
    {
        if (hours < 1.0F)
            hours = 1.0F; // .....
        return(hours * billingRate);
    }
    public new string TypeName() // .....
    {
        return ("Civil Engineer");
    }
}

private void button1_Click(object sender, EventArgs e) {
    Engineer eng = new Engineer("George", 35.50F);
    CivilEngineer ceng = new CivilEngineer("Tom", 50F);
    label1.Text=eng.TypeName().ToString();
    label2.Text=eng.CalculateCharge(2F).ToString();
    label3.Text=ceng.TypeName().ToString();
    label4.Text=ceng.CalculateCharge(0.95F).ToString();
}

```

6. მოცემულია პროგრამული კოდის ფრაგმენტი. აღწერეთ თითოეული სტრუქტურული კომენტარის სახით და დაასრულეთ პროგრამული კოდი.

```

class A {
    public int i = 0;
}

class B : A // .....
{

```

10

```

new int i; // .....
public B (int b)
{
    i = b; // .....
}

public void Show() {
    MessageBox.Show("ცვლადი i მემკვიდრე კლასში = " + i);
}
}

```

სტატიკური კომპონენტები: static მოდიფიკატორი, მუდმივები. შემთხვევითი რიცხვების გენერატორი

static მოდიფიკატორი. ჩვეულებრივ, კლასის წევრთან მიმართება ხორციელდება ობიექტის გამოყენებით, მაგრამ ამასთანავე შესაძლებელია მიმართება ობიექტის შექმნის გარეშე. კლასის ასეთ წევრებს სტატიკური წევრები ეწოდებათ და მათი გამოცხადებისათვის გამოიყენება static სიტყვა. მოკლედ რომ ვთქვათ, C# ენაში მოდიფიკატორი static გამოიყენება ისეთი ელემენტების/წევრების შესაქმნელად, რომლებიც შეიძლება იყოს საერთო კლასის ყველა ეგზემპლარისათვის/ობიექტისათვის და არ საჭიროებს ეგზემპლარის/ობიექტის შექმნას მათ გამოსაყენებლად. სტატიკური ელემენტების დახმარებით შესაძლებელია მეხსიერების ოპტიმიზაცია და კოდის გამარტივება.

კლასის სტატიკურ წევრთან მიმართვისათვის უნდა მივუთითოთ: კლასის სახელი, ოპერატორი წერტილი (.) და წევრის სახელი. სტატიკურ მეთოდთან მიმართვაც ანალოგიურად სრულდება. სტატიკური მეთოდი ჩვეულებრივი მეთოდისაგან იმით განსხვავდება, რომ ის შეიძლება გამოძახებული იყოს კლასის სახელის გამოყენებით ამ კლასის რაიმე ობიექტის შექმნის გარეშე. თუ კლასის წევრი განისაზღვრება static მოდიფიკატორით, მაშინ ის მისაწვდომი ხდება ამ კლასის ობიექტების შექმნამდე. სტატიკური შეიძლება იყოს: ცვლადები, მეთოდები, თვისებები და თავად კლასებიც.

სტატიკური ველები არის ცვლადები, რომლებიც საერთოა კლასის ყველა ობიექტისათვის. ისინი გამოცხადებულია საკვანძო სიტყვით `static`. სტატიკური ველის მნიშვნელობა შენარჩუნებულია გამოძახებებს შორის და ხელმისაწვდომია კლასის ყველა ეგზემპლარის/ობიექტისათვის.

სტატიკური ცვლადი ფაქტიურად გლობალური ცვლადია. მისი კლასის ობიექტების გამოცხადებისას არ იქმნება ცვლადის ასლი. სამაგიეროდ კლასის ყველა ობიექტი ერთობლივად იყენებს სტატიკურ ცვლადს, რომლის ინიციალიზება ხდება მისი კლასის ჩატვირთვისას. თუ საწყისი მნიშვნელობა არ არის აშკარად მითითებული, ჩუმათობის პრინციპის მიხედვით ასეთ ცვლადს ენიჭება მნიშვნელობა 0 (რიცხვითი ტიპის ცვლადისთვის), `null` მნიშვნელობა (მიმართვითი ტიპის ცვლადისთვის) ან `false` მნიშვნელობა (`bool` ტიპის ცვლადისთვის). ამრიგად, `static` მოდიფიკატორიან ცვლადს ყოველთვის აქვს რაიმე მნიშვნელობა. როგორც ვხედავთ სტატიკური ცვლადის ინიციალიზება ხდება მისი კლასის რაიმე ობიექტის შექმნის გარეშე.

სტატიკური მეთოდები არის ისეთი მეთოდები, რომელთა გამოძახება შესაძლებელია კლასის ეგზემპლარის/ობიექტის შექმნის გარეშე. ისინი გამოცხადებულია საკვანძო სიტყვით `static` და არ შეუძლიათ კლასის არასტატიკური ველებთან ან მეთოდებთან წვდომა.

სტატიკურ მეთოდებთან მიმართებით უნდა გავითვალისწინოთ შემდეგი შეზღუდვები:

- არ აქვთ `this` მიმართვა;
- შეუძლიათ უშუალოდ გამოიძახონ მხოლოდ სტატიკური მეთოდები და არ შეუძლიათ თავიანთი კლასის ობიექტის მეთოდები;
- შეუძლიათ პირდაპირ მიმართონ მხოლოდ სტატიკურ მონაცემებს;
- სტატიკურ მეთოდს მხოლოდ თავისი კლასის ობიექტის გამოყენებით შეუძლია მიმართოს თავისი კლასის ობიექტის მეთოდებს და ცვლადებს (ე.ი. ობიექტის ცვლადთან ან მეთოდთან მიმართვისათვის საჭიროა ამ ობიექტის სახელის მითითება).

სტატიკური თვისებები არის ისეთი სახის თვისებები, რომლებიც ხელმისაწვდომია კლასის ეგზემპლარის/ობიექტის შექმნის გარეშე. ისინი გამოცხადებულია საკვანძო სიტყვით `static`. სტატიკური თვისებები შეიძლება გამოყენებულ იქნას

სტატიკურ ცვლადებზე წვდომისთვის, ასევე გამოიყენება გლობალურ მონაცემებზე წვდომის მისაღებად, რომელიც ხელმისაწვდომი პროგრამის სხვადასხვა ადგილიდან.

სტატიკური კონსტრუქტორი არის ისეთი სახის სპეციალური მეთოდი, რომელის გამოძახება ხდება ავტომატურად ერთჯერ, კლასზე პირველივე მიმართვისას. იგი გამოცხადებულია static საკვანძო სიტყვით და არ შეიძლება ჰქონდეს პარამეტრები.

სტატიკური კლასი არის ისეთი სახის კლასი, რომელსაც აქვს მხოლოდ სტატიკური ელემენტები და ის არ შეიძლება იყოს სხვა კლასიდან მიღებული/წარმოებული. სტატიკური კლასები შეიძლება შეიცავდეს მხოლოდ სტატიკურ ველებს, მეთოდებს, თვისებებს და მოვლენებს.

სტატიკურ კლასს აქვს ორი ძირითადი თვისება: 1. სტატიკური კლასის ობიექტების შექმნა შეუძლებელია; 2. სტატიკური კლასი უნდა შეიცავდეს მხოლოდ სტატიკურ წევრებს. სტატიკური კლასები ძირითადად გამოიყენება ორ შემთხვევაში: პირველი - გაფართოების მეთოდის შექმნისას საჭიროა სტატიკური კლასი. გაფართოების მეთოდები ძირითადად ასოცირდება LINQ ენასთან; მეორე - ერთმანეთთან დაკავშირებული სტატიკური მეთოდების კოლექციის შესანახად.

ახლა კი განვიხილოთ მაგალითი, სადაც აღწერილია სტატიკური ცვლადი და მეთოდი:

```
public class ოთახი {  
    // თუ დაწეროთ "public int count;", მაშინ ეს ველი ექნება ნებისმიერ ობიექტს  
    // თუ არ შეიქმნება არცერთი ობიექტი, ეს ველი არც კი იარსებებს  
    // ამდენად ველი გავხადოთ სტატიკური  
    public static int count;  
    public ოთახი() // კონსტრუქტორი  
    {    count++;    }  
    // სტატიკური მეთოდი, რომელიც გამოიტანს შექმნილი ობიექტების რაოდენობას  
    public static void ობიექტების_რაოდენობა() {  
        MessageBox.Show("ობიექტების რაოდენობა = "+ოთახი.count.ToString());  
    }  
    private void button1_Click(object sender, EventArgs e){  
        // ობიექტების რაოდენობა ობიექტის შექმნის გარეშე  
        // 0-ის ტოლია, რადგან არ შეგვიქმნია არც ერთი ობიექტი
```



```

ოთახი.ობიექტების_რაოდენობა();

// შევექმნათ 3 ოთახი
ოთახი ოთახი1 = new ოთახი();
ოთახი ოთახი2 = new ოთახი();
ოთახი ოთახი3 = new ოთახი();

ოთახი.ობიექტების_რაოდენობა(); // გამოიტანს 3-ს
}

```

როგორც ვხედავთ, სტატიკური მეთოდი გვაძლევს საშუალებას, გამოვიძახოთ ისეთი მეთოდი, რომელსაც არ გააჩნია არც ერთი ობიექტი. ობიექტის სახელის ნაცვლად პირდაპირ გამოვიყენეთ კლასის სახელი:

```

ოთახი.ობიექტების_რაოდენობა();

```

ამდენად სტატიკური მეთოდის გამოსაძახებლად არ არის საჭირო ცალკე ობიექტის შექმნა. ახლა კი განვიხილოთ მაგალითი, სადაც აღწერილია სტატიკური კონსტრუქტორი:

```

public class MyClass{
    public static int a;
    public int b;
    // სტატიკური კონსტრუქტორი
    static MyClass(){
        a = 77;
    }
    // ჩვეულებრივი კონსტრუქტორი
    public MyClass() {
        b = 999;
    }
}

private void button1_Click(object sender, EventArgs e){
    label1.Text = "წვდომა კლასის ეგზემპლართან a-სთან = " + MyClass.a;
    MyClass obj = new MyClass();
    label2.Text = "წვდომა კლასის ეგზემპლართან b-სთან = " + obj.b;
}

```

როგორც ვხედავთ, სტატიკური კონსტრუქტორის გამოძახება ხდება ავტომატურად კლასის პირველივე გამოძახებისთანავე. აქედან შეიძლება დავასკვნათ: სტატიკური კონსტრუქტორი უნდა შესრულდეს ნებისმიერი ეგზემპლარის კონსტრუქტორის გამოძახებამდე უფრო ადრე. უფრო მეტიც, სტატიკურ კონსტრუქტორებს არ აქვთ წვდომის მოდიფიკატორები. ისინი იყენებენ წვდომას ჩუმათობის პრინციპის მიხედვით, შესაბამისად, მათი გამოძახება პროგრამიდან შეუძლებელია.

ახლა კი გადავიდეთ სტატიკური კლასის კოდის აღწერაზე:

```
static class MyMath{
    // რიცხვის მთელი ნაწილი
    static public int round(double d) {
        return (int)d;
    }
    // რიცხვის წილადი ნაწილი
    static public double doub(double d){
        return d - (int)d;
    }
    // რიცხვის კვადრატი
    static public double sqr(double d){
        return d * d;
    }
    // კვადრატული ფესვი რიცხვიდან
    static public double sqrt(double d){
        return Math.Sqrt(d);
    }
}

private void button1_Click(object sender, EventArgs e){
    label1.Text = "თავდაპირველი რიცხვი: 11.44 " + "\n";
    label1.Text+="რიცხვის მთელი ნაწილი=" + MyMath.round(11.44) + "\n";
    label1.Text+="რიცხვის წილადი ნაწილი=" + MyMath.doub(11.44) + "\n";
    label1.Text+="რიცხვის კვადრატი = " + MyMath.sqr(11.44) + "\n";
    label1.Text+="რიცხვის კვადრატული ფესვი="+MyMath.sqrt(11.44)+"\n";
}
```

}

მუდმივები. როგორც ვიცით ცვლადების მნიშვნელობები ყოველთვის შეგვიძლია შევცვალოთ, მაგრამ C# ენაში შეგვიძლია შემოვიტანოთ ისეთი ცვლადები რომელთა მნიშვნელობაც შეიძლება არ შეიცვალოს და დარჩეს მუდმივ მნიშვნელობად. ასეთ ცვლადებს უწოდებენ მუდმივებს.

მუდმივას გამოცხადებისათვის გამოიყენება `const` საკვანძო სიტყვა. კლასი შეიძლება შეიცავდეს მუდმივებს, რომლებიც ავტომატურად არის სტატიკური. მუდმივას გამოცხადებისთანავე აუცილებლად უნდა მივანიჭოთ მნიშვნელობა. ამის შემდეგ, მისი მნიშვნელობის შეცვლა აღარ შეიძლება. ეს, თავის მხრივ, ძალიან კარგია, როდესაც კოდის წერის დროს შეგიძლია გამოაცხადო ისეთი ცვლადი, რომელიც არასდროს იცვლება და არ ემორჩილება არანაირ მანიპულაციას.

მუდმივას გამოსახვის ზოგადი ფორმა შემდეგია:

`const` ტიპი მუდმივას_სახელი = მნიშვნელობა;

სადაც `const` - საკვანძო სიტყვა აფიქსირებს, რომ სწორედ ის არის მუდმივა; ტიპი - არის მონაცემთა ტიპი, რომელსაც იმახსოვრებს მუდმივა; მუდმივას_სახელი არის მუდმივას დასახელება; მნიშვნელობა არის მუდმივას მნიშვნელობა, რომელიც არ ექვემდებარება ცვლილებას.

საინტერესოა, რომელი ტიპის ობიექტები შეიძლება გამოვაცხადოთ მუდმივად? მუდმივად შეიძლება გამოვაცხადოთ რიცხვითი, ლოგიკური და სტრიქონული ტიპის მნიშვნელობები ან ობიექტები, მაგრამ, მაგალითად, `DateTime` ტიპის ობიექტი არ შეიძლება გამოყენებულ იქნას როგორც მუდმივა.

მუდმივას აღწერის მაგალითები:

```
const decimal m = 333897987.3949M; // decimal ტიპის მუდმივა
```

```
const double pi = 3.1415; // double ტიპის მუდმივა
```

```
const bool YES = true; // bool ტიპის მუდმივა
```

```
const int Max = 255; // int ტიპის მუდმივა
```

```
const char AnswerYes = 'Y'; // char ტიპის მუდმივა
```

```
const string Str = "დილა მშვიდობისა"; // string ტიპის მუდმივა
```

ახლა კი ვნახოთ მუდმივას ჩაწერის სწორი ვარიანტები:

```
const int a = 11;
```

```
const float b = a * 7.9f;
```

```
const string s1 = "დილა მშვიდობისა ";
```

```
const string s2 = s1 + " საქართველო";
```

მუდმივას ჩაწერის არასწორი ვარიანტები:

```
const int a = Math.Cos(4) * 2;    // შეცდომაა
```

```
const string s1 = "Hello, world!";    // სწორია
```

```
const string s2 = s1.Substring(0, 6) + " Universe"; // შეცდომაა
```

შემთხვევითი რიცხვების გენერატორი. შემთხვევითი რიცხვების გენერირება შესაძლებელია განვახორციელოთ Random კლასის მეთოდებზე დაყრდნობით. Random კლასის მეთოდებია:

Next() - მეთოდი პარამეტრის გარეშე, გასცემს მთელ დადებით int ტიპის რიცხვებს დადებით დიაპაზონში.

Next(max) - გასცემს დადებით შემთხვევით მთელ რიცხვებს, დიაპაზონში [0, max].

NextDouble() - გასცემს შემთხვევით წილად რიცხვებს, რომელთა მნიშვნელობაც მოთავსებულია 0.0-სა და 1.0-ს შორის.

Next (min, max) - გასცემს დადებით შემთხვევით მთელ რიცხვებს, დიაპაზონში [min, max].

NextBytes(byte[] მასივი) - მასივს შეავსებს შემთხვევითი რიცხვებით, რომელთა ტიპია byte.

ახლა კი გადავიდეთ ამოცანების განხილვაზე.

ამოცანა 10.1. პროგრამაში ხდება სტატიკურ კლასთან, ცვლადთან და მეთოდთან მუშაობის დემონსტრირება.

```
static class კალკულატორი {  
    // კლასის ველი/ცვლადი  
    public static float PI = 3.1416F;  
    public static int ჯამი(int value1, int value2) //კლასის მეთოდი  
    {  
        return value1 + value2;  
    }  
}  
  
private void button1_Click(object sender, EventArgs e) {  
    int value = კალკულატორი.ჯამი(7, 9);  
}
```

```

        label1.Text = value.ToString(); //16
        label2.Text = კალკულატორი.PI.ToString(); //3.1416
    }

```

ამოცანა 10.2. სტატიკურ ცვლადთან და კონსტრუქტორთან მუშაობის დემონსტრირების მაგალითი.

```

class Account {
    public static decimal bonus = 100;
    public decimal totalSum;
    public Account(decimal sum)
    {
        totalSum = sum + bonus;
    }
}

private void button1_Click(object sender, EventArgs e) {
    label1.Text = Account.bonus.ToString(); // 100
    Account.bonus += 200;
    Account account1 = new Account(150);
    label2.Text = account1.totalSum.ToString(); // 450
    Account account2 = new Account(1000);
    label3.Text = account2.totalSum.ToString(); // 1300
}

```

ამოცანა 10.3. მოცემულია მთელი ტიპის ერთგანზომილებიანი მასივი. დავწეროთ პროგრამული კოდი, სადაც მოხდება ერთგანზომილებიანი მასივის შემთხვევითი რიცხვებით შევსება.

პირველი ვარიანტი. შემთხვევითი რიცხვების გენერირება და მასივის შევსება დიაპაზონში [0; 50] Next მეთოდის გამოყენებით. შემთხვევითი რიცხვების რაოდენობა შევიტანოთ textBox კომპონენტიდან. შედეგი კი ავსახოთ label კომპონენტში. ფორმაზე განვალაგოთ ერთი textBox, ერთი label და ერთი button კომპონენტები. პროგრამული კოდი განვათავსოთ button1 კომპონენტის Click მოვლენაზე (Events). პროგრამული კოდი:

```

private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    Random rand_ricxvebi = new Random();

```

```

string S = textBox1.Text;
int A = int.Parse(S);
for (int i = 0; i <= A; ++i)
    label1.Text += i + " = " + rand_ricxvebi.Next(50) + '\n';
}

```

მეორე ვარიანტი. პროგრამული კოდი **Random** კლასის მეთოდების გამოყენებით.

```

private void button1_Click(object sender, EventArgs e){
    label1.Text = ""; label2.Text = ""; label3.Text = "";
    const int INITRND = 77;
    Random REALRND = new Random();
    Random REPEATRND = new Random(INITRND);
    //შემთხვევითი რიცხვები [0,1] დიაპაზონში
    for (int i = 0; i <= 5; ++i)
label1.Text+=i+" = " + Math.Round(REALRND.NextDouble(),3) + '\n';
    //შემთხვევითი რიცხვები [min,max] დიაპაზონში
    int min = -100, max = 100;
    for (int i = 0; i <= 5; ++i)
label2.Text += i + " = " + REALRND.Next(min, max) + '\n';
    // შემთხვევითი რიცხვების ბაიტების ტიპის მასივი [0,255] დიაპაზონში
    byte[] BARRND = new byte[100];
    REPEATRND.NextBytes(BARRND);
    for (int i = 0; i < 10; ++i)
label3.Text += i + " = " + BARRND[i] + '\n'; }

```

ამოცანა 10.4. მოცემულია მთელი ტიპის ერთგანზომილებიანი მასივი. მასივი შეიცავს 15 ელემენტს, როგორც დადებითს, ასევე უარყოფითს. დავწეროთ პროგრამული კოდი, სადაც მოხდება:

1. მასივის შევსება შემთხვევითი რიცხვების გენერირებით [-50; 50] დიაპაზონში;
2. მასივის უარყოფითი ელემენტების განულება.

გენერირებული (საწყისი) მასივის მნიშვნელობები გამოვიტანოთ listBox1 კომპონენტში, ხოლო მიღებული მასივი - listBox2 კომპონენტში. ფორმაზე განვალაგოთ ორი label, ორი button და ორი listBox კომპონენტები. პროგრამული კოდი დავწეროთ როგორც button1, ისე button2 კომპონენტებზე. პროგრამული კოდი:

თავდაპირველად მასივი გამოვაცხადოთ, როგორც გლობალური:

```
int[] mas = new int[15];
```

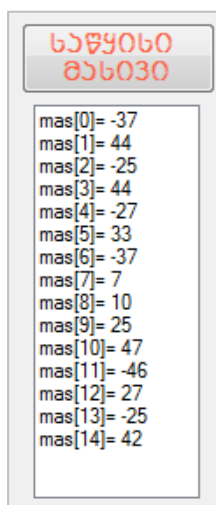
პროგრამული კოდის ერთი ნაწილი დავწეროთ button1 კომპონენტის click მოვლენაზე (Events).

```
private void button1_Click(object sender, EventArgs e){  
    listBox1.Items.Clear();  
    Random rand_ricxvebi = new Random();  
    for (int i = 0; i < 15; ++i) {  
        mas[i] = rand_ricxvebi.Next(-50, 50);  
        listBox1.Items.Add("mas[" + i.ToString() + "]=" + mas[i].ToString()); }  
}
```

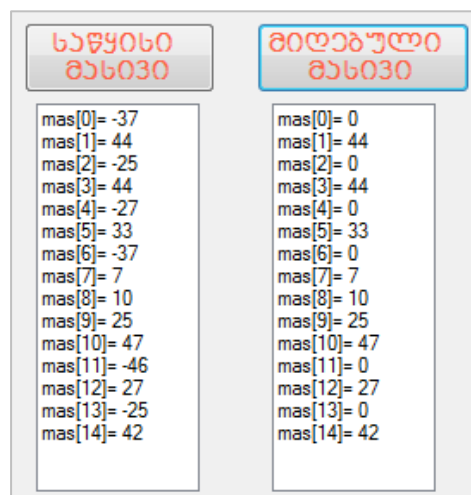
პროგრამული კოდის მეორე ნაწილი დავწეროთ button2 კომპონენტის click მოვლენაზე (Events).

```
private void button2_Click(object sender, EventArgs e){  
    listBox2.Items.Clear();  
    for (int i = 0; i < 15; ++i) {  
        if (mas[i] < 0) mas[i] = 0;  
        listBox2.Items.Add("mas[" + i.ToString() + "]=" + mas[i].ToString());  
    }  
}
```

ამოცანა 10.4-ის შესრულების შედეგი მოცემულია მე-40 და 41-ე ნახაზებზე.



ნახ. 40. ამოცანა 10.4-ის შესრულების შედეგი. სწყისი მასივი.



ნახ. 41. ამოცანა 10.4-ის შესრულების შედეგი. მიღებული მასივი.

ამოცანა 10.5. მოცემული გვაქვს მთელი ტიპის ერთგანზომილებიანი მასივი.

დავწეროთ პროგრამული კოდი, სადაც მოხდება:

- მასივის შევსება ათი შემთხვევითი რიცხვით დიაპაზონში [-10;10];
- მასივის გამოტანა listBox კომპონენტში;
- მასივის დადებითი ელემენტების ჯამის დაბეჭდვა;
- მასივის მინიმალური და მაქსიმალური მნიშვნელობების და მათი რიგით ნომერის (ინდექსის) დაბეჭდვა. ამ საკითხის გადასაჭრელად გამოვიყენოთ System.Array კლასის მეთოდები და თვისებები.

ფორმაზე განვალაგოთ სამი button, სამი label და ერთი listBox კომპონენტები. პროგრამული კოდი დავწეროთ button1, button2 და button3 კომპონენტების click მოვლენაზე (იხ. ნახ. 42.).

თავდაპირველად გამოვაცხადოთ გლობალური ცვლადები:

```
int[] mas = new int[7];  
Random rnd_ricxvebi = new Random();
```

პროგრამული კოდის ერთი ნაწილი დავწეროთ button1 კომპონენტის click მოვლენაზე (Events).

```
private void button1_Click(object sender, EventArgs e) {  
    listBox1.Items.Clear();  
    for (int i = 0; i < 7; ++i) {  
        mas[i] = rnd_ricxvebi.Next(-10, 11);  
        listBox1.Items.Add(string.Format("mas[{0},2]= {1,3}", i, mas[i]));  
    }  
}
```

პროგრამული კოდის მეორე ნაწილი დავწეროთ button2 კომპონენტის click მოვლენაზე (Events).

```
private void button2_Click(object sender, EventArgs e){  
    int sum = 0;  
    for (int i = 0; i < 7; ++i) {  
        if (mas[i] >= 0) sum += mas[i];  
    }  
    label1.Text = "დადებითი ელემენტების ჯამი" + '\n' + sum.ToString();  
}
```


პროგრამული კოდის მესამე ნაწილი დავწეროთ button3 კომპონენტის click მოვლენაზე (Events).

```
private void button3_Click(object sender, EventArgs e){
    //მინიმალური მნიშვნელობის პოვნა
    int min_ricxvi = mas.Min();
    //მინიმალური მნიშვნელობის ინდექსის პოვნა
    int index_min = Array.IndexOf(mas, min_ricxvi);
    label2.Text = "MIN = " + min_ricxvi.ToString() + '\n'
        + "MIN_INDEX = " + index_min.ToString();
    //მაქსიმალური მნიშვნელობის პოვნა
    int max_ricxvi = mas.Max();
    //მაქსიმალური მნიშვნელობის ინდექსის პოვნა
    int index_max = Array.IndexOf(mas, max_ricxvi);
    label3.Text = "MAX = " + max_ricxvi.ToString() + '\n'
        + "MAX_INDEX = " + index_max.ToString();
}
```

ნახ. 42. ამოცანა 10.5-ის შესრულების შედეგი

ორგანზომილებიანი მასივი. ორგანზომილებიან მასივებთან მუშაობისას სასურველია გამოვიყენოთ კომპონენტი **dataGridView**. კომპონენტი **dataGridView** გვებმარება ორგანზომილებიანი მასივი ავსახოთ ცხრილის სახით. ორგანზომილებიან მასივს მეორენაირად **მატრიცასაც** უწოდებენ.

ამოცანა 10.6. მოცემულია მთელი ტიპის ორგანზომილებიანი მასივი. მასივის სტრიქონების რაოდენობა უდრის **ოთხს**, ხოლო სვეტების რაოდენობა - **ხუთს**. დავწეროთ პროგრამული კოდი, სადაც მოვახდენთ ორგანზომილებიანი მასივის

შემთხვევითი რიცხვებით შევსებას და მიღებული მასივის დაბეჭდვას **label** კომპონენტში. ფორმაზე განვალაგოთ ერთი **button** და ერთი **label** კომპონენტები. პროგრამული კოდი დავწეროთ **button1** კომპონენტის **click** მოვლენაზე (Events).

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    int[,] myarr = new int[4, 5];
    Random ran = new Random();
    for (int i = 0; i < 4; ++i) //სტრიქონი = 4
        for (int j = 0; j < 5; ++j) //სვეტი = 5
            myarr[i, j] = ran.Next(1, 15);
    //ორგანზომილებიანი მასივის გამოტანა კომპონენტში label
    //მოხდება 20 ცალი (გვაქვს 4 სვეტი და 5 სტრიქონი ე.ი. 4*5 = 20)
    //შემთხვევითი რიცხვის გამოტანა [1,15] დიაპაზონში
    for (int i = 0; i < 4; ++i) // სტრიქონი
    {
        for (int j = 0; j < 5; ++j) //სვეტი
            label1.Text += myarr[i, j].ToString() + " ";
        label1.Text += '\n';
    }
}
```

ამოცანა 10.7. მოცემულია ნამდვილი ტიპის ორგანზომილებიანი მასივი. მასივის **სტრიქონების** რაოდენობა უდრის **სამს**, ხოლო **სვეტების** რაოდენობა **ექვსს**. დავწეროთ პროგრამული კოდი, სადაც მოვახდენთ ორგანზომილებიანი მასივის შემთხვევითი რიცხვებით შევსებას და მიღებული მასივის დაბეჭდვას **textBox** კომპონენტში. ნამდვილი ტიპის შემთხვევითი რიცხვების დიაპაზონია: [10.00, 50.00]. ნამდვილი ტიპის მასივის მნიშვნელობები დავბეჭდოთ ორი რიცხვის სიზუსტით. ფორმაზე განვალაგოთ ერთი **button** და ერთი **textBox** კომპონენტები. **textBox** კომპონენტის თვისება **Multiline-ში** ავირჩიოთ მდგომარეობა **true**; პროგრამული კოდი დავწეროთ **button1** კომპონენტის **click** მოვლენაზე (იხ. ნახ. 43.).

```
private void button1_Click(object sender, EventArgs e){
    double[,] myarr = new double[3, 6];
    Random random = new Random();
```

```

double temp;
textBox1.Clear();
// სტრიქონი = 3 სვეტი = 6
for (int i = 0; i < 3; ++i)
    for (int j = 0; j < 6; ++j) {
temp = Convert.ToDouble(random.Next(1000,5000))/100;
        myarr[i, j] += temp;
    }
//ორგანზომილებიანი მასივის გამოტანა textBox კომპონენტში
for (int i = 0; i < 3; ++i){
    for (int j = 0; j < 6; ++j)
        textBox1.Text += myarr[i, j].ToString() + " ";
        textBox1.Text += "\r\n";
    }
}

```

double ტიპის შემთხვევითი რიცხვები					
40,25	49,97	44,09	40,59	30,2	30,23
20,75	28,05	41,1	15,08	15,51	33,95
33,67	43,67	20,14	10,81	22,93	10,96

ნახ. 43. ამოცანა 10.7.-ის შესრულების შედეგი

ამოცანა 10.8. მოცემული გვაქვს მთელი ტიპის ორგანზომილებიანი მასივი. დავწეროთ პროგრამული კოდი, სადაც მოვახდენთ ორგანზომილებიანი მასივის შემთხვევითი რიცხვებით შევსებას. სტრიქონების და სვეტების რაოდენობა შევიყვანოთ **textBox** კომპონენტიდან. ფორმის დიზაინი ავაგოთ ნახ. 44-ზე ნაჩვენები სქემის სახით ან საკუთარი შეხედულებისამებრ. ფორმაზე განვალაგოთ ორი **textBox**, ორი **label**, ორი **button** და ერთი **dataGridView1** კომპონენტები.

თავდაპირველად გამოვაცხადოთ გლობალური ცვლადები:

```

int m, n;
public int[,] myarr = new int[50, 50];

```

button1 კომპონენტის **Click** მოვლენაზე განვათავსოთ ქვემოთ მოცემული კოდის ფრაგმენტი:

```

private void button1_Click(object sender, EventArgs e) {
    if ((textBox1.Text != "") && (textBox2.Text != ""))
    {
        m = int.Parse(textBox1.Text);
        n = int.Parse(textBox2.Text);
        dataGridView1.ColumnCount = n;
        dataGridView1.RowCount = m;
        dataGridView1.AutoSizeColumnsMode=DataGridViewAutoSizeColumnsMode.AllCells;
    }
    else
    {
        MessageBox.Show("შეცდომა!!! სვეტი? სტრიქონი?", "!!!",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}

```

button2 კომპონენტის Click მოვლენაზე განვათავსოთ ქვემოთ მოცემული კოდის ფრაგმენტი:

```

private void button2_Click(object sender, EventArgs e) {
    Random rnd = new Random();
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j)
            myarr[i,j] = rnd.Next(-30, 30);
    }
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j)
            dataGridView1.Rows[i].Cells[j].Value = myarr[i,j];
    }
}

```

▶	1	3	14
	-2	9	-19
	21	28	-5
	-15	-2	5
*	22	6	-26

ნახ. 44. ამოცანა 10.8-ის შესრულების შედეგი

ამოცანა 10.9. მოცემულია მთელი ტიპის კვადრატული მასივი (მატრიცა). კვადრატულია მასივი, რომლის სტრიქონების და სვეტების რაოდენობა ერთმანეთის ტოლია. დავწეროთ პროგრამული კოდი, სადაც მოვახდენთ კვადრატული მასივის შემთხვევითი რიცხვებით შევსებას. კვადრატული მასივის განზომილება შევიყვანოთ კლავიატურიდან textBox კომპონენტის საშუალებით. მასივი გამოვიტანოთ dataGridView კომპონენტში. ფორმაზე განვალაგოთ ერთი textBox, ერთი button და ერთი dataGridView1 კომპონენტები (იხ. ნახ. 45.). პროგრამული კოდი დავწეროთ button1 კომპონენტის Click მოვლენაზე (Events). პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e){
    int N = int.Parse(textBox1.Text);
    dataGridView1.ColumnCount = N;
    dataGridView1.RowCount = N;
    int[,] myarr = new int[N, N];
    Random rnd = new Random();
    for (int i = 0; i < dataGridView1.ColumnCount; ++i) {
        for (int j = 0; j < dataGridView1.RowCount; ++j) {
            myarr[i, j] = rnd.Next(100);
            dataGridView1.Rows[i].Cells[j].Value = myarr[i, j].ToString(); }
        }
}
```

95	75	75	69	72	29	38	45	21	96	65
27	50	67	98	27	6	19	94	18	14	30
68	52	67	95	53	88	30	13	87	8	31
23	13	19	31	19	5	63	18	49	84	40
17	60	73	61	12	54	57	22	88	52	19
64	6	23	1	76	76	50	15	8	8	57
95	26	53	78	96	86	13	31	0	65	46
27	34	7	21	91	75	76	56	65	19	79
4	48	12	8	18	29	86	67	34	76	32
60	64	47	7	33	58	70	8	57	52	54
18	79	88	93	54	84	74	38	51	42	37

ნახ. 45. ამოცანა 10.9-ის შესრულების შედეგი

ამოცანა 10.10. მოცემული გვაქვს მთელი ტიპის ორგანზომილებიანი კვადრატული მასივი. მატრიცის სტრიქონების და სვეტების რაოდენობა უდრის რვას. დავწეროთ პროგრამული კოდი, სადაც მოვახდენთ ორგანზომილებიანი მასივის შემთხვევითი რიცხვებით შევსებას. დავბეჭდოთ მატრიცის დამატებით დიაგონალზე მყოფი მაქსიმალური ელემენტის მნიშვნელობა და შედეგი გამოვიტანოთ textBox კომპონენტში. ორგანზომილებიანი მასივი ავსახოთ dataGridView კომპონენტში. ფორმაზე განვაღებოთ ერთი textBox, ერთი button, ერთი label და ერთი dataGridView1 კომპონენტები. პროგრამული კოდი განვათავსოთ button1 კომპონენტის Click მოვლენაზე (იხ. ნახ. 46.). პროგრამული კოდი:

```
private void button1_Click(object sender, EventArgs e) {  
    dataGridView1.ColumnCount = 6;  
    dataGridView1.RowCount = 6;  
    int[,] myarr = new int[6, 6];  
    Random rnd = new Random();//მატრიცის შემთხვევითი რიცხვებით შევსება  
    for (int i = 0; i < 6; ++i)  
        for (int j = 0; j < 6; ++j)  
            myarr[i, j] = rnd.Next(-100,100);  
    //მატრიცის მნიშვნელობების გამოტანა dataGridView1 კომპონენტში  
    for (int i = 0; i < 6; ++i)  
        for (int j = 0; j < 6; ++j)  
            dataGridView1.Rows[i].Cells[j].Value = myarr[i, j].ToString();  
    //მაქსიმალური ელემენტის ძებნა დამატებით დიაგონალზე  
    int m = int.MinValue;  
    for (int i = 0; i < 6; ++i)  
        if (myarr[i, 5 - i] > m) m = myarr[i, 5 - i];  
    textBox1.Text="დამატებით დიაგონალზე მყოფი მაქსიმალური ელემენტი="+  
        m.ToString(); }  

```

The screenshot shows a Windows application window. At the top, there is a 6x6 data grid (DataGridView) displaying random integers. A red diagonal line is drawn across the grid from the top-left to the bottom-right. Below the grid, there is a label (Label) with the text "დამატებით დიაგონალზე მაქსიმალური ელემენტი = 76".

8	-19	39	-13	83	-55
-65	73	44	46	76	71
-65	53	1	3	-38	-15
-93	67	-56	-100	-37	37
1	-13	54	62	-56	-53
-100	-96	-27	-6	-25	75

ნახ. 46. ამოცანა 10.10-ის შესრულების შედეგი

ამოცანა 10.11. გამოვაცხადოთ ორგანზომილებიანი მასივი, სადაც სვეტების და სტრიქონების რაოდენობა ერთმანეთის ტოლია ე. ი. მასივი კვადრატულია. მასივის განზომილება შევიყვანოთ **textBox** კომპონენტის დახმარებით. მოვახდინოთ მასივის შევსება შემთხვევითი რიცხვების გენერირების გზით დიაპაზონში 1-დან 10-მდე. დავბეჭდოთ თითოეული სტრიქონისა და სვეტის ჯამი. ფორმაზე განვალაგოთ ერთი **textBox**, ერთი **button** და სამი **label** კომპონენტები. პროგრამული კოდი განვათავსოთ **button1** კომპონენტის **Click** მოვლენაზე (Events). **პროგრამული კოდი:**

```
private void button1_Click(object sender, EventArgs e) {
    label1.Text = ""; label2.Text = ""; label3.Text = "";

    // კვადრატული მატრიცის მნიშვნელობა
    int n = int.Parse(textBox1.Text);
    int[,] a = new int[n, n];
    Random RND = new Random();

    // მატრიცის შევსება შემთხვევითი რიცხვებით
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            a[i, j] = 1 + RND.Next(10);

    // მატრიცის დაბეჭდვა
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++)
            label1.Text += a[i, j].ToString() + " ";
        label1.Text += '\n';
    }

    int[] b = new int[n]; //სტრიქონის ჯამი
    int[] c = new int[n]; //სვეტის ჯამი

    // თითოეული სტრიქონის ჯამი
    for (int i = 0; i < n; i++){
        int s = 0;
        for (int j = 0; j < n; j++)
```

```

        s += a[i, j];
        b[i] = s;
        label2.Text += b[i].ToString() + " ";
        label2.Text += '\n';
    }
    // თითოეული სვეტის ჯამი
    for (int j = 0; j < n; j++)
    {
        int s = 0;
        for (int i = 0; i < n; i++)
            s += a[i, j]; c[j] = s;
        label3.Text += c[j].ToString() + " ";
    }
}

```

დამოუკიდებელი სამუშაო

1. აღწერეთ კომენტარების დართვით ქვემოთ მოცემული კოდის თითოეული სტრიქონი:

```

class myCircle //.....
{
    public static double SqrCircle(int radius) //.....
    {
        return Math.PI * radius * radius; //.....
    }
    public static double LongCircle(int radius) //.....
    {
        return 2 * Math.PI * radius; //.....
    }
}
private void button1_Click(object sender, EventArgs e)
{
    int r = 11; //.....
    //.....
    label1.Text = " ..... = " + myCircle.SqrCircle(r);
}

```



```

//.....
label2.Text = " ..... = " +
myCircle.LongCircle(r);
}

```

2. ქვემოთ მოცემულ კოდში არის შეცდომები. გაასწორეთ იგი და წარმოადგინეთ კოდის სწორი ვარიანტი.

```

class MyMath {
    public const double PI = 3.14;
    public const double MY1;
    public MyMath()
    {
        MY1 = 8.4;
    }
}

private void button1_Click(object sender, EventArgs e) {
    double val = 10 * MyMath.PI;
    MathLib.PI = 3.8;
}

```

3. მოცემულია მთელი ტიპის ერთგანზომილებიანი მასივი. მასივი შეიცავს 15 ნამდვილი ტიპის ელემენტს. დაწერეთ პროგრამული კოდი, სადაც მოხდება:

- მასივის შეავსება შემთხვევითი რიცხვებით დიაპაზონში [-10.0;10.0];
- მასივის დაბეჭდვა;
- მასივის ლუწი ელემენტების ჯამის პოვნა;
- მასივის დადებითი ელემენტების რაოდენობის დათვლა;

მიღებული შედეგი გამოიტანეთ **label** კომპონენტებში. პროგრამული კოდი განათავსეთ ბრძანებით **button** კომპონენტზე.

4. მოცემულია მთელი ტიპის ერთგანზომილებიანი მასივი. მასივის შეიცავს 15 მთელი ტიპის ელემენტს. მოახდინეთ მასივის შეავსება შემთხვევითი რიცხვებით. დაწერეთ პროგრამული კოდი, რომელიც იპვის მასივის იმ ელემენტების საშუალო არითმეტიკულს, რომელთა მნიშვნელობებიც აღემატება 11-ს. თუ ასეთი ელემენტები არ არის, მაშინ გამოიტანეთ შესაბამისი შეტყობინება. მიღებული

შედეგი გამოიტანეთ **label** კომპონენტში. პროგრამული კოდი განათავსეთ ბრძანებით **button** კომპონენტზე.

5. ამოცანა 10.11 გადააკეთეთ შემდეგი სახით: კვადრატული მასივის ნაცვლად შემოიტანეთ ჩვეულებრივი მასივი ანუ ისეთი მასივი, სადაც სვეტებისა და სტრიქონების რაოდენობა არ არის თანაბარი $i \neq j$.

მასივის განზომილება (სვეტებისა და სტრიქონების რაოდენობა) და ელემენტების მნიშვნელობები შეიტანეთ **textBox1**, **textBox2** და **textBox3** კომპონენტებიდან.

იპოვეთ: ა) ორგანზომილებიანი მასივის მთავარ დიაგონალზე მყოფი მინიმალური ელემენტი და მისი ინდექსი; ბ) მთავარი დიაგონალის ზევით განთავსებული ელემენტები დაბეჭდეთ ერთგანზომილებიანი მასივის სახით.

მიღებული შედეგი გამოიტანეთ **label** კომპონენტში. პროგრამული კოდი განათავსეთ ბრძანებით **button** კომპონენტზე.

6. მოცემულია ორგანზომილებიანი მასივი. სტრიქონების და სვეტების რაოდენობა თანაბარია და უდრის ხუთს. ორგანზომილებიანი მასივი შეავსეთ მთელი ტიპის შემთხვევითი რიცხვებით დიაპაზონში $[-20; 20]$:

- ფორმა ააგეთ დამოუკიდებლად საკუთარი შეხედულებისამებრ.
- დაბეჭდეთ მასივი.
- დაადგინეთ, არის თუ არა მასივში ელემენტი 0. თუ არის, დაითვალეთ რამდენია და დაბეჭდეთ მათი შესაბამისი ინდექსები.
- დაადგინეთ, არის თუ არა მასივში 5-ის ჯერადი ელემენტები. თუ არის ასეთი ელემენტები დაბეჭდეთ, თუ არა და შესაბამისი შეტყობინება - „მასივში 5-ის ჯერადი ელემენტები არ არის“.
- დაადგინეთ, არის თუ არა მასივში ორი ერთნაირი ელემენტი.
- დაადგინეთ, რომელიმე სვეტი შეიცავს თუ არა მხოლოდ დადებით ელემენტებს. თუ შეიცავს, დაბეჭდეთ ასეთი სვეტის რიგითი ნომერი, თუ არა დაბეჭდეთ, შემდეგი სახის შეტყობინება - „მხოლოდ დადებითი ელემენტების მქონე სვეტი არ არის მასივში“ .

11 ინტერფეისის არსი. ინტერფეისების რეალიზება. სახელების სივრცე (namespace). using დირექტივა. ჩადგმული სივრცე.

C# ენის ერთ-ერთი მნიშვნელოვანი შემადგენელი ნაწილია ინტერფეისი. ინტერფეისი განსაზღვრავს მეთოდების, თვისებების, ინდექსატორებისა და მოვლენების ნაკრებს, რომელთა რეალიზება შესაძლებელია კლასის საშუალებით. პროგრამის წერის დროს ზოგჯერ წამოიჭრება შემდეგი საკითხი: მოხდეს კლასის მიერ შესრულებული მოქმედებების განსაზღვრა მათი რეალიზების გზის მითითების გარეშე.

C# ენაში შეიძლება მთლიანად განვაცალკევოთ კლასის ინტერფეისი ამ კლასის რეალიზაციისაგან/განხორციელებისაგან. ინტერფეისის გამოცხადების შემდეგ მისი რეალიზება/ამუშავება შესაძლებელია ერთი ან მეტი კლასის საშუალებით. ამასთან, ასეთი კლასი შეიძლება დამატებით შეიცავდეს საკუთარ ელემენტებს.

ინტერფეისის გამოცხადების სინტაქსია:

interface ინტერფეისის_სახელი

{ ინტერფეისის ტანი }

ინტერფეისი - ესაა შეთანხმება, რომელიც ობიექტის ყოფაქცევას განსაზღვრავს. ინტერფეისი წარმოდგენილი არის როგორც მიმართვითი ტიპი, რომელიც შეიცავს თვისებების და მეთოდების მხოლოდ ნაკრებს, მაგრამ C#-ის მე-8.0 ვერსიიდან დაწყებული ინტერფეისს უკვე შეუძლია მეთოდის რეალიზება/შესრულება. ინტერფეისი ადგენს კლასის წევრთა სიას, მაგრამ არაფერს ამბობს მათი რეალიზაციის/შესრულების/მუშაობის შესახებ. ობიექტში დასაშვებია რამდენიმე ინტერფეისის რეალიზაცია, ხოლო ერთი და იგივე ინტერფეისი შეიძლება რეალიზებულ იქნას სხვადასხვა კლასში.

ინტერფეისის ტანი შეიძლება შეიცავდეს მეთოდებს, თვისებებს, ინდექსატორებსა და მოვლენებს. ინტერფეისი არ შეიძლება შეიცავდეს ცვლადებს (ვარსებებს), სტატიკურ წევრებს, კონსტრუქტორებს, დესტრუქტორებსა და ოპერატორულ მეთოდებს. ინტერფეისში ხდება მეთოდების, თვისებების, ინდექსატორებისა და მოვლენების გამოცხადება/აღწერა. მეთოდისათვის დამატებით ეთითება

დასაბრუნებელი მნიშვნელობის ტიპი. გარდა ამისა, მეთოდის ხელმოწერა/სიგნატურა ზუსტად უნდა შეესაბამებოდეს interface-ის განსაზღვრაში მითითებულ სიგნატურას (მეთოდის სიგნატურა ხელმოწერა/სიგნატურა - ესაა მეთოდის სახელი, მის მიერ დაბრუნებული მნიშვნელობის ტიპი და პარამეტრების სია).

ინტერფეისი არის რომელიმე კლასის წარმომადგენელი პროგრამული კოდის დაწერის გარეშე. ნებისმიერი ინტერფეისი ჩუმათობის პრინციპის მიხედვით არის public და ამიტომ ინტერფეისების გამოცხადებისას წვდომის მოდიფიკატორი არ გამოიყენება. ასევე მეთოდებისთვისაც არ გამოიყენება წვდომის მოდიფიკატორი public, რადგან მეთოდებიც ჩუმათობის პრინციპის მიხედვით არის public. ინტერფეისში მეთოდებს არ გააჩნიათ არანაირი ტანი/კოდი, არის მხოლოდ მეთოდის დასახელება. მოდით, ვნახოთ ფრაგმენტი და გავარჩიოთ, თუ რა შეცდომაა ასახული:

```
interface Itoken {  
/* შეცდომა: 1. მეთოდის სახელის წინ მითითებულია მოდიფიკატორი public; 2. ასევე ვხედავთ  
მეთოდის ტანს/კოდს */  
    public int LineNumber()  
    { ... };  
}
```

ობიექტებს, რომლებშიც რეალიზებულია რომელიმე ინტერფეისი, შეუძლიათ ერთმანეთთან ურთიერთმოქმედება სწორედ ინტერფეისის საშუალებით. მიღებულია ინტერფეისთა სახელები დაიწყოს „I“ ასოთი, რათა კოდის კითხვადობა გახდეს უფრო მარტივი. მაგალითად, გვაქვს ობიექტი ავტომობილი, ასეთ შემთხვევაში შეგვიძლია ინტერფეისის რეალიზაცია დასახელებით Idrivable და ასევე მეთოდების - „მოძრაობა_წინ, მოძრაობა_უკან, გაჩერება“ რეალიზაცია/განხორციელება, იგივე ინტერფეისი შეიძლება გამოვიყენოთ და ავამუშავოთ სხვა კლასებშიც, როგორიცაა მაგალითად, „სატვირთო_მანქანა“ ან „კატერი“. შედეგად, ამ ობიექტებს შეუძლია ურთიერთმოქმედება ობიექტთან „მძღოლი“. ეს უკანასკნელი მთლიანად უხილავია ინტერფეისის რეალიზაციისთვის, რომელთანაც იგი მოქმედებს, მისთვის ცნობილია მხოლოდ ინტერფეისი. განვიხილოთ კიდევ ერთი მაგალითი: ქვაქვს ინტერფეისი IShape, რომელიც განსაზღვრავს ერთადერთ მეთოდს - CalculateArea. ამასთანავე კლასი Circle, რომელშიც რეალიზებულია ეს

ინტერფეისი, დაითვლის ფიგურის ფართობს სრულიად განსხვავებულად, კიდრე კლასი Square, რომელშიც ასევე რეალიზებულია ეს ინტერფეისი.

მიუხედავად ამისა, ობიექტს, რომელსაც სჭირდება ურთიერთქმედება Ishape-ის გამოყენებით, შეუძლია მეთოდის CalculateArea გამოძახება Circle ან Square კლასიდან და ამასთანავე სწორი/კორექტული შედეგის მიღება.

ინტერფეისები განსაზღვრავს კლასის ყოფაქცევას. ერთი და იგივე ინტერფეისი შეიძლება რეალიზებულ იქნას სხვადასხვა კლასში, რაც უზრუნველყოფს მათი ურთიერთმოქმედების შესაძლებლობას, ამგვარად პოლიმორფიზმის კონცეფციის განხორციელებას.

ინტერფეისების რეალიზება. როგორც კი ინტერფეისი განისაზღვრება, ის შეიძლება რეალიზებული იყოს ერთი ან მეტი კლასის მიერ. ინტერფეისის რეალიზაციისათვის კლასის სახელის შემდეგ უნდა მივუთითოთ ორი წერტილი და ინტერფეისის სახელი. თუ კლასი ახდენს ინტერფეისის რეალიზებას, ის ვალდებულია მისი რეალიზება მოახდინოს მთლიანად.

კლასებს, რომლებიც ახდენს ინტერფეისების რეალიზებას, შეუძლიათ განსაზღვრონ, აგრეთვე, დამატებითი წევრები. ერთ კლასს შეუძლია რამდენიმე ინტერფეისის რეალიზება. ამ შემთხვევაში კლასში უნდა შესრულდეს ყველა ინტერფეისის ელემენტების რეალიზება.

ახლა კი განვიხილოთ ინტერფეისი პრაქტიკულ მაგალითზე: ვთქვათ, გვინდა დავწეროთ ინტერფეისი, რომელიც შეიცავს მეთოდს Say, მეთოდის დასახელებიდან გამომდინარე, ინტერფეისსაც დავარქვათ შესაბამისი სახელი: ISpeakable. დავწეროთ ქვემოთ მოცემული კოდი:

```
interface ISpeakable
{
    void Say();
}
```

როგორც ვხედავთ, მეთოდი მხოლოდ გამოვაცხადეთ, მაგრამ მისი მუშაობა/რეალიზება არ განვახორციელეთ. ახლა კი დავწეროთ ისეთი კლასი რომელიც მოახდენს ზემოთ დაწერილი ინტერფეისის განხორციელებას:

```
public class Person : ISpeakable {
    public void Say()
```

```

    {
        MessageBox.Show("დილა მშვიდობის!");
    }
}

```

როგორც ვხედავთ, აღვწერეთ კლასი Person და ინტერფეისი, რომლის განხორციელებაც/შესრულებაც გვინდა. ინტერფეისის განხორციელებისას, კლასში აუცილებლად უნდა გამოვაცხადოთ და ავამუშაოთ ინტერფეისის ყველა მეთოდი და თვისება, რომელიც უნდა შევასრულოთ ანუ თუ ინტერფეისს აქვს მეთოდი Say, ჩვენ უნდა გამოვაცხადოთ და განვახორციელოთ/ავამუშაოთ იგი კლასში. ახლა კი დავწეროთ კლასის ობიექტი:

```

Person p = new Person();
p.Say();

```

გავარკვიოთ თუ რისთვის გვინდა ინტერფეისი, თუ ინტერფეისის გარეშეც შეგვიძლია კლასში პირდაპირ აღვწეროთ მეთოდი და ა. შ.

ვთქვათ გვაქვს ორი კლასი. ორივე კლასში აღწერილია ველი Name და მეთოდი Move:

```

public class Person {
    public string Name { get; set; }
    public void Move()
    {
        MessageBox.Show("ადამიანი გადაადგილდება...");
    }
}

public class Animal {
    public string Name { get; set; }
    public void Move()
    {
        MessageBox.Show("ცხოველი გადაადგილდება...");
    }
}

```

როგორც ვხედავთ, მეთოდის რეალიზება/განხორციელება არის განსხვავებული: ერთ კლასში გადაადგილდება ადამიანი, ხოლო მეორე კლასში კი ცხოველი. ახლა კი გვინდა შევქმნათ ისეთი ფუნქცია, რომელიც დააფიქსირებს ადამიანის ან

ცხოველის გადაადგილებას რომელიმე გარკვეულ წერტილამდე/პუნქტამდე ანუ ამ მეთოდში ადამიანი ან ცხოველი მიდის დანიშნულების ადგილამდე. ეს კი შეგვიძლია შევასრულოთ პარამეტრის გადაცემით, როგორც ქვემოთ არის ნაჩვენები:

```
void PersonGoToPoint(Person p)
{
    MessageBox.Show("დანიშნულების ადგილზე მისვლა");
}
```

აქ კი წამიჭრება შემდეგი პრობლემა: მოგვიწევს თითოეული კლასისათვის შევქმნათ თავისი მეთოდი ანუ ახლა მოგვიწევს კიდევ ერთი მეთოდის შექმნა კლასისათვის Animal:

```
void AnimalGoToPoint(Animal animal)
{
    MessageBox.Show("დანიშნულების ადგილზე მისვლა");
}
```

თუ მოგვიწევს კლასების რაოდენობის გაზრდა, მაგალითად, კლასი Fish-სათვის მოგვიწევს მეთოდის თავიდან დაწერა და ა.შ., ასეთ შემთხვევაში კოდი გახდება ძალიან დიდი და „გაიბერება“, ისევე როგორც გვებერება მუცელი „კოკა-კოლის“ დალევის შემდეგ ☺. მოდით ამ პრობლემას შევხედოთ ინტერფეისების საშუალებით და ვიპოვოთ ცოტა უფრო მარტივი გამოსავალი.

შევქმნათ ინტერფეისი Imovable, რომელიც თავისთავში შეინახავს თვისებას Name და მეთოდს Move:

```
public interface IMoveable {
    string Name { get; set; }
    void Move();
}
```

ახლა კი უკვე შესაძლებელია კლასებმა Person და Animal განახორციელონ მოცემული ინტერფეისი:

```
public class Person : IMoveable {
    public string Name { get; set; }
    public void Move()
    {
        MessageBox.Show("ადამიანი გადაადგილდება...");
    }
}

public class Animal : IMoveable {
    public string Name { get; set; }
}
```

```

public void Move()
{
    MessageBox.Show("ცხოველი გადაადგილდება...");
}
}

```

თითქოს ყველაფერი იგივეა, მაგრამ ჩვენ შეგვიძლია შევასრულოთ ერთი ძალიან მნიშვნელოვანი რამ: მეთოდის (რომელიც პასუხისმგებელია დანიშნულების ადგილამდე მისვლაზე) პარამეტრში გადავცეთ არა კონკრეტული კლასი, არამედ ინტერფეისი.

```

void GoToPoint(IMoveable moveable)
{
    MessageBox.Show("დანიშნულების ადგილზე მისვლა");
}

```

რას ნიშნავს ინტერფეისის გადაცემა? ახლა, ფუნქციის გამოძახებისას, ჩვენ შეგვიძლია გადავცეთ ნებისმიერი კლასი, რომელიც ამ ინტერფეისს ახორციელებს/ამუშავებს. შეგვიძლია შევადაროთ მემკვიდრეობითობას: თუ ფუნქცია შეიცავს საბაზო კლასის პარამეტრებს, მაშინ შეგვიძლია გადავცეთ ყველა მისი შვილობილი კლასი.

```

IMoveable p = new Person();
p.Name = "ლუკა";
p.Move();
GoToPoint(p);

```

ინტერფეისის საფუძველზე შეგვიძლია შევექმნათ Person კლასის ობიექტი და გადავცეთ მას ფუნქცია. რდგან Person კლასის ობიექტი ახდენს მითითებული ინტერფეისის რეალიზებას, მაშინ შეგვიძლია ეს ჩანაწერი ავკრიფოთ თავისუფლად, მაგრამ თუ კლასი Person არ ახდენს მითითებული ინტერფეისის რეალიზებას/განხორციელებას, მაშინ ასეთი ჩანაწერი არ შესრულდებოდა. სხვათა შორის, ჩვენ არ დაგვჭირდა კლასის შექმნა ინტერფეისის სახელით. ჩვენ შეგვიძლია უბრალოდ შევექმნათ Person კლასის ობიექტი და სწორედ ობიექტი გადავცეთ მეთოდს:

```

Person p = new Person();
p.Name = "ანა";

```



```
p.Move();
GoToPoint(p);
```

ასეთი დამოკიდებულებით პროგრამა ხდება მოქნილი და კოდის სიგრძე საკმაოდ მცირდება. თუ ჩვენ პროგრამულ კოდს დავამატებთ კლასს Fish, აღარ იქნება საჭირო დავწეროთ კიდეც ერთხელ ამ კლასს დაქვემდებარებული მეთოდი. საკმარისი იქნება რომ კლასმა Fish განახორციელოს/აამუშაოს ინტერფეისი Imovable, შემდეგ კი თავისუფლად შევძლებთ ფუნქციას პარამეტრის სახით გადავცეთ Fish კლასის ობიექტი.

კლასების მემკვიდრეობითობა და ინტერფეისის რეალიზება. კლასებს შეუძლიათ ერთზე მეტი ინტერფეისის რეალიზება. ამ შემთხვევაში ინტერფეისების სახელები ერთმანეთისაგან მძიმეებით უნდა გამოიყოს.

კლასი შეიძლება იყოს წინაპარი კლასის მემკვიდრე და ახდენდეს ერთი ან მეტი ინტერფეისის რეალიზებას. ასეთი შემთხვევების დროს ინტერფეისების ჩამონათვალში წინაპარი კლასის სახელი უნდა იყოს ჩაწერილი პირველ ადგილზე.

თუ გვინდა რომ მემკვიდრეობითობის მექანიზმით ვისარგებლოთ, ჯერ უნდა დავწეროთ კლასის სახელი და შემდეგ ინტერფეისის. ალბათ გაგიჩნდებათ კითხვა არის თუ არა შესაძლებელი კლასი იყოს ერთზე მეტი ინტერფეისი მემკვიდრე, პასუხია შეიძლება. ვიცით, რომ კლასი შეიძლება იყოს მხოლოდ ერთი კლასის მემკვიდრე, მაგრამ ინტერფეისებზე ეს შეზღუდვა არ მოქმედებს, რაც უფრო მძლავრს ხდის ჩვენ მიერ დაწერილ კოდს. ქვემოთ მოცემულია მემკვიდრეობითობის ინტერფეისებში გამოყენების მაგალითი:

```
interface Ipod {
    void mac();
}

interface Iphone {
    void mac();
}

class Apple : Ipod, Iphone {
    void Ipod.mac()
    {
        MessageBox.Show("Ipod");
    }
}
```

```

void Iphone.mac()
{
    MessageBox.Show("Iphone");
}

private void button1_Click(object sender, EventArgs e) {
    Ipod ipod = new Apple();
    ipod.mac();
    Iphone iphone = new Apple();
    iphone.mac();
}

```

განხილული მაგალითი საკმაოდ საინტერესოა, ვხედავთ, რომ შესაძლებელია კლასი რამდენიმე ინტერფეისის მემკვიდრე იყოს. აქვე არის სახელების დამთხვევის ე. წ. კოლიზიის შემთხვევა.

ვხედავთ, რომ ინტერფეისებს ერთი და იმავე დასახელების მეთოდები აქვთ. იმისათვის, რომ თითოეულის განსაზღვრა ცალ-ცალკე მოხდეს უნდა გამოვიყენოთ სინტაქსი void Ipod.mac(), ალბათ ასევე შეამჩნიეთ, რომ მედიფიკატორი public არ უწერია მეთოდებს.

ინტერფეისული მიმართვები. ჩვენ შეგვიძლია გამოვაცხადოთ მიმართვითი ცვლადი, რომელსაც აქვს ინტერფეისული ტიპი, ანუ შეგვიძლია შევექმნათ ინტერფეისული მიმართვითი ცვლადი. ასეთი ცვლადი შეიძლება მიმართავდეს ნებისმიერ ობიექტს, რომელიც ახდენს მისი ინტერფეისის რეალიზებას.

ინტერფეისული მიმართვის საშუალებით ობიექტის მეთოდის გამოძახებისას გამოიძახება ის მეთოდი, რომლის რეალიზებას მოცემული ობიექტი ახდენს.

ცხადი რეალიზება. არის შემთხვევა, როცა რეალიზაციის დროს ინტერფეისის ელემენტის წინ ეთითება ინტერფეისის სახელი. კლასს შეუძლია ორი ინტერფეისის რეალიზება, რომლებშიც შეიძლება გამოცხადებული იყოს ერთნაირი სახელის ელემენტები. წამოიჭრება პრობლემა - როგორ განვასხვავოთ ასეთი ელემენტები. ეს პრობლემა წყდება ცხადი რეალიზაციის გამოყენებით.

ვთქვათ, კლასმა უნდა განახორციელოს ორი ინტერფეისი, რომელშიც არის ერთნაირი ხელმოწერის/სიგნატურის მქონე წევრი, მაგალითად, მეთოდი. ასეთი შემთხვევის დროს კლასში ამ წევრის განსაზღვრისას, ორივე ინტერფეისი

გამოიყენებს მას. ქვემოთ მოცემულ მაგალითში მეთოდზე Paint მიმართვისას ხდება ერთ მეთოდის გამოძახება.

```
interface IControl {
    void Paint();
}

interface ISurface {
    void Paint();
}

class SampleClass : IControl, ISurface {
//ორივე ინტერფეისი ISurface.Paint და IControl.Paint იძახებს მეთოდს Paint()
    public void Paint()
    {
        MessageBox.Show("დილა მშვიდობისა!");
    }
}

private void button1_Click(object sender, EventArgs e){
    SampleClass sc = new SampleClass();
    IControl ctrl = (IControl)sc;
    ISurface srfc = (ISurface)sc;
    // ერთი მეთოდის გამოძახება
    sc.Paint();
    ctrl.Paint();
    srfc.Paint(); }
```

შესაძლებელია ასეთი შემთხვევაც, თუ ორი ინტერფეისის წევრები ასრულებენ სხვადასხვა ფუნქციას, მაშინ ერთის ან ორივე ინტერფეისის განხორციელება მოხდება არასწორად. ასეთ შემთხვევაში ვიყენებთ ინტერფეისის წევრის ცხად რეალიზაციას. ცხადი რეალიზაციის დროს ინტერფეისის წევრის გამოსაძახებლად უნდა შევექმნათ კლასის ისეთი წევრი, რომელიც მიმართავს მხოლოდ ინტერფეისს და ასეთი წევრი ცნობს მხოლოდ ინტერფეისს.

როგორ მოხდება ინტერფეისის ცხადი რეალიზაციის ჩაწერა? ინტერფეისის სახელი -> წერტილი -> ინტერფეისის წევრის დასახელება. მაგალითად:

```
interface IControl {
    void Paint();
```

```

    }
    interface ISurface {
        void Paint();
    }
    public class SampleClass : IControl, ISurface {
        void IControl.Paint()
        {
            MessageBox.Show("ინტერფეისი - IControl.Paint");
        }
        void ISurface.Paint()
        {
            MessageBox.Show("ინტერფეისი - ISurface.Paint");
        }
    }
    private void button1_Click(object sender, EventArgs e){
        SampleClass obj = new SampleClass();
        //obj.Paint(); შეცდომაა
        IControl c = (IControl)obj;
        //IControl.Paint ინტერფეისის გამოძახება კლასიდან SampleClass
        c.Paint();
        ISurface s = (ISurface)obj;
        //ISurface.Paint ინტერფეისის გამოძახება კლასიდან SampleClass
        s.Paint();
    }

```

კლასის წევრი IControl.Paint წვდომადია მხოლოდ ინტერფეისის Icontrol-ის მხრიდან, ხოლო კლასის წევრი ISurface.Paint კი წვდომადია მხოლოდ ინტერფეისის ISurface-ის მხრიდან. მეთოდის თითოეული რეალიზაცია არის დამოუკიდებელი და კლასთან პირდაპირი წვდომა არ შეუძლია.

სახელების სივრცე. პროგრამის წერის დროს სახელების სივრცე გვადლევს საშუალებას, თავიდან ავიცილოთ კლასების სახელებს (სახელების დამთხვევა) შორის კონფლიქტი. თუ პროგრამაში არ არის გამოცხადებული სახელების სივრცე, მაშინ გამოიყენება ავტომატურად განსაზღვრული სახელების სივრცე. სახელების ასეთი სივრცე მოხერხებულია გამოვიყენოთ მოკლე და მარტივი პროგრამებისათვის.

დიდი ზომის, გამოყენებითი პროგრამები უნდა შედიოდეს სახელების რომელიმე სივრცეში ანუ სხვანაირად რომ ვთქვათ, კოდი უნდა იყოს ინკაფსულირებული სახელების სივრცეში სახელების კონფლიქტების თავიდან აცილების მიზნით.

სახელსივრცე (**namespace**) არის პროგრამირების ენის ნაწილი, რომელიც გვეხმარება კოდი გავხადოთ უფრო მეტად თანამიმდევრული, მოწესრიგებული და ლოგიკურად გამართული. ამასთანავე სახელსივრცეს აქვს სხვა ფუნქცია-დანიშნულებებიც.

მაგალითად: არის ისეთი შემთხვევა, როდესაც გვინდა კლასს დავარქვათ სახელი **Console**. ჩვენ ვიცით, რომ ასეთი დასახელების კლასი .NET framework-ის ბიბლიოთეკებში უკვე არსებობს და მას უკვე აქვს განსაზღვრული ფუნქცია. ამის გარდა, შეიძლება კიდევ დაგვჭირდეს სხვა კლასების შექმნა, რომელიც უკვე არსებობს ჩაშენებულ ბიბლიოთეკებში. სწორედ ასეთი სახის პრობლემის თავიდან აცილებაში გვეხმარება სახელსივრცე. პრობლემის გადაჭრა მარტივად შეგიძლია, თუ საკუთარ, ჩვენ მიერ შექმნილ სახელსივრცეში დავწერთ ისეთივე დასახელების კლასს, როგორი სახელიც უკვე ჰქვია .NET framework-ში ჩაშენებულ კლასს.

სახელსივრცე (namespace) ესაა მონაცემთა ტიპების დაჯგუფება. ყველა მონაცემთა ტიპის სახელს მოცემულ სახელთა სივრცეში ავტომატურად ემატება პრეფიქსი, რომელიც შედგენილია სახელსივრცის დასახელებისგან. ასევე შესაძლებელია ჩადგმული სახელსივრცეების შექმნა.

მაგალითად, საბაზო კლასების უმრავლესობა, რომლებიც ზოგადი გამოყენებისთვისაა დადგენილი, მოთავსებულია სახელთა სივრცეში System, ვებგვერდებისათვის - System.Web და ა.შ. C#-ის პროგრამის მაგალითზე შეიძლება შემდეგი კომენტარის გაკეთება:

```
// აქ სახელია Magazia.Web
namespace Magazia.Web {
    public class Checkout : PageBase
    { // და ა.შ.
```

დავუშვათ, რომ პროგრამა იყენებს სხვა პროგრამისტის მიერ შემუშავებულ კლასს, რომლის სახელია Manqana. დავუშვათ, ასევე, რომ ჩვენც დაგვჭირდა ამავე სახელის მქონე კლასის განსაზღვრა. ასეთ შემთხვევაში მოგვიწევს სხვა სახელის გამოყენება. სწორედ ასეთი პრობლემების გადასაწყვეტად გამოიყენება სახელების სივრცე.

სახელების სივრცე განსაზღვრავს გამოცხადების უბანს, რომელიც საშუალებას გვაძლევს, შევინახოთ სახელების თითოეული ნაკრები სხვა ნაკრებებისაგან ცალკე.

სახელების ერთ სივრცეში გამოცხადებული სახელები არ არის კონფლიქტში სახელების სხვა სივრცეში გამოცხადებულ ასეთივე სახელებთან.

System სახელების სივრცეში მრავალი კლასია განსაზღვრული, ამიტომ თითოეული პროგრამა იწყება დირექტივით: using System; სახელების სივრცის გამოცხადება ხდება namespace საკვანძო სიტყვის საშუალებით. მისი სინტაქსია:

```
namespace სახელების_სივრცის_სახელი  
{  
  
    სახელების სივრცის წევრები  
  
}
```

სახელების სივრცეში განსაზღვრული ყველა ელემენტი, როგორცაა კლასები, დელეგატები, ჩამოთვლები, ინტერფეისები ან სხვა სახელების სივრცეები, იმყოფება ამ სახელების სივრცის მხედველობის უბნის საზღვრებში.

რისთვის გვჭირდება სახელების სივრცე? თუ ჩვენი პროგრამული კოდი არის მცირე ზომის, მაშინ ჩვენ მიერ დაწერილი კლასი შეგვიძლია მოვათავსოთ იმ სახელების სივრცეში, რომელის გენერირებასაც ავტომატურად ახდენს პროექტი. თუ თქვენ გჭირდებათ დაამატოთ სხვადასხვა კლასი, მაშინ სასურველია დავყოთ ისინი სახელების სივრცეებში. ეს ძლიერ აადვილებს პროგრამულ კოდის მწარმოებულობას, განსაკუთრებით მაშინ, როცა ფაილები განთავსებულია შესაბამის საქალაქებში. თუ პროექტში დავამატებთ საქალაქდეს, ხოლო შემდეგ მასში კი დავამატებთ კლასს, მაშინ Visual Studio ავტომატურად განათავსებს მას შესაბამის სახელების სივრცეში.

ვთქვათ, შევქმენით პროექტი დასახელებით MyProject; შემდეგ ამ პროექტში შევქმენათ საქალაქდე დასახელებით MyFolder; ამ საქალაქდებში დამატებული კლასები, ჩუმათობის პრინციპის მიხედვით განთავსდებიან MyProject.MyFolder დასახელების სახელების სივრცეში.

ერთ-ერთი საუკეთესო მაგალითი იმისა თუ რისთვის გვჭირდება სახელების სივრცე არის თვითონ .NET framework. თუ ყველა კლასი განთავსდებოდა გლობალური სახელების სივრცეში, მაშინ ეს გამოიწვევდა უწყვეტობას და გაურკვევლობას! ამის თავიდან ასაცილებლად გვაქვს ძირითადი სახელების სივრცე System კლასების უდიდესი ნაწილისათვის. საბაზო კლასების უმრავლესობა, რომლებიც ზოგადი გამოყენებისთვისაა დანიშნული, მოთავსებულია სახელთა სივრცეში System; სისტემური სახელსივრცე, მოიცავს სისტემის ფუნდამენტურ კლასთა ერთობლიობას;

System.IO ქვე-სახელების სივრცე ინფორმაციის გადაცემა/მიღებისათვის;

System.Net - სახელების სივრცე მონაცემების მოძებნისათვის;

System.Net.Mail - სახელების სივრცე ელექტრონული ფოსტისათვის;

System.Web - სახელების სივრცე ვებ-გვერდებისათვის;

მაგალითად, namespace WindowsMyFirst არის ჩვენს მიერ, შექმნილი სახელსივრცე, აერთიანებს ჩვენ მიერ დაწერილ მეთოდებს და გამოიცხავს კონფლიქტს სხვა სახელსივრცეში არსებულ იმავე დასახელების მეთოდთან;

System.Windows.Forms - გრაფიკული ვინდოუს-აპლიკაციის საჭირო საბაზო კლასების სახელსივრცე. ინახება System.Windows.Forms.dll ფაილში;

System.Windows.Forms სახელების სივრცეში გამოცხადებულია კლასები, რომლებიც ვიზუალურ და არავიზუალურ მმართველ ელემენტებს აღწერს. ამიტომ, როცა Windows Forms Application ტიპის პროგრამა-დანართს ვქმნით, კოდის ფაილის დასაწყისში მოთავსებულ using დირექტივებს შორის ერთ-ერთი using System.Windows.Forms დირექტივა იქნება.

მართვის ელემენტების უმრავლესობა System.Windows.Forms.Control კლასის მემკვიდრეა. ამ კლასში მართვის ელემენტების ძირითადი ფუნქციონალური შესაძლებლობებია განსაზღვრული. ამიტომ, მართვის ელემენტების თვისებებისა და მოვლენების უმრავლესობა ერთნაირია.

using დირექტივა. ძალზე დამღლელია, ყოველთვის მივუთითოთ სახელების სივრცე, როცა პროგრამაში ხდება ხშირი მიმართვები სახელების სივრცის წევრებთან, ასეთ შემთხვევებში, სასურველია using დირექტივის გამოყენება. ჩვენს მიერ განხილული ყველა პროგრამისათვის System სახელების სივრცე ხილული ხდება using დირექტივაში მისი მითითების შემდეგ.

using დირექტივა შეგვიძლია, აგრეთვე, გამოვიყენოთ იმისათვის, რომ ხილული გავხადოთ ჩვენ მიერ შექმნილი სახელების სივრცე.

using დირექტივის სინტაქსია: **using სახელების_სივრცის_სახელი**

ყველა წევრი, განსაზღვრული მითითებული სახელების სივრცის ფარგლებში, ხდება მისაწვდომი ე. ი. მისაწვდომია მიმდინარე სახელების სივრცის ნაწილი და შეიძლება გამოყენებული იყოს ადგილმდებარეობის სრული მითითების გარეშე. **using** დირექტივა მითითებული უნდა იყოს პროგრამის საწყისი კოდის შემცველი ფაილის დასაწყისში, ყველა სხვა გამოცხადების წინ.

ჩადგმული სახელების სივრცე. ერთი სახელების სივრცე შეიძლება მოვათავსოთ მეორის შიგნით. ასეთი გზით შეგვიძლია შევქმნათ სახელების სივრცის იერარქია.

სახელების სივრცის იერარქია შეიძლება გადანაწილებული იყოს სხვადასხვა პროგრამაში (სხვადასხვა ფაილში). ასეთი გადანაწილების უპირატესობა ისაა, რომ პროგრამისტებს შეუძლიათ ცალ-ცალკე იმუშაონ ამ პროგრამებზე და შემდეგ ერთდროულად შეასრულონ მათი კომპილაცია.

სახელების ჩადგმული სივრცე შეგვიძლია გამოვაცხადოთ ერთ სტრიქონში, ერთი namespace საკვანძო სიტყვის გამოყენებით. მაგალითად, კოდის ფრაგმენტი:

```
namespace OuterNS
{
    namespace InnerNS
    {
        //კოდი..
    }
}
```

შეგვიძლია ასე გადავწეროთ:

```
namespace OuterNS.InnerNs
{
    // კოდი
}
```

ახლა კი განვიხილოთ სახელსივრცეების გამოყენების სხვადასხვა

მაგალითები:

მაგალითი 1. ჩვეულებრივი სახელების სივრცე

```
namespace OneSpace
{
    class One
    {
        public string ShowText()
        {
            return "დილა მშვიდობისა!";
        }
    }
}
```



```

    }
}

private void button1_Click(object sender, EventArgs e){
    OneSpace.One O = new OneSpace.One();
    label1.Text=O.ShowText();
}

```

მაგალითი 2. ჩადგმული სახელების სივრცე

```

namespace OneSpace
{
    namespace TwoSpace
    {
        class One
        {
            public string ShowText()
            {
                return "დილა მშვიდობის!";
            }
        }
    }
}

```

```

private void button1_Click(object sender, EventArgs e) {
    OneSpace.TwoSpace.One O = new OneSpace.TwoSpace.One();
    label1.Text=O.ShowText();
}

```

მაგალითი 3. ჩადგმული სახელების სივრცე . ოპერატორის დახმარებით

```

namespace OneSpace.TwoSpace
{
    class One
    {
        public string ShowText()
        {
            return "დილა მშვიდობის!";
        }
    }
}

```

```

    }
    private void button1_Click(object sender, EventArgs e) {
        OneSpace.One O = new OneSpace.One();
        label1.Text=O.ShowText();
    }

```

მაგალითი 4. using დირექტივა

```

namespace OneSpace
{
    class One
    {
        public string ShowText()
        {
            return "დილა მშვიდობისა!";
        }
    }
}
using OneSpace;
private void button1_Click(object sender, EventArgs e) {
    //OneSpace.TwoSpace.One O = new OneSpace.TwoSpace.One(); შეცდომა!!!
    One O = new One();
    label1.Text=O.ShowText();
}

```

ახლა გადავიდეთ ამოცანების განხილვაზე.

ამოცანა 11.1. ელექტროენერგიის დანახარჯის გადასახადის გამოთვლა ერთი თვის განმავლობაში. გამოვიყენოთ ინტერფეისი ჯამური გადასახადის დასთვლელად (იხ. ნახ. 47.).

```

public interface IresultShow {
    // ushort - უნიშნო მოკლე მთელი რიცხვი (დიაპაზონი 0 ÷ 65535)
    ushort ResultShow();
}
public interface ISum {
    // decimal - ციფრული ტიპი საფინანსო გამოთვლებისათვის (დიაპაზონი 1E-28 ÷ 7.9E+28)
    decimal Sum();
}

```

```

public class Counter : IResultShow, ISum {
    ushort showStart;
    ushort showEnd;
    decimal price;
    public Counter(ushort arg0, ushort arg1, decimal arg2) {
        showStart = arg0;
        showEnd = arg1;
        price = arg2;
    }
    public ushort ResultShow() {
        return (ushort)(showEnd - showStart);
    }
    public decimal Sum() {
        return Math.Round((decimal)(ResultShow() * price), 2);
    }
}

private void button1_Click(object sender, EventArgs e){
    ushort show_start = ushort.Parse(textBox1.Text);
    ushort show_end = ushort.Parse(textBox2.Text);
    decimal price = decimal.Parse(textBox3.Text);
    Counter C = new Counter(show_start, show_end, price);
    label5.Text = "ჯამური გადასახადი " + C.Sum() + " ლარი"; }

```

ნახ. 47. ამოცანა 11.1-ის შესრულების შედეგი

ამოცანა 11.2. ცხადი რეალიზების დემონსტრირების ამოცანა.

```

public interface IA //ცხადი რეალიზების დემონსტრირება
{
    void Message();
}

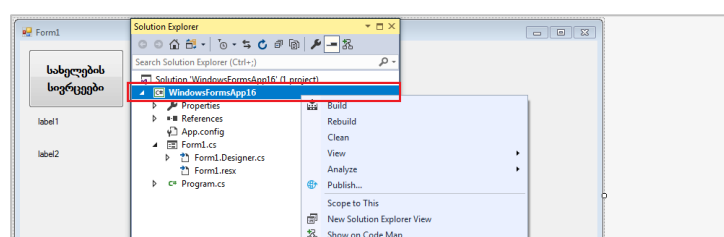
```

```

    }
    public interface IB
    {
        void Message();
    }
class One : IA, IB
{
    /* ცხადი რეალიზება: ინტერფეისის სრული სახელი -> წერტილი -> მეთოდის სახელი.
    ყურადღება!!! ცხადი რეალიზაციის დროს მოდიფიკატორების მითითება არ შეიძლება */
    void IA.Message() {
        MessageBox.Show("დილა მშვიდობისა საქართველო!");
    }
    void IB.Message() {
        MessageBox.Show("მალიან მიყვარხარ საქართველო!");
    }
}
private void button1_Click(object sender, EventArgs e) {
    //კლასის ობიექტიდან არ გვაქვს წვდომა მეთოდთან
    One o = new One();
    //შევქმნათ მიმართვა ინტერფეისზე IA
    IA Aref = 0;
    //მეთოდთან წვდომა მიმართვის საშუალებით
    Aref.Message();
    //შევქმნათ მიმართვა ინტერფეისზე IB
    IB Bref = 0;
    //მეთოდთან წვდომა მიმართვის საშუალებით
    Bref.Message();
}

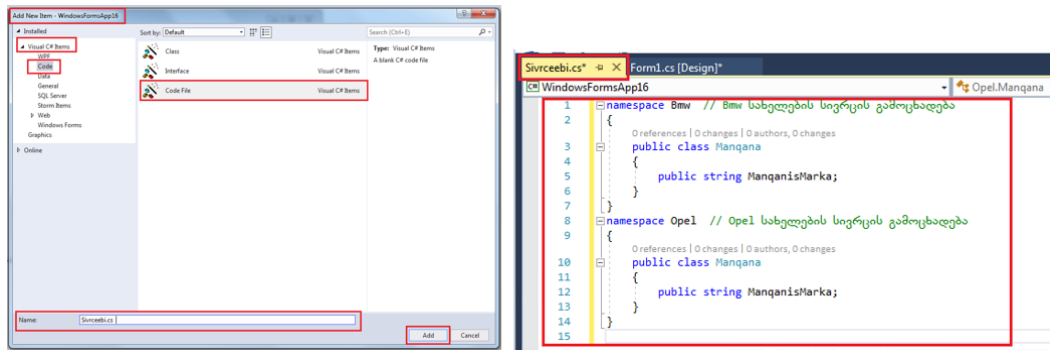
```

ამოცანა 11.3. პროგრამაში ხდება სახელების სივრცესთან მუშაობის დემონსტრირება. სახელების სივრცე Bmw და Opel მივუთითოთ ცალკე ფაილში. ამისათვის პროექტის დასახელებაზე (ფანჯარა Solution Explorer) დავაჭიროთ მაუსის მარჯვენა ღილაკს და ავირჩიოთ Add -> New Item (იხ. ნახ. 48.):



ნახ. 48. სახელების სივრცის ცალკე ფაილში ასახვა

შემდეგ, მოვნიშნოთ Visual C# Items განყოფილების ელემენტი Code File, ველში Name შევიტანოთ ფაილის სახელი, მაგალითად, Sivriceebi.cs ან დავტოვოთ არსებული სახელი. დავაჭიროთ Add ღილაკს და დავიწყოთ სახელების სივრცის ჩაწერა (იხ. ნახ. 49.):



ნახ. 49. სახელების სივრცის ცალკე ფაილში ასახვა და კოდის ჩაწერა

```
namespace Bmw // Bmw სახელების სივრცის გამოცხადება
{
    public class Manqana
    {
        public string ManqanisMarka;
    }
}

namespace Opel // Opel სახელების სივრცის გამოცხადება
{
    public class Manqana
    {
        public string ManqanisMarka;
    }
}
```

```
private void button1_Click(object sender, EventArgs e) {
    Bmw.Manqana ChemiManqana1 = new Bmw.Manqana();
    ChemiManqana1.ManqanisMarka = "BMV";
    label1.Text = ChemiManqana1.ManqanisMarka;
    Opel.Manqana ChemiManqana2 = new Opel.Manqana();
    ChemiManqana2.ManqanisMarka = "OPEL";
    label2.Text = ChemiManqana2.ManqanisMarka;
}
```

პროგრამაში განსაზღვრულია ორი კლასი, რომლებსაც ერთნაირი სახელები აქვთ - Manqana. იმისათვის, რომ ეს ორი კლასი ერთმანეთისაგან განვასხვავოთ, საჭიროა კლასის სახელის წინ სახელების სივრცის მითითება. ობიექტის შექმნის შემდეგ არ არის აუცილებელი სახელების სივრცის მითითება ობიექტის სახელის ან მისი წევრის წინ. მაგალითად: ChemiManqana1.ManqanisMarka = "BMW";

ამოცანა 11.4. ქვემოთ მოცემულია პროგრამა, რომელშიც using დირექტივა გამოიყენება იმისთვის, რომ System და Opel სახელების სივრცე ხილული გავხადოთ.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Opel;
namespace Opel // სახელსივრცის გამოცხადება Opel
{
    public class Manqana
    {
        public string ManqanisMarka;
    }
}
namespace WindowsFormsApp15
{
    public partial class Form1 : Form
```

```

{
    public Form1()
    {
        InitializeComponent();
    }
private void button1_Click(object sender, EventArgs e) {
    // Opel სიტყვის მითითება არ არის აუცილებელი. ობიექტის შექმნა
    Manqana chemimanqana = new Manqana();
    chemimanqana.ManqanisMarka = "OPEL";
    label1.Text = chemimanqana.ManqanisMarka;
}
}
}

```

დამოუკიდებელი სამუშაო

1. განხილული ამოცანა 11.1. გადააკეთეთ ისეთი სახით, რომ მან დათვალოს არა ერთი თვის, არამედ მთელი წლის ელექტროენერგიის ხარჯი. გამოიყენეთ ინტერფეისი და ყველა ის ცოდნა, რასაც გაეცანით წინა თემებიდან.

2. ქვემოთ მოცემულია პროგრამული კოდი: ა) კომენტარის სახით ჩაწერეთ თითოეული სტრიქონის ახსნა-განმარტება; ბ) დაადგინეთ, არის თუ არა ინტერფეისის ცხადი რეალიზება; გ) ააგეთ ფორმა, შეასრულეთ კოდი და გადმოიტანეთ შესრულების შედეგი;

```

interface IName {
    string type();
}
class A : IName {
    public string type()
    {
        return "A";
    }
}
class B : IName {
    public string type()

```

12

```
{
    return "B";
}

class C : IName {
    public string type()
    {
        return "C";
    }
}

private void button1_Click(object sender, EventArgs e) {
    label1.Text = "";
    IName[] imass = { new A(), new B(), new C() };
    foreach (IName i in imass) label1.Text +=(i.type()) + " ";
}
```

თვისებების არსი. get და set მეთოდები. მეთოდისთვის ობიექტის გადაცემა. მეთოდის მიერ ობიექტის დაბრუნება.

C# ენაში ჩვეულებრივი მეთოდების გარდა გვაქვს სპეციალური მეთოდები, რომლებთაც თვისებები ეწოდებათ. თვისებები საკმაოდ მარტივად უზრუნველყოფენ კლასის და სტრუქტურის ელემენტებზე/წევრებზე წვდომას. თვისება არის კლასის წევრი, რომელიც გამოიყენება კლასის მონაცემთა ჩასაწერად და მისაღებად. მნიშვნელოვანია, გვახსოვდეს, რომ თვისება არასდროს გამოიყენება მონაცემების შესანახად, ის მოქმედებს როგორც ინტერფეისი. თვისება მეორენაირად შეგვიძლია წარმოვიდგინოთ როგორც კლასის წევრების გაფართოება, რომელიც საშუალებას გვაძლევს, წავიკითხოთ, ჩაწეროთ ან შეცვალოთ ამ ცვლადების მნიშვნელობები.

თვისება შეიცავს კოდის ერთ ან ორ ბლოკს, რომელსაც აქსესორს უწოდებენ. ბლოკების ტანში გვაქვს get და set მეთოდები. get და set მეთოდების გამოყენებით,

შეგვიძლია შევცვალოთ კლასის შიგა რეალიზება/განხორციელება მოთხოვნებიდან გამომდინარე. თვისების ზოგადი სინტაქსი შემდეგია:

```
[მოდიფიკატორი] თვისების_ტიპი თვისების დასახელება
{
    get
    { // თვისების მნიშვნელობის დაბრუნება, ველის წაკითხვა
        // მოქმედება, რომელიც სრულდება თვისების მნიშვნელობის
        მიღებისას }
    set
    { // ახალი მნიშვნელობის დადგენა/დაყენება, მნიშვნელობის ჩაწერა
        // მოქმედება, რომელიც სრულდება თვისების მნიშვნელობის
        დადგენისას }
}
```

თვისების განსაზღვრისას თავიდან იწერება ხილვადობის წვდომის სხვადასხვა მოდიფიკატორი. შემდეგ მოდის თვისების ტიპი და თვისების სახელი. თვისების სრულყოფილი განსაზღვრა კი ხდება get და set ბლოკებში. ბლოკში get სრულდება თვისების მნიშვნელობის მიღება. ამავე ბლოკში ოპერატორ return-ის დახმარებით კი ვაბრუნებთ რაიმე შედეგს. ბლოკში set სრულდება თვისების მნიშვნელობის განსაზღვრა/დადგენა. ამავე ბლოკში პარამეტრ value-ს დახმარებით ვღებულობთ იმ მნიშვნელობას, რომელიც გადაეცემა თვისებას.

ბლოკები get და set იწოდება როგორც აქსესორები ანუ წვდომის მეთოდები. ზოგჯერ კი მათ „გეტერებსაც“ და „სეტერებსაც“ უწოდებენ.

მოდით, ახლა კიდევ უფრო კონკრეტულად განვიხილოთ თვისება ქვემოთ მოცემული ჩანაწერის მიხედვით:

```
public int StudentName {
    get
    {
        // შიგა ველის/ცვლადის წაკითხვის კოდი
    }
    set
    {
```

// შიგა ველში/ცვლადში ჩაწერის კოდი

}

}

სადაც public - ხილვადობის წვდომის მოდიფიკატორია; int - თვისების მიერ დასაბრუნებელი მნიშვნელობის ტიპია; get - აქსესორია, რომელიც გამოიყენება კლასის შიგა წვერიდან მნიშვნელობის წასაკითხად/ამოსაკითხად; set - აქსესორია, რომელიც გამოიყენება კლასის შიგა წვერში მნიშვნელობის ჩასაწერად; ამასთანავე უნდა ვიცოდეთ, რომ აქსესორია set იღებს არაცხად პარამეტრს value; არაცხადი პარამეტრი value კი თავისთავად იღებს იმ მნიშვნელობას, რომელიც ენიჭება თვისებას.

თუ თვისების სახელი ან გამოსახულებაში ან მინიჭების ოპერატორში (ოპერატორი ტოლობა „=“) გვხვდება მარჯვენა მხარეს, მაშინ ხდება get აქსესორის გამოძახება:

```
variable = obj.StudentName; // get
```

თუ თვისების სახელი ან გამოსახულებაში ან მინიჭების ოპერატორში (ოპერატორი ტოლობა „=“) გვხვდება მარცხენა მხარეს, მაშინ ხდება set აქსესორის გამოძახება:

```
obj.StudentName = expression; // set
```

აქ obj არის იმ კლასის ეგზემპლარის სახელი, რომელ კლასშიც არის გამოცხადებული თვისება StudentName.

თვისების აღწერისას ძალზე მნიშვნელოვანია განვსაზღვროთ წვდომის მოდიფიკატორი და დასაბრუნებელი მნიშვნელობის ტიპი, რათა პროგრამისტმა შეძლოს საჭიროებისამებრ ცვლილებების შეტანა კლასის ველებში/ცვლადებში.

თვისების კონსტრუქციაში, როცა თვისების წაკითხვა ხდება (მნიშვნელობის დაბრუნება) get მეთოდის კოდი იცვლება ყოველ ჯერზე, ხოლო set მეთოდის კოდი შესრულდება მაშინ, როდესაც თვისების გადასაზღვრა/ხელახლა განსაზღვრა მოხდება სხვა მნიშვნელობით.

მონაცემთა ასეთი სტრუქტურა იყოფა სამ ვარიანტად/ტიპად:

1. წაკითხვა-ჩაწერა (Read-Write). ასე იწოდება ის თვისება, რომელიც შეიცავს აქსესორების ორივე ბლოკს ანუ get და set მეთოდებს.

2. მხოლოდ წაკითხვისათვის (Read-Only). ასე იწოდება ის თვისება, რომელიც შეიცავს ერთ აქსესორ ბლოკს ანუ get მეთოდს.

3. მხოლოდ ჩაწერისათვის (Write-Only). ასე იწოდება ის თვისება, რომელიც შეიცავს ერთ აქსესორ ბლოკს ანუ set მეთოდს.

ამასთანავე მნიშვნელოვანია ვიცოდეთ, რომ თვისებას არ გააჩნია პარამეტრები, ხოლო მონაცემთა ტიპი ზუსტად იმ ველის ტიპს უნდა ემთხვეოდეს, რომელთათვისაც იქმნება თვისება. თვისებების გამოყენებისას უნდა დავიცვათ შემდეგი შეზღუდვები:

- თვისება არ შეიძლება გადაეცეს მეთოდს, როგორც ref ან out პარამეტრი;
- თვისება არ შეიძლება იყოს გადატვირთული;
- თვისებამ არ უნდა შეცვალოს ცვლადის მნიშვნელობა get მეთოდის გამოყენებით.

მოკლედ რომ ვთქვათ, თვისება არის კლასის წევრი, რომელიც საშუალებას გვაძლევს მეთოდების საშუალებით მივიღოთ ან შევცვალოთ ცვლადების (ველების) მნიშვნელობები. დაპროგრამების ზოგიერთ ენაში თვისება შეიძლება იყოს მხოლოდ ცვლადი, რაც შეეხება C# ენას, თვისება განსხვავდება ცვლადისაგან და წარმოადგენს მონაცემთა განსხვავებულ სტრუქტურას. ვთქვათ მოცემული გვაქვს კლასი window, რომელსაც უნდა დავამატოთ თვისება.

```
class window {  
    public int width;  
    public int height; }
```

ჯერჯერობით კლასში გამოვაცხადეთ ორი ცვლადი, ისინი ჯერ კიდევ არ წარმოადგენენ თვისებას. ვხედავთ, რომ ცვლადები არის ღია, ეს შეიძლება, მაგრამ გარკვეული შემთხვევების გარდა, სასურველი არ არის.

ძირითადად გამართულ პროგრამულ კოდებში კლასის ველები/ცვლადები უნდა დარჩეს დახურული, ხოლო იმისათვის რომ ისინი გავხადოთ ღია გვევმარება თვისება. ასეთ შემთხვევაში თვისება შეგვიძლია გამოვაცხადოთ ასე:

```
public int Width  
{  
    get  
    {  
        return width;  
    }  
    set  
    {  
        width = value;  
    }  
}
```

}

თვისება იქმნება სწორედ იმიტომ, რომ მისი მიმართვის მოდიფიკატორი არის ღია, ამიტომაც თვისების გამოცხადება ყოველთვის იწყება სიტყვით `public`. შემდეგ მოდის თვისების ტიპი (`int`) და სახელი (`Width`).

ზოგადად მიღებულია, რომ ცვლადის სახელი დაიწყოს ქვედა რეგისტრის სიმბოლოთი, ხოლო თვისების სახელის პირველი სიმბოლო კი ზედა რეგისტრის სიმბოლოთი. თვისების აღწერის შემდეგ არ იწერება მძიმე, იხსნება ფიგურული ფრჩხილი, სადაც ხდება ორი მეთოდის `get` და `set` რეალიზება. `get` მეთოდისათვის სრულდება ოპერატორი `return`, რომლის შემდეგაც ვუთითებთ იმ ცვლადს, რომელის მნიშვნელობაც უნდა დააბრუნოს თვისებამ. ჩვენს შემთხვევაში თვისება (`Width`) აბრუნებს `width` ცვლადის მნიშვნელობას ე. ი. თვისებასთან მიმართვის დროს (`get` მეთოდის შემთხვევაში) ჩვენ ვხედავთ `width` ცვლადის მნიშვნელობას.

`set` მეთოდისათვის უნდა მოვახდინოთ თვისების მნიშვნელობის შენახვა რომელიღაც ცვლადში. თვისების მნიშვნელობა ჩაწერილია ვირტუალურ (არაცხად) ცვლადში `value`. რატომ ვირტუალური/არაცხადი? `value` არსად არ არის გამოცხადებული და ყოველთვის არსებობს ფიგურულ ფრჩხილებში `set` საკვანძო სიტყვის შემდეგ. `value`-ს აქვს იგივე ტიპი, რაც თვისებას. თვისების შეცვლის შემთხვევაში თვისების მნიშვნელობა ინახება `width` ცვლადში.

ქვემოთ მოცემულ კოდში კი დაგვემატა შემოწმება: ფანჯრის სიგანე არ შეიძლება იყოს უარყოფითი, ნული ან 100-ზე მეტი. სწორედ თვისება ასეთი კონტროლის სახით იცავს კლასის დახურულ წევრს.

```
public int Width {  
    get  
    {  
        return width;  
    }  
    set  
    {  
        if (value > 0 && value < 100)  
            width = value;  
    }  
}
```

კლასის თვისების გამოცხადება შემდეგი სახისაა: ობიექტის_სახელი.თვისება

```
class window {  
    int width;  
    int height;  
    public int Width {  
        get  
        {  
            return width;  
        }  
        set  
        {  
            if (value > 0 && value < 100)  
                width = value;  
        }  
    }  
}
```

ქვემოთ მოცემული ფრაგმენტი აჩვენებს თვისების გამოცხადებას და გამოყენებას.

```
window obiecti = new window();  
objecti.Width = 10;  
int sigane = objecti.Width;  
label1.Text = sigane.ToString();
```

თავდაპირველად ვქმნით window კლასის ობიექტს.

მეორე სტრიქონში ფანჯრის სიგანეს ვანიჭებთ მნიშვნელობას 10-ს.

მესამე სტრიქონში კი ვკითხულობთ სიმაღლისათვის მინიჭებულ მნიშვნელობას და ვიმახსოვრებთ ცვლადში sigane. არსებობს ავტომატური სახის თვისება, რომელიც გაცილებით მარტივია. იგი ავტომატურად განსაზღვრავს მახასიათებლებში ადგილს, მაგრამ არ გვადლევს საშუალებას, ლოგიკური კოდი დავწეროთ. მისი გამოცხადება ხდება შემდეგი სახით:

```
მიმართვის_მოდული {  
    ტიპი თვისების_სახელი { get; set; }  
}
```

აქ get და set მეთოდებს არ გააჩნიათ ტანი:

```
class Book {  
    public int Price { get; set; }  
}
```

```

    }

    .....

    Book book = new Book();
    book.Price = 1200;
    label1.Text = book.Price.ToString();

```

ახლა კი განვიხილოთ მაგალითი, სადაც აღწერილია თვისების ორივე მეთოდი:

```

class ItemColors
{
    private string color;
    public string Color
    {
        get { return color; }
        set { color = value; }
    }
}

```

კოდის ფრაგმენტში აღწერილია თვისება „Color“. თვისება შეიცავს get მეთოდს (მნიშვნელობის დასაბრუნებლად) და set მეთოდს (მნიშვნელობის ხელახლა განსაზღვრისათვის).

მეთოდში get ჩვენ ან ვაბრუნებთ ან გამოვითვლით, ხოლო შემდეგ ვაბრუნებთ წევრის/ველის მნიშვნელობას. ამასთანავე მნიშვნელოვანია ვიცოდეთ, რომ get და set მეთოდები არ გამოიყენება ობიექტის მდგომარეობის შესაცვლელად! ამ მეთოდების დახმარებით, უბრალოდ, შეგვიძლია გავაფართოვოთ ცვლადების „ქცევა“:

```

class UserName {
    private string fullname = "georgia, caucasus";
    public string FullName
    {
        get
        {
            return fullname.ToUpper();
        }
        set
        {
            if (value == "georgia, caucasus")

```

```

        fullname = value;
    }
}

private void button1_Click(object sender, EventArgs e) {
    UserName u_obj = new UserName();
    u_obj.FullName = "Tbilisi";    //set
    label1.Text = u_obj.FullName; //get
    // შესრულების შედეგია: GEORGIA, CAUCASUS
}

```

ახლა კი ნამდვილად გაგვიჩნდება ასეთი შეკითხვა: რა საჭიროა თვისება? თავისუფლად შეგვიძლია გამოვიყენოთ კლასის ცვლადი ყოველგვარი დამატებითი თვისების გარეშე. მართალია, მაგრამ თვისება გვადლევს საშუალებას, ცვლადზე დავაწესოთ ერთგვარი დამატებითი კონტროლი ან ლოგიკა:

```

class Human {
    private int years;
    public int Years
    {
        set
        {
            if (value < 18)
            {
                MessageBox.Show("დაშვებულია 18 წელის ზევით ასაკის!");
            }
            else
            {
                years = value;
            }
        }
        get { return years; }
    }
}

private void button1_Click(object sender, EventArgs e) {
    Human u_obj = new Human();
}

```

```

        u_obj.Years = 45; // set
        label1.Text = u_obj.Years.ToString(); // get
    }

```

ქვემოთ განხილულია თვისების Read-Write მაგალითი:

```

class რიცხვი {
    int _რიცხვი;
    public int რიცხვია
    {
        get
        {
            return this._რიცხვი;
        }
        set
        {
            this._რიცხვი = value;
        }
    }
}

private void button1_Click(object sender, EventArgs e) {
    რიცხვი obieqti = new რიცხვი();
    obieqti.რიცხვია = 777; // set { }
    label1.Text = obieqti.რიცხვია.ToString(); // get { }
}

```

ქვემოთ განხილულია თვისების Read-only მაგალითი:

```

class ადამიანი {
    private string სახელი;
    private string ადგილი;
    public ადამიანი(string a, string b) {
        სახელი = a;
        ადგილი = b;
    }
    public string სახელი_get {

```



```

        get
        {
            return სახელი;
        }
    }

    public string ადგილი_get {
        get
        {
            return ადგილი;
        }
    }
}

private void button1_Click(object sender, EventArgs e) {
    ადამიანი obieqti = new
    ადამიანი("ვაჟა-ფშაველა", "ჩარგალი, საქართველო, კავკასია");
    label1.Text = "სახელი: " + obieqti.სახელი_get;
    label2.Text = "მისამართი: " + obieqti.ადგილი_get;
}

```

ამ მაგალითში გამოვიყენეთ თვისების წვდომის მხოლოდ ერთი მეთოდი get - ანუ თვისება ხელმისაწვდომია მხოლოდ წაკითხვისათვის. არ გვაქვს საშუალება, რათა დავაყენოთ/ მივანიჭოთ ახალი მნიშვნელობა set მეთოდის საშუალებით.

განვიხილოთ თვისების Write-only მაგალითი:

```

class Persona {
    private string _saxeligvari;
    public string SaxeliGvari
    {
        set
        {
            _saxeligvari = value;
        }
    }

    private string _misamarTi;
    public string MisamarTi

```

```

    {
        set
        {
            _misamarTi = value;
        }
    }
    public void Detalebi()
    {
        MessageBox.Show("სახელი და გვარი: " + _saxeligvari);
        MessageBox.Show("მისამართი: " + _misamarTi);
    }
}
private void button1_Click(object sender, EventArgs e) {
    Persona obieqti = new Persona();
    obieqti.SaxeliGvari = "ვაჟა ფშაველა";
    obieqti.MisamarTi = "ჩარგალი, საქართველო, კავკასია";
    // label1.Text = obieqti.SaxeliGvari; ასეთი ჩაწერა შეცდომაა!!!
    obieqti.Detalebi();
}

```

ამ მაგალითში კი მოვიქცით პირიქით: გამოვიყენეთ თვისების წვდომის მხოლოდ ერთი მეთოდი set - ანუ თვისება ხელმისაწვდომია მხოლოდ ჩაწერისათვის. არ გვაქვს საშუალება, დავაბრუნოთ მნიშვნელობა.

მეთოდისთვის ობიექტის გადაცემა. მეთოდს პარამეტრებად ჩვეულებრივი ტიპების მნიშვნელობების გარდა, შეგვიძლია გადავცეთ ობიექტებიც. C# ენაში არსებობს ორი ძირითადი გზა, რომლითაც შეიძლება არგუმენტი გადაეცეს მეთოდს.

პირველი გზა არის მეთოდისთვის არგუმენტის გადაცემა მნიშვნელობის მიხედვით (passing by value). ამ შემთხვევაში ხდება არგუმენტის მნიშვნელობის გადაწერა მეთოდის პარამეტრში. ასეთ შემთხვევაში, პარამეტრის ყველა ცვლილება, რომელიც მოხდება მეთოდში, არ იმოქმედებს არგუმენტზე, რომელიც გამოძახებისას გამოიყენება ანუ მეთოდში პარამეტრის შეცვლა არ გამოიწვევს არგუმენტის შეცვლას. მეთოდისთვის ჩვეულებრივი ტიპის მქონე მნიშვნელობების გადაცემისას,

როგორცაა int ან double, გამოიყენება გადაცემა მნიშვნელობის მიხედვით. შედეგად, პარამეტრის ცვლილება არ გამოიწვევს არგუმენტის ცვლილებას.

მეორე გზა არის მეთოდისთვის არგუმენტის გადაცემა მიმართვის მიხედვით (passing by reference). ამ შემთხვევაში, მეთოდის პარამეტრს გადაეცემა არგუმენტის მიმართვა (არგუმენტის მისამართი). ამრიგად, თუ შევცვლით მეთოდის პარამეტრს, შესაბამისად შეიცვლება არგუმენტი, რომელიც გამოვიყენეთ მეთოდის გამოძახებისას.

მეთოდს ობიექტი პირდაპირ არ გადაეცემა. მეთოდს გადაეცემა ობიექტზე მიმართვა. როგორც ძირითადი ტიპების შემთხვევაში, ობიექტის მიმართვის მეთოდზე გადაცემისას, ეს მიმართვა გადაიწერება მეთოდის პარამეტრში. ეს ნიშნავს, რომ მეთოდის პარამეტრი შეიცავს ზუსტად იმავე მნიშვნელობას, რასაც არგუმენტი, რომელიც გადაეცა მეთოდს.

განვიხილოთ რამდენიმე ამოცანა.

ამოცანა 12.1. მოცემულია კლასი TVSet, კლასში გამოცხადებულია დახურული ცვლადი volume, რომელიც აღწერს ხმის სიმძლავრეს. ამ ცვლადთან მიმართვისათვის გამოიყენეთ თვისება Volume. გაითვალისწინეთ, რომ ხმის სიმძლავრე მოთავსებულია დიაპაზონში 1-დან 100-მდე.

```
class TVset {  
    private int volume; //დახურული ველის გამოცხადება  
    public int Volume // თვისების გამოცხადება  
    {  
        get //ველიდან წაკითხვა  
        {  
            return volume;  
        }  
        set //ველში ჩაწერა  
        {  
            if (value < 1) volume = 1;  
            else  
                if (value > 100) volume = 100;  
            else volume = value;  
        }  
    }  
}
```

```

    }

    private void button1_Click(object sender, EventArgs e) {
        TVset tv = new TVset();
        label1.Text = "ტელევიზორის ხმის სიმძლავრე";
        tv.Volume = int.Parse(textBox1.Text);
        label2.Text = "მიმდინარე ხმის სიმძლავრე = " + v.Volume.ToString();
    }

```

ამოცანა 12.2. თვისების რეალიზება. მოცემულობა დაწერეთ დამოუკიდებლად.

```

class UserInfo
{
    int a;
    string s = "ლუკა";

    public int Age // თვისების გამოცხადება
    {
        get
        {
            if (a <= 0)
                return 1;
            return a;
        }
        set
        {
            a = value;
        }
    }

    public string Name // თვისება მხოლოდ წაკითხვისათვის
    {
        get
        {
            return s;
        }
    }
}

private void button1_Click(object sender, EventArgs e)

```

```

{
    UserInfo ui = new UserInfo();
    // დახურულ ცვლადთან მიმართვა თვისების გამოყენებით
    ui.Age = 26;
    label1.Text = ui.Age.ToString();
    // ცვლადი მხოლოდ წაკითხვისათვის
    string myName = ui.Name;
    label2.Text = myName;
}

```

ამოცანა 12.3. მეთოდისათვის ობიექტის გადაცემა. შექმენით Samkutxedi კლასი. მასში შექმენით მეთოდი, რომელსაც გადაეცემა ამავე ტიპის ობიექტი და რომელიც შეასრულებს ამ ობიექტის ფართობის გამოთვლას. კლავიატურიდან შეიტანეთ სამკუთხედის გვერდების შემდეგი მნიშვნელობები: 7, 8, 9;

ამოცანა 12.3-ის პირველი ვარიანტი:

```

class Samkutxedi {
    int gverdi1, gverdi2, gverdi3;
    public Samkutxedi(int par1, int par2, int par3)
    {
        gverdi1 = par1;
        gverdi2 = par2;
        gverdi3 = par3;
    }
    public double Gamotvla(Samkutxedi Sam1) {
        double nper, fartobi;
        nper = (Sam1.gverdi1 + Sam1.gverdi2 + Sam1.gverdi3)/2;
        fartobi = Math.Sqrt(nper * (nper - Sam1.gverdi1) *
            (nper - Sam1.gverdi2) * (nper - Sam1.gverdi3));
        return fartobi;
    }
}

private void button1_Click(object sender, EventArgs e) {
    int arg1 = int.Parse(textBox1.Text);
    int arg2 = int.Parse(textBox2.Text);
    int arg3 = int.Parse(textBox3.Text);
}

```

```

    Samkutxedi obieqti1 = new Samkutxedi(arg1, arg2, arg3);
    Samkutxedi obieqti2 = new Samkutxedi(arg1 + 1, arg2 + 2, arg3 + 3);
    double shedegi1 = obieqti1.Gamotvla(obieqti1);
    double shedegi2 = obieqti2.Gamotvla(obieqti2);
    label1.Text = Math.Round(shedegi1, 2).ToString();
    label2.Text = Math.Round(shedegi2, 2).ToString();
}

```

ამოცანა 12.3-ის მეორე ვარიანტი:

```

class Triangle {
double a, b, c; // სამკუთხედის გვერდები
// კონსტრუქტორი
public Triangle(double _a, double _b, double _c) {
    if (!IsTriangle()) return; // IsTriangle მეთოდის გამოძახება
    a = _a;
    b = _b;
    c = _c;
}

// მეთოდი, რომელიც განსაზღვრავს სამკუთხედის ფართობს
public double GetArea() {
    double p, s;
    // IsTriangle მეთოდის გამოძახება და შემოწმება
    if (!IsTriangle()) return 0.0;
    p = (a + b + c) / 2.0;
    s = Math.Sqrt(p * (p - a) * (p - b) * (p - c));
    return s;
}

// მეთოდი, რომელიც განსაზღვრავს სამკუთხედის უტოლობას:
// ნებისმიერი ორი გვერდის სიგრძეთა ჯამი მეტია მესამე გვერდის
სიგრძეზე
public bool IsTriangle() {
    if ((a + b) < c) return false;
    if ((b + c) < a) return false;
    if ((a + c) < b) return false;
}

```

```

        return true;
    }
}

private void button1_Click(object sender, EventArgs e) {
    // Triangle კლასის გამოყენება და მისი მეთოდები
    // კოდში გვაქვს ორი მიმართვა ერთსა და იმავე ობიექტზე
    // tr1 და tr2 მიმართავს ერთსა და იმავე ობიექტს
    Triangle tr1 = new Triangle(5, 6, 3);
    Triangle tr2 = tr1;
    double aa;
    aa = tr1.GetArea(); // aa = 7.483314
    label3.Text = aa.ToString();
    aa = tr2.GetArea(); // aa = 7.483314
    label4.Text = aa.ToString();
    tr2 = new Triangle(7, 7, 7);
    double area;
    area = tr1.GetArea(); // area = 7.483314
    label1.Text = area.ToString();
    bool isTriangle = tr2.IsTriangle(); // isTriangle = true
    area = tr2.GetArea(); // area = 21.217622
    label2.Text = area.ToString();
}

```

ამოცანა 12.4. თვისების გამოყენება წილადი ტიპის მასივის მნიშვნელობების დასაჯამებლად ანუ შესაკრებად. კოდის ბოლოში ხდება შემოწმება ანუ დატესტვა. თვისება აღწერილია სახელით Summary. თვისებაში მეთოდი get კრიზავს და ითვლის მასივის წევრების/ელემენტების მნიშვნელობებს, ხოლო მეთოდი set კი ანიჭებს მასივის წევრებს/ელემენტებს მნიშვნელობებს.

```

class მასივიწილადი {
    double[] მასივი; //წილადი ტიპის მასივი
    public მასივიწილადი(int size) //კონსტრუქტორი
    { //მასივისათვის მეხსიერების უჯრების/ადგილების გამოყოფა
        მასივი = new double[size];
        for (int i = 0; i < size; i++)

```

```

        მასივი[i] = 0.0; //მასივის შესვლა ნულოვანი მნიშვნელობებით
    }
    //კონსტრუქტორი
    public მასივიწილადი(double[] _მასივი) {
        მასივი = new double[_მასივი.Length];
        // მასივის გადაწერა
        for (int i = 0; i < _მასივი.Length; i++)
            მასივი[i] = _მასივი[i];
    }
    //ინდექსაცია, რიგითი ნომრის მიხედვით მასივის წევრის დაბრუნება
    public double this[int index] {
        get
        {
            if ((index >= 0) && (index < მასივი.Length))
                return მასივი[index];
            else
                return 0.0;
        }
    }
    public double Summary // თვისების გამოცხადება
    {
        get
        {
            // ელემენტების შეკრება
            double summ = 0;
            for (int i = 0; i < მასივი.Length; i++)
                summ += მასივი[i];
            return summ;
        }
        set
        {
            //მასივის წევრების თანაბარი განაწილება, მნიშვნელობების მინიჭება
            double წევრი = value/მასივი.Length;

```



```

        for (int i = 0; i < მასივი.Length; i++)
            მასივი[i] = წევრი;
    }
}

private void button1_Click(object sender, EventArgs e) {
    double[] მასივი_მას = { 1.0, 2.7, 2.1, 9.5, 0.0 };
    მასივიწილადი ობიექტი = new მასივიწილადი(მასივი_მას);
    double summ;
    summ = ობიექტი.Summary;
    label1.Text = "მასივის წილადი რიცხვების ჯამია: " + summ.ToString();
    //ობიექტის შევსება ერთიანებით
    ობიექტი.Summary = 1.0 * მასივი_მას.Length;
    double ტესტი;           // გადამოწმება
    ტესტი = ობიექტი[2];     //ტესტი = 1.0
    label2.Text = "შემოწმება : " + ტესტი.ToString();
}

```

ამოცანა 12.5. პროგრამაში ხდება მეთოდის მიერ სტრიქონის დაბრუნების დემონსტრირება.

```

class ChemiKlasi {
    string[] xili = { "ვაშლი", "მსხალი", "ლევდი", "ყურძენი" };
    public string StriqonisDabruneba(int indexi) {
        if ((indexi >= 0) & (indexi < xili.Length))
            return xili[indexi];
        else return "შეტანილია არასწორი ინდექსი";
    }
}

private void button1_Click(object sender, EventArgs e){
    // პროგრამაში ხდება ობიექტის დაბრუნების დემონსტრირება
    ChemiKlasi obieqti = new ChemiKlasi();
    label1.Text = obieqti.StriqonisDabruneba(3);
    label2.Text = obieqti.StriqonisDabruneba(15);
}

```

```
}
```

ამოცანა 12.6. პროგრამაში ხდება მეთოდის მიერ ობიექტის დაბრუნების დემონსტრირება. მოცემულობა დაწერეთ დამოუკიდებლად.

```
class ChemiKlasi_1 {
    public string xilis_saxeli;
    public int kodi;
    public ChemiKlasi_1(string par1, int par2)
    {
        xilis_saxeli = par1;
        kodi = par2;
    }
}

class ChemiKlasi_2 {
    string[] xili = { "ვაშლი ", "მსხალი ", "ლევდი ", "ყურბენი " };
    int[] kodebi = { 1, 0, 3, 7 };
    //
    public ChemiKlasi_1 ObieqtisDabruneba(int indexi) {
        if ((indexi >= 0) && (indexi < xili.Length))
            // ChemiKlasi_1 ობიექტის დაბრუნება
            return new ChemiKlasi_1(xili[indexi], kodebi[indexi]);
        else return new ChemiKlasi_1("შეტანილია არასწორი კოდი ", indexi);
    }
}

private void button1_Click(object sender, EventArgs e){
    ChemiKlasi_2 obieqti1 = new ChemiKlasi_2();
    ChemiKlasi_1 obieqti2;
    obieqti2 = obieqti1.ObieqtisDabruneba(3);
    label1.Text = obieqti2.xilis_saxeli +
                    obieqti2.kodi.ToString();
    obieqti2 = obieqti1.ObieqtisDabruneba(20);
    label2.Text = obieqti2.xilis_saxeli +
                    obieqti2.kodi.ToString();
}
```

დამოუკიდებელი სამუშაო

1. შექმენით კლასი Student. კლასში აღწერეთ დახურული (პრივატული) ცვლადი year, რომელიც არ შეიძლება იყოს 1-ზე ნაკლები და 4-ზე მეტი. დახურულ ცვლადთან მიმართვისათვის გამოიყენეთ თვისება Studyyear.

2. შექმენით კლასი, რომელშიც გამოცხადებულია პრივატული მთელი ტიპის ცვლადი. მასთან მიმართვისათვის გამოიყენეთ თვისება, რომელშიც set და get მეთოდებია განსაზღვრული. თვისება პრივატულ ცვლადს 5-ის ჯერად მნიშვნელობებს ანიჭებს.

3. შექმენით კლასი Paralelepipedi. მასში შექმენით მეთოდი, რომელსაც გადაეცემა ამავე ტიპის ობიექტი და რომელიც შეასრულებს ამ ობიექტის ფართობის გამოთვლას: $S = 2(a * b + b * c + a * c)$

4. შექმენით კლასი Cilindri. მასში შექმენით მეთოდი, რომელსაც გადაეცემა ამავე ტიპის ობიექტი და რომელიც შეასრულებს ამ ობიექტის მოცულობის გამოთვლას: $V = \pi r^2 h$

5. მოცემულია პროგრამული კოდი. ახსენით თუ რა დაიბეჭდება კოდის შესრულების შემდეგ: ან 1000? ან 300? ან 1200? და რატომ?

```
class Book
{
    private int _myPrice;
    public int MyPrice
    {
        get
        {
            return _myPrice;
        }
        set
        {
            if (value < 0)
                _myPrice = 0;
            else if (value > 1000)
                _myPrice = 1000;
            else
```

```

        _myPrice = value;
    }
}
}

private void button1_Click(object sender, EventArgs e)
{
    Book book = new Book();
    book.MyPrice = 1200;
    label1.Text=book.MyPrice.ToString();
}

```

6. ქვემოთ მოცემულ პროგრამულ კოდში რომელი გზით ხდება მეთოდისათვის არგუმენტის გადაცემა და დაასაბუთეთ, რატომ.

```

class PassingValue {
    public void IntValue(int t)
    {
        t = 100;
    }
    public void CharValue(char c)
    {
        c = 'F';
    }
    public void DoubleValue(double d)
    {
        d = 12.333;
    }
}

private void button1_Click(object sender, EventArgs e)
{
    PassingValue pv = new PassingValue();
    int x = 15;
    pv.IntValue(x);
    label1.Text = x.ToString();
    char y = 'M';
    pv.CharValue(y);
    label2.Text = y.ToString();
}

```

```

double z = 3.88;
pv.DoubleValue(z);
label3.Text = z.ToString(); }

```

7. ქვემოთ მოცემულ პროგრამულ კოდში, რომელი გზით ხდება მეთოდისათვის არგუმენტის გადაცემა. დაასაბუთეთ, რატომ. აღწერეთ თითოეული სტრიქონი.

```

class Triangle
{
    public double x1, y1, x2, y2, x3, y3;
    public Triangle(double _x1, double _y1, double _x2, double
        _y2, double _x3, double _y3)
    {
        x1 = _x1; y1 = _y1;
        x2 = _x2; y2 = _y2;
        x3 = _x3; y3 = _y3;
    }
    public double Perimeter(Triangle T)
    {
        double a, b, c;
        a = Math.Sqrt(Math.Pow(T.x1 - T.x2, 2) + Math.Pow(T.y1 -
            T.y2, 2));
        b = Math.Sqrt(Math.Pow(T.x1 - T.x3, 2) + Math.Pow(T.y1 -
            T.y3, 2));
        c = Math.Sqrt(Math.Pow(T.x2 - T.x3, 2) + Math.Pow(T.y2 -
            T.y3, 2));
        return a + b + c;    }
    }
    private void button1_Click(object sender, EventArgs e)
    {
        Triangle tr = new Triangle(1, 3, 4, 5, 0, 7);
        double p;
        p = tr.Perimeter(tr);
        label1.Text = p.ToString();
    }
}

```

8. ქვემოთ მოცემულია პროგრამა. მოცემულობა დაწერეთ დამოუკიდებლად; კომენტარის სახით აღწერეთ რას ასრულებს მეთოდი IsLetter; კომენტარის სახით

აღწერეთ, თუ რას აბრუნებს მეთოდი; კომენტარის სახით აღწერეთ, რას ასრულებს მეთოდი Reverse;

```
class DemoString
{ // .....
    public bool IsLetter(char c, string s)
    {
        for (int i = 0; i < s.Length; i++)
            if (s[i] == c) return true; return false; }
    //.....
    public string Reverse(string s)
    {
        string s2 = "";
        int len = s.Length;
        for (int i = 0; i < len; i++)
            s2 += s[len - i - 1];
        return s2;
    }
}

private void button1_Click(object sender, EventArgs e)
{
    DemoString ds = new DemoString();
    string s2;
    s2 = ds.Reverse("abcde"); // s2 = "edcba"
    label1.Text = s2;
    bool f = ds.IsLetter('p', "bestprog.net"); // f = True
    label2.Text = f.ToString();
}
```

9. მოცემულობა დაწერეთ დამოუკიდებლად; აღწერეთ თითოეული სტრიქონი; ახსენით, თუ რას აბრუნებს მეთოდი: ცვლადს? სტრიქონს თუ ობიექტს?

```
class Rect
{
    int width;
    int height;
    public Rect(int w, int h)
```

```

        {
            width = w;
            height = h;
        }
    public int Area()
    {
        return width * height;
    }
    public void Show()
    {
        MessageBox.Show(width + " " + height);
    }
    public Rect Enlarge(int factor) //
    {
        return new Rect(width * factor, height * factor);
    }
}

private void button1_Click(object sender, EventArgs e)
{
    Rect r1 = new Rect(4, 5);
    r1.Show();
    label1.Text = "მართკუთხედის ფართობი r1 " + r1.Area();
    Rect r2 = r1.Enlarge(2);
    r2.Show();
    label2.Text = "მართკუთხედის ფართობი r2= " + r2.Area();
}

```

13

შეტანა-გამოტანის ნაკადების კლასები. **FileStream** კლასი: ფაილის გახსნა, დახურვა, ბაიტების ჩაწერა და წაკითხვა. ფაილში სიმბოლოების შეტანა/გამოტანა: **StreamWriter** და **StreamReader** კლასები. ორობითი მონაცემების წაკითხვა და ჩაწერა: **BinaryReader** და **BinaryWriter** კლასები. ფაილებთან პირდაპირი მიმართვა: **Seek()** მეთოდი. ფაილისა და კატალოგის შესახებ ინფორმაციის მიღება. **File** და **FileInfo** კლასები და მათი თვისებები. **Directory** და **DirectoryInfo** კლასები და მათი თვისებები.

შეტანა-გამოტანის ნაკადების კლასები. C# ენაში შეტანა/გამოტანის ნაკადებთან ანუ ფაილებთან სამუშაოდ საკმაოდ ბევრი კლასი არსებობს (იხ. ცხრილი 11). შეტანა/გამოტანის ნაკადების კლასები განსაზღვრულია **System.IO** კლასში. ამიტომ, ნებისმიერი პროგრამის დასაწყისში, რომელიც იყენებს შეტანა/გამოტანის ნაკადების კლასებს, პროგრამულ კოდის დასაწყისში უნდა მივუთითოთ დირექტივა `using System.IO;`

FileStream კლასი: ფაილის გახსნა, დახურვა, ბაიტების ჩაწერა და წაკითხვა. კლასი **FileStream**-ის საშუალებით შესაძლებელია ფაილიდან ინფორმაციის წაკითხვისა და ჩაწერის უზრუნველყოფა. კლასი **FileStream**-ის დახმარებით შეგვიძლია ვიმუშაოთ როგორც ტექსტურ, ისე ბინარულ ფაილებთან.

კლასი **FileStream**-ის ტიპის ობიექტის შესაქმნელად შეგვიძლია გამოვიყენოთ ამ კლასის როგორც კონსტრუქტორები, ისე კლასი **File**-ის სტატიკური მეთოდები. კლასი **FileStream**-ის კონსტრუქტორს აქვს საკმაოდ ბევრი გადატვირთული ვერსა, მაგრამ ახლა აღვწეროთ ყველაზე უფრო მარტივი და გამოყენებადი ვარიანტი:

```
FileStream(string filename, FileMode mode);
```

აქ კონსტრუქტორს გადაეცემა ორი პარამეტრი: ფაილის მისამართი და ჩამოთვლა **FileMode**. მოცემული ჩამონათვალი მიუთითებს ფაილთან წვდომის ვარიანტებზე და **mode** პარამეტრს შეუძლია მიიღოს ქვემოთ ჩამოთვლილი მნიშვნელობები:

Append - თუ ფაილი არსებობს, მონაცემები/ტექსტი დაემატება ფაილის ბოლოში. თუ ფაილი არ არსებობს, მაშინ ის შეიქმნება. ფაილი იხსნება მხოლოდ მონაცემების ჩასაწერად.

Create - იქმნება ახალი ფაილი. თუ ასეთი ფაილი უკვე არსებობს, მაშინ მოხდება მასზე გადაწერა.

ცხრილი 11. შეტანა-გამოტანის ნაკადების კლასები

კლასი	გამოყენება
Binary Reader and Writer	მარტივი ინფორმაციის წაკითხვა/ჩაწერა
Directory, File, DirectoryInfo and FileInfo	შევექმნათ, წავშალოთ, გადავაადგილოთ ფაილები და მისამართები (საქაღალდეები). მივიღოთ კონკრეტული სახის ინფორმაცია ფაილების შესახებ ჩაშენებული მეთოდების გამოყენებით.
FileStream	მივწვდეთ ფაილს
MemoryStream	მივწვდეთ მეხსიერებაში შენახულ ინფორმაციას
StreamWriter and StreamReader	წავიკითხოთ და ჩავწეროთ ტექსტური ინფორმაცია
StringReader and StringWriter	წავიკითხოთ და ჩავწეროთ ტექსტური ინფორმაცია string-ის ბუფერით

CreateNew - იქმნება ახალი ფაილი. თუ ასეთი დასახელების ფაილი უკვე არსებობს, მაშინ აპლიკაცია/პროექტი გასცემს შეტყობინებას შეცდომის შესახებ.

Open - იხსნება არსებული ფაილი. თუ ასეთი დასახელების ფაილი არ არსებობს, მაშინ აპლიკაცია/პროექტი გასცემს შეტყობინებას განსაკუთრებული სიტუაციის შესახებ.

OpenCreate - თუ ფაილი არსებობს, მაშინ ის გაიხსნება, თუ არ არსებობს, მაშინ შეიქმნება ახალი ფაილი.

Truncate - თუ ფაილი არსებობს, მოხდება მასზე გადაწერა. ფაილი იხსნება მხოლოდ ჩაწერისათვის.

შეგვიძლია კლასი FileStream-ის ობიექტის შექმნა განსხვავებული სახითაც, კერძოდ კლასი File-ის მეთოდების დახმარებით:

1. `FileStream File.Open(string file, FileMode mode);` - ფაილის გახსნა და `FileStream` ნაკადის დაბრუნება;

2. `FileStream File.OpenRead(string file);` - ნაკადის გახსნა მონაცემების წასაკითხად;

3. `FileStream File.OpenWrite(string file);` - ნაკადის გახსნა მონაცემების ჩასაწერად;

ნაკადის დახურვა. კლასი `FileStream`-ის ფაილებთან დაკავშირებული ყველა რესურსის გამოთავისუფლების (ნაკადის დახურვას) განხორციელება შესაძლებელია ინტერფეისის `IDisposable`-ის საშუალებით. `FileStream`-თან მუშაობის დასრულების შემდეგ საჭიროა მასთან დაკავშირებული ფაილის გამოთავისუფლება. ასეთ გამოთავისუფლებაში გვხვდება მეთოდი `Dispose()`-ის გამოძახება. კორექტული დახურვისათვის კი შეგვიძლია გამოვიყენოთ მეთოდი `Close()`.

```
FileStream fstream = null;
try {
    fstream = new FileStream("faili.txt",
        FileMode.OpenOrCreate);
}
catch (Exception ex) { }
finally
{
    fstream.Close();
}
```

ან შეგვიძლია გამოვიყენოთ სტრუქტურა `using`, რომელიც ავტომატურად გამოათავისუფლებს `FileStream`-თან დაკავშირებულ ყველა რესურსს.

```
using (FileStream fstream = new
    FileStream("faili.txt", FileMode.OpenOrCreate)) {
    //fstream ობიექტი და კოდი..... }
```

განვიხილოთ კლასი `FileStream`-ის თვისებები:

თვისება `Length` - აბრუნებს ნაკადის სიგრძეს ბაიტებში;

თვისება `Position` - აბრუნებს ნაკადში მიმდინარე პოზიციას;

თვისება `Name` - აბრუნებს გზას `FileStream`-ის ფაილთან.

ინფორმაციის ფაილიდან წაკითხვის ან ფაილში ჩაწერის მიზნით შეგვიძლია გამოვიყენოთ კლასი `FileStream`-ის მეთოდები:

1. void CopyTo(Stream destination) - ნაკად destination-ში მიმდინარე ნაკადიდან მონაცემების ასლის აღება („დაკოპირება“);

2. Task CopyToAsync(Stream destination) - მეთოდი CopyTo-ს ასინქრონული ვერსია;

3. void Flush() - ფაილში ბუფერის შემცველობის ჩამოყრა;

4. Task FlushAsync() - მეთოდი Flush-ის ასინქრონული ვერსია.

int Read(byte[] array, int offset, int count) - თავიდან გადასაზღვრულ წარმოებულ კლასში, მიმდინარე ნაკადიდან კითხულობს ბაიტების თანამიმდევრობას და ნაკადში კურსორის პოზიციას გადაადგილებს/გადაიყვანს წაკითხული ბაიტების რაოდენობის/რიცხვის შემდეგ. მას აქვს სამი პარამეტრი: array - ბაიტების მასივი, რომელშიც გადავა ფაილიდან წაკითხული მონაცემები; offset - მასივში(array) ბაიტების გადაადგილება. მოხდება მასივში წაკითხული ბაიტების გადასვლა/განთავსება; count - იმ ბაიტების მაქსიმალური რიცხვი, რომელიც უნდა წავიკითხოთ.

Task<int> ReadAsync(byte[] array, int offset, int count) - მეთოდი Read-ის ასინქრონული ვერსია;

void Write(byte[] array, int offset, int count) - ბაიტების მასივიდან ინფორმაციის ჩაწერა/გადაწერა ფაილში; მას აქვს სამი პარამეტრი: array - ბაიტების მასივი, რომლიდანაც მონაცემები ჩაიწერება/გადაიწერება ფაილში; offset - მასივში (array) ბაიტების გადაადგილება. მოხდება მასივიდან ბაიტების ჩაწერა ნაკადში; count - იმ ბაიტების მაქსიმალური რიცხვი, რომელიც უნდა ჩაიწეროს.

Task WriteAsync(byte[] array, int offset, int count) - მეთოდი Write-ის ასინქრონული ვერსია;

StreamReader. კლასი StreamReader გამოიყენება ტექსტური სახის ფაილებიდან ინფორმაციის წასაკითხად/ამოსაკითხად.

ფაილში (file.txt) პირველივე ხაზზე წერია ტექსტი here is some kind of text:

```
label1.Text = "";  
string line;  
using (StreamReader reader = new StreamReader(@"D:\file.txt"))  
{  
    line = reader.ReadLine();  
}  
label1.Text += line;
```

კლასი `StreamReader` ხშირად გამოიყენება `using` ბრძანებასთან ერთად. ამგვარი კონსტრუქცია ხელს უწყობს მას რომ განკარგულებაში ჰქონდეს სისტემის რესურსები.

ამავდროულად `using` ბრძანება უზრუნველყოფს `StreamReader` ობიექტის დახურვას (ფაილის დახურვა მასთან მუშაობის შემდეგ აუცილებელია) და აღარ არის საჭირო რომ დავწეროთ შემდეგი ბრძანება: `reader.Close();`

როგორც ვხედავთ `StreamReadr` კლასის ობიექტს შექმნისას პარამეტრად გადავცევით ფაილის მისამართი. იმისთვის რომ პროგრამას აღექვა ფაილის მდებარეობა მისამართად და არა უბრალო `string` ტიპის ცვლადად, ჩვენ მის წინ დავწერეთ სიმბოლო „@“, შემდეგ `string` ტიპის ცვლადს `line`-ს მივანიჭეთ ტექსტური ფაილის პირველი ხაზიდან წაკითხული ინფორმაცია, ბრძანება `ReadLine`-მა კი უზრუნველყო, რომ წაკითხული იქნა ერთი ხაზი.

StreamWriter. კლასი `StreamWriter`-ის მეშვეობით შეგვიძლია ჩავწეროთ გარკვეული სახის ინფორმაცია ტექსტურ ფაილში:

```
label1.Text = "";  
using (StreamWriter writer = new StreamWriter(@"D:\file.txt"))  
{  
    writer.Write("ერთი");  
    writer.Write("ორი");  
    writer.Write("სამი");  
}
```

`StreamWriter`-ის სინტაქსი ძალიან ჰგავს `StreamReader` -ის სინტაქსს.

FileStream. კლასი `FileStream`-ის გამოყენებით შეგვიძლია ვიმუშაოთ ფაილებთან, ისე რომ ერთდროულად თან ჩავწეროთ და თან წავიკითხოთ ფაილიდან.

როდის გამოვიყენოთ `FileStream` კლასი?

1. ობიექტები რომლებთანაც ვმუშაობთ საშუალოდ არიან ერთ მეგაბაიტზე მეტი;
2. თუ სწრაფი კითხვა და წვდომა ფაილზე მნიშვნელოვანია;
3. თუ ვქმნით აპლიკაციას, რომელიც იყენებს შუალედურ კავშირს.

კლასს `FileStream` ასევე ძირითადად იყენებენ `SQL`-თან სამუშაოდ.

განვიხილოთ კლასი `FileStream`-ის გამოყენების რამდენიმე გზა:

FileStream კლასს აქვს Length მეთოდი, რომლის დახმარებითაც შეგვიძლია ვიპოვოთ ფაილში ჩაწერილი ინფორმაციის ზომა.

ქვემოთ მოცემული პროგრამული კოდიდან საკმაოდ კარგად ჩანს, თუ როგორ მუშაობს კლასი FileStream კლასებთან StreamReader და StreamWriter ერთად:

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    string line; string path = @"D:\file.txt";
    FileStream fs = new FileStream(path, FileMode.Open);
    StreamReader sr = new StreamReader(fs);
    //ფაილის წაკითხვა სანამ არ დამთავრდება ბოლო სტრიქონი
    while ((line = sr.ReadLine()) != null)
        { label1.Text += line + '\n'; }
    fs.Close();
    sr.Close();
    string pathnew = @"D:\filenew.txt";
    FileStream fsnew = new FileStream(pathnew,
    FileMode.Create);
    StreamWriter str = new StreamWriter(fsnew);
    str.Write("text for file");
    str.Close();
    fsnew.Close();
}
```

შეიქმნა FileStream კლასის ობიექტი. ობიექტს პარამეტრად გადავცემთ StreamReader კლასს, ხოლო მას შემდეგ, რაც დავამთავრებთ ფაილთან მუშაობას, ვხურავთ მას. FileStream კლასის ობიექტის შექმნისას, ამ კლასს პარამეტრად გადაეცა კიდეც ერთი მეთოდი FileMode.Open, ამ მეთოდის შინაარსი მდგომარეობს იმაში, რომ პროგრამას ვეუბნებით თუ რა გვინდა ამ ფაილისაგან. ამ შემთხვევაში ჩვენ გვინდა მხოლოდ ამ ფაილის გახსნა და იქიდან ინფორმაციის ამოკითხვა. FileMode.Create ბრძანებით პროგრამას ვეუბნებით, რომ ჩვენ ახალ ფაილს ვქმნით.

ორობითი მონაცემების წაკითხვა და ჩაწერა: BinaryReader და BinaryWriter კლასები. BinaryReader და BinaryWriter კლასები განკუთვნილია ორობითი ფორმატის მონაცემების წაკითხვისა და ჩაწერისათვის. კლასი BinaryReader გამოიყენება

მარტივი ტიპის მონაცემებისა და სტრიქონების წასაკითხად. მონაცემების წაკითხვისას შეგვიძლია მივუთითოთ სასურველი კოდირება. ჩუმათობის პრინციპის მიხედვით არჩეულია UTF-8 კოდირება.

განვიხილოთ კლასი BinaryReader-ის ძირითადი მეთოდები. ის ძირითადი მეთოდები, რომლებიც კითხულობენ მონაცემებს, იწყება პრეფიქსით Read.

მარტივი (სტანდარტული) ტიპის მონაცემების (int, double, char და სხვა) ორობით ფორმატში წასაკითხად გამოიყენება მრავალი მეთოდი. ასეთი მონაცემები კი უნდა ჩაიწეროს მეთოდით Write-ით.

ქვემოთ მოცემულია კლასი BinaryReader ძირითადი მეთოდების სია:

მეთოდი ReadBoolean() - მეთოდი ReadBoolean() მიმდინარე ნაკადიდან კითხულობს bool ტიპის მნიშვნელობებს და მიმდინარე პოზიციას გადაადგილებს/გადაწევს ერთი ბაიტით. მეთოდი ReadBoolean()-ის ზოგადი ფორმაა: public virtual bool ReadBoolean();

მეთოდები ReadByte() და ReadBytes() - უზრუნველყოფს ნაკადიდან ბაიტის ან ბაიტების წაკითხვას. მეთოდი ReadByte() კითხულობს მხოლოდ ერთ ბაიტს, მეთოდი ReadBytes() კი კითხულობს რამდენიმეს. მეთოდების ზოგადი ფორმაა:

```
public virtual byte ReadByte();
```

```
public virtual byte[] ReadBytes(int count);
```

სადაც count - წაკითხული ბაიტების რაოდენობაა.

მეთოდები ReadChar() и ReadChars(int) - უზრუნველყოფს ნაკადიდან char ტიპის ერთი სიმბოლოს ან რამდენიმე სიმბოლოს წაკითხვას. მეთოდების ზოგადი ფორმაა: public virtual char ReadChar(); public virtual char[] ReadChars(int count); სადაც count - იმ სიმბოლოების რაოდენობაა, რომელიც, ნაკადიდან უნდა წვიკითხვით.

მეთოდები ReadDecimal(), ReadDouble(), ReadSingle() - უზრუნველყოფენ მცოცავწერტილიანი ტიპის მონაცემების წაკითხვას. შესაბამისად კითხულობს decimal, double და float ტიპის მონაცემებს. მეთოდების ზოგადი ფორმაა: public virtual decimal ReadDecimal(); public virtual double ReadDouble(); public virtual float ReadSingle();

მეთოდები ReadInt16(), ReadInt32(), ReadInt64() - უზრუნველყოფენ int ტიპის მონაცემების წაკითხვას. მეთოდი ReadInt16() - მიმდინარე ნაკადიდან კითხულობს short ტიპის მონაცემებს და გადაყავს მიმთითებელი/კურსორი 2 ბაიტზე; მეთოდი

ReadInt32() - მიმდინარე ნაკადიდან კითხულობს int ტიპის მონაცემებს და გადაყავს მიმთითებელი/კურსორი 4 ბაიტზე; მეთოდი ReadInt64() - მიმდინარე ნაკადიდან კითხულობს long ტიპის მონაცემებს და გადაყავს მიმთითებელი/კურსორი 8 ბაიტზე; მეთოდების ზოგადი ფორმა: public virtual short ReadInt16(); public virtual int ReadInt32(); public virtual long ReadInt64();

მეთოდი ReadString() - უზრუნველყოფს მიმდინარე ნაკადიდან ერთი სტრიქონის წაკითხვას. მეთოდის ზოგადი ფორმა: public virtual string ReadString();

კლასი BinaryWriter გამოიყენება ნაკადში მარტივი ტიპის მონაცემებისა და სტრიქონების ჩასაწერად ორობით ფორმატში. მონაცემების წაკითხვისას შეგვიძლია მივუთითოთ სასურველი კოდირება. ჩუმათობის პრინციპის მიხედვით არჩეულია UTF-8 კოდირება.

კლასი BinaryWriter გამოიყენება მონაცემების ჩასაწერად ორობით (ბინარულ) ფორმატში. მონაცემების ჩაწერა შეიძლება მოხდეს ფაილებში, ქსელში, საცავში, მეხსიერებაში და ა.შ. თუ გვჭირდება შევექმნათ ობიექტი BinaryWriter ტიპისა, მაშინ უნდა მივმართოთ კონსტრუქტორს. ახლა განვიხილოთ კონსტრუქტორის უფრო გავრცელებული ვერსია:

BinaryWriter(Stream stream);

განვიხილოთ კლასი BinaryWriter-ის ძირითადი მეთოდები:

Close() - უზრუნველყოფს ნაკადის დახურვას და რესურსების გამოთავისუფლებას;

Flush() - წმენდს ბუფერს. მასში დარჩენილ მონაცემების გადაწერს ფაილში;

Seek() - ადგენს პოზიციას ნაკადში;

Write() - ნაკადში მონაცემების ჩაწერა. მეთოდის Write() პარამეტრი იღებს მარტივ მნიშვნელობებს: Write(bool); Write(byte); Write(char); Write(decimal); Write(double); Write(Half); Write(short); Write(int);

ასევე შეგვიძლია გადავცეთ byte და char ტიპის მონაცემები: Write(byte[]); Write(char[]); Write(ReadOnlySpan<byte>); Write(ReadOnlySpan<char>); ნაკადში მასივის ჩაწერის დროს ასევე შეგვიძლია მივუთითოთ დამატებითი პარამეტრები: მასივის რომელი ელემენტიდან უნდა მოხდეს ჩაწერა და ჩასაწერი ელემენტების რაოდენობა: Write(byte[], int, int); Write(char[], int, int);

ახლა კი განვიხილოთ მაგალითი: ფაილში filibinari.bin მეთოდის Write()-ის დახმარებით, ჩავწეროთ სხვადასხვა ტიპის მონაცემი, ხოლო შემდეგ კი წავიკითხოთ

ეს მონაცემები, რათა დავრწმუნდეთ, მართლა მოხდა თუ არა ჩაწერა ანუ გავაკონტროლოთ კოდი სისწორეზე.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
private void button1_Click(object sender, EventArgs e) {
    // filibinari.bin ფაილში მონაცემების ჩაწერა
    // *.bin გაფართოების ფაილი ინახავს ორობით ინფორმაციას
    // *.bin ფაილები განსხვავებულია ტექსტური ფაილებისაგან
    using (FileStream fs = new FileStream("filibinari.bin", FileMode.Create))
    {
        using (BinaryWriter bw = new BinaryWriter(fs, Encoding.Default)){
            bool b = false;
            int i = 233;
            double x = 2.338;
            decimal d = 0.011199m;
            float f = -30.88f;
            bw.Write(b);
            bw.Write(i);
            bw.Write(x);
            bw.Write(d);
            bw.Write(f);
            bw.Write(1234567890123L);
        }
    }
    // filibinari.bin ფაილიდან მონაცემების წაკითხვა
    using (FileStream fs=new FileStream("filibinari.bin",FileMode.Open))
    {
```



```

using (BinaryReader br = new BinaryReader(fs, Encoding.Default))
{
    bool b2 = br.ReadBoolean();
    int i2 = br.ReadInt32();
    double x2 = br.ReadDouble();
    decimal d2 = br.ReadDecimal();
    float f2 = br.ReadSingle();
    long l2 = br.ReadInt64();
    label11.Text = ""; label12.Text = "";
    label13.Text = ""; label14.Text = "";
    label15.Text = ""; label16.Text = "";
    label11.Text = b2.ToString();
    label12.Text = i2.ToString();
    label13.Text = x2.ToString();
    label14.Text = d2.ToString();
    label15.Text = f2.ToString();
    label16.Text = l2.ToString();
}

```

განვიხილოთ კიდეც ერთი მაგალითი, სადაც მოცემული იქნება ნაკადში ორობითი (ბინარული) ჩაწერა/წაკითხვის ასახვა. მოხდება BinaryReader და BinaryWriter კლასების გამოყენება ნაკადში ორობითი/ბინარული ინფორმაციის ჩასაწერად და ამოსაკითხად. ფაილში „testdata“ ჯერ ჩაიწერება და შემდეგ ამოიკითხება სხვადასხვა ტიპის მონაცემი. ასევე ხდება შემოწმება შეცდომებზე.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
private void button1_Click(object sender, EventArgs e) {

```

```

BinaryWriter moncemebischawera;
BinaryReader monacemebiswakitxva;
int i = 11;
double d = 1027.39;
bool b = true;
string str = "დილა მშვიდობისა!";
try {
moncemebischawera = new
    BinaryWriter(new FileStream("failitesti", FileMode.Create));
}
catch (IOException exc) {
MessageBox.Show("ფაილის გახსნისას დაფიქსირდა შეცდომა:\n" + exc.Message);
    return;
}
try
{
    MessageBox.Show("ჩაწერა: " + i);
    moncemebischawera.Write(i);
    MessageBox.Show("ჩაწერა: " + d);
    moncemebischawera.Write(d);
    MessageBox.Show("ჩაწერა " + b);
    moncemebischawera.Write(b);
    MessageBox.Show("ჩაწერა: " + 11.3 * 9.7);
    moncemebischawera.Write(11.3 * 9.7);
    MessageBox.Show("ჩაწერა: " + str);
    moncemebischawera.Write(str);
}
catch (IOException exc)
{
    MessageBox.Show("შეცდომა:\n" + exc.Message);
}
finally
{
    moncemebischawera.Close();
}
try
{

```

```

        monacemebiswakitxva = new
            BinaryReader(new FileStream("failitesti", FileMode.Open));
    }
    catch (IOException exc) {
        MessageBox.Show("ფაილის გახსნისას მოხდა შეცდომა:\n" + exc.Message);
        return;
    }
    try
    {
        i = monacemebiswakitxva.ReadInt32();
        label1.Text = "წაკითხვა: " + i;
        d = monacemebiswakitxva.ReadDouble();
        label2.Text = "წაკითხვა: " + d;
        b = monacemebiswakitxva.ReadBoolean();
        label3.Text = "წაკითხვა: " + b;
        d = monacemebiswakitxva.ReadDouble();
        label4.Text = "წაკითხვა: " + d;
        str = monacemebiswakitxva.ReadString();
        label5.Text = "წაკითხვა: "+ str;
    }
    catch (IOException exc) {
        MessageBox.Show("დაფიქსირდა შეტანა/გამოტანის შეცდომა:\n" + exc.Message);
    }
    finally
    {
        monacemebiswakitxva.Close();
    }
}

```

თუ შევამოწმებთ „failitesti“ ფაილის შიგთავსს, რომელიც შეიქმნა პროგრამის შესრულებაზე გაშვების შემდეგ, აღმოვაჩენთ, რომ ის შეიცავს არა ადამიანისათვის ადვილად წასაკითხ ტექსტურ ინფორმაციას, არამედ ორობით მონაცემებს, რომელიც გაუგებარი და გაურკვეველია.

ახლა კი შევეცადოთ ზემოთ აღწერილი კოდი გავხადოთ უფრო ფუნქციური, რათა დავინახოთ, თუ როგორი ეფექტური შეიძლება იყოს ორობითი ჩაწერა/წაკითხვა.

კოდში „საწყობის მარაგი“ აღწერილია საწყობში მოცემული ნივთების დასახელება, ფასი, რაოდენობა, გაყიდულების რაოდენობა, დამთვრდა თუ არა ეს ნივთი საწყობში და ასე შემდეგ. ახლა კი ვნახოთ უფრო ფუნქციური ვარიანტი:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
private void button1_Click(object sender, EventArgs e) {
// საწყობის მარაგი
    BinaryWriter moncemebischawera;
    BinaryReader monacemebiswakitxva;
    string nivTi; // ნივთის დასახელება,
    int maragi; // არის თუ არა ნივთი საწყობში; არის თუ არა მარაგი.
    double fasi; // ფასი.
    try {
        moncemebischawera = new
        BinaryWriter(new FileStream("inventari.dat", FileMode.Create));
    }
    catch (IOException exc) {
        MessageBox.Show("შეუძლებელია მარაგის ფაილის გახსნა მონაცემების გამოსატანად!!!");
        MessageBox.Show("მიზეზი: " + exc.Message);
        return;
    }
    try
    {
        moncemebischawera.Write("ცალხაზიანი რვეული A4");
```

```

moncembischawera.Write(21);
moncembischawera.Write(1.10);
moncembischawera.Write("მათემატიკის რვეული");
moncembischawera.Write(34);
moncembischawera.Write(1.10);
moncembischawera.Write("სახატავი რვეული");
moncembischawera.Write(17);
moncembischawera.Write(4.95);
moncembischawera.Write("შავგრაფიტიანი ფანქარი");
moncembischawera.Write(30);
moncembischawera.Write(2.30);
moncembischawera.Write("აკვარელის ფანქრები, 12 ფერი");
moncembischawera.Write(14);
moncembischawera.Write(30.80);
moncembischawera.Write("აკვარელის ფანქრები, 24 ფერი");
moncembischawera.Write(20);
moncembischawera.Write(40.95);
moncembischawera.Write("ბურთულიანი კალამი");
moncembischawera.Write(190);
moncembischawera.Write(1.20);
moncembischawera.Write("ნოტების რვეული");
moncembischawera.Write(200);
moncembischawera.Write(2.50);
moncembischawera.Write("გლობუსი, 30 სმ");
moncembischawera.Write(40);
moncembischawera.Write(70.20);
}
catch (IOException exc) {
    MessageBox.Show("საკანცელარო ნივთების ჩაწერისას მოხდა შეცდომა!!!");
    MessageBox.Show("მიზეზი: " + exc.Message); }
finally {
    moncembischawera.Close(); }
try {

```

```

monacemebiswakitxva = new
    BinaryReader(new FileStream("inventari.dat", FileMode.Open));
}
catch (IOException exc) {
    MessageBox.Show("შეუძლებელია მარაგის ფაილის გახსნა მონაცემების შესატანად!!! ");
    MessageBox.Show("მიზეზი: " + exc.Message);
    return;
}
// მომხმარებლის მიერ შეტანილი ნივთის ძებნა
string zebna = textBox1.Text;
try {
    for ( ; ; ) //უსასრულო ციკლი
    {
        // ნივთის მონაცემების წაკითხვა
        nivTi = monacemebiswakitxva.ReadString();
        maragi = monacemebiswakitxva.ReadInt32();
        fasi = monacemebiswakitxva.ReadDouble();
        // ნივთის დასახელების შემოწმება სისწორეზე
        if (nivTi.Equals(zebna, StringComparison.OrdinalIgnoreCase)) {
            label2.Text=nivTi+": " + maragi + "ცალი." + "ერთი ცალის ფასი:" + fasi.ToString();
            label3.Text = nivTi + "-ის მთლიანი ფასი: " + (fasi * maragi).ToString();
            break;
        }
    }
}
catch (EndOfStreamException)
{
    MessageBox.Show("ნივთი არ მოიძებნა");
}
catch (IOException exc)
{
    MessageBox.Show("ფაილიდან მონაცემების წაკითხვის
შეცდომა!!!");
}

```

```

        MessageBox.Show("მიზეზი: " + exc.Message);
    }
    finally
    {
        monacemebiswakitxva.Close();
    }
}

private void Form1_Load(object sender, EventArgs e) {
    label1.Text = "შეიტანეთ ნივთის დასახელება ";
}

```

Seek() მეთოდი. ბინარულ ფაილებს აქვს გარკვეული სახის სტრუქტურა. მაგალითად მონაცემთა ბაზის SQLite-ის ფაილში პირველი 100 ბაიტი გამოიყოფა შემდეგი სახის მონაცემებისათვის: ვერსიის მითითება, გვერდების რაოდენობა და ა. შ. იმის გამო, რომ ვიცნობთ და ვიცით ფაილის სტრუქტურა, ჩვენ შეგვიძლია, მივიღოთ ნებისმიერი/პირდაპირი წვდომა ფაილის ნაწილთან ან ჩავწეროთ აუცილებელი მონაცემები.

ფაილის გარკვეულ ნაწილთან პირდაპირი წვდომისათვის გამოიყენება მეთოდი Seek(). Seek() მეთოდის დახმარებით ნაკადში შეგვიძლია ვმართოთ კურსორის პოზიცია, რომელი პოზიციიდანაც დაიწყება ფაილის წაკითხვა ან ფაილში ჩაწერა.

მეთოდი Seek() იღებს ორ პარამეტრს: offset გადაადგილება და კურსორის პოზიცია ფაილში. კურსორის პოზიცია ფაილში აღიწერება შემდეგი სამი მნიშვნელობით:

SeekOrigin.Begin: ფაილის დასაწყისი;

SeekOrigin.End: ფაილის დასასრული;

SeekOrigin.Current: ფაილში მიმდინარე პოზიცია;

ნაკადის კურსორი, საიდანაც იწყება წაკითხვა ან ჩაწერა, გადაადგილდება წინ offset-ში ჩაწერილი მნიშვნელობით, მეორე პარამეტრის პოზიციიდან. გადაადგილება შეიძლება იყოს უარყოფითი ნიშნით, მაშინ კურსორი მოძრაობს უკან, თუ დადებითი ნიშნითაა, მაშინ მოძრაობს წინ.

განვიხილოთ ფაილთან პირდაპირი მიმართვის მაგალითი:

```

private void button1_Click(object sender, EventArgs e) {

```

```

string path = "note333.dat";
string text = "hello world";
using (FileStream fstream = new FileStream(path, FileMode.OpenOrCreate))
{
    // ტექსტის გარდაქმნა/კოდირება ბაიტებში
    byte[] input = Encoding.Default.GetBytes(text);
    // ბაიტების ჩაწერა ფაილში
    fstream.Write(input, 0, input.Length);
    MessageBox.Show("ტექსტი ჩაწერილია ფაილში");
}

// ფაილის ნაწილის წაკითხვა
using (FileStream fstream = new FileStream(path, FileMode.OpenOrCreate))
{
    //კურსორის/მიმთითებლის გადატანა ფაილის ბოლოში, ფაილის დასარულამდე 5
    //ბაიტია, მინუს 5 სიმბოლო ნაკადის ბოლოდან
    fstream.Seek(-5, SeekOrigin.End);
    // ვკითხულობთ ოთხ სიმბოლოს მიმდინარე პოზიციიდან
    byte[] output = new byte[5];
    fstream.ReadAsync(output, 0, output.Length);
    // ბაიტების სტრიქონად გარდაქმნა/დეკოდირება
    string textFromFile = Encoding.Default.GetString(output);
    MessageBox.Show( "ტექსტი ფაილიდან: " + textFromFile); // world
}
}

```

ფაილის შესახებ ინფორმაციის მიღება: **File** და **FileInfo** კლასები. კლასები **File** და **FileInfo** ასევე გამოიყენება ფაილებთან სამუშაოდ. კლასების **File**-ს და **FileInfo**-ს დახმარებით შეგვიძლია შევქმნათ, წავშალოთ, გადავაადგილოთ ფაილები და კიდევ სხვა უამრავი ოპერაცია შევასრულოთ ფაილებთან მიმართებით.

კლასი **FileInfo**-ს ზოგიერთი მეთოდი და თვისება: მეთოდი **CopyTo(path)** - ფაილის გადაწერა მითითებულ ახალ ადგილზე; მეთოდი **Create()** - ფაილის შექმნა; მეთოდი **Delete()** - ფაილის წაშლა; მეთოდი **MoveTo(destFileName)** - ფაილის გადატანა ახალ მისამართზე; თვისება **Directory** - იღებს მშობელ კატალოგს ობიექტი

DirectoryInfo-ს სახით; თვისება DirectoryName - იღებს მშობელი კატალოგის სრულ მისამართს; თვისება Exists - მიუთითებს ფაილის არსებობას; თვისება Length - იღებს ფაილის ზომას; თვისება Extension - იღებს ფაილის გაფართოებას; თვისება Name - იღებს ფაილის სახელს; თვისება FullName - იღებს ფაილის სრულ სახელს;

კლასი File უზრუნველყოფს კლასი FileInfo-ს მსგავს ფუნქციურობას სტატიკური მეთოდების დახმარებით: მეთოდი CopyTo(path) - ფაილის გადაწერა ახალ ადგილზე; მეთოდი Create() - ფაილის შექმნა; მეთოდი Delete() - ფაილის წაშლა;

ფაილებთან სამუშაოდ არსებობს კიდევ სხვა მეთოდებიც, რომლებიც გვცხმარება დავადგინოთ: შექმნილია თუ ახლა ვქმნით ფაილს, ვკითხულობთ თუ ვწერთ მასში; ახლა ვისაუბროთ File კლასის რამდენიმე ახალ მეთოდზე, რომლებსაც ვიყენებთ ფაილებთან სამუშაოდ. ფაილებთან მუშაობის ოთხი ახალი მეთოდია: ReadAllLines, ReadAllText, WriteAllText, WriteAllLines.

განიხილოთ თითოეული მათგანი:

მეთოდი ReadAllLines ფაილიდან კითხულობს ყველა ხაზს სათითაოდ და შემდეგ თითოეულ ხაზს ყრის string ტიპის მასივში შესაბამის ადგილზე ანუ მეთოდის ReadAllLines დასაბრუნებელი მნიშვნელობა არის string ტიპის მასივი, ხოლო მას პარამეტრად გადაეცემა ფაილის ადგილმდებარეობა (path).

მეთოდი ReadAllText კი ფაილის მთლიან ინფორმაციას აღიქვამს ერთ მთლიან ტექსტად ანუ მთლიანად string ტიპის ცვლადად. მეთოდის ReadAllText დასაბრუნებელი მნიშვნელობა string ტიპისაა, ხოლო პარამეტრად ფაილის მისამართი გადაეცემა.

მეთოდს WriteAllLines პარამეტრად გადაეცემა ფაილის მისამართი, სადაც უნდა ჩაიწეროს ინფორმაცია და string ტიპის მასივი. შემდეგ მეთოდი WriteAllLines მითითებულ ფაილში თითოეულ ხაზზე წერს მასივის თითოეულ ელემენტს; მეთოდის WriteAllLines დასაბრუნებელი მნიშვნელობა void ტიპისაა.

მეთოდის WriteAllText-ის დასაბრუნებელი მნიშვნელობაც ასევე void ტიპისაა, მას პარამეტრად გადაეცემა ფაილის მისამართი და string ტიპის ცვლადი. ის მითითებულ ფაილში წერს string ტიპის ცვლადის მნიშვნელობას. ტექსტურ ფაილში file.txt პირველივე ხაზზე წერია ტექსტი: Georgian Technical University (Tbilisi).

```
label1.Text = "";  
string path = @"D:\file.txt";  
string[] lines = File.ReadAllLines(path);
```

```

foreach (string var in lines) label1.Text += var + '\n';
string str = File.ReadAllText(path);
label2.Text = str;
string a = "create file";
File.WriteAllText(@"D:\filea.txt", a);
string [] m = {"Hello" , "And", "Welcom"};
File.WriteAllLines(@"D:\fileb.txt", m);

```

კატალოგებთან/საქაღალდეებთან მუშაობა. საქაღალდეებთან/კატალოგებთან სამუშაოდ გამოიყენება ორი კლასი: Directory და DirectoryInfo;

კლასი Directory - არის სტატიკური კლასი, რომელიც წარმოადგენს სტატიკურ მეთოდებს საქაღალდეებთან სამუშაოდ.

კლასი DirectoryInfo - არის კლასი, რომელიც წარმოადგენს ინფორმაციას კონკრეტული საქაღალდის შესახებ.

თუ გვჭირდება ფაილების ან საქაღალდეებია/ქვესაქაღალდეების დასახელებების წაკითხვა, მაშინ უფრო მიზანშეწონილი იქნება კლასის Directory-ის სტატიკური მეთოდების გამოყენება, ვიდრე კლასის DirectoryInfo-ს ობიექტის შექმნა და ა. შ. თუ გვჭირდება რაიმე ერთი ოპერაციის შესრულება, მაშინ უკეთესია გამოვიყენოთ კლასის Directory-ის სტატიკური მეთოდები.

იმ შემთხვევაში, თუ საჭიროა შექმნათ, წაშალოთ, ჩამოთვალოთ და ა. შ., უმჯობესია გამოვიყენოთ კლასი DirectoryInfo. თუ თქვენ უბრალოდ უნდა წაკითხოთ ფაილების ან დირექტორიების სახელები, მაშინ უფრო მიზანშეწონილი იქნება დირექტორიის კლასის სტატიკური მეთოდების გამოყენება, ვიდრე DirectoryInfo კლასის ინსტანციის შექმნა და ა. შ. თუ ერთი ოპერაცია გჭირდებათ, გამოიყენეთ დირექტორია.

თუ თქვენ უნდა შექმნათ, წაშალოთ, ჩამოთვალოთ რამე და ა. შ., უმჯობესია გამოიყენოთ DirectoryInfo კლასი.

კლასი Directory და DirectoryInfo მეთოდებსა და თვისებებს გავეცნოთ პროგრამული კოდების მაგალითებზე.

მაგალითი 1. ქვემოთ მოცემული კოდი გვაჩვენებს C: დისკის საქაღალდეებს და ფაილებს.

```

private void button1_Click(object sender, EventArgs e){
    string dirName = "C:\\";

```

```

if (Directory.Exists(dirName)) // თუ საქაღალდე არსებობს
{
    label1.Text = " საქაღალდეები: " + '\n';
    string[] dirs = Directory.GetDirectories(dirName);
    foreach (string s in dirs)
    {
        label1.Text += s + '\n';
    }
    label2.Text = "ფაილები: " + '\n';
    string[] files = Directory.GetFiles(dirName);
    foreach (string s in files)
    {
        label2.Text += s + '\n';
    }
}

```

მაგალითი 2.

```

private void button1_Click(object sender, EventArgs e) {
    // D: დისკზე არსებული საქაღალდეების სია
    DirectoryInfo[] massDirs = new
    DirectoryInfo(@"d:\").GetDirectories();
    // ფაილში საქაღალდეების სახელის დამატება სტრიქონების სახით
    using (StreamWriter SW = new StreamWriter("NamesofDirs.txt"))
    {
        foreach (DirectoryInfo dir in massDirs)
        {
            SW.WriteLine(dir.Name);
        }
    }
    string strline = ""; // ფაილში ჩაწერილი სტრიქონების წაკითხვა
    using (StreamReader SR = new StreamReader("NamesofDirs.txt"))
    {
        while ((strline = SR.ReadLine()) != null)
        {

```

```

        label1.Text += strline+ '\n';
    }
}

```

მაგალითი 3. კიდევ ერთი მაგალითი საქაღალდეებთან დაკავშირებით:

```

DirectoryInfo dir = new DirectoryInfo(@"C:\windows");
label1.Text = "ინფორმაცია საქაღალდეზე windows: " + '\n';
label1.Text += "1. საქაღალდის სრული სახელი: " + dir.FullName + '\n';
label1.Text += "2. საქაღალდის სახელი: " + dir.Name + '\n';
label1.Text += "3. მშობელი საქაღალდე: " + dir.Parent + '\n';
label1.Text += "4. შექმნის დრო: " + dir.CreationTime + '\n';
label1.Text += "5. ატრიბუტები: " + dir.Attributes + '\n';
label1.Text += "6. ძირითადი დისკი: " + dir.Root + '\n';

```

მაგალითი 4. მეთოდის GetFiles()-ის გამოყენებით მოვიძიოთ ინფორმაცია იმ ფაილებზე, რომელთა გაფართოებაც არის *.jpg და იმყოფებიან C:\windows საქაღალდეში.

```

DirectoryInfo dir1 = new
    DirectoryInfo(@"C:\Windows\Web\Wallpaper\Nature");
// მივიღოთ ის ფაილები, რომელთა გაფართოებაც არის *.jpg
FileInfo[] imagefiles =
dir1.GetFiles("*.jpg", SearchOption.AllDirectories);
label1.Text = "ნაპოვნია " + imagefiles.Length + "*.jpg ფაილი "+'\n';
foreach (FileInfo f in imagefiles) {
    label1.Text += "ფაილის სახელი " + f.Name + '\n';
    label1.Text += "ფაილის ზომა " + f.Length + '\n';
    label1.Text += "შექმნის დრო " + f.CreationTime + '\n';
    label1.Text += "ატრიბუტები " + f.Attributes + '\n';
}

```

ფაილების არჩევა. OpenFileDialog კლასი. პროგრამებთან მუშაობისას ფაილის სახელის მითითებისათვის საჭიროა პროგრამის კოდის შეცვლა, რაც ბუნებრივია, გარკვეულ წილად ართულებს სხვადასხვა ფაილთან მუშაობას. ამ პრობლემის გადასაწყვეტად გამოიყენება OpenFileDialog კლასი, რომელიც განსაზღვრულია System.Windows.Forms სახელების სივრცეში. მისი ShowDialog() მეთოდი

გამოიყენება ეკრანზე ფაილის გახსნის სტანდარტული დიალოგური ფანჯრის გამოსატანად.

განვიხილოთ მაგალითი, სადაც გამოვიყენებთ კლასს OpenFileDialog-ს. პროგრამაში ხდება Open ფანჯრის გახსნის დემონსტრირება.

```
label1.Text = "";
OpenFileDialog FailisGaxsna = new OpenFileDialog();
if (FailisGaxsna.ShowDialog() == DialogResult.OK)
{
    FileInfo faili = new FileInfo(FailisGaxsna.FileName);
    label1.Text += faili.FullName + '\n';
    label1.Text += faili.Length.ToString() + '\n';
    label1.Text += faili.LastAccessTime.ToString() + '\n';
    label1.Text += faili.DirectoryName + '\n';
}
```

ახლა კი განვიხილოთ მაგალითი, სადაც გამოვიყენებთ კლასს OpenFileDialog. პროგრამაში ხდება ფაილის წაკითხვის დემონსტრირება Open დიალოგური ფანჯრის საშუალებით:

```
label1.Text = ""; FileStream file_in;
string striqoni;
OpenFileDialog FailisGaxsna = new OpenFileDialog();
if ( FailisGaxsna.ShowDialog() == DialogResult.OK )
{
    file_in = new FileStream(FailisGaxsna.FileName, FileMode.Open);
    StreamReader file_stream_in = new StreamReader(file_in);
    for ( ; ( striqoni = file_stream_in.ReadLine() ) != null; )
        label1.Text += striqoni + '\n';
    file_stream_in.Close();
}
```

განვიხილოთ კიდევ ერთი მაგალითი, სადაც გამოვიყენებთ კლასს OpenFileDialog-ს. პროგრამაში ხდება ფაილში ჩაწერის დემონსტრირება Save დიალოგური ფანჯრის საშუალებით:

```
string striqoni;
FileStream file_out;
```

```

        SaveFileDialog FailisGaxsna = new SaveFileDialog();
        if (FailisGaxsna.ShowDialog() == DialogResult.OK) {
            file_out = new FileStream(FailisGaxsna.FileName, FileMode.Create);
            StreamWriter file_stream_out = new StreamWriter(file_out);
            striqoni = textBox1.Text;
            // ფაილში striqoni სტრიქონის ჩაწერა
            file_stream_out.Write(striqoni);
            file_stream_out.Close();
        }

```

განვიხილოთ რამდენიმე ამოცანა.

ამოცანა 13.1. ფაილის შექმნა და მასში ინფორმაციის ჩაწერა. ამოცანის გადასაწყვეტად ვიყენებთ კლასს **File**. პროგრამული კოდის გადასაწყვეტად using დირექტივების შემდეგ უნდა დავამატოთ using System.IO; დირექტივა. დასმული ამოცანა განვიხილოთ სამ ეტაპად:

1. შევამოწმოთ არსებობს თუ არა ასეთი ფაილი;
2. თუ არ არსებობს, მაშინ ვქმნით მას და ვწერთ მასში სტრიქონებს;
3. გავხსნათ შექმნილი ფაილი და წავიკითხოთ ჩაწერილი სტრიქონი.

```

private void button1_Click(object sender, EventArgs e) {
    label1.Text = "";
    //ცვლადი ფაილის მისამართს შესანახად
    string pathOfFile = @"c:\Examples\FileTestApp.txt";
    //ამოწმებს ფაილის არსებობას
    if (!File.Exists(pathOfFile))
    { //თუ ფაილი არ არსებობს მაშინ ვქმნით მას და ვწერთ მასში ინფორმაციას
        using (StreamWriter sw = File.CreateText(pathOfFile)) {
            sw.WriteLine("საქართველოს ");
            sw.WriteLine("ტექნიკური");
            sw.WriteLine("უნივერსიტეტი");
        }
    }
    // ფაილის გახსნა და მისი გამოტანა
    using (StreamReader sr = File.OpenText(pathOfFile))

```

```

{ // ცვლადი რომელიც მოახდენს ფაილის შემცველობის წაკითხვას
    string str = "";
    // ფაილის წაკითხვა სანამ არ დამთავრდება ბოლო სტრიქონი
    while ((str = sr.ReadLine()) != null) {
        label1.Text += str + '\n';
    }
}

```

ამოცანა 13.2. კლასი StreamReader და StreamWriter ხშირად გამოიყენება using ბრძანებასთან ერთად. ამგვარი კონსტრუქცია ხელს უწყობს მას, რომ განკარგულე-ბაში ჰქონდეს სისტემის რესურსები. ამავდროულად using ბრძანება უზრუნველ-ყოფს StreamReader და StreamWriter ობიექტის დახურვას (ფაილთან მუშაობის შემდეგ აუცილებელია ფაილის დახურვა) და აღარ არის საჭირო, რომ დავწეროთ შემდეგი ბრძანება: reader.Close();

აქ using არის ოპერატორი, რომელიც განსაზღვრავს მოქმედების უბანს, რომლის ბოლოსაც ობიექტი წაიშლება.

```

label1.Text = "";
//ცვლადი ფაილის მისამართს შესანახად
string pathOfFile = @"c:\Examples\FileTestApp.txt";
if (!File.Exists(pathOfFile)) // ამოწმებს ფაილის არსებობას
{ // თუ ფაილი არ არსებობს, მაშინ ვქმნით მას და ვწერთ მასში ინფორმაციას
    using (StreamWriter sw = File.CreateText(pathOfFile)) {
        sw.WriteLine("საქართველოს ");
        sw.WriteLine("ტექნიკური");
        sw.WriteLine("უნივერსიტეტი");
    }
}

// AppendText მეთოდის გამოყენება ინფორმაციის დასამატებლად
using (StreamWriter sw = File.AppendText(pathOfFile)) {
    sw.WriteLine("ივ. ჯავახილშვილის ");
    sw.WriteLine("სახელობის");
    sw.WriteLine("სახელმწიფო უნივერსიტეტი");
}

```

```

    }
    // ფაილის გახსნა და წაკითხვა
    using (StreamReader sr = File.OpenText(pathOfFile)) {
    // ცვლადი რომელიც მოახდენს ფაილის შემცველობის წაკითხვას
        string str = "";
        // ფაილის წაკითხვა სანამ არ დამთავრდება ბოლო სტრიქონი
        while ((str = sr.ReadLine()) != null)
        {
            label1.Text += str + '\n';
        }
    }
}

```

ამოცანა 13.3. ფაილის შესახებ ინფორმაციის მიღება.

```

private void button1_Click(object sender, EventArgs e) {
    label1.Text = "";
    //ცვლადი ფაილის მისამართის შესანახად
    string path = @"c:\Examples\FileTestApp.txt";
    FileInfo fileInf = new FileInfo(path)
    /*თუ ფაილი არსებობს, მაშინ გასცემს true მნიშვნელობას, საწინააღმდეგო შემთხვევაში კი
    false მნიშვნელობას*/
        if (fileInf.Exists) {
            label1.Text = "ფაილის სახელი: " + fileInf.Name;
            label2.Text = "ფაილის შექმნის დრო: " +
            fileInf.CreationTime;
            label3.Text = "ფაილის ზომა: " + fileInf.Length;
        }
    }
}

```

ამოცანა 13.4. შევადგინოთ პროგრამა, რომელიც მოახდენს ფაილის წაკითხვა/ჩაწერას. პროგრამამ უნდა შეძლოს ფაილის არჩევა წასაკითხად, მისი კორექტირება და ამ ფაილის ჩაწერა დისკზე.

ფორმაზე განვალაგოთ კომპონენტები: button1, button2, RichTextBox და Label.

მართვის ელემენტი button1 თვისება Text = „ფაილის გახსნა“;

მართვის ელემენტი button2 თვისება Text = „ფაილის შენახვა“;

მართვის ელემენტი RichTextBox თვისება WordWrap = „false“;

მართვის ელემენტი RichTextBox არის მრავალსტრიქონიანი ფორმატირებული ტექსტური ველი, რომელიც მუშაობს RTF (გაფართოებული ტექსტური ფორმატი, RTF) ფორმატის ტექსტთან.

იმისათვის, რომ ავირჩიოთ ფაილი წაკითხვისთვის, უნდა გამოვიყენოთ მართვის ელემენტი OpenFileDialog. ეს ელემენტი არის Windows-ის სტანდარტული დიალოგური ფანჯარა, რომელიც განკუთვნილია ფაილის გახსნისათვის.

ფორმაზე დავამატოთ კომპონენტი OpenFileDialog.

using ჩამონათვალში დავამატოთ: using System.IO;

პროგრამაში უნდა დავამატოთ ლოკალური ცვლადები: f_open - განსაზღვრავს, აირჩია თუ არა მომხმარებელმა ფაილი; f_save - განსაზღვრავს, იქნა თუ არა შენახული ადრე გახსნილი ფაილი (f_open = true); bool f_open, f_save - ფაილის არჩევის და ფაილის შენახვის განსაზღვრა. Load მოვლენა აღიძვრება ფორმის ჩატვირთვის დროს დანართის შესრულებაზე გაშვების შემდეგ. მოვლენის დამამუშავებელში საჭიროა მოვახდინოთ ცვლადების საწყისი ინიციალიზება. ჩვენს შემთხვევაში ეს არის: f_open, f_save, label1, richTextBox1.

label1-ში აისახება არჩეული ფაილის მისამართი, ჩვენს შემთხვევაში ეს არის ფაილი student1.txt. მასში შაწერილია ქვემოთ მოცემული ტექსტი (ან ტექსტი შეგიძლიათ შეარჩიოთ შეხედულებისამებრ):

HISTORY OF FOUNDATION AND DEVELOPMENT OF THE GEORGIAN TECHNICAL

UNIVERSITY 1917 - Russian Emperor issued the order, according to which was set up Polytechnic Institute in Tbilisi, the first Higher Educational Institution in Caucasian region.

1922 - Was founded as a polytechnic faculty of Tbilisi State University.

1928 - The Departments of the polytechnic faculty merged into an independent Institute and named Georgian Polytechnic Institute (GPI).

In the 70's - The Institute consisted of 15 full-time and 13 part-time faculties.

1985 – 1987 - For the volume of the advanced scientific researches and works carried out by the students, Polytechnical Institute took first place in the USSR higher educational institutions. During this period, the Institute became the largest higher educational institution in the Caucasian region for its students (total 40 000) and academic staff (total 5000).

1990 - Georgian Polytechnic Institute was granted the university status and named as the Georgian Technical University.

1995 - Due to reforms and restructuring of curriculum, GTU gradually began installation of new training standards introducing Credit System –120 of the UK credits.

2001 - GTU became a full member of the European University Association - EUA.

2005 – GTU joined Bologna process and introduced 60 ECTS credits.

2005 - Due to the reorganizations conducted at GTU, 8 Faculties were set up.

2007 - GTU was awarded accreditation by the National Center for Educational Accreditation.

richTextBox1 -ში აისახება არჩეული ფაილის შემცველობა.

ფორმის ჩატვირთვის მოვლენის დამამუშავებელი:

```

private void Form1_Load(object sender, EventArgs e){
bool f_open, f_save; // ფაილის არჩევის და ფაილის შენახვის განსაზღვრა
private void Form1_Load(object sender, EventArgs e) {
    f_open = false; // ფაილი არ არის გახსნილი
    f_save = false;
    label1.Text = "-";
    richTextBox1.Text = "";
}
}

```

button1 კომპონენტის Click მოვლენის დამამუშავებელზე (button1_Click()-ზე) დავწეროთ ქვემოთ მოცემული კოდი:

```

private void button1_Click(object sender, EventArgs e) {
    // დიალოგური ფანჯრის გახსნა და შემოწმება არჩეულია თუ არა ფაილი
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    { // ფაილის სახელის გამოტანა label1 კომპონენტში
        label1.Text = openFileDialog1.FileName;
        // f_open და f_save ალმის დაყენება
        f_open = true;
        f_save = false;
        // richTextBox1-ში ფაილის წაკითხვა, richTextBox1-ის გასუფთავება
        richTextBox1.Clear();
        // StreamReader კლასის ობიექტის შექმნა და მონაცემების წაკითხვა ფაილიდან
        StreamReader sr = File.OpenText(openFileDialog1.FileName);
        // დამატებითი ცვლადი ფაილიდან სტრიქონების წასაკითხად
        string line = null;
        line = sr.ReadLine(); // პირველი სტრიქონის წაკითხვა
        //ფაილის სტრიქონების წაკითხვის ციკლი, თუ სტრიქონები დასრულდა, მაშინ line=null
        while (line != null)
        { // richTextBox1-ში დავამატოთ სტრიქონი
            richTextBox1.AppendText(line);
            // დავამატოთ სტრიქონის გადატანის სიმბოლო
            richTextBox1.AppendText("\r\n");
            // წავიკითხოთ მომდევნო სტრიქონი

```

```

        line = sr.ReadLine();
    }
    sr.Close(); // ფაილთან კავშირის დახურვა
}
}

```

ოპერაციული სისტემა Windows-ის სტანდარტული დიალოგური ფანჯრის გამოსამახებლად ვიყენებთ openFileDialog1 კომპონენტის ShowDialog() მეთოდს.

არჩეული ფაილი შეინახება (ინახება) openFileDialog-ის ობიექტის FileName თვისებაში.

ტექსტური ფაილის წასაკითხად გამოიყენება კლასი StreamReader, რომელიც ფაილიდან სიმბოლური მონაცემების წაკითხვას განაპირობებს.

იმისათვის, რომ შევქმნათ StreamReader კლასის ეგზემპლარი, უნდა გამოვიყენოთ File კლასის OpenText() მეთოდი.

იმისათვის, რომ ფაილიდან წავიკითხოთ სტრიქონი, პროგრამაში ხდება ReadLine() მეთოდის გამოყენება. ReadLine() მეთოდი კითხულობს მიმდინარე ნაკადის სიმბოლოების სტრიქონს და მონაცემებს აბრუნებს სტრიქონის სახით. თუ მიღწეულია ფაილის ბოლომდე, მაშინ მეთოდი აბრუნებს null.

სტრიქონების წაკითხვა ხორციელდება ლოკალური ცვლადით line.

იმისათვის, რომ დავამატოთ სტრიქონი richTextBox1 ობიექტში, უნდა გამოვიყენოთ მეთოდი AppendText().

richTextBox1 მართვის ელემენტში დავამატოთ შემდეგი ტექსტი:

1922 - was founded as a Polytechnical Faculty of Tbilisi State University
 1990 - Georgian Polytechnic Institute was granted the university status and was named the
 Georgian Technical University (GTU)

თუ ფაილის ტექსტში მოხდება ცვლილება, მაშინ f_save-ის ალამი უნდა იყოს false. richTextBox1 კომპონენტს აქვს მოვლენა TextChanged, რომელიც გენერირდება რედაქტორში ტექსტში ცვლილების შეტანის მომენტში.

მოვლენის TextChanged-ის დამამუშავებელს აქვს შემდეგი სახე:

```

private void richTextBox1_TextChanged(object sender, EventArgs e) {
    f_save = false; }

```

კომპონენტში richTextBox1 დამატებული მონაცემების ფაილში შესანახად გამოიყენება StreamWriter კლასის მეთოდი:

```

private void button2_Click(object sender, EventArgs e) {

```

```

// შემოწმება გახსნილია თუ არა ფაილი
    if (!f_open) return;
// თუ ფაილი გახსნილია, შემოწმდება შენახულია თუ არა ფაილი
    if (f_save) return;
// StreamReader კლასის ობიექტის შექმნა და სტრიქონული მონაცემების მიღება
    StreamWriter sw = File.CreateText(openFileDialog1.FileName);
// richTextBox1-დან სტრიქონების წაკითხვა და ფაილში დამატება
    string line;
    for (int i = 0; i < richTextBox1.Lines.Length; i++) {
        // ერთი სტრიქონის წაკითხვა
        line = richTextBox1.Lines[i].ToString();
        // ფაილში წაკითხული სტრიქონის დამატება
        sw.WriteLine(line);
    }
    sw.Close(); // sw-ობიექტის დახურვა
}

```

button2_Click მოვლენის დამამუშავებელში ვქმნით StreamWriter ტიპის ობიექტს (ცვლადს) sw სახელით. ობიექტის შესაქმნისას გამოიყენება File ტიპის მეთოდი CreateText(), რომელიც აბრუნებს StreamWriter ტიპის ეგზემპლარს.

ფაილის სახელი შეინახება თვისებაში openFileDialog1.FileName.

richTextBox კომპონენტში დამატებულ სტრიქონებთან წვდომისათვის, გამოიყენება თვისება Lines, რომელიც წარმოადგენს სტრიქონების მასივს.

იმისათვის, რომ დავამატოთ ერთი სტრიქონი, უნდა გამოვიძახოთ StreamWriter ტიპის sw ობიექტის WriteLine() მეთოდი.

ამოცანა 13.5. პროგრამაში ხდება ფაილში ჩაწერის დემონსტრირება StreamWriter კლასის გამოყენებით. ნაკადების კლასები განსაზღვრულია System.IO კლასში. ამიტომ, ნებისმიერი პროგრამის დასაწყისში, რომელიც იყენებს ნაკადების კლასებს, უნდა ჩავრთოთ ინსტრუქცია using System.IO; ე. ი. მივუთითოთ using ჩამონათვალის შემდეგ.

```

string strqoni;
FileStream file_out;
file_out = new FileStream("text.txt", FileMode.Create);

```

```

StreamWriter file_stream_out = new StreamWriter(file_out);
// textBox კომპონენტში შეტანილი ტექსტის მინიჭება striqoni ცვლადისათვის
striqoni = textBox1.Text;
// text.txt ფაილში striqoni სტრიქონის ჩაწერა
file_stream_out.Write(striqoni);
file_stream_out.Close();

```

ამოცანა 13.6. მოყვანილი პროგრამა კითხულობს ტექსტურ ფაილს და მისი შემცველობა ეკრანზე გამოაქვს. ტექსტური ფაილიდან წაკითხვის დემონსტრირება ხდება StreamReader კლასის გამოყენებით.

using ჩამონათვლის შემდეგ მივუთითოთ: using System.IO;

პროგრამაში ფაილიდან სტრიქონების წაკითხვა მთავრდება მაშინ, როცა ReadLine() მეთოდის მიერ გაცემული სტრიქონი მიიღებს ნულოვან მნიშვნელობას. სწორედ ეს მიუთითებს ფაილის დასასრულზე.

```

FileStream file_in;
string striqoni;
file_in = new FileStream("text.txt", FileMode.Open);
StreamReader file_stream_in = new StreamReader(file_in);
while ((striqoni = file_stream_in.ReadLine()) != null)
{
    label1.Text += striqoni + '\n';
}
file_stream_in.Close();

```

ამოცანა 13.7. მოყვანილი პროგრამა ერთი ფაილიდან წაკითხავს სიმბოლოს, მოათავსებს მათ სიმბოლოების მასივში და მასივის ელემენტებს ჩაწერს მეორე ფაილში. using ჩამონათვლის შემდეგ მივუთითოთ: using System.IO;

მიმდინარე კატალოგის WindowsFormsApplication1-ის bin ქვედირექტორიის Debug ქვედირექტორიაში შექმენით notepad ტპის ორი ფაილი: file1 და file2.

file1-ში ჩაწერეთ შემდეგი ინფორმაცია: „Georgian Technical University (GTU) is one of the biggest educational and scientific institutions in Georgia.“

```

label1.Text = "";
FileStream file_in;
char[] simboloebis_masivi = new char[19];

```

```

file_in = new
FileStream("D:\\student\\WindowsFormsApplication1\\WindowsFormsAppli
cation1\\bin\\Debug\\file1.txt", FileMode.Open);
StreamReader file_stream_in = new StreamReader(file_in);
int wakitxuli_simboloebis_raodenoba =
    file_stream_in.Read(simboloebis_masivi, 0, 19);
for (int i = 0; i < simboloebis_masivi.Length; i++)
    label1.Text += simboloebis_masivi[i].ToString();
file_stream_in.Close();
FileStream file_out;
file_out = new FileStream("file2.txt", FileMode.Create);
StreamWriter file_stream_out = new StreamWriter(file_out);
    // file2.txt ფაილში simboloebis_masivi სტრიქონის ჩაწერა
file_stream_out.Write(simboloebis_masivi);
file_stream_out.Close();

```

ამოცანა 13.8. ფაილში ბაიტებისა და სიმბოლოების წაკითხვისა და ჩაწერის გარდა შესაძლებელია სხვა ტიპის მონაცემების წაკითხვა და ჩაწერა, როგორიცაა int, double და ა.შ. მათი წაკითხვისა და ჩაწერისათვის გამოიყენება **BinaryReader** და **BinaryWriter** ნაკადები. ასეთი მონაცემების წაკითხვა და ჩაწერა ხდება შიგა ორობით ფორმატში და არა ტექსტური ფორმით.

```

BinaryReader file_in;
BinaryWriter file_out;
int mteli1 = 10, mteli2;
double wiladi1 = 1001.47, wiladi2, wiladi3;
bool b1 = true, b2;
file_out = new BinaryWriter(new FileStream("file1.dat", FileMode.Create));
file_out.Write(mteli1); // ფაილში მთელი რიცხვის ჩაწერა
file_out.Write(wiladi1); // ფაილში წილადის ჩაწერა
file_out.Write(b1); // ფაილში ლოგიკური მონაცემის ჩაწერა
file_out.Write(10.2 * 2.3); // ფაილში იწერება 10.2 * 2.3 გამოთვლის შედეგი
file_out.Close();
    // ფაილიდან წაკითხვა
file_in = new BinaryReader(new FileStream("file1.dat", FileMode.Open));

```

```

mteli2 = file_in.ReadInt32(); //მთელი რიცხვის წაკითხვა ფაილიდან
label1.Text = mteli2.ToString();
wiladi2 = file_in.ReadDouble(); //წილადის წაკითხვა ფაილიდან
label2.Text = wiladi2.ToString();
b2 = file_in.ReadBoolean(); //ლოგიკური მონაცემის წაკითხვა ფაილიდან
label3.Text = b2.ToString();
wiladi3 = file_in.ReadDouble(); //წილადის წაკითხვა ფაილიდან
label4.Text = wiladi3.ToString();
file_in.Close();

```

ამოცანა 13.9. ფაილში ჩაწერა/წაკითხვის ოპერაციების შესრულება პირდაპირი მიმართვით.

```

FileStream file;
file = new FileStream("file.txt", FileMode.Create);
char simbolo;
label1.Text = " ";
for (int i = 0; i < 26; i++) //ასოების ჩაწერა ანბანის მიხედვით
    { file.WriteByte((byte)('a' + i)); }
//ასოების წაკითხვა ფაილიდან
file.Seek(0, SeekOrigin.Begin); //პირველი ბაიტის არჩევა
simbolo = (char)file.ReadByte(); //პირველი ბაიტის წაკითხვა
label1.Text += "პირველი მნიშვნელობა " + simbolo + '\n';
file.Seek(4, SeekOrigin.Begin); //მეხუთე ბაიტის არჩევა
simbolo = (char)file.ReadByte(); //მეხუთე ბაიტის წაკითხვა
label1.Text += "მეხუთე მნიშვნელობა " + simbolo + '\n';
file.Close();

```

დამოუკიდებელი სამუშაო

1. ქვემოთ მოცემულია სამი პროგრამული კოდი. ააწყვეთ კოდის შესაბამისი ფორმა და გაუშვით შესრულებაზე. შესრულების შედეგი გადმოიტანეთ.

პროგრამული კოდი N1.

```

private void button1_Click(object sender, EventArgs e) {
    if(colorDialog1.ShowDialog() == DialogResult.OK) {
        richTextBox1.Select();
    }
}

```

```

richTextBox1.SelectionColor = colorDialog1.Color;
    }
}
private void button2_Click(object sender, EventArgs e){
    richTextBox1.Clear(); }

```

პროგრამული კოდი N2.

```

private void button1_Click(object sender, EventArgs e) {
    if(fontDialog1.ShowDialog() == DialogResult.OK) {
        richTextBox1.Select();
        richTextBox1.SelectionFont = fontDialog1.Font;
    }
}
private void button2_Click(object sender, EventArgs e) {
    richTextBox1.Clear(); }

```

პროგრამული კოდი N3.

//ტექსტის ჩატვირთვის დილაგი

```

private void button1_Click(object sender, EventArgs e) {
    openFileDialog1.Filter = "Text (*.txt) | *.txt";
    if(openFileDialog1.ShowDialog() == DialogResult.OK) {
        richTextBox1.LoadFile(openFileDialog1.FileName,
                                RichTextBoxStreamType.PlainText);
    }
}

```

//ტექსტის შენახვის დილაგი

```

private void button2_Click(object sender, EventArgs e) {
    saveFileDialog1.Filter = "Text (*.txt) | *.txt";
    if(saveFileDialog1.ShowDialog() == DialogResult.OK) {
        richTextBox1.SaveFile(saveFileDialog1.FileName,
                                RichTextBoxStreamType.PlainText); }
}

```

//დილაგი ფერი

```

private void button3_Click(object sender, EventArgs e) {
    if(colorDialog1.ShowDialog() == DialogResult.OK) {
        richTextBox1.Focus();
    }
}

```



```

        richTextBox1.SelectionColor = colorDialog1.Color; }
    }

//დილაკი ფონტი
private void button4_Click(object sender, EventArgs e) {
    if(fontDialog1.ShowDialog() == DialogResult.OK) {
        richTextBox1.Focus();
        richTextBox1.SelectionFont = fontDialog1.Font; }
    }

//დილაკი გასუფთავება
private void button5_Click(object sender, EventArgs e) {
    richTextBox1.Clear();
    richTextBox1.Focus();
    }
}

```

2. მე-13 თემის თეორიულ ნაწილში აღწერილია კოდი „საწყობის მარაგი“. კოდი ითვლის erTi cali ნივთის მთლიანი რაოდენობის და ერთი ცალი ნივთის ფასის ნამრავლს ანუ თანხის საერთო რაოდენობას. ეს კოდი გადავაკეთოთ ისეთი სახით, რომ გადსახდი გამოთვალოს მომხმარებლის მიერ არჩეული ნივთის რაოდენობის და ფასის მიხედვით. ასე მაგალითად: თუ აირჩია ნივთი - „გლობუსი“ და რაოდენობა - ორი ცალი, მაშინ კოდმა უნდა გამოთვალოს გლობუსის ფასი გადამრავლებული მომხმარებლის მიერ მითითებულ რაოდენობაზე და არა საერთო რაოდენობაზე, როგორც არის მოცემული თეორიულ ნაწილში.

დელეგატის არსი. დელეგატების მრავალმისამართიანობა. მოვლენის არსი. ფართოსამუწყებლო მოვლენა

დელეგატის არსი. დელეგატების მრავალმისამართიანობა. დელეგატს შეგვიძლია ვუწოდოთ სპეციალური კლასი, რომელიც შეიცავს მონაცემებს მეთოდის ხელმოწერის შესახებ. ნებისმიერი დელეგატი არის System.Delegate და System.MulticastDelegate ტიპის ობიექტი.

დელეგატის განსაზღვრა:

- დელეგატი არის ისეთი ობიექტი, რომელიც მიუთითებს მეთოდზე ანუ დელეგატი არის მიმთითებელი მეთოდზე და მისი დახმარებით შეგვიძლია გამოვიძახოთ მეთოდი ან მეთოდები.
- დელეგატი არის ისეთი კონსტრუქცია, რომლის დახმარებითაც შეგვიძლია გამოვიძახოთ მეთოდი. ამასთანავე შეგვიძლია გამოვიძახოთ არა ერთი, არამედ რამდენიმე მეთოდიც კი. მეთოდის გამოძახებისას კი მოხდება იმ მეთოდის გამოძახება, რომლის ხელმოწერაც ემთხვევა დელეგატის ხელმოწერას. (გავიხსენოთ, რა არის მეთოდის ხელმოწერა: ეს არის მეთოდის სახელი, მის მიერ დაბრუნებული შედეგის ტიპი და პარამეტრების სია).

როგორც ვიცით, ობიექტზე ორიენტირებულ პროგრამირებაში მეთოდი ცალკე და განყენებულად არ იწერება განსაკუთრებული შემთხვევების გარდა. მეთოდის გამოსაძახებლად უნდა არსებობდეს კლასი და ობიექტი ანუ მეთოდი პირდაპირ

კავშირშია კლასის ობიექტთან. იმ შემთხვევაში კი, როდესაც გვინდა გადავცეთ მეთოდი, მაშინ დაწვრილებითი ინფორმაცია გადასაცემი მეთოდის შესახებ შეგვიძლია განვათავსოთ დელეგატი. დელეგატი შეიცავს მეთოდის შესახებ დაწვრილებით ინფორმაციას.

ყველა მეთოდი .NET გარემოში შეიძლება დაიყოს ორ ჯგუფად: სტატიკური (static) და ჩვეულებრივი/ეგზემპლარის (instance).

თუ დელეგატი მიმართავს სტატიკურ მეთოდს, მაშინ ყველაფერი შედარებით მარტივია. ვინაიდან ამ შემთხვევაში გვაქვს ყველა საჭირო ინფორმაცია მეთოდის გამოსაძახებლად: მეთოდის მისამართი და პარამეტრები.

თუ დელეგატი მიმართავს ეგზემპლარის მეთოდს, მაშინ ამოცანა უფრო რთულდება. ეგზემპლარის მეთოდის გამოსაძახებლად, დელეგატმა უნდა იცოდეს იმ ობიექტის მიმთითებელი, რომელზეც ეს კონკრეტული მეთოდია მიბმული. გამოდის, რომ ეს მითითება ინახება თავად დელეგატის ობიექტში და მიეთითება მისი შექმნისას. ერთმანეთისაგან განსხვავებული რამდენიმე დელეგატი შეიძლება იყოს მიბმული ერთ მოვლენაზე.

დელეგატის გამოცხადების ზოგადი სინტაქსი:

public delegate დასაბრუნებელი_მნიშვნელობის_ტიპი;

დელეგატის_სახელი(პარამეტრის_ტიპი პარამეტრის_სახელი);

მაგალითად: `public delegate int MyDelegate(string a, double b);`

დელეგატის სახელია `MyDelegate()`. დელეგატს `MyDelegate()`-ს შეუძლია გამოიძახოს ნებისმიერი მეთოდი, რომელსაც დასაბრუნებელი მნიშვნელობის ტიპი აქვს `int`; პირველი პარამეტრი აღწერილი აქვთ ტიპით `string a`, ხოლო მეორე პარამეტრი კი ტიპით `double b`. ყურადღება მივაქციოთ იმას, რომ აქ საერთოდ არც კი გვიხსენებია მეთოდის სახელი. აქედან დასკვნა ასეთია: დელეგატის დახმარებით შესაძლებელია ნებისმიერი მეთოდის გამოძახება, თუ მისი ხელმოწერა (შეკვეცილი, სახელის გარეშე) შეესაბამება დელეგატის ხელმოწერას.

რა უნდა ვიცოდეთ დელეგატის თვისებების შესახებ:

1. დელეგატი არის მიმართვითი ტიპის, მაგრამ ისინი მიუთითებენ არა ობიექტებზე, არამედ მეთოდებზე, რომლებიც ასრულებენ მიმთითებლის როლს ანუ შესაძლებელია დელეგატის დახმარებით მეთოდის დამუშავება არგუმენტის როლში;

2. ერთმა მოვლენამ შეიძლება გამოიძახოს ერთდროულად რამდენიმე მეთოდი;

3. დელეგატს არ გააჩნია მეთოდის ტანი/კოდი;
4. დელეგატები ჩუმათობის პრინციპის მიხედვით არიან ობიექტორიენტირებული და უსაფრთხო ტიპებთან მიმართებით;
5. დელეგატი კოდში ახდენს საჭირო მეთოდის/მეთოდების ინკაპსულირებას;
6. დელეგატები საერთოდ არ არიან მგრძნობიარენი შეტანილ შეცდომებთან;
7. მეთოდის ხელმოწერა უნდა ემთხვეოდეს დელეგატის ხელმოწერას; მეთოდს, რომელიც ემატება დელეგატს, უნდა ჰქონდეს იგივე ტიპი და ხელმოწერა, რაც დელეგატს;

ახლა კი უფრო დაწვრილებით განვიხილოთ დელეგატის ჩაწერის სინტაქსი ეტაპების მიხედვით:

პირველი ეტაპი - გამოცხადება;

```
modifier delegate ReturnType DelegateName([Parameter_1]);
```

მეორე ეტაპი - ინიციალიზება; დელეგატის ინიციალიზირება ნიშნავს მისთვის მეთოდის სახელის გადაცემას არგუმენტის ფორმით:

```
DelegateName DelgObjectName = new DelegateName(MethodName);
```

მესამე ეტაპი - გამოძახება; დელეგატის გამოძახება შესაძლებელია პარამეტრის მითითებითაც:

```
DelegateObjectName([Parameter_1]);
```

დელეგატი შეიძლება შეგვხვდეს შემდეგი ვარიანტებით:

1. პარამეტრის გარეშე და დასაბრუნებელი მნიშვნელობის ტიპის გარეშე:

```
private delegate ExampleDelegate();
```

2. პარამეტრებით, მაგრამ ტიპის გარეშე:

```
private delegate ExampleDelegate(object item1, object item1); ან
```

```
private delegate ExampleDelegate(String text);
```

3. პარამეტრებით და ტიპებით:

```
private delegate int ExampleDelegate(object item1, object item2);
```

დელეგატს შეუძლია გამოიძახოს ის მეთოდი, რომლის ხელმოწერა შეესაბამება დელეგატის ხელმოწერას ე. ი. დელეგატი იწახავს გამოსაძახებელი მეთოდის სახელს, მის მიერ დაბრუნებული შედეგის ტიპსა და პარამეტრების სიას. მეთოდი შეიძლება იყოს ჩვეულებრივი მეთოდი ან კლასში განსაზღვრული სტატიკური

მეთოდი. მეთოდი მხოლოდ მაშინ შეიძლება გამოვიძახოთ, როცა მისი ხელმოწერა შეესაბამება დელეგატის ხელმოწერას.

```
delegate void MyCalculation(int a, double b); // დელეგატის აღწერა
void CalcSum(int a, double b) { ... } // მეთოდი შეესაბამება დელეგატს
int CalcSum(int a, double b) { ... } // მეთოდი არ შეესაბამება დელეგატს
void CalcSum(double a, int b) { ... } // მეთოდი არ შეესაბამება დელეგატს
```

დელეგატის შექმნის შემდეგ, ვიღებთ ობიექტს, რომელიც შეიცავს მეთოდზე მიმართვას. ამდენად დელეგატები გამოიყენება მეთოდების გამოძახებისათვის. ამასთან, გამოსაძახებელი მეთოდის არჩევა ხდება დინამიკურად, პროგრამის შესრულების დროს. ამიტომ, დელეგატები იმ შემთხვევაში გამოიყენება, როცა წინასწარ არ ვიცით, თუ რომელი მეთოდი უნდა გამოვიძახოთ.

მაგალითი 1. დელეგატის დახმარებით შევკრიბოთ ორი მთელი ტიპის რიცხვი.

// დელეგატი, რომელიც მიმართავს მეთოდს ორი პარამეტრით და მთელი ტიპით

```
delegate int delegatishekreba(int ricxvi1, int ricxvi2);
class Program {
    // სტატიკური მეთოდი, რომელსაც აქვს იგივე ხელმოწერა, რაც დელეგატს
    public static int shekreba(int ricxvi1, int ricxvi2) {
        int jami;
        jami = ricxvi1 + ricxvi2;
        return jami;
    }
}

private void button1_Click(object sender, EventArgs e) {
    label1.Text = "***** მიმართვა დელეგატზე *****";
    /* დელეგატის მეთოდის შექმნა და ფუნქციის პარამეტრისათვის ფუნქციის გადაცემა
    არგუმენტად */
    delegatishekreba delegatis_metodi = new
        delegatishekreba(Program.shekreba);
    int shedegi = delegatis_metodi(7, 9); // დელეგატის გამოძახება
    label2.Text="ორი მთელი ტიპის რიცხვის შეკრება: "+shedegi.ToString(); }
```

მაგალითი 2. ქვემოთ მოცემულია პროგრამული კოდი, სადაც შექმნილია დელეგატის DoubleDelegat()-ის ტიპის ორი ობიექტი, აქედან ერთი მიმართავს

არგუმენტების შეკრების მეთოდს, ხოლო მეორე გამოკლების მეთოდს, ორივე ობიექტს აქვს ერთი და იგივე ტიპი, მაგრამ მიმართავს სხვადასხვა მეთოდს.

დელეგატის ობიექტის შექმნის დროს ჩვენ ვუთითებთ მეთოდს, რომელსაც მიმართავს დელეგატი. ამ შემთხვევაში ეს არის მეთოდი Sum, ხოლო ამ მეთოდის გამოძახება შექმნილი დელეგატის მიხედვით შეგვიძლია ასე მოვიქცეთ: double sumResult = sumDelegat(24.5, 21.4);

```
//დელეგატის გამოცხადება
delegate double DoubleDelegat(double aFirstArg, double aSecondArg);
class AAA {
    // მეთოდი, რომელიც აბრუნებს ორი არგუმენტის ჯამს
    public static double Sum(double aFirstSumArg, double aSecondSumArg){
        return aFirstSumArg + aSecondSumArg; }
    // მეთოდი, რომელიც აბრუნებს ორი არგუმენტის სხვაობას
    public static double Sub(double aFirstSumArg, double aSecondSumArg){
        return aFirstSumArg - aSecondSumArg; }
}
private void button1_Click(object sender, EventArgs e){
    // DoubleDelegat-ის ეგზემპლარის შექმნა, რომელიც მიმართავს Sum მეთოდს
    DoubleDelegat sumDelegat = new DoubleDelegat(AAA.Sum);
    // Sum მეთოდის გამოძახება sumDelegat დელეგატით
    double sumResult = sumDelegat(24.5, 21.4);
    // შედეგი + "ლიმილა"
    label1.Text = "sumResult(მეთოდი Sum): " + sumResult + " \u263B ";
    // DoubleDelegat-ის ეგზემპლარის შექმნა, რომელიც მიმართავს Sub მეთოდს
    DoubleDelegat subDelegat = new DoubleDelegat(AAA.Sub);
    // Sub მეთოდის გამოძახება sumDelegat დელეგატით
    double subResult = subDelegat(24.5, 21.4);
    // შედეგი + "ლიმილა"
    label2.Text = "subResult(მეთოდი Sub): " + subResult + " \u263B ";
}
```

შეგვიძლია თუ არა დელეგატის გამოყენებით გამოვიძახოთ სხვადასხვა მეთოდი? დიახ, შესაძლებელია. თუ დელეგატი მიმართავს მეთოდს „მეთოდი1“, იმისათვის, რომ

გამოვიძახოთ „მეთოდი2“, საჭიროა, ამ დელეგატი შევცვალოთ მიმართვა ამ მეთოდზე ე. ი. მივმართოთ „მეთოდი2“ მეთოდს. პროგრამის შესრულების დროს ერთი და იგივე დელეგატი შეგვიძლია გამოვიყენოთ სხვადასხვა მეთოდის გამოსაძახებლად, ე. ი. მეთოდი, რომელსაც იძახებს დელეგატი, განისაზღვრება არა პროგრამის კომპილაციის დროს, არამედ მისი შესრულებისას.

მაგალითი 3. დელეგატი მიმართავს როგორც ჩვეულებრივ, ისე სტატიკურ მეთოდს:

```
// დელეგატის ტიპის გამოცხადება
delegate void StringProcessor(string input);

class Person {
    string name;
    public Person(string name)
    {
        this.name = name;
    }

    // ჩვეულებრივი მეთოდის გამოცხადება
    public void Say(string message) {
        MessageBox.Show(name + " " + message);
    }
}

class Background {
    // სტატიკური მეთოდის გამოცხადება
    public static void Note(string note){
        MessageBox.Show(note);
    }
}

private void button1_Click(object sender, EventArgs e) {
    Person luka = new Person("ლუკა");
    Person nino = new Person("ნინო");

    //დელეგატის სამი ეგზემპლარი
    StringProcessor lukaVoice, ninoVoice, background;
    lukaVoice = new StringProcessor(luka.Say);
    ninoVoice = new StringProcessor(nino.Say);
    background = new StringProcessor(Background.Note);
}
```

```

// დელეგატის ეგზემპლარის გამოძახება
lucaVoice("დილა მშვიდობისა " + " \u263B");
// დელეგატის ეგზემპლარის გამოძახება
ninoVoice.Invoke(" გამარჯობა " + " \u263B");
// დელეგატის ეგზემპლარის გამოძახება
background(" დღეს მზიანი და თბილი ამინდია " + " \u263A");
}

```

მაგალითი 4. დელეგატებს შეუძლიათ გამოძახონ მეთოდები სახელის მითითების გარეშე. ეს მოცემულია ქვემოთ კოდში, სადაც აღწერილია ორი კლასი:

```

class B {
    public void M(int i) {
        MessageBox.Show("B.M:" + i);
    }
}
class A {
    public void M1(int i) {
        MessageBox.Show("A.M1:" + i);
    }
    static public void M2(int i) {
        MessageBox.Show("A.M2:" + i);
    }
}
delegate void TestMethod(int i);
void Test(TestMethod m) {
    m(7);
}
private void button1_Click(object sender, EventArgs e) {
    var a = new A();
    Test(a.M1);
    Test(A.M2);
    var b = new B();
    Test(b.M);
}

```


დელეგატს შეუძლია შეინახოს რამდენიმე მეთოდის მისამართი. ამ მისამართების მიმდევრობით ინიციალიზების გზით დელეგატს შეუძლია ერთმანეთის მიყოლებით გამოიძახოს შესაბამისი მეთოდები. დელეგატის ამ თვისებას მრავალმისამართიანობა ეწოდება.

გამოსაძახებელი მეთოდების მისამართების ასეთი მიმდევრობა ანუ მეთოდებზე მიმართვების მწკრივი, შემდეგნაირად იქმნება: თავდაპირველად იქმნება დელეგატი (დელეგატი-ობიექტი), შემდეგ კი გამოიყენება += ოპერატორი მწკრივისათვის მეთოდების მისამართების დასამატებლად. შედეგად, დელეგატი მრავალმისამართიანი ხდება. მწკრივიდან მეთოდის მისამართების წასაშლელად გამოიყენება -= ოპერატორი. ერთადერთი შეზღუდვაა ის, რომ დელეგატს, რომელიც ინახავს რამდენიმე მიმართვას, უნდა ჰქონდეს დასაბრუნებელი მნიშვნელობის void ტიპი.

მაგალითი 5. დელეგატების მრავალმისამართიანობის დემონსტრირება.

```
delegate void Message();
class ABC_student {
    public static void Hello() {
        MessageBox.Show("დილა მშვიდობისა!");
    }
    public static void HowAreYou() {
        MessageBox.Show("როგორ ხარ? მომენატრე \u263B ");
    }
}

private void button1_Click(object sender, EventArgs e) {
    Message mes1 = ABC_student.Hello;
    mes1 += ABC_student.HowAreYou; // mes1 მიუთითებს ორივე მეთოდზე
    mes1(); // Hello და HowAreYou მეთოდების გამოძახება
    mes1 -= ABC_student.HowAreYou; // HowAreYou მეთოდის წაშლა
    mes1(); // Hello მეთოდის გამოძახება
    Message mes2 = ABC_student.HowAreYou;
    Message mes3 = mes1 + mes2; // დელეგატების გაერთიანება
    mes3(); // ორივე მეთოდის გამოძახება mes1-დან და mes2-დან
}
```

დელეგატები მჭიდროდ არიან დაკავშირებული C# ენის სხვა ელემენტებთან:

- დელეგატებზე შეიძლება განხორციელდეს დამატებისა და გამოკლების ოპერატორები;

- დელეგატები მუშაობენ მეთოდებთან, ანონიმურ მეთოდებთან და ლამბდა გამოსახულებებთან;

- დელეგატები მიიღებული არიან მოვლენებთან.

ზემოთ მოცემული ჩამონათვალი, სურვილის შემთხვევაში, შეიძლება განხორციელდეს კლასებისა და ინტერფეისების გამოყენებით, მაგრამ ენის შესაძლებლობებს დელეგატის ფუნქციონალი უფრო ფართოდ წარმოაჩენს და საშუალებას გვაძლევს, დაწეროთ შედარებით მოკლე კოდი. თუ შესაძლებელია დაწეროთ კოდი დელეგატების გარეშე, დაწერეთ იგი, მაგრამ თუ იმავე კოდი შეიძლება უფრო სწრაფად და მარტივად დაიწეროს დელეგატების დახმარებით, მაშინ უფრო მართებული იქნებოდა დელეგატების გამოყენება.

მოვლენის არსი. ფართოსამაუწყებლო მოვლენა. ადამიანები რეალურ სამყაროში მუდმივად გარშემორტყმული ვართ სხვადასხვა მოვლენებით: სამსახურში წასვლა, სამსახურიდან დაბრუნება, ხელ-პირის დაბანა, ნაბიჯის გადაგმა წინ, თავის მობრუნება, თვალის ჩაპაჭუნება ან თვალის დახუჭვა და ა. შ. ყველა ამ მოვლენის საპასუხოდ ჩვენს ორგანიზმში/თავში/გულში წარმოიქმნება საპასუხო რეაქციები.

მოვლენები შეიძლება დაიყოს ორ ტიპად: ის, რომელსაც ვიწვევთ ჩვენ თვითონ და ის, რომელიც არის გამოწვეული/ინიცირებული სხვის მიერ.

მაგალითად, სამსახურიდან დავბრუნდით სახლში. სამსახურიდან სახლში დაბრუნებამ გამოიწვია რაღაცა მოვლენა, რომლის საპასუხოდ ჩვენ რაღაცა გავაკეთეთ: გადავედით მდგომარეობაში „სახლი“ ან მდგომარეობაში „მყუდრო სახლი“, მაგრამ თუ მივდივართ ქუჩაში და შემთხვევით უცხო ადამიანი დაგვეჯახა, მაშინ ჩვენში სხვა ადამიანმა გამოიწვია რეაქცია და სწორედ ის სხვა ადამიანი არის მოვლენის გამომწვევი/ინიციატორი.

პროგრამული გარემოში ყველაფერის ასახვა ხდება ცხოვრებისეული პროცესების მსგავსად ანუ პროგრამირება ზუსტად ასახავს რეალურ ცხოვრებისეულ პროცესებს.

არსებობს მოვლენები/ქმედებები, რომლებიც ინიციალიზებულია თავად აპლიკაციის (ე. წ. პროგრამის) მიერ და არიან ისეთი მოვლენებიც, რომლებიც

თვითონ მომხმარებელის მიერ არის ინიციალიზებული, მაგალითად, აპლიკაციის გრაფიკულ ინტერფეისში წარმოდგენილ ბრძანებით ღილაკზე button-ზე დაჭერა.

მოვლენები არ არსებობს განმარტოებულ მდგომარეობაში, მოვლენები აუცილებლად საჭიროებს რაიმე სახის რეაქციას/ქმედებას.

რა სახის რეაქცია შეიძლება იყოს პროგრამაში ანუ როგორ გამოისახება რეაქცია პროგრამულად? მოვლენისათვის პასუხის გასაცემად უნდა გაეშვას/დაიწყოს მუშაობა რაღაცა სხვა პროგრამამ, რომლის ტანშიც მოხდება ისეთი ინფორმაციის დამუშავება, რომელიც დაკავშირებულია მიმდინარე მოვლენასთან. ასეთი სახით გაშვებულ პროგრამას ეწოდება მოვლენის დამამუშავებელი. თუ მოვლენის დასახელებაზე ორჯერ დააწკაპუნებთ, პროგრამული გარემო მომხმარებლისათვის/პროგრამისტისათვის ავტომატურად შექმნის და გახსნის ამ მოვლენის ცარიელ დამამუშავებელს (ე. წ. შაბლონის).

ცარიელი დამამუშავებელი (შაბლონი) შეიცავს მხოლოდ მეთოდის სათაურს და ცარიელ ტანს, რომელშიც პროგრამისტს შეუძლია შეიტანოს მოვლენით გამოწვეული რეაქციის ამსახველი პროგრამის კოდები/ბრძანებები.

მაგალითად, ფორმაზე გადმოვიტანოთ ბრძანებითი ღილაკი (button). თუ ბრძანებითი ღილაკზე „მაუსით“ დავაჭრეთ ორჯერ, მაშინ ეს დაჭერა გადაგვიყვანს დაჭერის მოვლენის დამამუშავებელზე. სწორედ ამ დამამუშავებელის ტანში შეგვიძლია ჩავწეროთ კოდი. „ბრძანებითი ღილაკის button-ის მიმდინარე ფერის შეცვლა და ლურჯი ფერით წარმოდგენა“.

თუ ყურადღებით დავაკვირდებით მოვლენის დამამუშავებლების სათაურებს, ვნახავთ, რომ ყველა აწყობილია ერთსა და იმავე სტანდარტის მიხედვით:

მოვლენის_დამამუშავებლის_სახელი (როგორც წესი, სახელი უკავშირდება იმ ობიექტის სახელს, რომლის მოვლენაც უნდა დამუშავდეს; მაგალითად, ღილაკი სახელით Button1-სათვის მოვლენის დამამუშავებელს ჰქვია Button1_Click) და **მრგვალ ფრჩხილებში მოთავსებული ორი პარამეტრი** (Object sender, EventArgs e).

ახლა კი ვნახოთ მოვლენის დამამუშავებლის ორი მაგალითი: მაუსით ფორმაზე დაჭერა და მაუსით ბრძანებითი ღილაკზე დაჭერა (ბრძანებითი ღილაკი იმყოფება ფორმაზე):

```
private void Form1_Load(object sender, EventArgs e) { }  
private void button1_Click(object sender, EventArgs e) { }
```

დამამუშავებლის პარამეტრებია: თავად ობიექტი, რომელმაც წარმოქმნა მოვლენა (sender შეიცავს ამ ობიექტზე მიმართვას/მიმთითებელს) და მოვლენის დამამუშავების მონაცემები, რომლებიც იმყოფებიან EventArgs-ს ტიპის ობიექტში e. პარამეტრის ტიპი EventArgs-ს არის object კლასის მემკვიდრე, ამიტომ მოვლენის დამამუშავებლებს შეგვიძლია გადავცეთ ყველაფერი რაც გვინდა. თუ კარგად დავაკვირდებით მოვლენის დამამუშავებლებს, დავინახავთ, რომ ყველა ისინი აბრუნებს ერთსა და იმავე ტიპის მნიშვნელობას და ეს ტიპი არის void (სინამდვილეში void ტიპის გამო, მოვლენის დამამუშავებელი არაფერსაც არ აბრუნებს, ასრულებს მხოლოდ მის ტანში მოცემული კოდების ჩანაწერებს). მოვლენის ყველა დამამუშავებელს აქვს პარამეტრების და ტიპების ერთი და იგივე ნაკრები.

იმდენად, რამდენადაც პარამეტრის ტიპი არის object, მოვლენის დამამუშავებელს ჩვენ შეგვიძლია გადავცეთ ყველაფერი, რაც გვსურს. ყველა მოვლენის დამამუშავებელს აქვს მნიშვნელობის დაბრუნების ერთი და იგივე ტიპი ანუ void ე. ი. დამამუშავებელი არაფერს აბრუნებს, არამედ ასრულებს დამამუშავებლის ტანში მოცემულ კოდს. ამასთანავე მოვლენის დამამუშავებლებს აქვთ პარამეტრების და ტიპების ერთი და იგივე ნაკრები. ყველაფერი ზემოთქმულიდან გამომდინარე უნდა ვიფიქროთ, რომ მოვლენის დამამუშავებლის ხელმოწერების ერთნაირი სტანდარტის გამო, ასევე უნდა არსებობდეს მოვლენის დამამუშავებლის გაშვების საერთო მექანიზმი და ეს საერთო მექანიზმი არის - დელეგატი.

სწორედ დელეგატს შეუძლია გაუშვას (გამოიძახოს მუშაობისათვის/გაშვებისათვის) ასეთი სახის მეთოდები, მხოლოდ იმ შემთხვევაში, თუ დელეგატის ხელმოწერა/სიგნატურა დაემთხვევა ანუ იგივე იქნება, რაც მოვლენის დამამუშავებლის ხელმოწერა/სიგნატურა.

მოვლენები დელეგატების გამოყენების კერძო შემთხვევაა. თავად დელეგატი არსებითად არის მითითება მეთოდზე მკაცრად განსაზღვრული ხელმოწერით. ამა თუ იმ მოვლენის შესრულებისათვის პროგრამას შეუძლია შესაბამისი რეაგირება. მოვლენაზე რეაქცია ხორციელდება ე. წ. მოვლენის დამამუშავებლის საშუალებით. რა არის მოვლენის დამამუშავებელი? მოვლენის დამამუშავებელი ეს არის ჩვეულებრივი მეთოდი, რომელიც ასრულებს რაიმე ქმედებას, იმ შემთხვევაში, თუ მოხდა მოვლენის გენერირება.

მოვლენა იქმნება დელეგატების საფუძველზე ანუ მოვლენა მუშაობს დელეგატთან ერთად კავშირში. ასეთი კავშირი გვაძლევს საშუალებას, მოვახდინოთ მოვლენის დამამუშავებლის (მეთოდების) მწკრივების ფორმირება, რომელთა გამოძახებაც მოხდება მოვლენის გენერირების დროს. მოვლენა - ეს არის დელეგატთან მიმთითებლის დაკავშირების საშუალება, ხოლო დელეგატი კი მეთოდზე მიმთითებულია ანუ ჩვენ უბრალოდ შეგვიძლია ჩავრთოთ ან გამოვრთოთ დელეგატი.

ნებისმიერი მოვლენის შიგნით არის ორი მეთოდი: დამატება და წაშლა, რომელიც უბრალოდ ჩართავენ ან გამორთავენ დელეგატს.

მოვლენები წარმოადგენენ კლასის წევრებს და მათი გამოცხადება ხდება event საკვანძო სიტყვის გამოყენებით. მოვლენების გამოცხადების სინტაქსია:

event დელეგატი_სახელი მოვლენის_სახელი;

დელეგატი_სახელი - დელეგატის ტიპის სახელი, რომელიც გამოიყენება მოვლენის გამოცხადებისათვის; მოვლენის_სახელი - „მოვლენის“ ტიპის ობიექტის (ცვლადის) კონკრეტული სახელი. მაგალითად, გამოვაცხადოთ მოვლენა სახელით MyEvent დელეგატის MyDelType ტიპისა: `event MyDelType MyEvent;`

რა მოთხოვნებს უნდა აკმაყოფილებდეს დელეგატი, რომელსაც გამოვიყენებთ მოვლენის დამამუშავებისათვის? იმისათვის რომ მოვლენის დამამუშავებლის (მეთოდის) სია, მწკრივი დავამუშაოთ, დელეგატმა არ უნდა დააბრუნოს მნიშვნელობა ე. ი. დელეგატმა უნდა დააბრუნოს მხოლოდ void ტიპი. თუ დელეგატი აბრუნებს მნიშვნელობას მაშინ, მოხდება მხოლოდ ბოლო მეთოდის გამოძახება ფორმირებული მეთოდების მწკრივიდან.

რა მოთხოვნებს უნდა აკმაყოფილებდეს მეთოდი, რომელიც შეიძლება იყოს რომელიმე მოვლენის დამამუშავებელი? ძირითადი მოთხოვნა არის ის, რომ მეთოდს უნდა ჰქონდეს იგივე სიგნატურა რაც დელეგატს, რომლის საფუძველზეც არის გამოცხადებული მოვლენა. თუ საჭიროა მოვახდინოთ მეთოდების მწკრივის ორგანიზება, რომლებიც გამოიძახება მოვლენის გამოძახებისას, დელეგატის და მეთოდის ტიპებმა უნდა დააბრუნონ void მნიშვნელობა.

როგორ ხდება მეთოდის რეგისტრაცია მოვლენაში? იმისათვის, რომ მოვახდინოთ მეთოდის რეგისტრაცია მოცემული მოვლენის დამამუშავებისათვის, უნდა გამოვიყენოთ ოპერატორები '=' ან '+='.

ახლა კი განვიხილოთ მოვლენა ცოტა უფრო პრაქტიკული კუთხით. ვთქვათ, რომელიმე კლასში უნდა გამოვაცხადოთ:

- მოვლენა სახელით MyEvent;
- მოვლენის დამამუშავების ერთი მეთოდი (მოვლენის დამამუშავებელი) MyMethod1();
- მოვლენის დამამუშავებელი მეორე მეთოდი MyMethod2();
- მოვლენის დამამუშავებელი მესამე მეთოდი MyMethod3().

მეთოდებს აქვთ ზუსტად იგივე სიგნატურა, რაც დელეგატის ტიპს, რომლის საფუძველზეც ვაცხადებთ MyEvent მოვლენას. იმისათვის, რომ დავარეგისტრიროთ MyEvent მოვლენის მეთოდები MyMethod1(), MyMethod2(), MyMethod3() შეგვიძლია დავწეროთ შემდეგი პროგრამული კოდი:

```
// MyEvent მოვლენის მეთოდების რეგისტრაცია
MyEvent += MyMethod1(); // MyEvent => MyMethod1
MyEvent += MyMethod2(); // MyEvent => MyMethod1 -> MyMethod2
MyEvent += MyMethod3(); // MyEvent => MyMethod1 -> MyMethod2 -> MyMethod3
```

ამ შემთხვევაში წარმოიქმნება სია (მწკრივი), რომელიც შედგება სამი მეთოდისაგან. სიაში მეთოდები ზუსტად იმ მიმდევრობით იმყოფება, როგორც მოხდა მათი რეგისტრირებისას მოვლენაში.

მაგალითი 5.

```
delegate void MyEventHandler(); // დელეგატის ტიპის გამოცხადება
// კლასის გამოცხადება. შეიცავს მოვლენას.
class MyEvent {
    public event MyEventHandler SomeEvent;
    // მეთოდი გამოიძახება მოვლენის წარმოქმნისას
    public void OnSomeEvent() {
        /* მოვლენის დამამუშავებელი წარმოიქმნება მხოლოდ მაშინ, თუ მოვლენა SomeEvent
        არ არის ცარიელი. მოვლენის შემოწმება, რათა ის არ იყოს ცარიელი */
        if (SomeEvent != null)
            SomeEvent(); }
    // მოვლენის დამამუშავებელი
    public static void Handler() {
```

```

        MessageBox.Show("წარმოიქმნა მოვლენა ");
    }
}

private void button1_Click(object sender, EventArgs e) {
    MyEvent evt = new MyEvent();
    // დავამატოთ მეთოდი Handler() მოვლენის სიას
    evt.SomeEvent += MyEvent.Handler;
    // მოვლენის გაშვება
    evt.OnSomeEvent();
}

```

OnSomeEvent მეთოდის გამოძახებას მივყავართ ყველა იმ მოვლენის გამოძახებასთან, რომელიც მოვლენის დამამუშავებლის მიერ არის დარეგისტრირებული. ამ შემთხვევაში დარეგისტრირებულია მხოლოდ ერთი მოვლენის დამამუშავებელი, მაგრამ ისინი შეიძლება იყოს მეტი, როგორც ქვემოთ მაგალითში არის ნაჩვენები. ამდენად მოვლენა არის დელეგატის სპეციალური ობიექტი, იგი წარმოადგენს მეთოდების შენახვის ადგილს, ეს მეთოდები შესრულდება ერთდროულად, როგორც კი წარმოიქმნება მოვლენა.

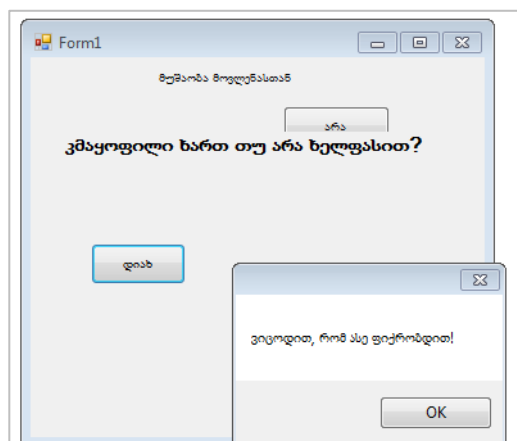
მაგალითი 6. განვიხილოთ მოვლენის წარმოდგენის საკმაოდ მარტივი და ფუნქციური კოდი (იხ. ნახ. 50.):

```

private void button1_Click(object sender, EventArgs e) {
    MessageBox.Show("ვიცოდით, რომ ასე ფიქრობდით!"); }

private void button2_MouseMove(object sender, MouseEventArgs e) {
    button2.Top -= e.Y;
    button2.Left += e.X;
    if (button2.Top < -10 || button2.Top > 100) button2.Top = 60;
    if (button2.Left < -80 || button2.Left > 250) button2.Left = 120;}

```



ნახ. 50. მე-6 მაგალითის შესრულების შედეგი

ვიზუალური კომპონენტების უმრავლესობის მოვლენის დამამუშავებელს აქვს ორი პარამეტრი:

1. sender - ეს არის ობიექტი, რომელიც გამოიძახა მოვლენამ. იგი არის Object ტიპის;
2. არგუმენტი e - ინახავს ინფორმაციას მოვლენის შესახებ. არგუმენტი e არის EventArgs კლასის არგუმენტი და საჭიროა, რათა მოვლენის დამამუშავებელს გადავცეთ რაიმე დამატებითი მონაცემები.

კლასი EventArgs წარმოადგენს იმ კლასების საბაზო კლასს, რომლებიც წარმოადგენენ მოვლენის მონაცემებს.

ანონიმური მეთოდები. ანონიმური მეთოდი არ არის ისეთი, როგორც ტრადიციული მეთოდი ე. ი. არ არის რაიმე კლასის მეთოდი. ანონიმური მეთოდი იქმნება მხოლოდ იმისთვის, რომ გამოვიყენოთ როგორც მიზნობრივი მეთოდი დელეგატისთვის.

ანონიმური მეთოდი ეს არის მეთოდი, რომელსაც არ გააჩნია სახელი და აქვს მხოლოდ პროგრამული კოდის ბლოკი, რომელსაც ის ასრულებს. ანონიმური მეთოდი მოხერხებულია გამოვიყენოთ დელეგატებთან ერთად (მიმართებით).

ანონიმური მეთოდის გამოძახება ხდება მხოლოდ დელეგატის დახმარებით. როდის არის მიზანშეწონილი ანონიმური მეთოდის გამოყენება? ხშირად ხდება, რომ დელეგატი მიმართავს მხოლოდ ერთადერთ მეთოდს. ასეთ დროს მეთოდის სახელის გამოყენება არ არის აუცილებელი. საჭიროა მხოლოდ მეთოდის პროგრამული ბლოკის მითითება ე. ი ხდება უსახელო პროგრამული კოდის გამოცხადება.

ანონიმური მეთოდის გამოყენებას აქვს შემდეგი უპირატესობა:

- მარტივია გამოსაყენებლად;
- არ არის საჭირო ცალკე მეთოდის გამოცხადება, რომელიც განკუთვნილი იქნებოდა აუცილებლად დელეგატისათვის გადასაცემად.

ანონიმური მეთოდის შექმნის სინტაქსია:

delegate (პარამეტრები)

{

// ანონიმური მეთოდის კოდი


```
};
```

პარამეტრები უნდა შეესაბამებოდეს დელეგატის პარამეტრებს.

მაგალითი 6. ანონიმური მეთოდის გამოყენების დემონსტრირება. მეთოდი არც იღებს და არც აბრუნებს მნიშვნელობას. მეთოდს ფორმაზე გამოაქვს შეტყობინება „Hello world!!!!“.

```
delegate void HELLO();
private void button1_Click(object sender, EventArgs e){
    // ტექსტის გამოტანა დელეგატით
    // დელეგატის ობიექტის სახელია HW, მეთოდს სახელი არ გააჩნია
    HELLO HW = delegate
    {
        // მეთოდის პროგრამული კოდი
        label1.Text = "დილა მშვიდობისა სამყაროვ!";
    };
    HW(); // ანონიმური მეთოდის გამოძახება დელეგატის დახმარებით
}
```

ანონიმურ მეთოდს შეიძლება გადავცეთ პარამეტრები, მაგრამ როგორ მოვახდინოთ მნიშვნელობის დაბრუნება. ამისათვის გამოიყენება საკვანძო სიტყვა return.

ანონიმური მეთოდის განზოგადებული სახე, რომელიც იღებს პარამეტრებს და აბრუნებს მნიშვნელობას, არის შემდეგი სახის:

```
delegate (პარამეტრების_სია)
{
    // მეთოდის პროგრამული კოდი
    return სიდიდე;
};
```

სადაც, სიდიდე ეს არის ცვლადი, გამოსახულება ან მნიშვნელობა, რომელსაც აბრუნებს დელეგატი.

მაგალითი 7. დავწეროთ ანონიმური მეთოდი, რომელმაც უნდა გამოთვალოს სამკუთხედის ფართობი ჰერონის ფორმულით. მეთოდი პარამეტრებად იღებს სამკუთხედის სამ გვერდს: a, b, c. თუ მივაწოდებთ არაკორექტულ მნიშვნელობას, მაშინ დაგვიბრუნოს უარყოფითი რიცხვი (-1).

```
delegate double SquareTriangle(double a, double b, double c);
private void button1_Click(object sender, EventArgs e) {
```

```

SquareTriangle ST;
ST = delegate(double a, double b, double c) {
    double s, p, d;
    p = (a + b + c) / 2.0;
    d = p * (p - a) * (p - b) * (p - c);
    if (d < 0) return -1.0;
    else
    {
        s = Math.Sqrt(p * (p - a) * (p - b) * (p - c));
        return s;
    }
};
double aa, bb, cc; //a, b, c
aa = double.Parse(textBox1.Text);
bb = double.Parse(textBox2.Text);
cc = double.Parse(textBox3.Text);
double area; //დელეგატის გამოძახება
area = ST(aa, bb, cc);
label1.Text="S = "+area.ToString(); //ფორმაზე შედეგის გამოტანა
}

```

განვიხილოთ რამდენიმე ამოცანა.

ამოცანა 14.1. დავწეროთ პროგრამული კოდი დელეგატის დახმარებით, რომელიც დაადგენს, დღის რომელი ნახევარია: პირველი თუ მეორე?

```

delegate void Message(); // დელეგატის გამოცხადება
class ABC {
    public static void GoodMorning() {
        MessageBox.Show("დილა მშვიდობისა, დღის პირველი ნახევარი");
    }
    public static void GoodEvening() {
        MessageBox.Show("სალამო მშვიდობისა, დღის მეორე ნახევარი");
    }
}
private void button1_Click(object sender, EventArgs e) {
    label1.Text = "";
}

```

```

Message mes; // დელეგატის ობიექტის შექმნა
if (DateTime.Now.Hour < 12)
{
    mes = ABC.GoodMorning;
}
else
{
    mes = ABC.GoodEvening;
}
mes(); // მეთოდის გამოძახება
}

```

ამოცანა 14.2. განვიხილოთ „ბუშტივური“ დალაგების ალგორითმის გადაწყვეტა დელეგატის გამოყენებით. შევქმნათ სამი კლასი შემდეგი დასახელებებით: BubbleSortClass; Student; Program;

კლასი BubbleSortClass შეიცავს სტატიკურ მეთოდს Sort(), რომელსაც არ აქვს დასაბრუნებელი მნიშვნელობა. არგუმენტად მას გადაეცემა მასივი, რომელსაც დავალაგებთ ორი ელემენტის ერთმანეთთან შედარებით:

```

public class BubbleSortClass {
    static public void Sort(object[] sortArray,
                           Program.CompareDelegategtMethod)
    {
        for (int i = 0; i < sortArray.Length; i++)
        {
            for (int j = 0; j < sortArray.Length; j++)
            {
                if (gtMethod(sortArray[j], sortArray[i]))
                {
                    object temp = sortArray[i];
                    sortArray[i] = sortArray[j];
                    sortArray[j] = temp;
                }
            }
        }
    }
}

```

```

    }
}

class Student {
    private string name;
    private int rollno;
    private int marks;
    // კლასის ობიექტის ინიციალიზება
    public Student(string name, int rollno, int marks) {
        this.name = name;
        this.rollno = rollno;
        this.marks = marks;
    }

    // მეთოდის თავიდან განსაზღვრა შედეგების გამოსატანად
    public override string ToString() {
        return string.Format( name + ", " + rollno + ", " + marks);
    }

    //შედარების სამომხმარებლო ფუნქცია, აბრუნებს bool ტიპის მნიშვნელობას
    public static bool RhsIsGreater(object lhs, object rhs) {
        Student stdLhs = (Student)lhs;
        Student stdRhs = (Student)rhs;
        return stdRhs.marks > stdLhs.marks;
    }
}

public class Program {
    //დელეგატის გამოცხადება, რომელიც მიმართავს მეთოდს
    //გვაქვს ორი პარამეტრისა და bool ტიპის შედეგი
    public delegate bool CompareDelegate(object lhs, object rhs);
}

private void button1_Click(object sender, EventArgs e) {
    label1.Text = "";
    // Student.cs კლასის ობიექტების შექმნა
    Student[] students = {
        new Student("ლუკა", 1, 799),
        new Student("დათი", 2, 545),
    }
}

```

```

        new Student("ლაშა", 3, 999),
        new Student("სერგი", 4, 228),
        new Student("დარიკო", 5, 768),
        new Student("მიხეილი", 6, 367),
        new Student("ნინო", 7, 799),
        new Student("მარიამი", 8, 867),
        new Student("ბექა", 9, 978),
        new Student("ნიკოლოზი", 10, 567) };

Program.CompareDelegate StudentCompareOp = new
    Program.CompareDelegate(Student.RhsIsGreater);
    BubbleSortClass.Sort(students, StudentCompareOp);
    for (int i = 0; i < students.Length; i++)
    {
        label1.Text +=students[i].ToString() + '\n';
    }
}

```

ამოცანა 14.2. გამოვაცხადოთ დელეგატი (ტიპი double; სახელი CalcFigure), რომელიც მიმართავს მეთოდს. მოთხოვნა მეთოდის სიგნატურისადმი შემდეგია: მეთოდი იღებს პარამეტრად double ტიპის ცვლადს; მეთოდი აბრუნებს double ტიპის მნიშვნელობას;

მეთოდის გამოძახება ხდება დელეგატის მიერ.

მეთოდი იღებს რადიუსის R მნიშვნელობას და ითვლის: წრის სიგრძეს $D = 2 * \pi * R$; წრის ფართობს $S = \pi * R^2$; სფეროს მოცულობას $V = 4/3 * \pi * R^3$.

მეთოდები გამოვაცხადოთ როგორც სტატიკური. გამოვიყენოთ საკვანძო სიტყვა static.

```

//დელეგატის ტიპის CalcFigure გამოცხადება
delegate double CalcFigure(double r);

//მეთოდების გამოცხადება
public double Get_Length(double r) {
    double length;
    length = 2 * 3.1415 * r;
    return length;
}

```

```

    }
    public double Get_Area(double r) {
        double area;
        area = 3.1415 * r * r;
        return area;
    }
    public double Get_Volume(double r) {
        double volume;
        volume = 3.1415 * r * r * r * 4.0 / 3.0;
        return volume;
    }
    private void button1_Click(object sender, EventArgs e) {
        double radius, length, area, volume;
        radius = Convert.ToDouble(textBox1.Text); // რადიუსი
        CalcFigure CF = Get_Length; // დელეგატის გამოძახება
        length = CF(radius); // Get_Length() მეთოდის გამოძახება
        CF = Get_Area;
        area = CF(radius); // Get_Area() მეთოდის გამოძახება
        CF = Get_Volume;
        volume = CF(radius); // Get_Volume() მეთოდის გამოძახება
        label1.Text = "რადიუსი=";
        label2.Text = "წრის სიგრძე=" + length.ToString();
        label3.Text = "წრის ფართობი=" + area.ToString();
        label4.Text = "სფეროს მოცულობა=" + volume.ToString();
    }

```

ამოცანა 14.3. მოვლენის დახმარებით მეთოდების მუშაობის დემონსტრირება. გეომეტრიული ფიგურის წრის მახასიათებლების გამოთვლა ხდება დელეგატების მოვლენის დამამუშავებელი მეთოდების დახმარებით. შევქმნათ მეთოდების მწკრივი, რომელებიც გამოიძახება მოვლენიდან.

```

delegate void CalcFigure(double R); // დელეგატის ტიპის გამოცხადება
event CalcFigure ECF; // მოვლენის გამოცხადება სახელით ECF
// მოვლენის დამამუშავებელი მეთოდები

```

```

// მეთოდებს აქვთ იგივე სიგნატურა, რაც დელეგატის ტიპს
void GetLength(double R) // წრის სიგრძე
{
    double res;
    const double Pi = 3.1415;
    res = 2 * Pi * R;
    label1.Text = "წრის სიგრძე = " + res.ToString();
}
void GetArea(double R) // წრის ფართობი
{
    double res;
    const double Pi = 3.1415;
    res = Pi * R * R;
    label2.Text = "წრის ფართობი = " + res.ToString();
}
void GetVolume(double R) // სფეროს მოცულობა
{
    double res;
    const double Pi = 3.1415;
    res = 4.0 / 3.0 * Pi * R * R * R;
    label3.Text = "სფეროს მოცულობა = " + res.ToString();
}
private void button1_Click(object sender, EventArgs e) {
    // მოვლენის დახმარებით მეთოდების მუშაობის დემონსტრირება
    // შევექმნათ მეთოდების მწკრივი, რომელებიც გამოიძახებიან ECF მოვლენიდან
    ECF = GetLength; // ECF => GetLength()
    ECF += GetArea; // ECF => GetLength() -> GetArea()
    ECF += GetVolume; // ECF => GetLength() -> GetArea() -> GetVolume()
    // მოვლენის გამოძახება პარამეტრით
    // გამოიძახება მიმდევრობით სამი: GetLength(), GetArea(), etVolume()
    ECF(4.0);
}

```

ამოცანა 14.4. მრავალმისამართიანი მოვლენის დემონსტრირება.

```

// დელეგატის ტიპის გამოცხადება მოვლენისათვის
delegate void MyEventHandler();
// კლასის გამოცხადება, რომელიც შეიცავს მოვლენას
class MyEvent {
    public event MyEventHandler SomeEvent;
    // ეს მეთოდი გამოიძახება მოვლენის გასაშვებად
    public void OnSomeEvent() {
        if (SomeEvent != null)
            SomeEvent();
    }
}
class X {
    public void Xhandler()
    {
        MessageBox.Show("X კლასის ობიექტის მიერ მიღებული
მოვლენა");
    }
}
class Y {
    public void Yhandler()
    {
        MessageBox.Show("Y კლასის ობიექტის მიერ მიღებული მოვლენა");
    }
}
class EventDemo2 {
    public static void Handler()
    {
        MessageBox.Show("EventDemo კლასის ობიექტის მიერ მიღებული მოვლენა"); }
}
private void button1_Click(object sender, EventArgs e) {
    MyEvent evt = new MyEvent();
    X xOb = new X();
    Y yOb = new Y();
    // მოვლენის სიაში დამამუშავებლის დამატება

```


// შევექმნათ მეთოდების მწკრივი, რომელებიც გამოიძახებინ SomeEvent მოვლენიდან

```
    evt.SomeEvent += EventDemo2.Handler;
    evt.SomeEvent += xOb.Xhandler;
    evt.SomeEvent += yOb.Yhandler;
    //მოვლენის გაშვება
    evt.OnSomeEvent();
    // მოვლენის წაშლა
    evt.SomeEvent -= xOb.Xhandler;
    evt.OnSomeEvent();
}
```

ამოცანა 14.5. დელეგატების მრავალმისამართიანობა. დელეგატს შეუძლია ერთმანეთის მიყოლებით გამოიძახოს არა ერთი, არამედ რამოდენიმე მეთოდი:

```
public delegate void Message();
public void Hello()
{
    MessageBox.Show("დილა მშვიდობისა");
}
public void HowAreYou()
{
    MessageBox.Show("როგორ ხართ?");
}
private void button1_Click(object sender, EventArgs e) {
    Message mes1 = Hello;
    mes1 += HowAreYou; //mes1 მიმართავს ორ მეთოდს,
    mes1(); //ორივე მეთოდის - Hello и HowAreYou - გამოიძახება
}
```

დამოუკიდებელი სამუშაო

1. მოცემულია პროგრამული კოდი. დაწერეთ მისი მოცემულობა და გადააკეთეთ იგი ისეთი სახით, რომ გამოიყენოთ კომპონენტი textBox.

```
public delegate int AddDelegate(int num1, int num2);

public int Add(int num1, int num2)
```

```

    {

        int sumation;

        sumation = num1 + num2;

        return sumation;

    }

private void button1_Click(object sender, EventArgs e)

{ //

    AddDelegate funct1 = new AddDelegate(Add);

    //

    int k = funct1(7, 2);

    label1.Text="Sumation = "+ k;

}

```

2. მოცემულია პროგრამული კოდი. დაწერეთ მისი მოცემულობა, ასევე აღწერეთ თითოეული სტრიქონი კომენტარების დართვით, გადააკეთეთ იგი ისეთი სახით, რომ გამოიყენით კომპონენტი textBox.

```

public delegate int Operation(int I, int j);
// მეთოდები ერთსა და იმავე სიგნატურით
public static int summa(int x, int y)
{
    return x + y;
}
public static int subtract(int x, int y)
{
    return x - y;
}
public static int multiply(int x, int y)
{
    return x * y;
}

```

```

    }
    public static int divide(int x, int y)
    {
        return x / y;
    }
    private void button1_Click(object sender, EventArgs e)
    {
        int result;
        Operation op; //
        op = new Operation(summa); //
        result = op(2, 8);
        label1.Text = "ჯამი: " + result;
        op = new Operation(subtract); //
        result = op(2, 8);
        label2.Text = "სხვაობა: " + result;
    }

```

3. ქვემოთ მოცემული პროგრამულ კოდში არის შეცდომები. გაასწორეთ იგი და წარმოადგინეთ შესწორებული კოდი და შესრულების შედეგი.

```

public delegate void showDelegate(string s);
class Program
{
    public static void Display(string)
    {
        MessageBox.Show(title);
    }
    public static void Show(string title)
    {
        MessageBox.Show(title);
    }
}
private void button1_Click(object sender, EventArgs e)
{
    showDelegate s = Display;

```

```

        s += Show;
15    s("გამარჯობა");
        s("ლუკა");
    }

```

განსაკუთრებული სიტუაციები. try, catch, throw და finally ოპერატორები. პროგრამის გამართვის საშუალებები. წყვეტის წერტილების შექმნა. ცვლადების მნიშვნელობათა ნახვა. პროგრამის შესრულება ბიჭობრივ რეჟიმში.

განსაკუთრებული სიტუაციები. try, catch, throw და finally ოპერატორები. შეცდომების დამუშავება Visual C# ენაში ხდება try/catch/throw/finally ბლოკის მიერ. სიტყვა try ითარგმნება როგორც „სცადე“, ხოლო catch - „დაიჭირე“. ამ ბლოკმა უნდა მოიცვას პროგრამული კოდის ის ნაწილი, სადაც შეიძლება წარმოიქმნას განსაკუთრებული სიტუაციები. გამართულად დაწერილი პროგრამის ბლოკის დასაწყისში ყოველთვის მოთავსებულია ოპერატორი try. ბლოკი catch-ის კოდი კი შეგვიძლია განვათავსოთ იქ, სადაც ჩვენ თვითონ გადავწყვეტთ, თუ როგორი ტიპის შეცდომები უნდა მოვძებნოთ. ბლოკი finally ყოველთვის სრულდება და შეიცავს

რესურსების განთავისუფლების ქვეპროგრამას განსაკუთრებული სიტუაციები-სათვის. განსაკუთრებული სიტუაციების ტიპებია:

- `Exception` - განსაკუთრებული სიტუაციების ძირითადი კლასი. ბლოკი `catch`, რომელშიც მითითებული იქნება `Exception` ტიპი „დაიჭერს“ ნებისმიერ განსაკუთრებულ სიტუაციას.
- `FormatException` - ოპერენდის ან არგუმენტის არაკორექტული ფორმატი. (მეთოდისათვის გადაცემის შემთხვევა).
- `NullReferenceException` - ობიექტის ეგზემპლარს არ აქვს მიმართვა თვითონ ობიექტზე, ობიექტი არ არის შექმნილი.
- `IndexOutOfRangeException` - მასივის ინდექსი დიაპაზონს გასცდა.
- `FileNotFoundException` - ფაილი არ არის ნაპოვნი.
- `DivideByZeroException` - ნულზე გაყოფა.

რისთვის არის საჭირო ბლოკი `finally`? ერთი შეხედვით, ბლოკის `try/catch`-ის გამოყენების შემდეგ ჩვენ შეგვიძლია გავათავისუფლოთ რესურსები `finally`-ის გამოყენების გარეშეც, მაგრამ `finally` ნებისმიერ შემთხვევაში ახდენს კოდის შესრულების გენერირებას ყველაფრის მიუხედავად. თუ ბლოკში `try/catch` მოხდება მეთოდიდან გამოსვლა `return` ოპერატორით, `finally` მაინც შესრულდება.

ზოგჯერ პროგრამაში წარმოიქმნება ისეთი სახის შეცდომები, რომელთა გამომწვევი მიზეზები მეტისმეტად პრიმიტიულია. შეცდომების ძებნა მთელს კოდში ძალიან რთულია და ზოგ შემთხვევაში თითქმის შეუძლებელი, რადგან გრძელი კოდის შემთხვევაში ეს მოითხოვს ძალიან დიდ დროს. C# ენას აქვს მექანიზმი, რომელიც გვუბნობს წინასწარ ვიზრუნოთ მეტისმეტად ცხადი შეცდომების თავიდან აცილებაზე. ასეთ შეცდომების მიეკუთვნება ნულზე გაყოფა, მასივის გადავსება, მასივის ინდექსის გასვლა დიაპაზონის გარეთ, არარსებული ფაილის გახსნის მცდელობა და ა. შ. ასეთი სახის შეცდომები იწვევს პროგრამის შესრულების ავარიულად დამთავრებას.

შეცდომა არის არასტანდარტული/განსაკუთრებული სიტუაცია, მისი დამუშავება კი არასტანდარტული/განსაკუთრებული სიტუაციის დამუშავებაა. C# ენაში არასტანდარტული სიტუაციები წარმოდგენილია `Exception` კლასით, რომელიც მიეკუთვნება `System.Exception` საბაზო კლასს.

C# ენაში განსაზღვრულია არასტანდარტული სიტუაციების ორი კატეგორია: გენერირებული C# ენის შესრულების გარემოს (CLR) მიერ და გენერირებული პროგრამა-დანართების (Application) მიერ. პირველი მათგანი აღწერილია System.Exception კლასში, მეორე კი - System.ApplicationException კლასში.

ხშირად წარმოქმნილი არასტანდარტული სიტუაციები (System.ApplicationException კლასი):

1. არასტანდარტული სიტუაცია ArrayTypeMismatchException - მნიშვნელობის ტიპი არ შეესაბამება მასივის ტიპს;
2. არასტანდარტული სიტუაცია DivideByZeroException - ნულზე გაყოფის შეცდომა;
3. არასტანდარტული სიტუაცია IndexOutOfRangeException - მასივის ინდექსი გადის დიაპაზონის გარეთ;
4. არასტანდარტული სიტუაცია InvalidCastException - არაკორექტული გარდაქმნა პროგრამის შესრულების პროცესში;
5. არასტანდარტული სიტუაცია OutOfMemoryException - new ოპერატორის გამოძახება იყო არაკორექტული მეხსიერების უკმარისობის გამო;
6. არასტანდარტული სიტუაცია OverflowException - გადავსება არითმეტიკული ოპერაციის შესრულების დროს;
7. არასტანდარტული სიტუაცია StackOverflowException - სტეკის გადავსება.

კლასის Exception-ის თვისებები (System.Exception კლასი):

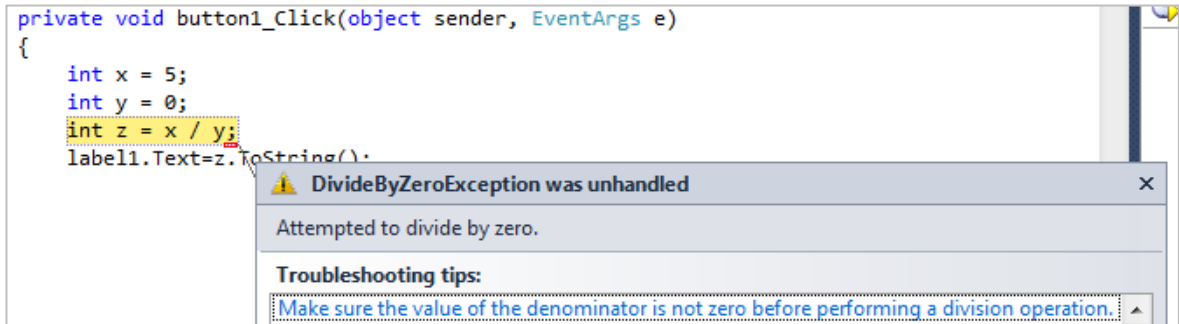
1. თვისება HelpLink - თვისებას HelpLink-ს შეიძლება მივანიჭოთ ფაილის სახელი ან ვებგვერდის მისამართი, რომელიც შეიცავს დამატებით ინფორმაციას არასტანდარტული სიტუაციის შესახებ (ტიპი string);
2. თვისება Message - შეიცავს არასტანდარტული სიტუაციის აღწერას (ტიპი string);
3. თვისება Source - შეიცავს არასტანდარტული სიტუაციის გამომწვევი პროგრამის სახელს (ტიპი string);
4. თვისება StackTrace - შეიცავს არასტანდარტული სიტუაციის გამომწვევი მეთოდისა და კლასის სახელს (ტიპი string);
5. თვისება TargetSite - შეიცავს იმ მეთოდის სახელს, საიდანაც მოხდა არასტანდარტული სიტუაციის გამოძახება MethodBase (ტიპი MethodBase).

მაგალითი 1. განვიხილოთ მარტივი მაგალითი. ქვემოთ მოცემულია კოდი:

```
int x = 5;
```

```
int y = 0;
int z = x / y;
label1.Text = z.ToString();
```

მოცემული კოდის შესრულებაზე გაშვების შემდეგ მივიღებთ შეტყობინებას შეცდომის შესახებ (ნახ. 51.):



ნახ. 51. განსაკუთრებული სიტუაციის გამოჭერა/დაჭერა

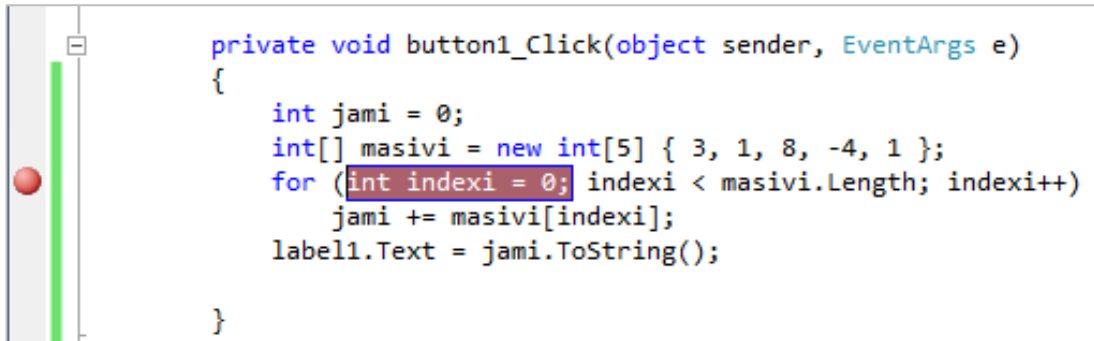
იმისათვის, რომ ასეთი სახის შეცდომები თავიდან ავიცილოთ, უნდა გამოვიყენოთ try-catch კონსტრუქცია:

```
private void button1_Click(object sender, EventArgs e) {
    try {
        int x = 5; int y = 0;
        int z = x / y;
        label1.Text = z.ToString();
    }
    catch {
        label1.Text = "შეცდომა!!! ნულზე გაყოფა არ შეიძლება!";
        label1.ForeColor = System.Drawing.Color.Red;
    }
}
```

გამართვა. .NET გარემოს შემადგენლობაში შედის პროგრამა გამმართველი. ის საშუალებას გვაძლევს, პროგრამის კოდი შევასრულოთ ბიჯობრივ რეჟიმში და ვნახოთ პროგრამის ცვლადების მნიშვნელობები. ეს, თავის მხრივ, აადვილებს პროგრამის კოდში შეცდომების პოვნას.

წყვეტის წერტილების შექმნა. წყვეტის წერტილი არის პროგრამული კოდის ის ადგილი (სტრიქონი), სადაც წყდება პროგრამის შესრულება. ამ სტრიქონში მოთავსებული გამოსახულება (ოპერატორი ან მეთოდი) არ შესრულდება და მართვა მომხმარებელს გადაეცემა.

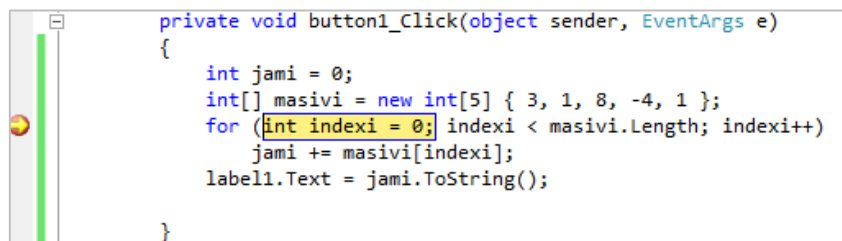
წყვეტის წერტილის შესაქმნელად საჭირო გამოსახულების გასწვრივ მარცხენა ველში მოვათავსოთ კურსორი და დავაჭიროთ თავის მარცხენა ღილაკს. გამოჩნდება მუქი წრე. წყვეტის წერტილის შესაქმნელად შეგვიძლია, აგრეთვე, კურსორი მოვათავსოთ საჭირო გამოსახულების შიგნით და დავაჭიროთ F9 ღილაკს. პროგრამის კოდში შეგვიძლია შევქმნათ წყვეტის რამდენიმე წერტილი (ნახ. 52.).



ნახ. 52. წყვეტის წერტილის შექმნა

წყვეტის წერტილის გასაუქმებლად უნდა დავაჭიროთ მუქ წრეს ან F9 ღილაკს. წყვეტის წერტილთან პროგრამული კოდის შესრულება წყდება. ამის შემდეგ, პროგრამის კოდი შეგვიძლია ბიჯობრივ რეჟიმში შევასრულოთ და დავაკვირდეთ ცვლადების მნიშვნელობებს. გამმართველის გაშვება ხდება F5 ღილაკით.

თუ გვინდა კოდის შესრულება გამმართველის გარეშე, მაშინ უნდა დავაჭიროთ Ctrl+F5 ღილაკებს. დავაჭიროთ F5 ღილაკს. გამოჩნდება ფორმა. შემდეგ დავაჭიროთ button ღილაკს. გამმართველი კოდს წყვეტის წერტილამდე ასრულებს და მართვას მომხმარებელს გადასცემს (იხ. ნახ. 53.).



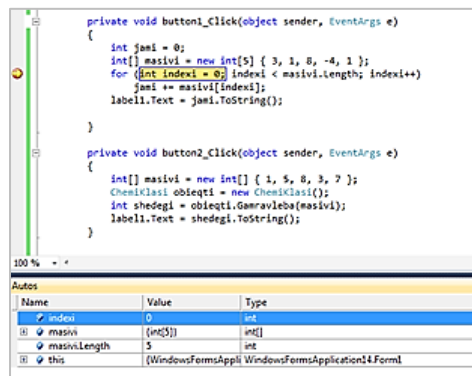
ნახ. 53. გამმართველის გაშვების დემონსტრირება

შესაბამისი გამოსახულების (int indexi = 0;) გასწვრივ მარცხენა ველში გამოჩნდება ისარი. ეს გამოსახულება ჯერ არ არის შესრულებული. ცვლადების მნიშვნელობების ნახვა როგორც აღვნიშნეთ, წყვეტის წერტილთან გამმართველი წყვეტს კოდის შესრულებას. ამ დროს შეგვიძლია ვნახოთ ცვლადების მნიშვნელობები.

ფანჯარა Autos. ეს ფანჯარა მოთავსებულია გამმართველის ფანჯრის ქვემოთ. თუ ის არ ჩანს, მაშინ ვხსნით Debug მენიუს, Windows ქვემენიუს და ვასრულებთ Autos ბრძანებას Debug -> Windows -> Autos.

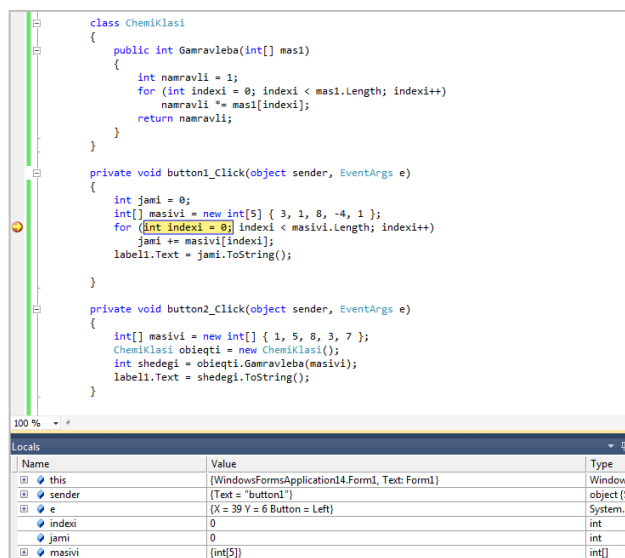
Windows ქვემენიუს ბრძანებები გამოჩნდება წყვეტის წერტილთან პროგრამის შეჩერების შემდეგ. ამ ფანჯარაში ჩანს იმ ცვლადების მნიშვნელობები, რომლებიც გამოიყენება მიმდინარე და წინა გამოსახულებებში.

აგრეთვე, ცვლადის მნიშვნელობა შეგვიძლია ვნახოთ, თუ პროგრამის კოდში მის სახელზე გავაჩერებთ კურსორს. თუ Autos ფანჯარაში დავაჭერთ masivi სახელის მარცხნივ მოთავსებულ + ნიშანზე, მაშინ გამოჩნდება ამ მასივის ელემენტების მნიშვნელობები (ნახ. 54).



ნახ. 54. ფანჯარა Autos

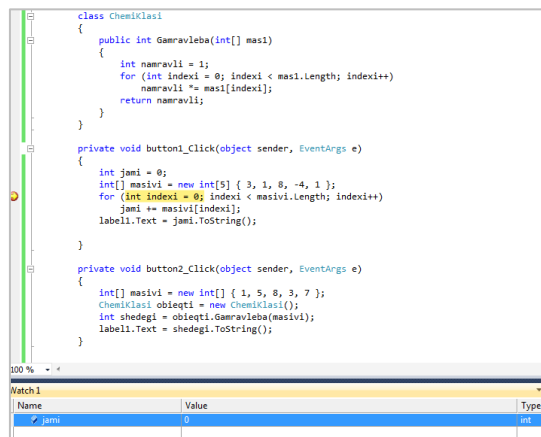
ფანჯარა Locals. ეს ფანჯარა მოთავსებულია გამმართველის ფანჯრის ქვემოთ. მის გასახსნელად უნდა დავაჭიროთ Locals ჩანართს. თუ ეს ფანჯარა არ ჩანს, მაშინ Debug -> Windows -> Locals. მასში ჩანს იმ ცვლადების მნიშვნელობები, რომლებიც იმყოფებიან ხილვადობის მოცემულ უბანში (ნახ. 55.).



ნახ. 55. ფანჯარა Locals

ფანჯარა Watch. ეს ფანჯარა მოთავსებულია გამმართველის ფანჯრის ქვემოთ. მის გასახსნელად უნდა დავაჭიროთ Watch ჩანართს. თუ ეს ფანჯარა არ ჩანს, მაშინ Debug → Windows → Watch → Watch1. მასში ჩანს სპეციალურად მითითებული ცვლადების მნიშვნელობები. ეს საჭიროა მაშინ, როცა ცვლადების დიდი რაოდენობიდან გვაინტერესებს მხოლოდ რამდენიმე.

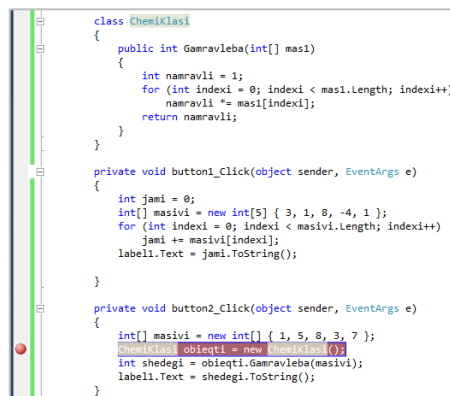
Watch ფანჯარაში ცვლადის დასამატებლად პროგრამის კოდში ცვლადის სახელზე უნდა დავაჭიროთ თავისი მარჯვენა ღილაკს და კონტექსტური მენიუდან შევასრულოთ Add Watch ბრძანება (ნახ. 56.).



ნახ. 56. ფანჯარა Watch

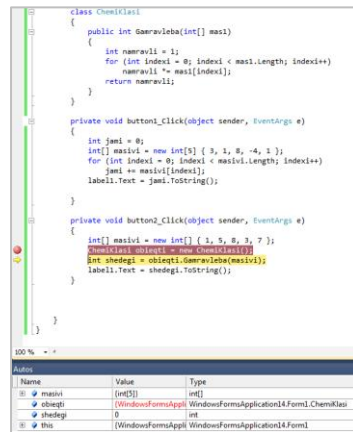
პროგრამის კოდის შესრულება ბიჯობრივ რეჟიმში. პროგრამის გამართვის დროს შეგვიძლია ცვლადების მნიშვნელობების ნახვა და ოპერატორების ბიჯობრივ რეჟიმში შესრულება. მიმდინარე გამოსახულების შესასრულებლად შეგვიძლია დავაჭიროთ F10 ღილაკს ან შევასრულოთ Debug -> Step Over ბრძანება. თუ გვინდა მეთოდის კოდის ბიჯობრივ რეჟიმში შესრულება, მაშინ უნდა დავაჭიროთ F11 ღილაკს ან შევასრულოთ Debug -> Step Into ბრძანება. მეთოდის კოდიდან გამოსასვლელად ვასრულებთ Debug->Step Out ბრძანებას ან ვაჭერთ Shift+F11 ღილაკებს. მეთოდიდან გამოსვლისას მართვა გადაეცემა ამ მეთოდის გამომძახებელ გამოსახულებას.

სტრუქტურაში, რომელშიც ხდება მეთოდის Gamravleba() გამოიძახება შევექმნათ წყვეტის წერტილი (იხ. ნახ. 57.).



ნახ. 57. წყვეტის წერტილის შექმნა მეთოდის გამოძახებისას
დავაჭიროთ F5 ღილაკს. გაიხსნება ფორმა. დავაჭიროთ მეორე button ღილაკს.
პროგრამის შესრულება შეწყდება წყვეტის წერტილში.

ამის შემდეგ, თუ გვინდა მეთოდის Gamravleba() შესრულება ბიჯობრივ რეჟიმში, უნდა დავაჭიროთ F11 ღილაკს. ამ ღილაკზე ყოველი დაჭერის შემდეგ შეგვიძლია ვნახოთ ცვლადების მნიშვნელობები (იხ. ნახ. 58.).



ნახ. 58. მეთოდის შესრულება ბიჯობრივ რეჟიმში

ახლა კი გადავიდეთ განსაკუთრებული სიტუაციების დემონსტრირების მაგალითების განხილვაზე.

მაგალითი 1. ქვემოთ მოცემული კოდი გვაძლევს საშუალებას შევიტანოთ რიცხვითი მნიშვნელობა. თუ შევიტანთ რიცხვისაგან განსხვავებულ სიმბოლოს, გამოვა გამაფრთხილებელი შეტყობინება (იხ. ნახ. 59).

`private void button1_Click(object sender, EventArgs e)`

```
{
    string result = "";
    try
    {
        int a = int.Parse(textBox1.Text);
        result = "თქვენ შეიყვანეთ რიცხვი " + a;
    }
    catch (FormatException)
    {

```

```
result="წარმოიქმნა განსაკუთრებული სიტუაცია!!!\n შეიყვანეთ რიცხვითი მნიშვნელობა!";
```

```
    label2.ForeColor = System.Drawing.Color.Red;
```

```
    }
```

```
    label2.Text = result.ToString();
```

```
    }
```

```
private void Form1_Load(object sender, EventArgs e) {
```

```
    label1.Text = "შეიტანეთ რიცხვითი მნიშვნელობა";
```

```
}
```

ნახ. 59. მაგალითი 1-ის შესრულების შედეგი

მაგალითი 2. შევადგინოთ პროგრამა, რომელიც გამოთვლის სფეროს ფართობს. სფეროს გამოსათვლელი ფორმულაა: $S = 4\pi R^2$

```
private void button1_Click(object sender, EventArgs e) {
    const double Pi = 3.1415;
    double R, S;
    R = Double.Parse(textBox1.Text);
    S = 4 * Pi * R * R;
    label1.Text = "სფეროს ფართობი = " + S.ToString();
}
```

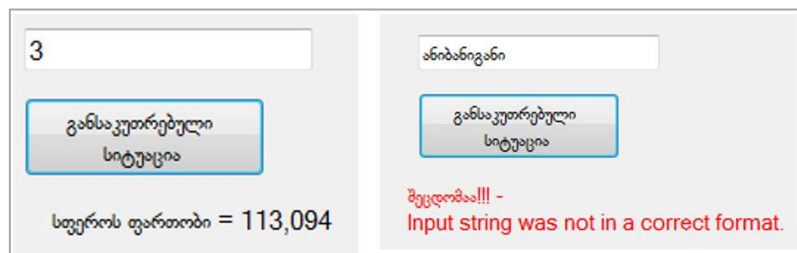
ზემოთ მოცემულ კოდში არ არსებობს დაცვა რადიუსის არაკორექტულ შეტანასთან დაკავშირებით, ამიტომ button1_Click მეთოდი შეგვიძლია ასე დავწეროთ (იხ. ნახ. 60.):

```
private void button1_Click(object sender, EventArgs e) {
    try {
        const double Pi = 3.1415;
```

```

        double R, S;
        R = Double.Parse(textBox1.Text);
        S = 4 * Pi * R * R;
        label1.Text = "სფეროს ფართობი = " + S.ToString(); }
    catch (FormatException ex) {
        label1.Text = "შეცდომა!!! - " + '\n' + ex.Message;
        label1.ForeColor = System.Drawing.Color.Red;
    }
}

```



ნახ. 60. მაგალითი 2-ის რულების შედეგი

მაგალითი 3. განსაკუთრებული სიტუაციის დამუშავება, რომელიც წარმოიქმნება ნულზე გაყოფის დროს (იხ. ნახ. 61.):

```

class Program {
    public decimal MyDel(decimal x, decimal y)
    {
        return (x/y);
    }
}

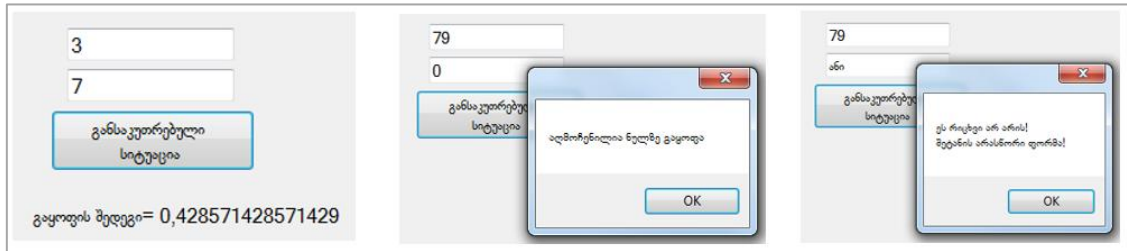
private void button1_Click(object sender, EventArgs e) {
    Program objeqti = new Program();
    decimal result;
    try
    {
        decimal x1 = decimal.Parse(textBox1.Text);
        decimal y1 = decimal.Parse(textBox2.Text);
        result = objeqti.MyDel(x1, y1);
        label1.Text = "გაყოფის შედეგი = " + result.ToString();
    }
    catch (DivideByZeroException) {

```

```

        MessageBox.Show("აღმოჩენილია ნულზე გაყოფა ");
    }
    catch (FormatException) {
        MessageBox.Show("ეს რიცხვი არ არის!\nშეტანის არასწორი ფორმა!");
    }
}

```



ნახ. 61. მაგალითი 3-ის შესრულების შედეგი

მაგალითი 4. მე-3 მაგალითში მოცემული კოდი გადავაკეთოთ ისეთი სახით, სადაც მოხდება განსაკუთრებული მნიშვნელობის გენერირება, იმ შემთხვევაში, თუ მონაცემები არ გვაქვს textBox1 და textBox2 კომპონენტებში (იხ. ნახ. 62.):

```

class Program
{
    public decimal MyDel(decimal x, decimal y)
    {
        return (x/y);
    }
}

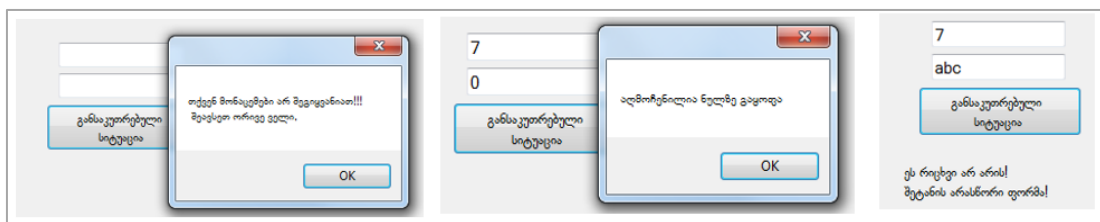
private void button1_Click(object sender, EventArgs e)
{
    Program objeqti = new Program();
    decimal result;
    try
    {
        if ((textBox1.Text == "") || (textBox2.Text == ""))
            MessageBox.Show("თქვენ მონაცემები არ შეგიყვანიათ!!!\n შეავსეთ ორივე  
ველი.");
        else
        {
            decimal x1 = decimal.Parse(textBox1.Text);

```

```

        decimal y1 = decimal.Parse(textBox2.Text);
        result = obieqti.MyDel(x1, y1);
        label1.Text = "გაყოფის შედეგი= " + result.ToString();
    }
}
catch (DivideByZeroException)
{
    MessageBox.Show("აღმოჩენილია ნულზე გაყოფა ");
}
catch (FormatException)
{
    label1.Text = "ეს რიცხვი არ არის!\nშეტანის არასწორი  
ფორმა!";
}
}
}

```



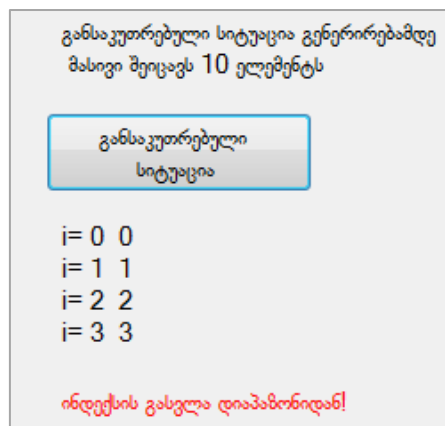
ნახ. 62. მაგალითი 4-ის შესრულების შედეგი

ჩადგმული try ბლოკები. შესაძლებელია try ბლოკების ერთმანეთში მოთავსება. შიგა try ბლოკში გენერირებული განსაკუთრებული სიტუაცია, რომელიც ვერ დაიჭირა ამ შიგა try ბლოკის შესაბამისმა catch ბლოკმა, ვრცელდება გარე try ბლოკზე. მოყვანილ მაგალითში გენერირებული IndexOutOfRangeException განსაკუთრებული სიტუაცია, რომელიც არ იყო დაჭერილი შიგა catch ბლოკის მიერ, დაჭერილი იქნება გარე catch ბლოკის მიერ.

ხშირად ჩადგმული try ბლოკები გამოიყენება სხვადასხვა სახის შეცდომის დასამუშავებლად. გარე try ბლოკი შეგვიძლია გამოვიყენოთ სერიოზული შეცდომების დასაჭერად. ხოლო შიგა try ბლოკები ნაკლებად სერიოზული შეცდომების დასამუშავებლად. შეგვიძლია, აგრეთვე, გამოვიყენოთ გარე try ბლოკი უპარამეტრებო catch ოპერატორით იმ შეცდომების დასაჭერად, რომლებიც არ მუშავდებიან შიდა try ბლოკში.

მაგალითი 5. განსაკუთრებული სიტუაციების გენერირება (იხ. ნახ. 63.)

```
private void button1_Click(object sender, EventArgs e) {  
    label2.Text = "";  
    int[] nums = new int[4];  
    try  
    { // განსაკუთრებული სიტუაციების გენერირება. ინდექსის დიაპაზონიდან გასვლა.  
        for (int i = 0; i < 10; i++)  
        {  
            nums[i] = i;  
            label2.Text += "i= " + i + " " + nums[i] + "\n";  
        }  
    }  
    catch (IndexOutOfRangeException)  
    {  
        // განსაკუთრებული სიტუაციის დაჭერა  
        label3.Text = ("ინდექსის გასვლა დიაპაზონიდან!");  
        label3.ForeColor = System.Drawing.Color.Red;  
    }  
}  
  
private void Form1_Load(object sender, EventArgs e){  
    label1.Text=("განსაკუთრებული სიტუაცია გენერირებამდე\i მასივი შეიცავს10 ელემენტს");  
}
```



ნახ. 63. მე-5 მაგალითის შესრულების შედეგი

ახლა კი გადავიდეთ ამოცანების განხილვაზე.

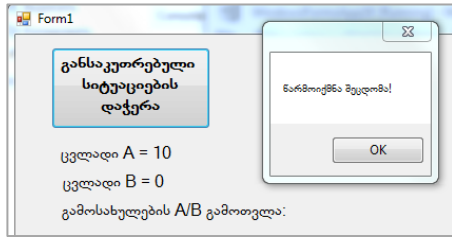
ამოცანა 15.1. პროგრამულ კოდში განსაკუთრებული სიტუაციების აღმოჩენა (იხ. ნახ. 64. ა.ბ.გ.დ.ე.ვ.ზ.).


```

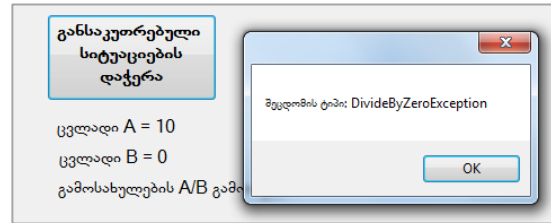
private void button1_Click(object sender, EventArgs e) {
    label1.Text = "პროგრამის შესრულების დაწყება";
    // ცვლადების გამოცხადება
    int A = 10, B = 0;
    label2.Text = "ცვლადი A = " + A;
    label3.Text = "ცვლადი B = " + B;
    //კოდზე კონტროლის დაწესება
    try {
        label4.Text = "გამოსახულების A/B გამოთვლა: ";
        //გაყოფის მცდელობა
        label5.Text = "";
        label5.Text = "შედეგი: " + A/B;
    }
    // განსაკუთრებული სიტუაციების დამუშავება.
    catch (Exception ex)
    {
        // შეტყობინების გამოტანა
        MessageBox.Show("წარმოიქმნა შეცდომა!");
        // შეცდომის ტიპი:
        MessageBox.Show("შეცდომის ტიპი: " + ex.GetType().Name);
        // შეცდომის აღწერა:
        MessageBox.Show("აღწერა: " + ex.Message);
        // შეცდომის წყარო (პროგრამა):
        MessageBox.Show("პროგრამა: " + ex.Source);
        // მეთოდი, სადაც წარმოიშვა შეცდომა:
        MessageBox.Show("მეთოდი: " + ex.TargetSite.Name);
        // შეცდომის სტრიქონი:
        MessageBox.Show("შეცდომის სტრიქონი: " + ex.StackTrace);
    }
    MessageBox.Show("პროგრამამ დაასრულა მუშაობა");
    //შეტყობინება
    Close();
}

```

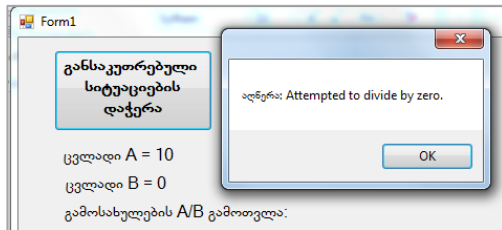
}



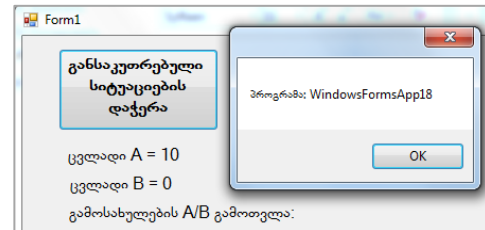
ა)



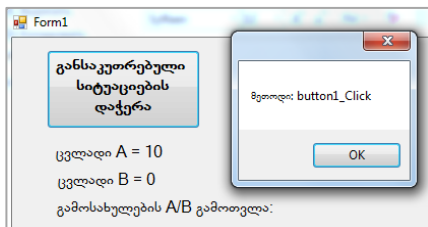
ბ)



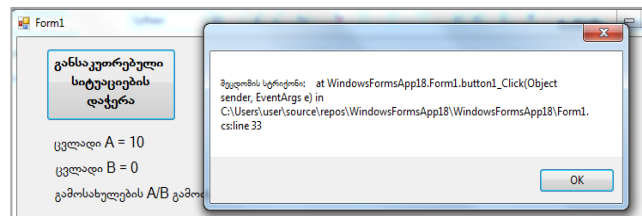
გ)



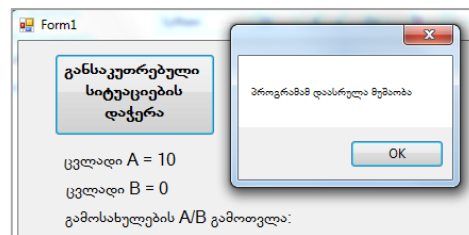
დ)



ე)



ვ)



ზ)

ნახ. 64. ამოცანა 15.1-ის: ა) - ზ) შესრულების შედეგები

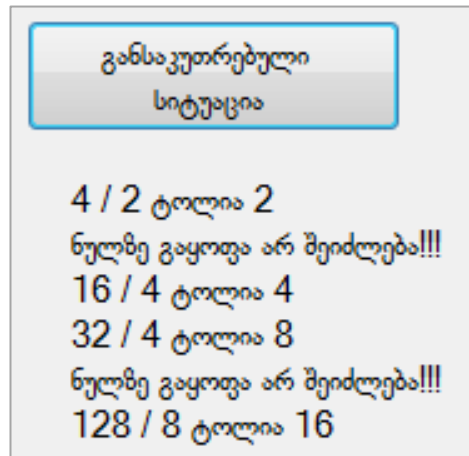
ამოცანა 15.2. განსაკუთრებული სიტუაციების დამუშავების ერთ-ერთი უპირატესობა არის ის, რომ ის საშუალებას გაძლევთ, პროგრამულ კოდში აღწეროთ სავარაუდო შეცდომა და შემდეგ განაგრძოთ კოდის შესრულება. მაგალითად, ნულზე გაყოფის მცდელობა არ გამოიწვევს პროგრამის შესრულებისას პროგრამიდან გასვლას. შეცდომა დამუშავდება გამოწვევის - DivideByZeroException გენერირებით პროგრამის შესრულების შეწყვეტის გარეშე (იხ. ნახ. 65.).

```
private void button1_Click(object sender, EventArgs e) {
    label1.Text = "";
    int[] numer = { 4, 8, 16, 32, 64, 128 };
    int[] denom = { 2, 0, 4, 4, 0, 8 };
}
```

```

for (int i = 0; i < numer.Length; i++)
{
    try {
label1.Text += numer[i] + "/" + denom[i] + " ტოლია " +
                                                    numer[i] / denom[i] + "\n";
    }
    catch (DivideByZeroException) { // შეცდომის დაჭერა:
        label1.Text += "ნულზე გაყოფა არ შეიძლება!!!" + "\n";
    }
}
}

```



ნახ. 65. ამოცანა 15.2.-ის შესრულების შედეგი

ამოცანა 15.3. განსაკუთრებული სიტუაციების დამუშავებისას შეგვიძლია გამოვიყენოთ შეცდომის „უნივერსალური“ - `FormatException` დაჭერა. მნიშვნელოვანია ვიცოდეთ, რომ „უნივერსალური“ შეცდომის დაჭერა უნდა განვათავსოთ `catch` ბლოკის ბოლოს.

```

private void button1_Click(object sender, EventArgs e){
    Random rnd = new Random();
    int[] nums;
    int x;
    for (int k = 1; k < 10; k++)
    {
label1.Text = "მასივის ელემენტების რაოდენობა - " + k.ToString();
        try
        {

```

```

        nums = new int[2 * rnd.Next(3) - 1];
        x = 1 / rnd.Next(3);
        nums[rnd.Next(2) - 1] = x;
        nums[0] = int.Parse("ნოლი");
    }
    catch (OverflowException)
    {
        MessageBox.Show("შეცდომა №1: მასივის ზომა გასულია ");
    }
    catch (DivideByZeroException)
    {
        MessageBox.Show("შეცდომა №2: ნულზე გაყოფა");
    }
    catch (IndexOutOfRangeException)
    {
        MessageBox.Show("შეცდომა №3: ელემენტის ინდექსი არ არის სწორი");
    }
    catch (FormatException)
    {
        MessageBox.Show("შეცდომა №4: რიცხვის არასწორი ტიპი");
    }
    }
    MessageBox.Show("პროგრამამ დაასრულა მუშაობა");
    Close();
}

```

ამოცანა 15.4. მოცემულია ერთგანზომილებიანი მასივი. დავწეროთ პროგრამული კოდი, სადაც მოხდება მასივის ელემენტების რაოდენობისა და მნიშვნელობების შეტანა textBox კომპონენტიდან. რაოდენობის და მნიშვნელობების შეტანა დავაფიქსიროთ Enter ღილაკზე დაჭერით. შედეგი ავსახოთ შესაბამის label კომპონენტებში. ამასთან აუცილებლად გამოვიყენოთ მონაცემების შეტანის დროს შეცდომის დაჭერის სავარაუდო ვარიანტები. ფორმის დიზაინი ავსაგოთ ქვემოთ ნაჩვენები სქემის სახით (იხ. ნახ. 66.) ან ააწყვეთ ფორმა საკუთარი შეხედულებისამებრ.

ფორმაზე განვალაგოთ ორი textBox, ოთხი label და ორი button კომპონენტები.

კომპონენტში textBox1 შევიტანოთ მასივის ელემენტების რაოდენობა.

კომპონენტში textBox2 შევიტანოთ მასივის ელემენტების მნიშვნელობები.

მოცემული ამოცანის გადასაწყვეტად პროგრამული კოდი განვთავსოთ ორივე textBox და ერთ button კომპონენტზე. განვიხილოთ თითოეული მათგანი ცალ-ცალკე.

თავდაპირველად გამოვაცხადოთ გლობალური ცვლადები:

```
public int size;  
int i = 0;  
int[] mas;
```

button1 კომპონენტის Click მოვლენაზე დავწეროთ ქვემოთ მოცემული კოდის ფრაგმენტი.

```
private void button1_Click(object sender, EventArgs e) {  
    i = 0;  
    Array.Clear(mas, 0, mas.Length);  
    label3.Text = ""; label4.Text = "";  
}
```

Button2 კომპონენტის Click მოვლენაზე დავწეროთ ქვემოთ მოცემული კოდის ფრაგმენტი.

```
private void button2_Click(object sender, EventArgs e){  
    MessageBox.Show("პროგრამამ დაასრულა მუშაობა"); //შეტყობინება.  
    Close();  
}
```

პროგრამული კოდის ერთი ფრაგმენტი დავწეროთ textBox1 კომპონენტის KeyPress მოვლენაზე.

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e){  
    try  
    {  
        if (e.KeyChar == 13)  
        {  
            size = int.Parse(textBox1.Text);  
            label3.Text = ""; label4.Text = "";  
            label3.Text = "მასივი შეიცავს " + size + " ელემენტს";  
        }  
    }  
}
```

```

        catch
        {
            MessageBox.Show("შეცდომა!!! შეყვანეთ თავიდან და დააჭირეთ ENTER-ს !!!");
        }
    }

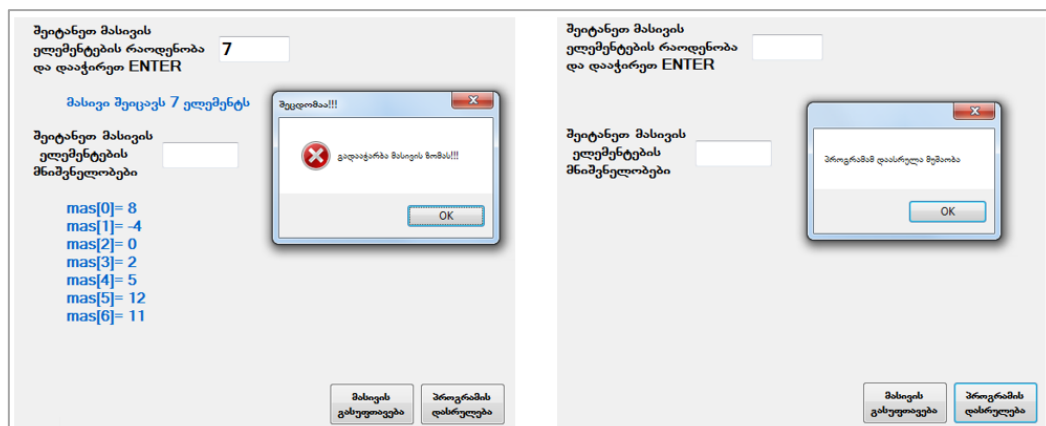
```

პროგრამული კოდის მეორე ფრაგმენტი დავწეროთ textBox2 კომპონენტის KeyPress მოვლენაზე.

```

private void textBox2_KeyPress(object sender, KeyPressEventArgs e)
{
    try
    {
        mas = new int[size];
        if (e.KeyChar == 13)
        {
            mas[i] = int.Parse(textBox2.Text);
            label4.Text += "mas[" + i.ToString() + "]= " + " " + mas[i].ToString() + '\n';
            ++i;
            textBox2.Text = "";
        }
    }
    catch (Exception)
    {
        textBox2.Text = "";
        MessageBox.Show("გადაჭარბა მასივის ზომას!!!", "შეცდომა!!!",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```



ამოცანა 15.5. მოყვანილ მაგალითში ხდება DivideByZeroException კლასის nulze_gayofa ობიექტის შექმნა და მისი მითითება throw ოპერატორის შემდეგ.

```
int ricxvi1, ricxvi2, shedegi;
DivideByZeroException nulze_gayofa = new DivideByZeroException();
nulze_gayofa.HelpLink = "დამატებითი ინფორმაცია იხ. Readme.txt ფაილში";
nulze_gayofa.Source = "შეცდომის გამომწვევია - პროგრამა 10.12";
try
{ // ეს ოპერატორები შესრულდება throw ოპერატორის წინ.
    ricxvi1 = Convert.ToInt32(textBox1.Text);
    ricxvi2 = Convert.ToInt32(textBox2.Text);
    // განსაკუთრებული სიტუაციის გენერირება.
    if (ricxvi2 == 0) throw nulze_gayofa;
    shedegi = ricxvi1/ricxvi2;
}
catch (DivideByZeroException arg)
{ label1.Text = "განსაკუთრებული სიტუაცია დაჭერილია";
  label2.Text = arg.HelpLink; label3.Text = arg.Source; }
```

ამოცანა 15.6. finally ბლოკი გამოიძახება იმისგან დამოუკიდებლად, გენერირდება თუ არა განსაკუთრებული სიტუაცია. ამრიგად, არა აქვს მნიშვნელობა try ბლოკი როგორ დამთავრდება - ნორმალურად თუ აღიძვრება განსაკუთრებული სიტუაცია. finally ბლოკის კოდი ყოველთვის შესრულდება. მოყვანილ პროგრამაში ხდება finally ბლოკის გამოყენების დემონსტრირება.

```
class finally1 {
    public static void gen_finally(int par1)
    {
        int ricxvi;
        int[] masivi = new int[5];
        MessageBox.Show(par1.ToString() + "-ის მიღება");
        try
        {
            switch (par1)
            {
```

```

        case 0: ricxvi = 10 / par1; // ნულზე გაყოფის შეცდომა
            break;
        case 1: masivi[7] = 7; // დიაპაზონის გარეთ ინდექსის გასვლის შეცდომა
            break;
        case 2: return; // try ბლოკიდან გამოსვლა
    }
}
catch (DivideByZeroException)
{
    MessageBox.Show("ნულზე გაყოფის შეცდომა!");
    return;
}
catch (IndexOutOfRangeException)
{
    MessageBox.Show(" დიაპაზონის გარეთ ინდექსის გასვლის შეცდომა!");
    return;
}
finally
{
    MessageBox.Show("try ბლოკიდან გამოსვლა");
}
}
private void button1_Click(object sender, EventArgs e)
{
    for (int indeqsi = 0; indeqsi < 3; indeqsi++)
    {
        finally1.gen_finally(indeqsi);
    }
}

```

ამოცანა 15.7. პროგრამაში ხდება საკუთარი განსაკუთრებული სიტუაციის შექმნის დემონსტრირება

```

public class Gansakutrebuli_Situacia : ApplicationException {
    public Gansakutrebuli_Situacia(string Message): base(Message)

```



```

{
    this.HelpLink = "დამატებითი ინფორმაცია იხ. Readme.txt ფაილში";
    this.Source = "შეცდომის გამომწვევია - პროგრამა 10.14";
}
}

private void button1_Click(object sender, EventArgs e) {
try
{
throw new Gansakutrebuli_Situacia("ჩემი შეტყობინება
Gansakutrebuli_Situacia კლასში");
}
catch (Gansakutrebuli_Situacia arg)
{
    label1.Text = arg.HelpLink;
    label2.Text = arg.Message;
    label3.Text = arg.Source;
    label4.Text = arg.StackTrace;
    label5.Text = arg.TargetSite.ToString();
}
}
}

```

დამოუკიდებელი სამუშაო

1. ქვემოთ მოცემულია პროგრამული კოდის შესრულების შედეგები. დამოუკიდებლად დაწერეთ თითოეული შემთხვევისთვის პროგრამული კოდი, რომლიც შესაბამისობაში მოვა ქვემოთ ნაჩვენებ შესრულების შედეგებთან (იხ. ნახ. 67. ა, ბ, გ, დ, ე) ანუ ვხედავთ ფორმას (ვიზუალურ მხარეს) და ვწერთ კოდს ☺. წარმატებები...

პროგრამული შეცდომების აღმოფხვრა

X =

Y =

გაყოფა

Z=X/Y label5

ა)

პროგრამული შეცდომების აღმოფხვრა

X = 125

Y = 5

გაყოფა

გაყოფა შეცდომის გამორიცხვით

Z=X/Y 5

ბ)

პროგრამული შეცდომების აღმოფხვრა

X = 125

Y = 0

გაყოფა

გაყოფა შეცდომის გამორიცხვით

Z=X/Y შეცდომა: ნულზე გაყოფა არ შეიძლება

გ)

პროგრამული შეცდომების აღმოფხვრა

X = 125

Y = qw

გაყოფა

გაყოფა შეცდომის გამორიცხვით

Z=X/Y შეცდომა: შეიტანეთ რიცხვითი მნიშვნელობა

დ)

პროგრამული შეცდომების აღმოფხვრა

X =

Y =

გაყოფა

გაყოფა შეცდომის გამორიცხვით

Z=X/Y შეცდომა: მნიშვნელობა არ არის შეტანილი არც ერთ ველში

ე)

ნახ. 67. დამოუკიდებელი სამუშაოს: ა) ბ) გ) დ) ე) შესრულების შედეგები

16

თარიღთან და დროსთან სამუშაო მეთოდები. სტრიქონებთან სამუშაო მეთოდები. სტრიქონების მასივები. გეომეტრიული ფიგურების ხატვა, გამოსახულებასთან მუშაობა

თარიღთან და დროსთან მუშაობა. თარიღთან და დროსთან სამუშაოდ C# ენაში არის სტრუქტურა: `DateTime`. იგი წარმოგვიდგენს თარიღს და დროს შემდეგ დიაპაზონში: 0001 წლის 1 იანვრის 00:00:00-დან ÷ 9999 წლის 31 დეკემბრის 23:59:59-მდე.

`DateTime`-ის ახალი ობიექტის შექმნის დროს შეგვიძლია გამოვიყენოთ კონსტრუქტორი. ცარიელი კონსტრუქტორი ქმნის საწყის თარიღს. შევქმნათ `datetime` სტრუქტურის ახალი ობიექტი:

```
label1.Text = "";  
DateTime dt = new DateTime();  
label1.Text = dt.ToString() + '\n';
```

შესრულების შედეგი: 0001-01-01 0:00:00

მიღებული შედეგი საკმაოდ მარტივია. იგი შეესაბამება `DateTime.MinValue`-ს მნიშვნელობას.

სტრუქტურა `DateTime` შეიცავს ბევრ სასარგებლო თვისებას, მათ შორის ყველაზე საინტერესო არის `DateTime.Now` თვისება:

```
label1.Text = "";  
DateTime dt = DateTime.Now;  
label1.Text = dt.ToString() + '\n';
```

შესრულების შედეგი: 2023-09-11 14:01:07

პროგრამამ გასცა მიმდინარე თარიღი და დრო, მაგრამ უფრო ხშირად დაგვჭირდება წარმოვადგინოთ კონკრეტული დრო და თარიღი. `DateTime`-ს აქვს ზოგიერთი კონსტრუქტორი, რომელიც დაგვცხმარება ამ ამოცანის გადაწყვეტაში. იმისათვის, რომ დავაყენოთ კონკრეტული თარიღი, საჭიროა გამოვიყენოთ კონსტრუქტორებიდან ერთ-ერთი ვარიანტი, რომელიც იღებს პარამეტრებს:

```
label1.Text = "";
DateTime dt = new DateTime(2092, 12, 24); // წელი-თვე-დღე
label1.Text = dt.ToString() + '\n';
```

შესრულების შედეგი: 2092-12-24 0:00:00

კიდევ ერთი ვარიანტი, სადაც სრულად ხდება თარიღისა და დროის ასახვა:

```
label1.Text = "";
//წელი - თვე - დღე -საათი - წუთი - წამი
DateTime dt = new DateTime(2092, 12, 24, 18, 30, 25);
label1.Text = dt.ToString() + '\n';
```

შესრულების შედეგი: 2092-12-24 18:30:25

რა ვქნათ, თუ გვინდა წარმოვადგინოთ თარიღის მხოლოდ ნაწილი - კარგი იქნებოდა, თუ გამოვიყენებდით მხოლოდ Date კლასს DateTime-ის ნაცვლად, მაგრამ კლასი Date განყენებულად არ არსებობს. ასეთ შემთხვევაში გამოსავალი არის კლასის DateTime-ის თვისება Date:

```
DateTime dt = new DateTime(2042, 12, 24, 18, 42, 0);
DateTime date = dt.Date;
label1.Text = date.ToString() + '\n';
```

შესრულების შედეგი: 2092-12-24 0:00:00

პროგრამული კოდიდან ვხედავთ, რომ თვისება Date ასევე არის DateTime-ის ობიექტი, სადაც თარიღის ნაწილი წარმოდგენილია როგორც თარიღი და დრო (0:00:00). ეს შედეგი ცოტა დამაბნეველია, მაგრამ გამოსავალი არის ის, რომ DateTime შეგვიძლია წარმოვიდგინოთ როგორც კონტეინერი იმ მონაცემებით, რომელიც შეგვიძლია მივაწოდოთ მომხმარებელს სხვადასხვა თვალსაზრისით და ფორმატით, ჩვენ გამოვიყენეთ სტანდარტული მეთოდი ToString(), თუმცა არსებობს ბევრი ხერხი, რათა მივიღოთ DateTime ობიექტის სხვადასხვა სახის მონაცემი. თუ საჭიროა მიმდინარე დროისა და თარიღის მიღება, შეგვიძლია გამოვიყენოთ DateTime-ის ზოგიერთი თვისება:

```
label1.Text = DateTime.Now.ToString() + '\n';
label1.Text += DateTime.UtcNow.ToString() + '\n';
label1.Text += DateTime.Today.ToString() + '\n';
```

შესრულების შედეგი: 2023-09-11 14:27:04

2023-09-11 10:27:04

DateTime.Now თვისება გვამღევს კომპიუტერის მიმდინარე თარიღსა და დროს, DateTime.UtcNow - გვამღევს თარიღსა და დროს გრინვიჩის დროსთან (GMT) მიმართებით, ხოლო DateTime.Today - მხოლოდ მიმდინარე თარიღია. თარიღებთან მუშაობისას გაითვალისწინეთ, რომ ჩუმათობის პრინციპის მიხედვით, თარიღების დასადგენად გამოიყენება გრიგორიანული კალენდარი.

გრინვიჩი (ინგლ. Greenwich) - არის ქალაქი ლონდონის საზღვაო ჭიშკარი. ლონდონის ისტორიული გარეუბანი, ამჟამად ადგილობრივი თვითმმართველობის ერთ-ერთი ერთეული ქალაქის სამხრეთ-აღმოსავლეთით, მდინარე ტემზის მარჯვენა სანაპიროზე. გრინვიჩი ითვლება ლონდონის ერთ-ერთ პრესტიჟულ რაიონად. გრინვიჩი ცნობილია იმით, რომ აქ გადის ნულოვანი მერიდიანი, რომელიც ისტორიულად უკავშირდება გრინვიჩის ობსერვატორიას.

დედამიწაზე, სადაც არ უნდა ვცხოვრობდეთ, შუადღე ყოველთვის დღის 12:00-ზე დგება; ეს ის დროა, როდესაც ჩვენი მზე ცამრგვალზე ყველაზე მაღალ წერტილში - ზედა კულმინაციაში იმყოფება. თუმცა, პლანეტა დედამიწა იმდენად დიდია, რომ როდესაც თბილისში 12:00 საათია, იმავდროულად ტოკიოში 17:00 საათი დგება, ხოლო სინგაპურში 22:00. როდესაც ასტრონომები ციური მოვლენების დროს განსაზღვრავენ, ისინი სარგებლობენ გრინვიჩის საშუალო დროით Greenwich Mean Time (GMT) ანუ იმ დროით, რომელსაც უჩვენებს ინგლისში, დედამიწის ნულოვან მერიდიანზე განთავსებული ქალაქის - გრინვიჩის საათი.

თუ გვინდა გრინვიჩის საშუალო დრო - GMT ჩვენს დროს შევუსაბამოთ, მაშინ მას საათები უნდა მივუმატოთ ან გამოვაკლოთ. ზემოთ მოყვანილი ტაბულა გვიჩვენებს, თუ რამდენი საათი უნდა მივუმატოთ გრინვიჩის დროს, თუ ნულოვანი მერიდიანისაგან აღმოსავლეთით გადავინაცვლებთ; და რამდენი საათი უნდა გამოვაკლოთ, თუ ამ მერიდიანიდან დასავლეთით გავემგზავრებით. მაგალითად: ნიუ-იორკი გრინვიჩის მერიდიანიდან დასავლეთით განთავსდება, ამიტომ მისი ადგილობრივი დრო გრინვიჩის დროზე 5 საათით უფრო წინ არის, ანუ 12:47 GMT ნიუ იორკის ადგილობრივი დროით, დილის 07:47 იქნება. თუ გრინვიჩიდან აღმოსავლეთით მდებარე ტოკიოში ვიმყოფებით, სადაც ადგილობრივი დრო გრინვიჩის საშუალო დროს 9 საათით ჩამორჩება, მაშინ 12:47 GMT ტოკიოს ადგილობრივი დროით 21:47 იქნება.

სტრუქტურა `DateTime`-ის ბეჭდვა. სტრუქტურა `DateTime`-ის ბეჭდვაზე უფრო მეტი კონტროლის დასამყარებლად შეგვიძლია გამოვიყენოთ სხვა ხერხებიც. ყველაზე სწრაფი ხერხი ეს არის მეთოდი `To*`-ს გამოყენება, მაგ. `ToShortDateString`:

```
label1.Text = DateTime.Now.ToShortDateString() + '\n';
```

შესრულების შედეგი: 2023-09-11. დაიბეჭდა თარიღი მოკლე ფორმატით, დროის მთლიანად გამოტოვებით. შეგვიძლია გამოვიყენოთ `ToLongDateString` მეთოდი:

```
label1.Text = DateTime.Now.ToLongDateString();
```

შესრულების შედეგი: 2023 წლის 11 09, ორშაბათი. ამ კოდში აღსანიშნავია ის, რომ თვე გამოდის დასახელებით და არა რიცხვითი ფორმატით.

სტრუქტურა `DateTime` ფორმატის სტანდარტული სტრიქონი. იმისათვის, რომ შევძლოთ თარიღის და დროის ბეჭდვაზე უფრო მკაცრი კონტროლი, შეგვიძლია გამოვიყენოთ `DateTime` ფორმატის სტანდარტული სტრიქონები. ეს არის ერთი სიმბოლო, რომლიც გამოიყენება თარიღის და დროის ასახვისათვის (იხ. ნახ. 68.)

```
DateTime dt = new DateTime(2092, 12, 24, 18, 42, 0);  
label1.Text = "Short date pattern (d): " + dt.ToString("d") + '\n';  
label1.Text += "Long date pattern (D): " + dt.ToString("D") + '\n';  
label1.Text += "Full date/time pattern (F): " + dt.ToString("F") + '\n';  
label1.Text += "Year/month pattern (y): " + dt.ToString("y") + '\n';
```

```
Short date pattern (d): 2092-12-24  
Long date pattern (D): 2092 წლის 24 12, ოთხშაბათი  
Full date/time pattern (F): 2092 წლის 24 12, ოთხშაბათი 18:42:00  
Year/month pattern (y): დეკემბერი 2092
```

ნახ. 68. სტრუქტურა `DateTime` ფორმატის სტანდარტული სტრიქონი

სტანდარტული ფორმატი საშუალებას გაძლევთ, გარკვეულწილად გააკონტროლოთ ბეჭდვა, მაგრამ ზოგჯერ ეს საკმარისი არ არის - ასეთ შემთხვევაში გვჭირდება `DateTime` ფორმატის სამომხმარებლო სტრიქონები.

სტრუქტურის `DateTime`-ის სამომხმარებლო ფორმატის სტრიქონი. სრული კონტროლისთვის, გვაქვს `DateTime` ფორმატის სამომხმარებლო სტრიქონები. სტრუქტურა `DateTime` ფორმატის სამომხმარებლო სტრიქონები ასოებისა და სპეციალური სიმბოლოების ერთობლიობას წარმოადგენს, რომელსაც გადასვცემთ `ToString()` მეთოდს, რათა ავსახოთ თარიღის ან დროის ზუსტი ფორმატი. ვნახოთ, როგორ მუშაობს ფორმატის დაზუსტების სრული სია (იხ. ნახ. 69.):

```
DateTime dt = new DateTime(2092, 12, 24, 18, 42, 0);
```

```
label1.Text = dt.ToString("MM '/' 'dd yyyy") + '\n';
label1.Text += dt.ToString("dd.MM.yyyy") + '\n';
label1.Text += dt.ToString("MM.dd.yyyy HH:mm") + '\n';
label1.Text += dt.ToString("dddd, MMMM (yyyy): HH:mm:ss") + '\n';
```

```
12/24 2092
24.12.2092
12.24.2092 18:42
ოთხშაბათი, დეკემბერი (2092): 18:42:00
```

ნახ. 69. სტრუქტურა DateTime სამომხმარებლო ფორმატის სტრიქონები

ზოგადად, ერთი ან რამდენიმე d გასცემს დღის გარკვეულ ვარიანტს, ერთი ან მეტი M გასცემს თვეს (ქვედა რეგისტრის m საჭიროა წუთებისათვის) და y კი - წელიწადს.

თარიღებთან მუშაობისას გაითვალისწინეთ, რომ ჩუმათობის პრინციპის მიხედვით, თარიღების დასადგენად გამოიყენება გრიგორიანული კალენდარი.

თარიღების დასამატებლად გამოიყენება მრავალი მეთოდი:

- Add(DateTime date) - date-ს დაამატებს თარიღს;
- AddDays(double value) - მიმდინარე თარიღს დაამატებს რამდენიმე დღეს;
- AddHours(double value) - მიმდინარე თარიღს დაამატებს რამდენიმე საათს;
- AddMinutes(double value) - მიმდინარე თარიღს დაამატებს რამდენიმე წუთს;
- AddMonths(int value) - მიმდინარე თარიღს დაამატებს რამდენიმე თვეს;
- AddYears(int value) - მიმდინარე თარიღს დაამატებს რამდენიმე წელს;

მოცემულ დროს დავამატოთ 3 საათი (იხ. ნახ. 70.):

```
// წელი-თვე-დღე-საათი-წუთი-წამი.
```

```
DateTime date1 = new DateTime(2015, 7, 20, 6, 30, 25);
```

```
label1.Text = date1.ToString() + '\n';
```

```
label1.Text += date1.AddHours(3).ToString();
```

```
2015-07-20 6:30:25
2015-07-20 9:30:25
```

ნახ. 70. მიმდინარე საათს დავუმატეთ 3 საათი

გამოსაკლებად შეგვიძლია გამოვიყენოთ მათოდი Subtract():

```
//წელი-თვე-დღე-საათი-წუთი-წამი.
```



```
DateTime date1 = new DateTime(2015, 7, 20, 6, 30, 25);
DateTime date2 = new DateTime(2015, 7, 20, 3, 30, 25);
label1.Text = date1.Subtract(date2).ToString();
```

შესრულების შედეგი: 03:00:00

როგორც ვხედავთ, Subtract() მეთოდს არ შეუძლია ცალკე მოახდინოს დღეების, წლების და ა. შ. გამოკლება, მაგრამ ჩვენ შეგვიძლია გადავცეთ AddDays() და სხვა მეთოდები. პარამეტრის უარყოფითი მნიშვნელობით - გამოვაკლოთ სამი საათი:

// წელი-თვე-დღე-საათი-წუთი-წამი

```
DateTime date1 = new DateTime(2015, 7, 20, 6, 30, 25);
label1.Text = date1.ToString() + '\n';
label1.Text += date1.AddHours(-3).ToString(); // სამი საათის გამოკლება
```

გამოვაკლოთ სამი დღე:

// წელი-თვე-დღე-საათი-წუთი-წამი

```
DateTime date1 = new DateTime(2015, 7, 20, 6, 30, 25);
label1.Text = date1.ToString() + '\n';
label1.Text += date1.AddDays(-3).ToString(); // სამი დღის გამოკლება
```

ახლა კი განვიხილოთ თარიღის და დროის რიგი სტრიქონული ფორმატებისა:

სიმბოლო D - თარიღის სრული ფორმატი. 2007 წლის 04 02, კვირა

სიმბოლო d - თარიღის მოკლე ფორმატი. 17.07.2007

სიმბოლო F - თარიღის და დროის სრული ფორმატი. 2007 წლის 04 02, კვირა 9:36:51

სიმბოლო f - თარიღის სრული და დროის მოკლე ფორმატი. 2007 წლის 04 02 კვირა 9:36

სიმბოლო G - თარიღის მოკლე და დროის სრული ფორმატი. 04.02.2007 9:36:51

სიმბოლო g - თარიღის და დროის მოკლე ფორმატი. 04.02.2007 9:36

სიმბოლო M, m - დღეების ფორმატი. 04 02

სიმბოლო R, r - დრო გრინვიჩის მიხედვით. Sun, 04 Feb 2007 09:42:17 GMT

სიმბოლო s - თარიღის და დროის დალაგებული ფორმატი. 2007-02-04T09:42:17

სიმბოლო T - დროის სრული ფორმატი. 9:42:17

სიმბოლო t - დროის მოკლე ფორმატი. 9:42

სიმბოლო U - უნივერსალური სრული ფორმატი. 2007 წლის 04 02, კვირა 5:42:17

სიმბოლო u - უნივერსალური მოკლე ფორმატი. 2007-02-04 09:42:17Z

სიმბოლო Y, y - წლის ფორმატი. თებერვალი 2007

თარიღის და დროის სტრიქონული ფორმატები წარმოვადგინოთ პროგრამულად (იხ. ნახ. 71.):


```

DateTime datetm1 = DateTime.Now;
label1.Text += "D: " + datetm1.ToString("D") + '\n';
label1.Text += "d: " + datetm1.ToString("d") + '\n';
label1.Text += "F: " + datetm1.ToString("F") + '\n';
label1.Text += "f: " + datetm1.ToString("f") + '\n';
label1.Text += "G: " + datetm1.ToString("G") + '\n';
label1.Text += "g: " + datetm1.ToString("g") + '\n';
label1.Text += "M: " + datetm1.ToString("M") + '\n';
label1.Text += "R: " + datetm1.ToString("R") + '\n';
label1.Text += "s: " + datetm1.ToString("s") + '\n';
label1.Text += "T: " + datetm1.ToString("T") + '\n';
label1.Text += "t: " + datetm1.ToString("t") + '\n';
label1.Text += "U: " + datetm1.ToString("U") + '\n';
label1.Text += "u: " + datetm1.ToString("u") + '\n';
label1.Text += "Y: " + datetm1.ToString("Y") + '\n';

```

```

D: 2023 წლის 12 09, სამშაბათი
d: 2023-09-12
F: 2023 წლის 12 09, სამშაბათი 11:14:32
f: 2023 წლის 12 09, სამშაბათი 11:14
G: 2023-09-12 11:14:32
g: 2023-09-12 11:14
M: 12 09
R: Tue, 12 Sep 2023 11:14:32 GMT
s: 2023-09-12T11:14:32
T: 11:14:32
t: 11:14
U: 2023 წლის 12 09, სამშაბათი 7:14:32
u: 2023-09-12 11:14:32Z
Y: სექტემბერი 2023

```

ნახ. 71. მიმდინარე საათს დავუმატეთ 3 საათი

დროის ინტერვალები. თარიღისა და დროის გარდა შეგვიძლია, აგრეთვე, ვიმუშაოთ დროის ინტერვალებთან. დროის ინტერვალებთან სამუშაოდ გამოიყენება System.TimeSpan კლასი. კლასის System.TimeSpan-ის კონსტრუქტორს აქვს რამდენიმე გადატვირთული ვერსია. მათი სინტაქსია:

`TimeSpan(int საათი, int წუთი, int წამი);`

`TimeSpan(int დღე, int საათი, int წუთი, int წამი);`

`TimeSpan(int დღე, int საათი, int წუთი, int წამი, int მილიწამი);`

`TimeSpan(int წიკი);`

დროის ინტერვალის თვისებების Days, Hours, Minutes, Seconds, Milliseconds და Ticks აღწერა:

//დროითი ინტერვალის შექმნა საათის, წუთისა და წამის მითითებით

```
TimeSpan Drois_Intervali1 = new TimeSpan(5, 15, 45);
```

// დროითი ინტერვალის შექმნა დღის, საათის, წუთისა და წამის მითითებით

```
TimeSpan Drois_Intervali2 = new TimeSpan(2, 5, 15, 45);
```

// დროითი ინტერვალის შექმნა დღის, საათის, წუთის, წამისა და მილიწამის მითითებით

```
TimeSpan Drois_Intervali3 = new TimeSpan(2, 5, 15, 45, 80);
```

// დროითი ინტერვალის შექმნა წიკის მითითებით

```
TimeSpan Drois_Intervali4 = new TimeSpan(1000);
```

```
label1.Text = Drois_Intervali1.ToString() + '\n'; // 05:15:45
```

```
label1.Text += Drois_Intervali2.ToString() + '\n'; // 2.05:15:45
```

```
label1.Text += Drois_Intervali3.ToString() + '\n'; // 2.05:15:45.0800000
```

```
label1.Text += Drois_Intervali4.ToString() + '\n'; // 00:00:00.0001000
```

სტრიქონებთან სამუშაო მეთოდები: Compare, Concat, Equals, Format, Insert, IndexOf, LastIndexOf, EndsWith, Remove, Replace, Split, Join, StartsWith, SubstringToString, Trim, TrimEnd, TrimStart. სტრიქონების მასივები.

C# ენაში სტრიქონები განეკუთვნება String კლასს. კლასი String წარმოადგენს ძირითად ტიპს სტრიქონებთან მიმართებით და განსაზღვრულია System სახელების სივრცეში.

სტრიქონები გამოიყენება Unicode სიმბოლოების მიმდევრობის შესანახად. ერთი Unicode სიმბოლო იკავებს 2 ბაიტს ანუ 16 ბიტს. String კლასის ობიექტის მაქსიმალური ზომა არის 2 გბ, ე. ი. დაახლოებით ერთი მილიარდი სიმბოლო. დაპროგრამების მრავალ ენაში სტრიქონი წარმოადგენს სიმბოლოების მასივს, მაგრამ C# ენაში სტრიქონი არის ობიექტი. ამიტომ string ტიპი არის მიმართებითი ტიპი. ე. ი. სტრიქონი System.String კლასის მიმართებითი ტიპის მქონე ობიექტია.

სტრიქონული ტიპი აღიწერება string სიტყვის საშუალებით. string ტიპის მნიშვნელობა იწერება ბრჭყალებში და არა აპოსტროფებში.

```
int n = 12; // ეს მთელი რიცხვია
```

// ეს სტრიქონია. char-საგან განსხვავებით სტრიქონი იწერება ორმაგ ბრჭყალებში

```
string m = "12";
```

```
int n1 = 12; int m1 = 7; int k1 = n1 + m1; // შედეგი არის 19
```

```
string q1 = "12"; string z1 = "7"; string r1 = q1 + z1; // შედეგი არის 127
```

სტრიქონის (string ტიპის ობიექტის) შექმნის ყველაზე მარტივი საშუალებაა სტრიქონული ლიტერალის გამოყენება. მაგალითად, string str = “დილა მშვიდობისა, საქართველოვ”;

აქ str ცვლადი არის string ტიპის მიმართებითი ცვლადი, რომელსაც ენიჭება სტრიქონულ ლიტერალზე მიმართვა. ამ შემთხვევაში str ცვლადის ინიციალიზება ხდება სიმბოლოების მიმდევრობით. ისევე როგორც მასივებში, აქაც პირველი ელემენტის (სიმბოლოს) ინდექსია 0. ინდექსი გამოიყენება მხოლოდ სიმბოლოს მისაღებად. რადგან, სტრიქონები ობიექტებს წარმოადგენენ, ამიტომ მათ Length თვისება აქვთ. მასში ინახება სტრიქონის სიგრძე (სტრიქონში სიმბოლოების რაოდენობა). სტრიქონი ასევე შეიძლება შეიცავდეს მმართველ სიმბოლოებს (იხ. ცხრილი 12.).

ცხრილი 12. კლასი String-ის მეთოდები

კლასი String-ის მეთოდები	
მეთოდი	სტრუქტურა და გადატვირთული ვერსიები
სტრიქონების შედარება	
Compare()	1. public static int Compare(string strA, string strB) სტატიკური მეთოდი, რომელიც strA სტრიქონს ადარებს strB სტრიქონს; თუ strA მეტია strB-ზე, მაშინ გასცემს დადებით მნიშვნელობას; თუ strA ნაკლებია strB-ზე, მაშინ - უარყოფით მნიშვნელობას; და გასცემს ნულს, თუ strA და strB ერთმანეთის ტოლია; 2. public static int Compare(string strA, string strB, bool ignoreCase) შედარება ხორციელდება რეგისტრების და კულტურული თავისებურების გათვალისწინებით; თუ პარამეტრი ignoreCase იღებს ლოგიკურ true მნიშვნელობას, მაშინ სტრიქონების შედარებისას არ ხდება ზედა და ქვედა რეგისტრის სიმბოლოებს შორის განსხვავების გათვალისწინება; საწინააღმდეგო შემთხვევაში ეს განსხვავება გათვალისწინებულია; 3. public static int Compare(string strA, string strB, StringComparison comparisonType)

	პარამეტრი <code>comparisonType</code> განსაზღვრავს სტრიქონების შედარების კონკრეტულ ხერხს. 4. public static int Compare(string strA, string strB, bool ignoreCase, CultureInfo culture) კლასი <code>CultureInfo</code> განსაზღვრულია <code>System.Globalization</code> სახელების სივრცეში; 5. public static int Compare(string strA, int indexA, string strB, int indexB, int length) ერთმანეთს ადარებს <code>strA</code> და <code>strB</code> სტრიქონების ნაწილებს. შედარება იწყება სტრიქონული ელემენტებიდან <code>strA[indexA]</code> და <code>strB[indexB]</code> და მოიცავს იმ სიმბოლოების რაოდენობას, რომლებიც განსაზღვრულია <code>length</code> პარამეტრით. 6. public static int Compare(string strA, int indexA, string strB, int indexB, int length, bool ignoreCase) მეთოდი აბრუნებს დადებით მნიშვნელობას, თუ <code>strA</code> სტრიქონის ნაწილი მეტია <code>strB</code> სტრიქონის ნაწილზე; აბრუნებს უარყოფით მნიშვნელობას, თუ <code>strA</code> სტრიქონის ნაწილი ნაკლებია <code>strB</code> სტრიქონის ნაწილზე. აბრუნებს ნულს, თუ <code>strA</code> და <code>strB</code> სტრიქონების შესადარებელი ნაწილები ტოლია; 7. public static int Compare(string strA, int indexA, string strB, int indexB, int length, StringComparison comparisonType) 8. public static int Compare(string strA, int indexA, string strB, int indexB, int length, bool ignoreCase, CultureInfo culture) შედარება ხორციელდება რეგისტრებისა და კულტურული თავისებურებების გათვალისწინებით;
CompareOrdinal()	9. public static int CompareOrdinal(string strA, string strB) ასრულებს ყველაფერს იმავეს, რასაც <code>Compare()</code>. 10. public static int CompareOrdinal(string strA, int indexA, string strB, int indexB, int count) შედარების დროს არ მოცდება ენისა და კულტურის თავისებურებების გათვალისწინება.
CompareTo()	11. public int CompareTo(object value)

	გამომძახებელ სტრიქონს ადარებს value ობიექტს. აბრუნებს დადებით მნიშვნელობას, თუ გამომძახებელი სტრიქონი მეტია value სტრიქონზე; აბრუნებს უარყოფით მნიშვნელობას, თუ გამომძახებელი სტრიქონი ნაკლებია value სტრიქონზე; აბრუნებს ნულს, თუ შესადარებელი სტრიქონები ტოლია;
	12. public int CompareTo(string strB) გამომძახებელ სტრიქონს ადარებს strB (მითითებულ) სტრიქონს
Equals()	13. public override bool Equals(object obj) აბრუნებს ლოგიკურ true მნიშვნელობას, თუ გამომძახებელი სტრიქონი შეიცავს იმავე სიმბოლოების თანამიმდევრობას, რასაც ობიექტი obj-ს სტრიქონული წარმოდგენა
	14. public bool Equals(string value) აბრუნებს ლოგიკურ true მნიშვნელობას, თუ გამომძახებელი სტრიქონი შეიცავს იგივე სიმბოლოების თანამიმდევრობას, რასაც value სტრიქონი.
	15. public bool Equals(string value, StringComparison comparisonType) ხდება რიგითობითი შედარება რეგისტრის გათვალისწინებით, მაგრამ კულტურული თავისებურებების გათვალისწინების გარეშე; პარამეტრი comparisonType განსაზღვრავს სტრიქონების შედარების კონკრეტულ ხერხს.
	16. public static bool Equals(string a, string b) აბრუნებს ლოგიკურ true მნიშვნელობას, თუ სტრიქონი a შეიცავს სიმბოლოების იმავე თანამიმდევრობას, რასაც სტრიქონი b. 17. public static bool Equals(string a, string b, StringComparison comparisonType) ხორციელდება რიგითობითი შედარება რეგისტრის გათვალისწინებით, მაგრამ კულტურული გარემოს გათვალისწინების გარეშე; პარამეტრი comparisonType განსაზღვრავს სტრიქონების შედარების კონკრეტულ ხერხს.
სტრიქონების კონკატენაცია გაერთიანება	
Concat()	18. public static string Concat(string str0, string str1); გასცემს ორი ან მეტი სტრიქონის გაერთიანებას

	19. public static string Concat(params string[] values); კონკატენაციის დროს რეალურად მეხსიერებაში გამოიყოფა ადგილი, სადაც იქმნება string ტიპის ახალი ცვლადი, რომელსაც ენიჭება string ტიპის სტრიქონების „გაერთიანების“ შედეგად მიღებული მნიშვნელობა.
ძებნა სტრიქონში	
Contains()	20. public bool Contains(string value) მეთოდი ამოწმებს, შეიცავს თუ არა სტრიქონი გარკვეულ ქვესტრიქონს (value)
StartsWith()	21. public bool StartsWith(string value) აბრუნებს ლოგიკურ true მნიშვნელობას, თუ გამომძახებელი სტრიქონი იწყება value ქვესტრიქონით; წინააღმდეგ შემთხვევაში აბრუნებს ლოგიკურ false მნიშვნელობას. 22. public bool StartsWith(string value, StringComparison comparisonType) პარამეტრი comparisonType განსაზღვრავს ძიების განხორციელების კონკრეტულ ხერხს.
EndsWith()	23. public bool EndsWith(string value) აბრუნებს ლოგიკურ true მნიშვნელობას, თუ გამომძახებელი სტრიქონი მთავრდება value ქვესტრიქონით; წინააღმდეგ შემთხვევაში აბრუნებს ლოგიკურ false მნიშვნელობას. 24. public bool EndsWith(string value, StringComparison comparisonType) პარამეტრი comparisonType განსაზღვრავს ძიების განხორციელების კონკრეტულ ხერხს.
IndexOf()	25. public int IndexOf(char value) 26. public int IndexOf(string value) გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც პირველად იყო ნაპოვნი მითითებული ქვესტრიქონი; თუ ქვესტრიქონი ან სიმბოლო ვერ მოიძებნა, აბრუნებს -1 მნიშვნელობას;
	27. public int IndexOf(char value, int startIndex) 28. public int IndexOf(string value, int startIndex)

	29. public int IndexOf(char value, int startIndex, int count) 30. public int IndexOf(string value, int startIndex, int count) გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც პირველად იყო ნაპოვნი მითითებული ქვესტრიქონი; ძებნა დაიწყება იმ ელემენტიდან, რომლის ინდექსიც არის startIndex და მოიცავს, ყველა იმ ელემენტს რომელიც განსაზღვრულია პარამეტრით count. თუ ქვესტრიქონი ან სიმბოლო ვერ მოიძებნა, აბრუნებს -1 მნიშვნელობას;
LastIndexOf()	31. IndexOf() მეთოდის ანალოგიური გადატვირთული ვერსია
IndexOfAny()	32. public int IndexOfAny(char[] anyOf) 33. public int IndexOfAny(char[] anyOf, int startIndex) 34. public int IndexOfAny(char[] anyOf, int startIndex, int count) გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც უკანასკნელად იყო ნაპოვნი მითითებული ქვესტრიქონი გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც პირველად იყო ნაპოვნი anyOf სიმბოლოების მასივის ნებისმიერი სიმბოლო გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც უკანასკნელად იყო ნაპოვნი სიმბოლოების მასივის ნებისმიერი სიმბოლო
LastIndexOfAny	35. IndexOfAny() მეთოდის ანალოგიური გადატვირთული ვერსია
სტრიქონების დაყოფა და გაერთიანება	
Split	36. public string[] Split(params char[] separator) 37. public string[] Split(params char[] separator, int count) სტრიქონს ჰყოფს სტრიქონების მასივად მითითებული გამყოფის გამოყენებით
Join()	38. public static string Join(string separator, string[] value) 39. public static string Join(string separator, string[] value, int startIndex, int count) მეთოდი გამყოფს ათავსებს მასივის ელემენტებს შორის და გასცემს მიღებულ სტრიქონს
სტრიქონის შევსება და ამოჭრა	
Trim()	40. public string Trim(params char[] trimChars)

	გამომძახებელ სტრიქონში შლის საწყის და ბოლო ინტერვალებს. 41. public string TrimEnd(char[] სიმბოლოები) გამომძახებელი სტრიქონის ბოლოში შლის სიმბოლოების მასივში მითითებულ სიმბოლოებს. 42. public string TrimStart(char[] სიმბოლოები) გამომძახებელი სტრიქონის დასაწყისში შლის სიმბოლოების მასივში მითითებულ სიმბოლოებს.
PadLeft()	43. public string PadLeft(int totalWidth) 44. public string PadLeft(int totalWidth, char paddingChar) სტრიქონის სიმბოლოებს ასწორებს მარცხენა საზღვარზე, უმატებს რა სტრიქონის დასაწყისს ინტერვალებს ან მითითებულ სიმბოლოს იმდენჯერ, რომ სტრიქონის სიგრძე გახდეს მითითებული სიგანის ტოლი.
PadRight()	ანალოგიურია PadLeft() მეთოდის სტრიქონის სიმბოლოებს ასწორებს მარჯვენა საზღვარზე, უმატებს რა სტრიქონის ბოლოს ინტერვალებს ან მითითებულ სიმბოლოს იმდენჯერ, რომ სტრიქონის სიგრძე გახდეს მითითებული სიგანის ტოლი
სტრიქონის ჩასმა, წაშლა და შეცვლა	
Insert()	45. public string Insert(int startIndex, string value) გამომძახებელ სტრიქონში startIndex პოზიციიდან დაწყებული ჩასვამს მითითებულ სტრიქონს
Remove()	46. public string Remove(int startIndex) 47. public string Remove(int startIndex, int count) გამომძახებელ სტრიქონში startIndex პოზიციიდან დაწყებული წაშლის მითითებული რაოდენობის სიმბოლოს
Replace()	48. public string Replace(char oldChar, char newChar) 49. public string Replace(string oldValue, string newValue) გამომძახებელ სტრიქონში ძველ სტრიქონს ახალი სტრიქონით შეცვლის
რეგისტრის ცვლილება	
ToUpper()	50. public string ToUpper()

	გამომდახებული სტრიქონის ყველა სიმბოლო გადაყავს ზედა რეგისტრში
ToLower()	51. public string ToLower() გამომდახებული სტრიქონის ყველა სიმბოლო გადაყავს ქვედა რეგისტრში
სტრიქონიდან ქვესტრიქონის მიღება	
Substring()	52. public string Substring(int startIndex) 53. public string Substring(int startIndex, int length) გამომდახებული სტრიქონის startIndex პოზიციიდან გასცემს მითითებული რაოდენობის სიმბოლოს

string ტიპის ობიექტების ერთ-ერთი მნიშვნელოვანი თავისებურება არის ის, რომ სტრიქონში ერთხელ შექმნილი სიმბოლოების მიმდევრობა აღარ იცვლება ე. ი. არ შეიძლება ინდექსის გამოყენება სტრიქონის რომელიმე სიმბოლოსათვის ახალი მნიშვნელობის მისანიჭებლად.

მაგალითად, დაუშვებელია ასეთი მინიჭება:

```
string striqoni[1] = "ს"; // შეცდომაა!!!
```

განვიხილოთ კლასი String-ის მეთოდი Format() დაწვრილებით. Format() მეთოდი გამოიყენება სტრიქონების ფორმატირებისათვის და იყენებს ფორმატირების სიმბოლოებს (იხ. ცხრილი 12.1.): f ან F - რიცხვს გარდაქმნის წილადის ფორმატით; e ან E - რიცხვს აფორმატებს ექსპონენციალური წარმოდგენით; p ან P - რიცხვს აფორმატებს როგორც პროცენტს; n ან N - რიცხვს აფორმატებს, როგორც რიცხვს თანრიგების გამყოფებით; c ან C - რიცხვს აფორმატებს, როგორც თანხას ადგილობრივ ვალუტაში; d ან D - რიცხვს აფორმატებს, როგორც ათობით რიცხვს; g ან G - რიცხვს აფორმატებს, როგორც წილადს ან როგორც რიცხვს ექსპონენციალური წარმოდგენით; x ან X - მთელი რიცხვი გადაჰყავს თექვსმეტობით წარმოდგენაში;

ცხრილი 12.1. კლასი String-ის მეთოდი Format()

მაგალითები სტრიქონების ფორმატულ წარმოდგენაზე		
მეთოდი Format()	პროგრამული კოდი	შედეგი
ვალუტის ფორმატი; გამოიყენება სიმბოლო „C“	<pre>double number = 23.7; string result = String.Format("{0:C1}", number); label1.Text=result.ToString(); // 23.7 Lari</pre>	23.7 Lari

	<pre>string result2 = String.Format("{0:C2}", number); label2.Text = result2.ToString(); // 23.70 Lari</pre>	23.70 Lari
მთელი რიცხვების ფორმატი; გამოყენება სიმბოლო „d“	<pre>int number = 23; string result = String.Format("{0:d}", number); label1.Text=result.ToString(); // 23 string result2 = String.Format("{0:d4}", number); label2.Text = result2.ToString(); // 0023</pre>	23 0023
ათწილადი რიცხვების ფორმატი; გამოყენება სიმბოლო „F“	<pre>int number = 23; string result = String.Format("{0:f}", number); label1.Text=result.ToString(); // 23,00 double number2 = 45.08; string result2 = String.Format("{0:f4}", number2); label2.Text = result2.ToString(); // 45,0800 double number3 = 25.07; string result3 = String.Format("{0:f1}", number3); label3.Text = result3.ToString(); // 25,1</pre>	23,00 45,0800 25,1
პროცენტის ფორმატი; გამოყენება სიმბოლო „P“	<pre>decimal number = 0.15345m; string result = String.Format("{0:P1}", number); label1.Text=result.ToString(); // 15.3%</pre>	15.3%
„მორგებული“ ფორმატი; გამოყენება სიმბოლო „#“; მაგ. რიცხვი ტელეფონის ფორმატით: +x (xxx)xxx-xx-xx:	<pre>long number = 19876543210; string result = String.Format("{0:+# (###) ###-##-##}", number); // +1 (987) 654-32-10 label1.Text=result.ToString();</pre>	+1 (987) 654-32-10
მეთოდი ToString()	პროგრამული კოდი	შედეგი
მეთოდი ToString() შეუძლია ფორმატირება	<pre>long number = 19876543210; string result = number.ToString("+# (###) ###-##-##"); //+1 (987) 654-32-10 label1.Text = result.ToString(); double money = 24.8; string result2 = money.ToString("C2"); label2.Text = result2.ToString(); // 24,80 Lari</pre>	+1 (987) 654-32-10 24,80 Lari

string ტიპის ობიექტები არ არის დინამიკური ე. ი. არიან უცვლელი ობიექტები (არსებობენ მხოლოდ წაკითხვისათვის). ჩვენ არ შეგვიძლია მათი მნიშვნელობის შეცვლა მათი შექმნის შემდეგ. მეთოდები რომლებიც გამოიყენება string ტიპის

ობიექტების შესაცვლელად, ფაქტიურად ქმნის ახალ string ობიექტებს, რომლებიც შეიცავს ამ ცვლილებას. სტრიქონში სიმბოლოების შესაცვლელად, უნდა შევქმნათ ახალი სტრიქონი, მაგრამ უკვე არსებულის შეცვლა არ შეგვიძლია.

კლასი String არ გვადლევს საშუალებას, შევცვალოთ არსებული ობიექტები. ამ საკითხის გადასაჭრელად C# ენაში მოხდა ახალი კლასის StringBuilder-ის დამატება, რომელიც საშუალებას გვადლევს, შევქმნათ დინამიკური სტრიქონები. დინამიკურ სტრიქონებში, String კლასის სტრიქონებისაგან განსხვავებით, შესაძლებელია სიმბოლოების პირდაპირი ცვლილება.

დინამიკური სტრიქონი. დინამიკურ სტრიქონებთან სამუშაოდ გამოიყენება კლასი StringBuilder, რომელიც განსაზღვრულია System.Text სახელების სივრცეში. კლასი StringBuilder წარმოადგენს დინამიკურ სტრიქონს.

კლასი StringBuilder-ის მეთოდები მუშაობენ უფრო სწრაფად, ვიდრე String კლასი. კლასი StringBuilder-ის მეთოდები არ ქმნიან სტრიქონის ასლს და უშუალოდ მიმართავენ არსებულ სტრიქონს. დინამიკურ სტრიქონებთან მუშაობის დროს, პროგრამის დასაწყისში, სახელების სივრცეში უნდა მოვათავსოთ დირექტივა:

```
using System.Text;
```

დინამიკური სტრიქონის შექმნის დროს ტევადობა ჩუმათობის პრინციპით არის 16 სიმბოლო. ახალი სიმბოლოს დამატების შემდეგ დინამიკური სტრიქონის ტევადობა ავტომატურად იზრდება. დინამიკურ სტრიქონს შექმნის დროს ავტომატურად ენიჭება 2147483647-ის ტოლი მაქსიმალური ტევადობა.

StringBuilder კლასის კონსტრუქტორი გადატვირთულია, ამდენად არსებობს ამ კლასის კონსტრუქტორის რამდენიმე ვარიანტი (იხ. ცხრილი 13.).

ერთ-ერთი ვარიანტია ცარიელი დინამიკური სტრიქონის შექმნა:

```
StringBuilder DS1 = new StringBuilder();
```

დინამიკური სტრიქონის შექმნისას კონსტრუქტორს შეგვიძლია გადავცეთ: მაქსიმალური ტევადობა; სტრიქონი, რომელიც დინამიკურ სტრიქონში უნდა ჩაიწეროს; სტრიქონი; საწყისი ინდექსი, საიდანაც უნდა დაიწყოს სიმბოლოების გადაწერა; გადასაწერი სიმბოლოების რაოდენობა და დინამიკური სტრიქონის ტევადობა.

//ცარიელი სტრიქონის შექმნა, ზომა ჩუმათობის მიხედვით =16 სიმბოლო.

```
StringBuilder a =new StringBuilder();
```

//სტრიქონის ინიციალიზება, მეხსიერების გამოყოფა ოთხი სიმბოლოსათვის.

```
StringBuilder b = new StringBuilder("abcd");
```

//ცარიელი სტრიქონის შექმნა, მეხსიერების გამოყოფა ასი სიმბოლოსათვის.

```
StringBuilder c = new StringBuilder(100);
```

ცხრილი 13. კლასი StringBuilder-ის თვისებები

კლასი StringBuilder-ის თვისება	აღწერა
Capacity	გასცემს ან აყენებს დინამიკური სტრიქონის ტევადობას (აბრუნებს int მნიშვნელობას)
Length	გასცემს ან აყენებს დინამიკურ სტრიქონში სიმბოლოების რაოდენობას (აბრუნებს int მნიშვნელობას)
MaxCapacity	გასცემს დინამიკური სტრიქონის მაქსიმალურ ტევადობას (აბრუნებს int მნიშვნელობას)

StringBuilder ტიპის სტრიქონის შესაქმნელად გამოიყენება მეხსიერების უფრო მეტი ტევადობა, ვიდრე ეს სჭირდება String ტიპის სტრიქონს. მაგალითად:

```
StringBuilder sb = new StringBuilder("თბილისი!!!");
```

```
label1.Text= sb.Length.ToString(); // სიმბოლო =10
```

```
// გამოყოფილი მეხსიერების ტევადობა ჩუმად =16
```

```
label2.Text = sb.Capacity.ToString();
```

```
label3.Text = sb.MaxCapacity.ToString(); // 2147483647
```

ახლა კი განვიხილოთ დინამიკური სტრიქონის მეთოდები (იხ. ცხრილი 14):

ცხრილი 14. კლასი StringBuilder-ის მეთოდები

კლასი StringBuilder-ის მეთოდები	დაბრუნებული მნიშვნელობის ტიპი	აღწერა
Append()	StringBuilder	ობიექტის სტრიქონულ წარმოდგენას ამატებს დინამიკური სტრიქონის ბოლოში.
AppendFormat()	StringBuilder	დაფორმატებულ სტრიქონს ამატებს დინამიკური სტრიქონის ბოლოში.
EnsureCapacity()	int	ამოწმებს, ტოლია თუ მეტი დინამიკური სტრიქონის მიმდინარე ტევადობა მითითებულ მნიშვნელობაზე.

Equals()	bool	გასცემს true მნიშვნელობას თუ დინამიკური სტრიქონი მითითებული ობიექტის ტოლია, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას.
GetType()	Type	გასცემს ობიექტის ტიპს.
Insert()	StringBuilder	მითითებული ობიექტის სტრიქონულ წარმოდგენას ამატებს დინამიკურ სტრიქონში დაწყებული მითითებული პოზიციიდან.
Remove()	StringBuilder	დინამიკური სტრიქონიდან შლის მითითებული რაოდენობის სიმბოლოს დაწყებული მითითებული პოზიციიდან.
Replace()	StringBuilder	მითითებულ სიმბოლოს ან ქვესტრიქონს დინამიკურ სტრიქონში ყველგან ცვლის მითითებული სიმბოლოთი ან ქვესტრიქონით.
ToString()	string	დინამიკურ სტრიქონს გარდაქმნის სტრიქონად.

განვიხილოთ, თუ რა განსხვავებაა String და StringBuilder კლასებს შორის კონკატენაციის მაგალითზე. ვთქვათ გვაქვს ორი ან მეტი სტრიქონი, მათ გასაერთიანებლად ვიყენებთ ოპერატორს „+“, მაგრამ მისი გამოყენება დიდ ციკლებთან მიმართებით ითვლება არასწორად, რადგან ამ შემთხვევაში ყოველი იტერაციის დროს იქმნება სტრიქონის ახალი ეგზემპლარი. ამის შედეგად ხდება მეხსიერების დაკავება და პროგრამის მწარმოებლობა (სისწრაფე) საკმაოდ ეცემა. ასეთი შეცდომების თავიდან აცილების მიზნით გამოიყენება არა String კლასი, არამედ კლასი StringBuilder;

განსხვავების დასადგენად განვიხილოთ მაგალითები.

მაგალითი 1. მივიღოთ სტრიქონი, რომელიც მიმდევრობით შეიცავს სიტყვას „C#“. ეს ამოცანა შეგვიძლია გადავწყვიტოთ ორი გზით (იხ. ცხრილი 15): **პირველი გზა:** სტრიქონის პირდაპირი მიმატება.

```
string str = "";
for (int j = 0; j < count; j++) {
    str += " C# "; }
label1.Text = str;
```

მეორე გზა: StringBuilder-ის გამოყენება.

```

StringBuilder sb = new StringBuilder();
for (int j = 0; j < count; j++) {
    sb.Append(" C# ");
}
label1.Text = sb.ToString();

```

ცხრილი 15. თითოეული იტერაციაზე დახარჯული დრო და მეხსიერება

სიტყვა „C#“- ის რაოდენობა	სტრიქონის სიგრძე	String		StringBuilder	
		დრო	მეხსიერება	დრო	მეხსიერება
2	10	0.091	15	0.18	16
3	15	0.163	30	0.22	16
4	20	0.252	50	0.373	48
5	25	0.336	75	0.39	48
6	30	0.464	105	0.463	48
7	35	0.565	140	0.591	112
8	40	0.695	180	0.663	112
9	45	0.809	225	0.692	112
10	50	0.965	275	0.731	112
15	75	1.779	600	1.125	240
20	100	2.697	1050	1.354	240
25	125	3.811	1625	1.571	240
30	150	5.045	2325	2.144	496
35	175	6.441	3150	2.359	496
40	200	7.992	4100	2.615	496
45	225	9.59	5175	2.799	496
50	250	11.45	6375	3.03	496
60	300	15.57	9150	4.074	1008
70	350	20.55	12425	4.683	1008
80	400	26.15	16200	5.036	1008
90	450	32.13	20475	5.419	1008

მაგალითი 2. განვიხილოთ მეორე მაგალითი. თავდაპირველად გამოვიყენოთ კლასი String. პროგრამა ითვლის კომპილაციის დროს მილიწამებში. პროგრამაში ხდება 10000 იტერაციის შესრულება.

```

System.Diagnostics.Stopwatch time = new Stopwatch();

```

```

string test = string.Empty;
time.Start();
for (int i = 0; i < 10000; i++)
{
    test += i;
}
time.Stop();

```

```

label1.Text="Using String concatenation: "
           + time.ElapsedMilliseconds + " milliseconds";

```

შესრულების შედეგი კლასი String-ის შემთხვევაში:

Using String concatenation: 122 milliseconds

კოდი მუშაობს, მაგრამ ოპტიმალურია კი ასეთი ჩანაწერი? საინტერესოა, თუ რა შედეგს მოგვცემს იგივე კოდი კლასი StringBuilder-ის გამოყენებით:

```

System.Diagnostics.Stopwatch time = new Stopwatch();
StringBuilder test1 = new StringBuilder();
time.Reset();
time.Start();
for (int i = 0; i < 10000; i++) {
    test1.Append(i); }
time.Stop();

```

```

label1.Text="Using StringBuilder: "+time.ElapsedMilliseconds+"
milliseconds";

```

შესრულების შედეგი კლასი StringBuilder-ის შემთხვევაში:

Using String concatenation: 1 milliseconds

როგორც ვხედავთ, StringBuilder კლასი მუშაობს უფრო სწრაფად, რადგან იგი არ ქმნის სტრიქონის ასლს და იყენებს ნაკლებ მეხსიერებას.

მაგალითი 3. პროგრამა ითვლის კომპილაციის დროს წამებში.

```

string result = "";
DateTime dt = DateTime.Now;
for (int i = 1; i < 10000; i++)
{
    result = result + i.ToString(); }

```

```

label1.Text="შესრულების დრო: "+(DateTime.Now - dt).ToString() + " წამი";
StringBuilder result1 = new StringBuilder();
DateTime dt1 = DateTime.Now;

```



```
for (int i = 1; i < 10000; i++) {
    result1.Append(i.ToString());
}
```

```
label2.Text="შესრულების დრო: " + (DateTime.Now - dt1).ToString() + "წამი";
```

შესრულების შედეგი:

შესრულების დრო: 00:00:00.1150065 წამი

შესრულების დრო: 00:00:00.0010001 წამი

მესამე მაგალითში 10000 ელემენტიან ციკლში ყოველ სტრიქონს ვამატებთ რიცხვს: 1, 12, 123, 1234 და ა.შ. (ხდება სტრიქონების შეერთება).

პირველ შემთხვევაში იქმნება 10000 სტრიქონი (ობიექტი), რომელიც იკავებს მეხსიერებას.

მეორე შემთხვევაში კი არსებული ობიექტი იცვლება StringBuilder კლასით. ამდენად მეორე შემთხვევაში პროგრამის ეფექტური რეალიზაციის დრო გაცილებით უფრო მცირეა, ვიდრე String კლასისას.

გეომეტრიული ფიგურების ხატვა, ფიგურების შევსება ფერით, გამოსახულებასთან მუშაობა. კლასი Graphics განსაზღვრავს სახატავ უბანს (ფურცელს, ტილოს), კლასი Pen გამოიყენება ხაზებისა და გრაფიკული ფიგურების დასახატად, კლასი Brush გამოიყენება ფიგურების შესავსებად არჩეული ფერით, კლასი Rectangle განსაზღვრავს სახატავი უბნის საზღვრებს.

ყველა გრაფიკული ობიექტისათვის სიგრძის ერთეული არის პიქსელი. კოორდინატების დასაწყისად ითვლება Graphics ობიექტის ზედა მარცხენა კუთხე.

კლასი Graphics-ის ობიექტი წარმოადგენს სახატავ ზედაპირს, ასევე შეგვიძლია განვიხილოთ როგორც სახატავი ფურცელი (ტილო). კლასი Pen-ის ობიექტი შეიცავს დასახატი ხაზის თვისებებს.

გეომეტრიული ფიგურების ხატვა. განვიხილოთ კლასი Graphics-ის ზოგიერთი მეთოდი. კლასი Graphics-ის ობიექტი იქმნება CreateGraphics() მეთოდის საშუალებით. მაგალითად: `Graphics SaxataviPurceli = CreateGraphics();`

მეთოდი DrawArc() გამოიყენება ელიფსის ნაწილის (რკალის) დასახატად. მისი სინტაქსია:

```
void Graphics.DrawArc(Pen ფუნჯი, Rectangle სწორკუთხედი, float
საწყისი_კუთხე, float მოღუნვის_კუთხე);
```


მეთოდი DrawRectangle() გამოიყენება სწორკუთხედის დასახატად. მისი სინტაქსია:

```
void Graphics.DrawRectangle(Pen ფუნჯი, Rectangle სწორკუთხედი);
```

მეთოდი DrawEllipse() გამოიყენება ელიფსის დასახატად. მისი სინტაქსია:

```
void Graphics.DrawEllipse(Pen ფუნჯი, Rectangle სწორკუთხედი);
```

მეთოდი DrawPie() გამოიყენება სექტორის დასახატად. მისი სინტაქსია:

```
Void Graphics.DrawPie(Pen ფუნჯი, Rectangle სწორკუთხედი, float  
საწყისი_კუთხე, float მოღუნვის_კუთხე);
```

მეთოდი DrawLine() გამოიყენება ხაზის გასავლებად საწყისი წერტილიდან საბოლოო წერტილამდე. მისი სინტაქსია:

```
void Graphics.DrawLine(Pen ფუნჯი, float x1, float y1, float x2, float y2);
```

აქ x1 და y1 არის საწყისი წერტილის კოორდინატები, ხოლო x2 და y2 კი - საბოლოო წერტილისა.

მეთოდი DrawPolygon() გამოიყენება მრავალკუთხედის ასაგებად. მისი სინტაქსია:

```
void Graphics.DrawPolygon(Pen ფუნჯი, Point[] წერტილები);
```

მეთოდი DrawLines() გამოიყენება ხაზების გასავლებად საწყისი წერტილიდან საბოლოო წერტილამდე. მისი სინტაქსია:

```
void Graphics.DrawLines(Pen ფუნჯი, Point[] წერტილები);
```

კლასი Pen-ს კონსტრუქტორს აქვს 4 გადატვირთული ვერსია.

პირველი ვერსიის სინტაქსია: Pen ობიექტის_სახელი = new Pen(Brush ფუნჯი);

მეორე ვერსიის სინტაქსია: Pen ობიექტის_სახელი = new Pen(Color ფერი);

მესამე ვერსიის სინტაქსია: Pen ობიექტის_სახელი = new Pen(Brush ფუნჯი, float სიგანე);

მეოთხე ვერსიის სინტაქსია: Pen ობიექტის_სახელი = new Pen(Color ფერი, float სიგანე);

მაგალითი 4. რკალის დახაზვა (იხ. ნახ. 72.):

```
// სახატავი ფურცლის შექმნა.
```

```
Graphics SaxataviPurceli = CreateGraphics();
```

```
// კალმის შექმნა.
```

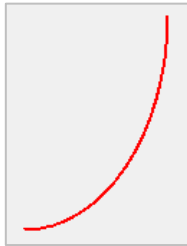
```
Pen Kalami = new Pen(Color.Red, 2);
```

```
// ხატვის უბნის შექმნა.
```

```
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
```

// რკალის ხატვა.

```
SaxataviPurceli.DrawArc(Kalami, Otxkutxedi, 0, 90);
```



ნახ. 72. მე-4 მაგალითის შესრულების შედეგი

როგორც პროგრამიდან ჩანს, ჯერ იქმნება სახატავი ფურცელი. შემდეგ იქმნება წითელი ფერის კალამი, რომლის სისქეა 2 პიქსელი. შემდეგ იქმნება ხატვის უბანი, რომელშიც უნდა მოთავსდეს რკალი. ამ უბნის საწყისი წერტილია (0,0), საბოლოო წერტილი კი - (200,300). ბოლოს სრულდება რკალის ხატვა.

მაგალითი 5. ნახევარი რკალის დახატვა (იხ. ნახ. 73.):

// სახატავი ფურცლის შექმნა.

```
Graphics SaxataviPurceli = CreateGraphics();
```

// კალმის შექმნა.

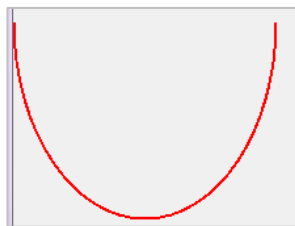
```
Pen Kalami = new Pen(Color.Red, 2);
```

//ხატვის უბნის შექმნა.

```
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
```

// რკალის ხატვა.

```
SaxataviPurceli.DrawArc(Kalami, Otxkutxedi, 0, 180);
```



ნახ. 73. მე-5 მაგალითის შესრულების შედეგი

მაგალითი 6. რკალის დახატვა (იხ. ნახ. 74.):

//სახატავი ფურცლის შექმნა.

```
Graphics SaxataviPurceli = CreateGraphics();
```

//კალმის შექმნა.

```
Pen Kalami = new Pen(Color.Red, 2);
```

//ხატვის უბნის შექმნა.

```
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
```

//რკალის ხატვა.

```
SaxataviPurceli.DrawArc(Kalami, Otxkutxedi, 0, 270);
```



ნახ. 74. მე-6 მაგალითის შესრულების შედეგი

მაგალითი 7. ელიფსის დახაზვა (იხ. ნახ. 75.):

//სახატავი ფურცლის შექმნა.

```
Graphics SaxataviPurceli = CreateGraphics();
```

// კალმის შექმნა.

```
Pen Kalami = new Pen(Color.Red, 2);
```

// ხატვის უბნის შექმნა.

```
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
```

// შეკრული რკალის ხატვა.

```
SaxataviPurceli.DrawArc(Kalami, Otxkutxedi, 0, 360);
```



ნახ. 75. მე-7 მაგალითის შესრულების შედეგი

განვიხილოთ მეთოდი DrawLines() გამოიყენება ხაზების გასავლებად.

მაგალითი 8. ხაზის დახაზვა (იხ. ნახ. 76.):

//წერტილების შექმნა, რომლებიც წრფეს განსაზღვრავს.

```
Point wertili1 = new Point(25, 25);
```

```
Point wertili2 = new Point(100, 25);
```

```
Point[] WertilebisMasivi = { wertili1, wertili2 };
```

```
Graphics SaxataviPurceli = CreateGraphics();
```

```
Pen Kalami = new Pen(Color.Red, 2);
```

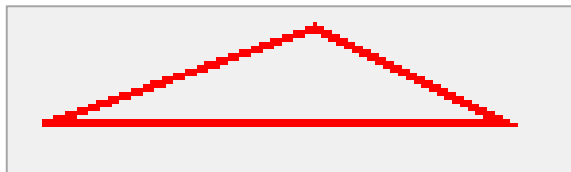
```
SaxataviPurceli.DrawLine(Kalami, WertilebisMasivi);
```



ნახ. 76. მე-8 მაგალითის შესრულების შედეგი

მაგალითი 9. სამკუთხედის დახაზვა (იხ. ნახ. 77.):

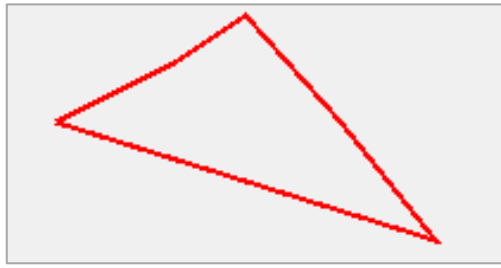
```
Point wertili1 = new Point(50, 50);  
Point wertili2 = new Point(120, 25);  
Point wertili3 = new Point(120, 25);  
Point wertili4 = new Point(170, 50);  
Point wertili5 = new Point(170, 50);  
Point wertili6 = new Point(50, 50);  
Point[] WertilebisMasivi = { wertili1, wertili2, wertili3,  
    wertili4, wertili5, wertili6 };  
Graphics SaxataviPurceli = CreateGraphics();  
Pen Kalami = new Pen(Color.Red, 2);  
SaxataviPurceli.DrawLine(Kalami, WertilebisMasivi);
```



ნახ. 77. მე-9 მაგალითის შესრულების შედეგი

მაგალითი 10. გეომეტრიული ფიგურის დახაზვა (იხ. ნახ. 78.):

```
Point wertili1 = new Point(50, 50);  
Point wertili2 = new Point(100, 25);  
Point wertili3 = new Point(130, 5);  
Point wertili4 = new Point(170, 50);  
Point wertili5 = new Point(210, 100);  
Point wertili6 = new Point(50, 50);  
Point[] WertilebisMasivi = { wertili1, wertili2, wertili3,  
    wertili4, wertili5, wertili6 };  
Graphics SaxataviPurceli = CreateGraphics();  
Pen Kalami = new Pen(Color.Red, 2);  
SaxataviPurceli.DrawLine(Kalami, WertilebisMasivi);
```

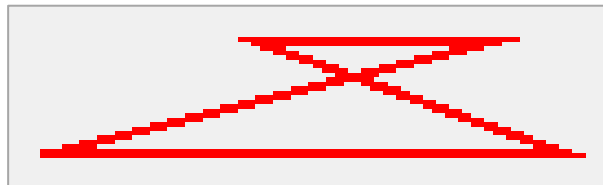


ნახ. 78. მე-10 მაგალითის შესრულების შედეგი

მაგალითი 11. გეომეტრიული ფიგურის დახაზვა (იხ. ნახ. 79.):

//წერტილების შექმნა, რომლებიც მრავალკუთხედს განსაზღვრავს.

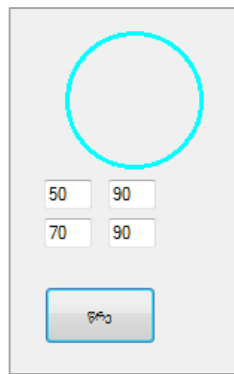
```
Point wertili1 = new Point(50, 50);
Point wertili2 = new Point(150, 25);
Point wertili3 = new Point(100, 25);
Point wertili4 = new Point(170, 50);
Point wertili5 = new Point(170, 50);
Point wertili6 = new Point(50, 50);
Point[] WertilebisMasivi = {wertili1, wertili2, wertili3,
wertili4, wertili5, wertili6 };
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
SaxataviPurceli.DrawLines(Kalami, WertilebisMasivi);
```



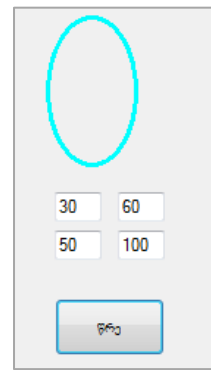
ნახ. 79. მე-11 მაგალითის შესრულების შედეგი

მაგალითი 12. წრის და ელიფსის დახაზვა (იხ. ნახ. 80. ა. ბ.):

```
Graphics SaxataviPurceli = CreateGraphics();
int m1 = int.Parse(textBox1.Text);
int m2 = int.Parse(textBox2.Text);
int m3 = int.Parse(textBox3.Text);
int m4 = int.Parse(textBox4.Text);
Pen kalami = new Pen(Color.Cyan, 3);
SaxataviPurceli.DrawEllipse(kalami, m1, m2, m3, m4);
```



ა)

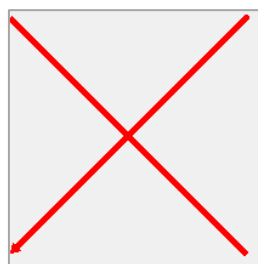


ბ)

ნახ. 80. გეომეტრიული ფიგურის დახაზვა: ა) წრე; ბ) ელიფსი

მაგალითი 13. გადამკვეთი ხაზები (იხ. ნახ. 81.):

```
int x1 = 5, y1 = 5, x2 = 200, y2 = 200;
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 5);
SaxataviPurceli.DrawLine(Kalami, x1, y1, x2, y2);
x1 = 5; y1 = 200; x2 = 200; y2 = 5;
Kalami.StartCap = LineCap.ArrowAnchor;
Kalami.EndCap = LineCap.Round;
SaxataviPurceli.DrawLine(Kalami, x1, y1, x2, y2);
Kalami.Dispose(); // რესურსების განთავისუფლება.
SaxataviPurceli.Dispose();
```



ნახ. 81. მე-13 მაგალითის შესრულების შედეგი

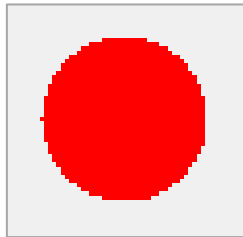
ფიგურების შევსება ფერით. კლასი Brush გამოიყენება სხვადასხვა ფერით და შაბლონით ფიგურების შესავსებად. ის აბსტრაქტულია და ამიტომ ამ კლასის ობიექტებს ვერ შევქმნით. კლასი Brush-დან წარმოებული კლასებია:

- SolidBrush - უბნის მთლიანად შევსება მითითებული ფერით;
- TextureBrush - უბნის შევსება გამოსახულებით (გამოსხულების (image) - წაკითხვა ხდება ფაილიდან);

- HatchBrush - უბნის შევსება წინასწარ განსაზღვრული ნიმუშით, მაგალითად, ხაზებიანი ნიმუშით;
- LinearGradientBrush - უბნის შევსება გარდამავალი ფერით, გამოიყენება წრფივი გრადიენტი;
- PathGradientBrush - უბნის შევსება გარდამავალი ფერით, სადაც ფერების ცვლილება წარმოდგენილია შედარებით რთული გრადიენტით.

მაგალითი 14. წრის დახაზვა (იხ. ნახ. 82.):

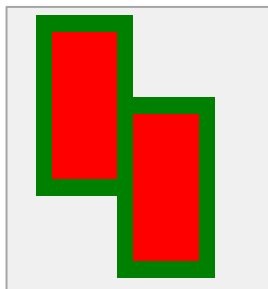
```
Graphics SaxataviPurceli = CreateGraphics();
SaxataviPurceli.FillEllipse(Brushes.Red, 10, 10, 40, 40);
```



ნახ. 82. მე-14 მაგალითის შესრულების შედეგი

მაგალითი 15. გეომეტრიული ფიგურების დახაზვა (იხ. ნახ. 83.):

```
Graphics SaxataviPurceli = CreateGraphics();
Pen p2 = new Pen(Color.Green, 4); // სისქე 4 პიქსელი
SaxataviPurceli.FillRectangle(Brushes.Red, 10, 10, 20, 40);
SaxataviPurceli.DrawRectangle(p2, 10, 10, 20, 40);
SaxataviPurceli.TranslateTransform(20, 20);
SaxataviPurceli.FillRectangle(Brushes.Red, 10, 10, 20, 40);
SaxataviPurceli.DrawRectangle(p2, 10, 10, 20, 40);
```



ნახ. 83. მე-15 მაგალითის შესრულების შედეგი

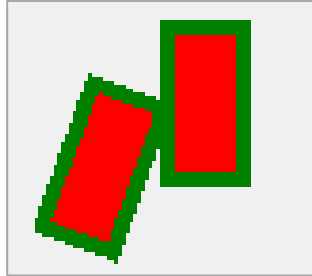
მაგალითი 16. გეომეტრიული ფიგურების დახაზვა (იხ. ნახ. 84.):

```
Graphics SaxataviPurceli = CreateGraphics();
Pen p2 = new Pen(Color.Green, 4); // სისქე 4 პიქსელი.
SaxataviPurceli.FillRectangle(Brushes.Red, 50, 50, 20, 40);
```

```

SaxataviPurceli.DrawRectangle(p2, 50, 50, 20, 40);
SaxataviPurceli.RotateTransform(20);
SaxataviPurceli.FillRectangle(Brushes.Red, 50, 50, 20, 40);
SaxataviPurceli.DrawRectangle(p2, 50, 50, 20, 40);

```



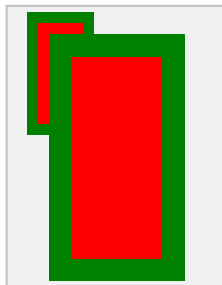
ნახ. 84. მე-16 მაგალითის შესრულების შედეგი

მაგალითი 17. გეომეტრიული ფიგურების დახაზვა (იხ. ნახ. 85.):

```

Graphics SaxataviPurceli = CreateGraphics();
Pen p2 = new Pen(Color.Green, 4); // სისქე 4 პიქსელი.
SaxataviPurceli.FillRectangle(Brushes.Red, 10, 10, 20, 40);
SaxataviPurceli.DrawRectangle(p2, 10, 10, 20, 40);
SaxataviPurceli.ScaleTransform(2, 2);
SaxataviPurceli.FillRectangle(Brushes.Red, 10, 10, 20, 40);
SaxataviPurceli.DrawRectangle(p2, 10, 10, 20, 40);

```



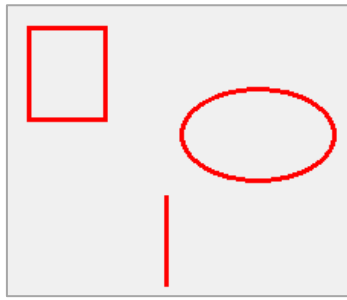
ნახ. 85. მე-17 მაგალითის შესრულების შედეგი

მაგალითი 18. გეომეტრიული ფიგურების დახაზვა (იხ. ნახ. 86.):

```

Graphics g = this.CreateGraphics();
Pen redPen = new Pen(Color.Red, 3); // კალამი 3 პიქსელიანი სისქით.
// კალმის დახმარებით ვხატავთ ხაზს, წრეს და ოთხკუთხედს.
g.DrawLine(redPen, 140, 170, 140, 230);
g.DrawRectangle(redPen, 50, 60, 50, 60);
g.DrawEllipse(redPen, 150, 100, 100, 60);
g.Dispose(); // გასუფთავება.

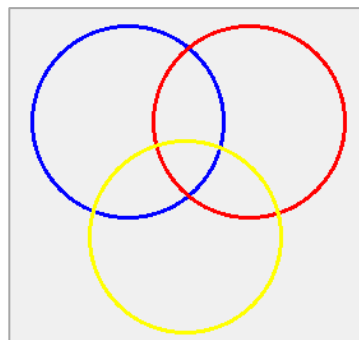
```

ნახ. 86. მე-18 მაგალითის შესრულების შედეგი

მაგალითი 19. გეომეტრიული ფიგურების დახაზვა (იხ. ნახ. 87.):

```
Graphics A = CreateGraphics(); Graphics B = CreateGraphics();
Graphics C = CreateGraphics(); Pen Bluepen = new
Pen(Color.Blue, 3);
Pen RedPen = new Pen(Color.Red, 3);
Pen BlackPen = new Pen(Color.Yellow, 3);
A.DrawEllipse(Bluepen, 100, 100, 150, 150);
B.DrawEllipse(RedPen, 195, 100, 150, 150);
C.DrawEllipse(BlackPen, 145, 190, 150, 150);
```



ნახ. 87. მე-19 მაგალითის შესრულების შედეგი

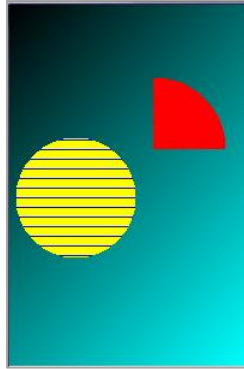
მაგალითი 20. გეომეტრიული ფიგურების დახაზვა (იხ. ნახ. 88.):

```
using System.Drawing.Drawing2D;
int x = 0; int y = 0; int Sigane = 200; int Simagle = 300;
Graphics SaxataviPurceli = CreateGraphics();
Rectangle Otxkutxedi = new Rectangle(x, y, Sigane, Simagle);
// ფუნჯის შექმნა თანაბარი ფერით შევსებისათვის.
Brush PunjiSolid = new SolidBrush(Color.Red);
// ფუნჯის შექმნა შაბლონის მიხედვით შევსებისათვის.
Brush PunjiHatch = new
HatchBrush(HatchStyle.Horizontal, Color.Blue, Color.Yellow);
// ფუნჯის შექმნა გრადიენტული შევსებისათვის.
```

```

Brush PunjiGradient = new
LinearGradientBrush(Otxkutxedi, Color.Black, Color.Aqua, 45, false);
// ოთხკუთხედის, ელიფსისა და სექტორის შევსება ფერით.
SaxataviPurceli.FillRectangle(PunjiGradient, Otxkutxedi);
SaxataviPurceli.FillEllipse(PunjiHatch, 5, 110, 100, 100);
SaxataviPurceli.FillPie(PunjiSolid, 60, 60, 120, 120, 270, 90);
PunjiSolid.Dispose(); PunjiHatch.Dispose();
PunjiGradient.Dispose(); SaxataviPurceli.Dispose();

```



ნახ. 88. მე-20 მაგალითის შესრულების შედეგი

მაგალითი 21. გამოსახულების შემოტანა (იხ. ნახ. 89.):

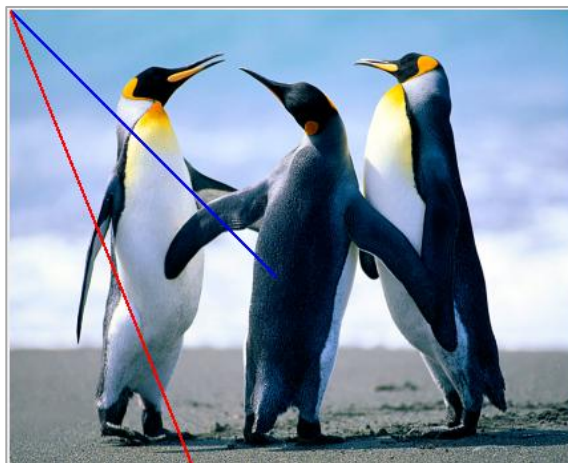
// ხატვა შესაძლებელია, აგრეთვე, pictureBox კომპონენტის შიგნით. გავავლოთ ორი ხაზი.

```

Graphics g = pictureBox1.CreateGraphics();
Pen pen1 = new Pen(Color.Blue, 2);
Pen pen2 = new Pen(Color.Red, 2);
g.DrawLine(pen1, 0, 0, 200, 200);
g.DrawLine(pen2, 0, 0, 200, 500);

```

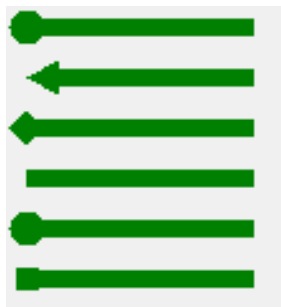
/* შეგვიძლია pictureBox კომპონენტის ფონად ავირჩიოთ რაიმე სურათი, რისთვისაც უნდა გამოვიყენოთ BackgroundImage თვისება */



ნახ. 89. 21-ე მაგალითის შესრულების შედეგი

მაგალითი 22. ისრიანი ხაზების გავლება (იხ. ნახ. 90.):

```
Graphics g = CreateGraphics();
Pen mypen = new Pen(Color.Green, 7);
mypen.StartCap = LineCap.RoundAnchor;
g.DrawLine(mypen, 10, 10, 100, 10);
mypen.StartCap = LineCap.ArrowAnchor;
g.DrawLine(mypen, 10, 30, 100, 30);
mypen.StartCap = LineCap.DiamondAnchor;
g.DrawLine(mypen, 10, 50, 100, 50);
mypen.StartCap = LineCap.Flat;
g.DrawLine(mypen, 10, 70, 100, 70);
mypen.StartCap = LineCap.RoundAnchor;
g.DrawLine(mypen, 10, 90, 100, 90);
mypen.StartCap = LineCap.SquareAnchor;
g.DrawLine(mypen, 10, 110, 100, 110);
```



ნახ. 90. 22-ე მაგალითის შესრულების შედეგი

სტრიქონების ხატვა. მეთოდი DrawString() გამოიყენება სტრიქონების დასახატად.

მისი სინტაქსია:

```
void Graphics.DrawString(string სტრიქონი, Font შრიფტი, Brush ფუნჯი, float x, float y);
```

მეთოდი SetPixel() გამოიყენება მითითებული პიქსელისთვის (მონიტორის ეკრანის წერტილისთვის) ფერის მისანიჭებლად. მისი სინტაქსია: void SetPixel(int x, int y, Color ფერი);

გამოსახულებასთან მუშაობა. გამოსახულებასთან სამუშაოდ გამოიყენება Bitmap ობიექტი. ის მუშაობს *.bmp, *.gif, *.jpg, *.png და *.tif ფაილებთან.

გადავიდეთ ამოცანების განხილვაზე,

ამოცანა 16.1. სტრუქტურა DateTime-ის კონსტრუქტორს გადაეცემა წელი, თვე

და დღე:

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
    int weli = 2007;
```

```
    int tve = 3;
```

```
    int dge = 19;
```

```
    int saati = 20;
```

```
    int wuti = 30;
```

```
    int wami = 25;
```

```
    int miliwami = 45;
```

/* იქმნება სტრუქტურა Tarigi, რომელიც შეიცავს წელს, თვეს, დღეს, საათს, წუთს, წამს და მილიწამს */

```
    DateTime Tarigi = new
```

```
    DateTime(weli,tve,dge,saati,wuti,wami,miliwami);
```

```
    label1.Text = Tarigi.Year.ToString() + " " + Tarigi.Month.ToString() + " " + Tarigi.Day.ToString() + " " + Tarigi.Hour.ToString() + " " + Tarigi.Minute.ToString() + " " + Tarigi.Second.ToString() + " " + Tarigi.Millisecond.ToString(); }
```

ამოცანა 16.2. თვისებებთან Date, Day, DayOfWeek, DayOfYear და TimeOfDay მუშაობის პროგრამული კოდი (იხ. ნახ. 91.):

```
private void button1_Click(object sender, EventArgs e)
```

```
{    DateTime Tarigi_Dro1 = DateTime.Now;
```

```
    label1.Text = Tarigi_Dro1.Date.ToString() + '\n';
```

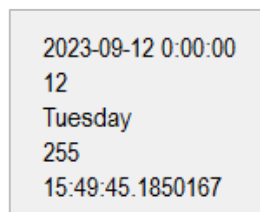
```
    label1.Text += Tarigi_Dro1.Day.ToString() + '\n';
```

```
    label1.Text += Tarigi_Dro1.DayOfWeek.ToString() + '\n';
```

```
    label1.Text += Tarigi_Dro1.DayOfYear.ToString() + '\n';
```

```
    label1.Text += Tarigi_Dro1.TimeOfDay.ToString() + '\n';
```

```
}
```



```
2023-09-12 0:00:00
12
Tuesday
255
15:49:45.1850167
```

ნახ. 91. ამოცანა 16.2.-ის შესრულების შედეგი

ამოცანა 16.3. მეთოდი DaysInMonth() გასცემს მითითებულ თვეში დღეების რაოდენობას.

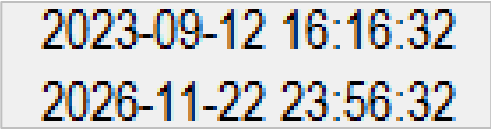
```
private void button1_Click(object sender, EventArgs e)
{
    int DgeebisRaodenoba = DateTime.DaysInMonth(2007, 2);
    label1.Text = DgeebisRaodenoba.ToString(); //28
}
```

ამოცანა 16.4. მეთოდი IsLeapYear() გასცემს true-ს თუ თვე ნაკიანია და false-ს საწინააღმდეგო შემთხვევაში.

```
private void button1_Click(object sender, EventArgs e)
{
    bool ArisNakiani = DateTime.IsLeapYear(2007);
    if (ArisNakiani == true)
        label1.Text = "წელი ნაკიანია";
    else
        label1.Text = " წელი არ არის ნაკიანი";
}
```

ამოცანა 16.5. მეთოდები: AddYears(), AddMonths(), AddDays(), AddHours() და AddMinutes(). სტრუქტურა DateTime-ის ეგზემპლარს შესაბამისად უმატებს წლებს, თვეებს, დღეებს, საათებსა და წუთებს (იხ. ნახ. 92.). ამ მეთოდებს აქვთ double ტიპის პარამეტრი:

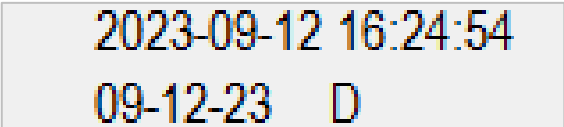
```
private void button1_Click(object sender, EventArgs e)
{
    DateTime Tarigi_Dro1 = DateTime.Now;
    label1.Text = Tarigi_Dro1.ToString() + '\n'; ;
    Tarigi_Dro1 = Tarigi_Dro1.AddYears(3);
    Tarigi_Dro1 = Tarigi_Dro1.AddMonths(2);
    Tarigi_Dro1 = Tarigi_Dro1.AddDays(10);
    Tarigi_Dro1 = Tarigi_Dro1.AddHours(7);
    Tarigi_Dro1 = Tarigi_Dro1.AddMinutes(40);
    label1.Text += Tarigi_Dro1.ToString();
}
```



ნახ. 92. ამოცანა 16.5.-ის შესრულების შედეგი

ამოცანა 16.6. მეთოდს ToString() შეგვიძლია პარამეტრად გადავცეთ სტრიქონი, რომლის მიხედვითაც შესრულდება DateTime სტრუქტურის ეგზემპლარის გარდაქმნა (იხ. ნახ. 93.).

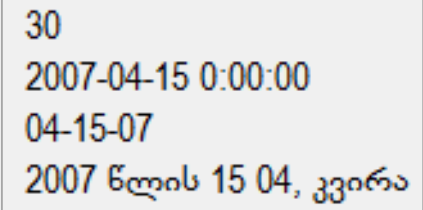
```
private void button1_Click(object sender, EventArgs e)
{
    DateTime Tarigi_Dro1 = DateTime.Now;
    label1.Text = Tarigi_Dro1.ToString() + '\n';
    label1.Text += Tarigi_Dro1.ToString("MM/dd/yy");
    label1.Text += " " + Tarigi_Dro1.ToString(" D");
}
```



ნახ. 93. ამოცანა 16.6.-ის შესრულების შედეგი

ამოცანა 16.7. პროგრამა გამოთვლის მითითებულ თვეში დღეების რაოდენობას და გამოიტანს მიმდინარე თარიღს სხვადასხვა ფორმატში (იხ. ნახ. 94.).

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    DateTime Tarigi_Dro = new DateTime(2007, 04, 15);
    int DgeebisRaodenoba = DateTime.DaysInMonth(2007, 4);
    label1.Text = DgeebisRaodenoba.ToString() + '\n';
    label1.Text += Tarigi_Dro.ToString() + '\n';
    label1.Text += Tarigi_Dro.ToString("MM/dd/yy") + '\n';
    label1.Text += Tarigi_Dro.ToString("D") + '\n';
}
```



ნახ. 94. ამოცანა 16.7.-ის შესრულების შედეგი

ამოცანა 16.8. მეთოდების Add() და Subtract() დემონსტრირება.

```
private void button1_Click(object sender, EventArgs e) {  
    label1.Text = "";  
    TimeSpan Drois_Intervali = new TimeSpan(1, 5, 10, 20);  
    DateTime Tarigi_Dro1 = new DateTime(2007, 5, 15, 17, 35, 30);  
    DateTime Tarigi_Dro2 = Tarigi_Dro1.Add(Drois_Intervali);  
        //16.05.2007 22:45:50  
    DateTime Tarigi_Dro3 = Tarigi_Dro1.Subtract(Drois_Intervali);  
        //14.05.2007 12:25:10  
    label1.Text = Tarigi_Dro2.ToString() + '\n';  
    label1.Text += Tarigi_Dro3.ToString() + '\n';  
}
```

თავდაპირველად პროგრამაში იქმნება დროის ინტერვალი: 1 დღე, 5 საათი, 10 წუთი და 20 წამი. შემდეგ Tarigi_Dro1 სტრუქტურა: 2007 წლის 15 მაისი, 17 საათი, 35 წუთი და 30 წამი. Add() მეთოდი Tarigi_Dro1 სტრუქტურას დაუმატებს Drois_Intervali დროის ინტერვალს, შედეგად Tarigi_Dro2 სტრუქტურას მიენიჭება მნიშვნელობა - 16.05.2007 22:45:50. Subtract() მეთოდი Tarigi_Dro1 სტრუქტურას გამოაკლებს Drois_Intervali დროის ინტერვალს. შედეგად, Tarigi_Dro3 სტრუქტურას მიენიჭება მნიშვნელობა - 14.05.2007 12:25:10.

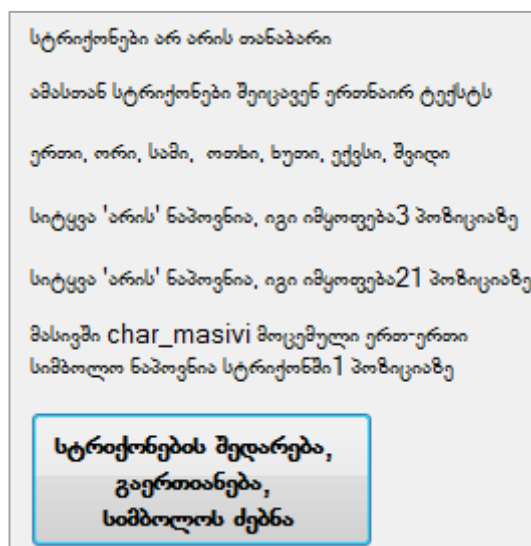
ამოცანა 16.9. კლასი String. მოცემულია სტრიქონული ტიპის ორი s1 და s2 ცვლადი, რომლებიც წინასწარ არიან განსაზღვრული. დავწეროთ პროგრამული კოდი, სადაც მოხდება სტრიქონების შედარება, გაერთიანება, სიმბოლოს ძებნა, კომპიუტერში ჩაწერილი ოპერაცილი სისტემის დადგენა. მიღებული შედეგი დავბეჭდოთ label კომპონენტში (იხ. ნახ. 95.). ფორმაზე განვალაგოთ ექვსი label და ერთი button კომპონენტი.

```
private void button1_Click(object sender, EventArgs e)  
{ //ორი სტრიქონის შედარება.  
    string s1 = "ეს არის სტრიქონი";  
    string s2 = "ეს არის სტრიქონი, ეს არის ტექსტი";  
    if (string.CompareOrdinal(s1, s2) != 0)  
        label1.Text = "სტრიქონები არ არის თანაბარი";  
        else label1.Text = "სტრიქონები თანაბარია";
```

```

if (string.Compare(s1, 0, s2, 0, 16, true) == 0)
    label2.Text = "ამასთან სტრიქონები შეიცავს ერთნაირ ტექსტს";
    //სტრიქონების გაერთიანება.
label3.Text = string.Concat("ერთი, ორი, სამი, ", " ოთხი, ხუთი,
ექვსი, შვიდი");
if (s2.IndexOf("არის") != -1)//ძებნა სტრიქონში, ქვესტრიქონის შემოსვლა.
    label4.Text = "სიტყვა 'არის' ნაპოვნია, იგი იმყოფება"
        + s2.IndexOf("არის") + " პოზიციაზე";
if (s2.LastIndexOf("არის") != -1) //ქვესტრიქონის ბოლო შემოსვლა.
    label5.Text = "სიტყვა 'არის' ნაპოვნია, იგი იმყოფება" +
        s2.LastIndexOf("არის") + " პოზიციაზე";
//მასივში მოცემული სიმბოლოს ძებნა.
char[] char_masivi = { 'ა', 'ს', 'ტ' };
if (s2.IndexOfAny(char_masivi) != -1)
    label6.Text = "მასივში char_masivi მოცემული ერთ-ერთი \სიმბოლო
ნაპოვნია სტრიქონში" + s2.IndexOfAny(char_masivi) + " პოზიციაზე";
}

```



ნახ. 95. ამოცანა 16.9.-ის შესრულების შედეგი

ამოცანა 16.10. კლასი String. მოცემულია სტრიქონული ტიპის წინასწარ განსაზღვრული ხუთი ცვლადი: str1, str2, str3, str4 და str5; დაწერეთ პროგრამული კოდი, სადაც მოხდება სტრიქონების შედარება კულტურული თავისებურებების და

რეგისტრის გათვალისწინებით. მიღებული შედეგი დავბეჭდოთ label კომპონენტში; ფორმაზე განვაღოთ თერთმეტი label და ერთი button კომპონენტები (იხ. ნახ. 96.).

```
private void button1_Click(object sender, EventArgs e)
{
    string str1 = "alpha";
    string str2 = "Alpha";
    string str3 = "Beta";
    string str4 = "alpha";
    string str5 = "alpha, beta, omega";
    int result;

    label1.Text = "სტრიქონების შედარება კულტურული თავისებურებებით";
    result = string.Compare(str1, str2, StringComparison.CurrentCulture);
    if (result < 0) label2.Text = str1 + " ნაკლებია " + str2;
        else if (result > 0) label2.Text = str1 + " მეტია " + str2;
            else label2.Text = str1 + " ტოლია " + str2;

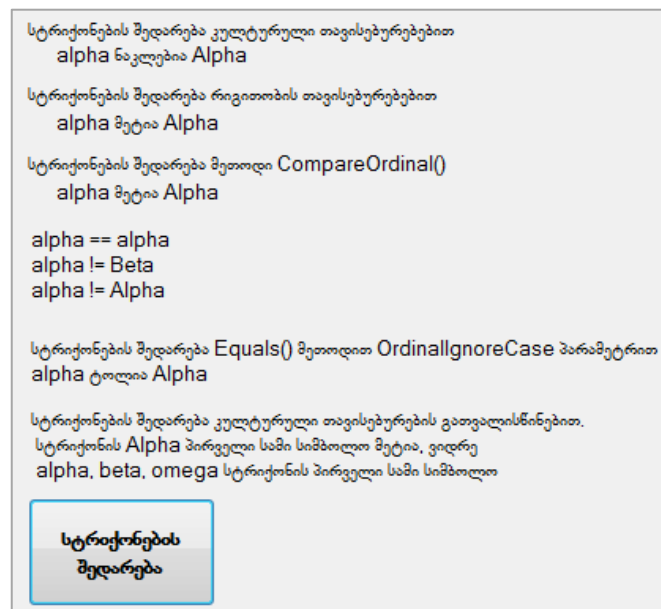
    label3.Text = "სტრიქონების შედარება რიგითობის თავისებურებებით";
    result = string.Compare(str1, str2, StringComparison.Ordinal);
    if (result < 0) label4.Text = str1 + " ნაკლებია " + str2;
        else if (result > 0) label4.Text = str1 + " მეტია " + str2;
            else label4.Text = str1 + " ტოლია " + str4;

    label5.Text = "სტრიქონების შედარება მეთოდი CompareOrdinal()";
    result = string.CompareOrdinal(str1, str2);
    if (result < 0) label6.Text = str1 + " ნაკლებია " + str2;
        else if (result > 0) label6.Text = str1 + " მეტია " + str2;
            else label6.Text = str1 + " ტოლია " + str4;
    if (str1 == str4) label7.Text = str1 + " == " + str4;
    if (str1 != str3) label8.Text = str1 + " != " + str3;
    if (str1 != str2) label9.Text = str1 + " != " + str2;
    if (String.Equals(str1, str2, StringComparison.OrdinalIgnoreCase))
    label10.Text = "სტრიქონების შედარება Equals() მეთოდით
OrdinalIgnoreCase პარამეტრით\n" + str1 + " ტოლია " + str2;
    if (String.Compare(str2, 0, str5, 0, 3, StringComparison.CurrentCulture) > 0)
```

```

{
label11.Text = "სტრიქონების შედარება კულტურული თავისებურების
გათვალისწინებით.\n " + "სტრიქონის " + str2 + " პირველი სამი
სიმბოლო მეტია, ვიდრე\n" + str5 + " სტრიქონის პირველი სამი
სიმბოლო"; }
}

```



ნახ. 96. ამოცანა 16.10.-ის შესრულების შედეგი

ამოცანა 16.11. კლასი String. მოცემულია სტრიქონული ტიპის Striqoni ცვლადი, რომლის მნიშვნელობაც მომხმარებელმა უნდა შეიყვანოს კლავიატურიდან. დაწერეთ პროგრამული კოდი, სადაც მოხდება სტრიქონის სიმბოლოების დათვლა და სიგრძის გამოთვლა. მიღებული შედეგი დავბეჭდოთ label კომპონენტში; ფორმაზე განვაღებთ ორი label, ერთი textBox და ერთი button კომპონენტები (იხ. ნახ. 97.).

```

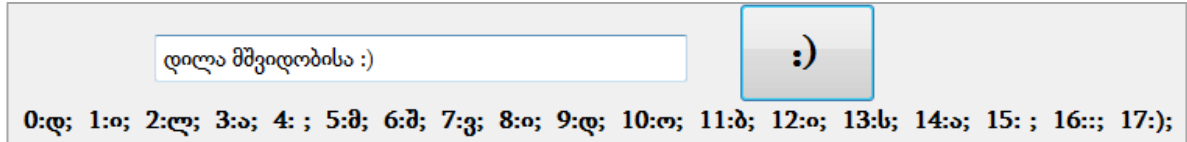
private void button1_Click(object sender, EventArgs e)
{
    string striqoni;
    char simbolo;
    int indeqsi;
    striqoni = textBox1.Text;
    label1.Text = "";
    for (indeqsi = 0; indeqsi < striqoni.Length; ++indeqsi)
    {

```

```

        simbolo = striqoni[indeqsi];
        label1.Text += indeqsi.ToString() + ":" + simbolo + "; ";
    }
}

```



ნახ. 97. ამოცანა 16.11.-ის შესრულების შედეგი

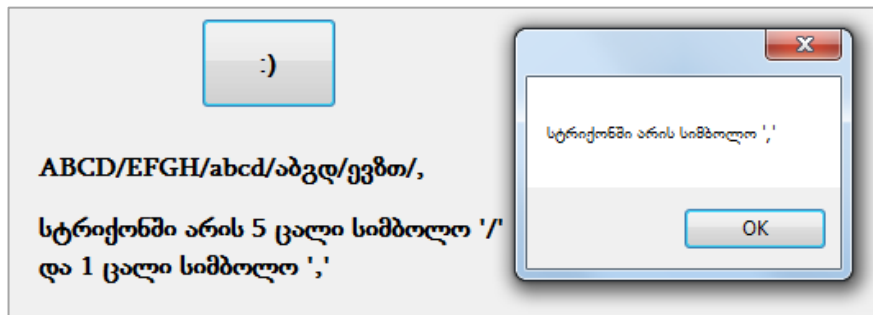
ამოცანა 16.12. კლასი String. მოცემულია სტრიქონული ტიპის ცვლადი, რომელიც წინასწარ არის განსაზღვრული. დავწეროთ პროგრამული კოდი, სადაც მოხდება მითითებული სიმბოლოს დათვლა (რაოდენობის დადგენა).

მიღებული შედეგი დავბეჭდოთ label კომპონენტში; ფორმაზე განვალაგოთ ორი label და ერთი button კომპონენტები (იხ. ნახ. 98.).

```

private void button1_Click(object sender, EventArgs e) {
    string striqoni = "ABCD/EFGH/abcd/აბგდ/ევზო/,";
    int count = 0;    int count1 = 0;
    label1.Text = striqoni;
    label2.Text = "";
    for (int indeqsi = 0; indeqsi < striqoni.Length; ++indeqsi)
    { if (striqoni[indeqsi] == '/') ++count;
      if (striqoni[indeqsi] == ',') ++count1;
    }
    label2.Text = "სტრიქონში არის " + count + " ცალი სიმბოლო '/'"
        + "\nდა " + count1 + " ცალი სიმბოლო ',' ";
    if (count > 0)
        MessageBox.Show("სტრიქონში არის სიმბოლო '/');
    else
        MessageBox.Show("სტრიქონში არ არის სიმბოლო '/');
    if (count1 > 0)
        MessageBox.Show("სტრიქონში არის სიმბოლო ','");
    else
        MessageBox.Show("სტრიქონში არ არის სიმბოლო ','");
}

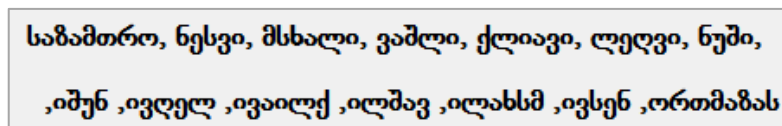
```



ნახ. 98. ამოცანა 16.12.-ის შესრულების შედეგი

ამოცანა 16.13. კლასი String. მოცემულია სტრიქონული ტიპის ცვლადი, რომელიც წინასწარ არის განსაზღვრული. დავწეროთ პროგრამული კოდი, რომელიც მოახდენს სტრიქონის შებრუნებას. მიღებული შედეგი დავბეჭდოთ label კომპონენტში; ფორმაზე განვალაგოთ ორი label და ერთი button კომპონენტები (იხ. ნახ. 99.).

```
private void button1_Click(object sender, EventArgs e){
    string striqoni= "საზამთრო, ნესვი, მსხალი, ვაშლი, ქლიავი, ლეღვი, ნუში";
    label1.Text = striqoni;
    char[] striqoni_shebruneba = striqoni.ToCharArray();
    Array.Reverse(striqoni_shebruneba);
    striqoni = new string(striqoni_shebruneba) + " ";
    label2.Text = " " + striqoni + " ";
}
```



ნახ. 99. ამოცანა 16.13.-ის შესრულების შედეგი

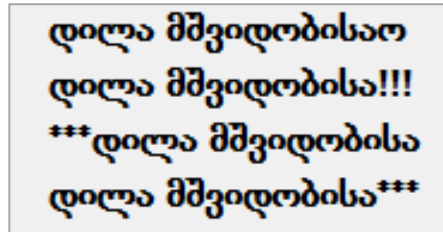
ამოცანა 16.14. კლასი String. მოცემულია სტრიქონული ტიპის წინასწარ განსაზღვრული ცვლადი. დავწეროთ პროგრამული კოდი, რომელიც მოახდენს სტრიქონში ერთი სიმბოლოს ჩამატებას სტრიქონის მარცხენა ან მარჯვენა მხარეს (იხ. ნახ. 100.).

```
private void button1_Click(object sender, EventArgs e){
    string striqoni = "დილა მშვიდობისა";
    string s1 = striqoni.PadRight(striqoni.Length + 1, 'ლ') + '\n';
    label1.Text = s1;
    string s2 = striqoni.PadRight(striqoni.Length + 3, '!') + '\n';
    label1.Text += s2;
}
```

```

string s3 = striqoni.PadLeft(striqoni.Length + 3, '*') + '\n';
label1.Text += s3;
string s4 = striqoni.PadRight(striqoni.Length + 3, '*') + '\n';
label1.Text += s4;
}

```



ნახ. 100. ამოცანა 16.14.-ის შესრულების შედეგი

ამოცანა 16.15. კლასი String. მეთოდი Substring(). მოცემულია სტრიქონული ტიპის წინასწარ განსაზღვრული მასივი. დაწერეთ პროგრამული კოდი, რომელიც მოახდენს სტრიქონული მასივიდან ქვესტრიქონის მიღებას. მიღებული შედეგი დავბეჭდოთ label კომპონენტში (იხ. ნახ. 101.).

```

private void button1_Click(object sender, EventArgs e) {
string[] striqoni = {"სახელი და გვარი: ადრიანო ჩელენტანო",
"საქმიანობა: მომღერალი, მსახიობი, კინორეჟისორი, საზოგადო მოღვაწე",
"დაბადების თარიღი: 1938 წლის 6 იანვარი",
"დაბადების ადგილი: იტალია, მილანი",
"ჯილდო: ოქროს ამბროსი (იტალიურად - Ambrogino d'oro)" };
int found = 0;
label1.Text = "*** თავდაპირველი მასივი ***\n";
foreach (string informacia in striqoni)
    label1.Text += informacia + '\n';
label1.Text += "\n*** მასივიდან მთავარი ინფორმაცია ***\n";
foreach (string informacia in striqoni)
{
    found = informacia.IndexOf(": ");
    label1.Text += informacia.Substring(found + 2) + '\n';
}
}

```

სტრიქონიდან ქვესტრიქონის მიღება:)

*** თავდაპირველი მასივი ***

სახელი და გვარი: ადრიანო ჩელენტანო

საქმიანობა: მომღერალი, მსახიობი, კინორეჟისორი, საზოგადო მოღვაწე

დაბადების თარიღი: 1938 წლის 6 იანვარი

დაბადების ადგილი: იტალია, მილანი

ჯილდო: ოქროს ამბროსი (იტალიურად - Ambrogino d'oro)

*** მასივიდან მთავარი ინფორმაცია ***

ადრიანო ჩელენტანო

მომღერალი, მსახიობი, კინორეჟისორი, საზოგადო მოღვაწე

1938 წლის 6 იანვარი

იტალია, მილანი

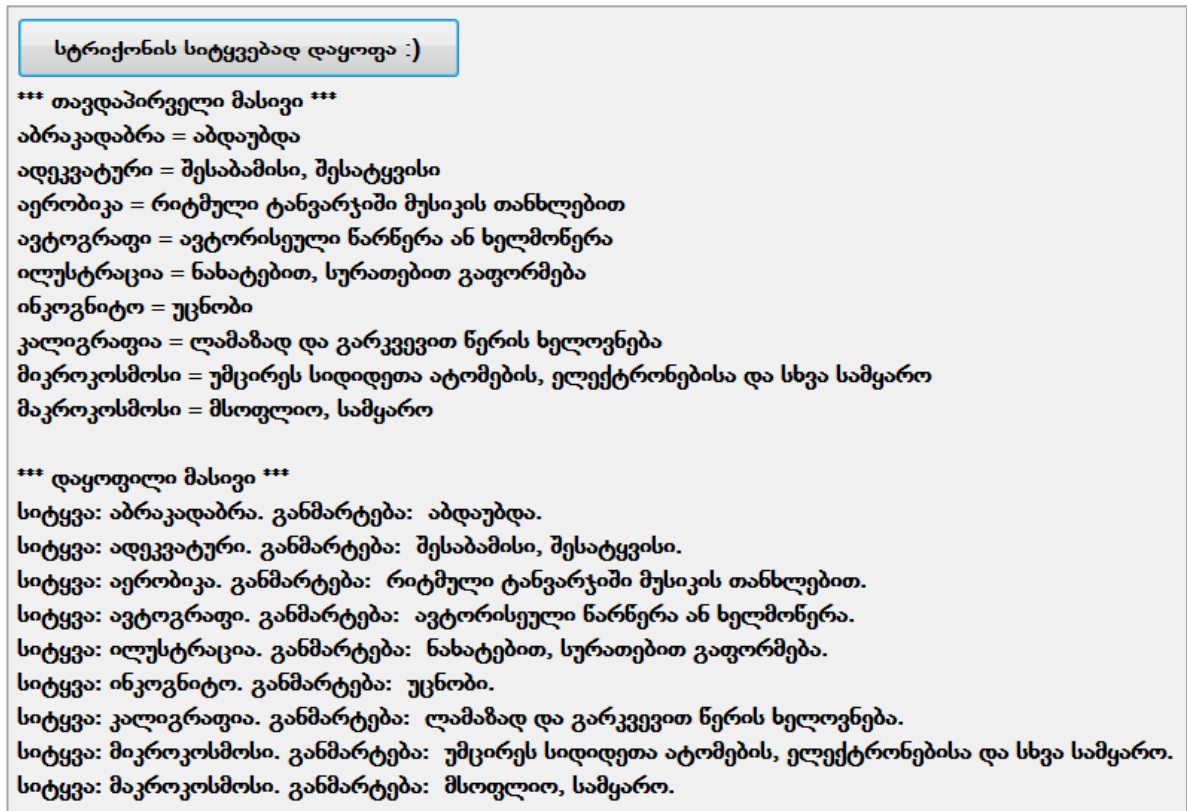
ოქროს ამბროსი (იტალიურად - Ambrogino d'oro)

ნახ. 101. ამოცანა 16.15.-ის შესრულების შედეგი

ამოცანა 16.16. კლასი String. მეთოდი Substring(). მოცემულია სტრიქონული ტიპის წინასწარ განსაზღვრული მასივი. დავწეროთ პროგრამული კოდი, რომელიც მოახდენს სტრიქონული მასივიდან ცალკეული სიტყვების მიღებას. მიღებული შედეგი დავბეჭდოთ label კომპონენტში (იხ. ნახ. 102.).

```
private void button1_Click(object sender, EventArgs e) {  
    string[] striqoni = { "აბრაკადაბრა = აბდაუბდა",  
        "ადეკვატური = შესაბამისი, შესატყვისი",  
        "აერობიკა = რიტმული ტანვარჯიში მუსიკის თანხლებით",  
        "ავტოგრაფი = ავტორისეული წარწერა ან ხელმოწერა",  
        "ილუსტრაცია = ნახატებით, სურათებით გაფორმება",  
        "ინკოგნიტო = უცნობი",  
        "კალიგრაფია = ლამაზად და გარკვევით წერის ხელოვნება",  
        "მიკროკოსმოსი = უმცირეს სიდიდეთა ატომების, ელექტრონებისა და სხვა  
სამყარო",  
        "მაკროკოსმოსი = მსოფლიო, სამყარო" };  
    int found = 0;  
    label1.Text = "*** თავდაპირველი მასივი ***\n";  
    foreach (string informacia in striqoni) label1.Text+=informacia+'\n';  
    label1.Text += "\n*** დაყოფილი მასივი ***\n";  
    foreach (var informacia in striqoni) {
```

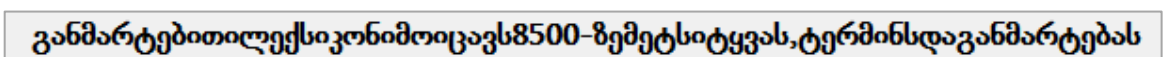
```
int pozicia = informacia.IndexOf(" = ");
if (pozicia <= 0) continue;
label1.Text += "სიტყვა: " + informacia.Substring(0, pozicia) + "." +
" განმარტება: " + informacia.Substring(pozicia + 2) + "." + '\n'; }
}
```



ნახ. 102. ამოცანა 16.16.-ის შესრულების შედეგი

ამოცანა 16.17. კლასი String. მეთოდი Split(). მოცემულია სტრიქონული ტიპის ცვლადი, რომელიც წინასწარ არის განსაზღვრული. სტრიქონში წავშალოთ გამოტოვებები. მიღებული შედეგი დავბეჭდოთ label კომპონენტში (იხ. ნახ. 103.).

```
private void button1_Click(object sender, EventArgs e){
label1.Text = "";
string striqoni = "განმარტებითი ლექსიკონი მოიცავს 8500-ზე მეტ " +
"სიტყვას, ტერმინს და განმარტებას";
string[] shedegi = striqoni.Split(new char[0],
StringSplitOptions.RemoveEmptyEntries);
foreach (string informacia in shedegi) label1.Text += informacia;
}
```

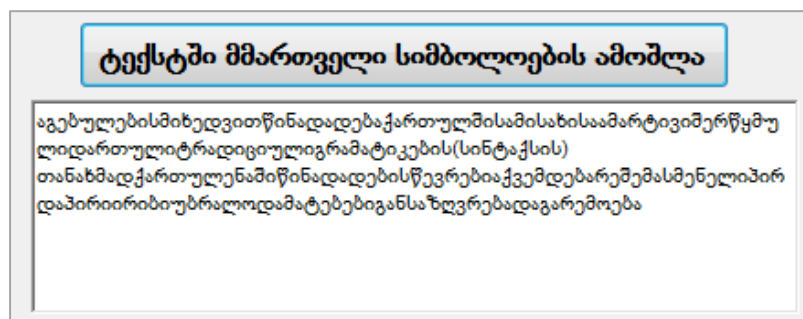


ნახ. 103. ამოცანა 16.17.-ის შესრულების შედეგი

ამოცანა 16.18. კლასი String. მეთოდი Split(). მოცემულია სტრიქონული ტიპის ცვლადი, რომელიც წინასწარ არის განსაზღვრული. სტრიქონში წავშალოთ მმართველი სიმბოლოები: გამოტოვება, მძიმე, წერტილი, ორი წერტილი და ჰორიზონტალური ტაბულაცია (' ', ',', '.', ':', '\t').

პროგრამის გამარტივების მიზნით, მმართველი სიმბოლოები აღვწეროთ, როგორც მასივი. მიღებული შედეგი დავბეჭდოთ richTextBox1 კომპონენტში (იხ. ნახ. 104.).

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    string striqoni = "აგებულების მიხედვით წინადადება ქართულში სამი
სახისაა: მარტივი, შერწყმული და რთული." + "ტრადიციული
გრამატიკების(სინტაქსის) თანახმად, ქართულ ენაში წინადადების წევრებია:"
+ "\tქვემდებარე, შემასმენელი, პირდაპირი, ირიბი, უბრალო დამატებები,
განსაზღვრება და გარემოება.";
    string[] shedegi = striqoni.Split(new char[]{' ', '.', ',', ':', '\t'});
    foreach (string informacia in shedegi)
    if (informacia.Trim() != "") richTextBox1.Text += informacia.ToString();
}
```



ნახ. 104. ამოცანა 16.18.-ის შესრულების შედეგი

ამოცანა 16.19. კლასი String. მეთოდი Split(). მოცემულია სტრიქონული ტიპის ცვლადი, რომელიც წინასწარ არის განსაზღვრული. შევცვალოთ ერთი სიმბოლო მეორე სიმბოლოთი. მიღებული შედეგი დავბეჭდოთ label კომპონენტში (იხ. ნახ. 105.).

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    string striqoni = "An example of a string is 'Welcome to the World
of C# Programming!'";
    striqoni = striqoni.Replace('o', 'a');
    richTextBox1.Text = striqoni.ToString(); }
}
```

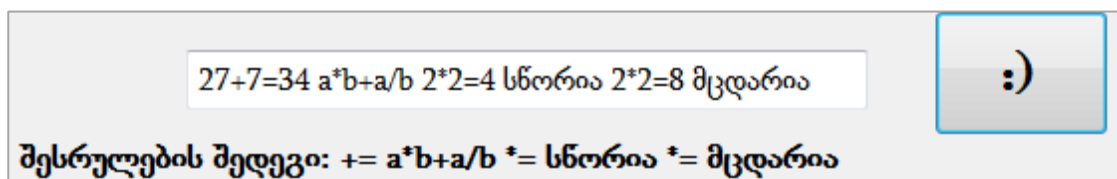

**ტექსტში ერთი სიმბოლოს
შეცვლა მეორეთი**

An example of a string is 'Welcome to the World of C# Programming!'

ნახ. 105. ამოცანა 16.19.-ის შესრულების შედეგი

ამოცანა 16.20. კლასი String. მოცემულია სტრიქონული ტიპის ცვლადი, რომლის მნიშვნელობაც მომხმარებელმა უნდა შეიყვანოს კლავიატურიდან (textBox კომპონენტი). დაწეროთ პროგრამული კოდი, სადაც მოხდება სტრიქონიდან სიმბოლო-ციფრის წაშლა. მიღებული შედეგი დავბეჭდოთ label კომპონენტში (იხ. ნახ. 106.).

```
private void button1_Click(object sender, EventArgs e){
    string striqoni;
    striqoni = textBox1.Text;
    string striqoni1 = "";
    label1.Text = "";
    for (int indeqsi = 0; indeqsi < striqoni.Length; ++indeqsi) {
        if (!char.IsDigit(striqoni[indeqsi])) {
            striqoni1 += striqoni[indeqsi]; }
        label1.Text = "შესრულების შედეგი: " + striqoni1;
    }
}
```



ნახ. 106. ამოცანა 16.20.-ის შესრულების შედეგი

ამოცანა 16.21. კლასი String. მოცემულია სტრიქონული ტიპის ცვლადი, რომელიც წინასწარ არის განსაზღვრული.

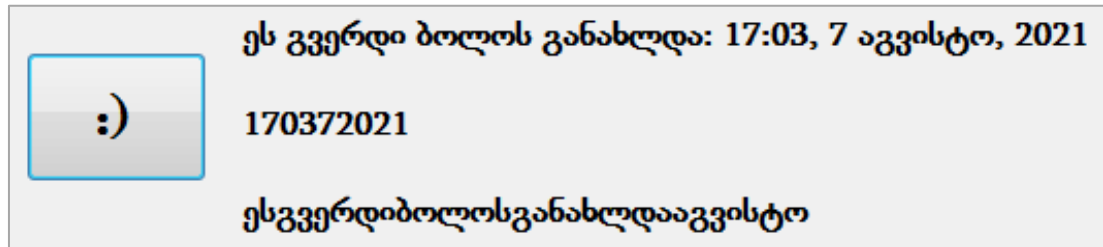
დაწეროთ პროგრამული კოდი, სადაც მოხდება სტრიქონიდან სიმბოლო-ციფრის წაშლა. მიღებული შედეგი დავბეჭდოთ label კომპონენტში (იხ. ნახ. 107.).

```
private void button1_Click(object sender, EventArgs e){
    string striqoni="ეს გვერდი ბოლოს განახლდა:17:03, 7 აგვისტო, 2021";
    label1.Text = striqoni;
    label2.Text = "";
```

```

label3.Text = "";
foreach (char informacia in striqoni) {
    if (char.IsDigit(informacia)) label2.Text += informacia;
    if (char.IsLetter(informacia)) label3.Text += informacia;
}
}

```



ნახ. 107. ამოცანა 16.21.-ის შესრულების შედეგი

ამოცანა 16.22. კლასი String. მოცემულია სტრიქონული ტიპის ცვლადი, რომელიც წინასწარ არის განსაზღვრული.

დავწეროთ პროგრამული კოდი, რომელიც მოცემულ სტრიქონში დაითვლის ბეჭდურ (ზედა რეგისტრის) სიმბოლოებს (იხ. ნახ. 108.).

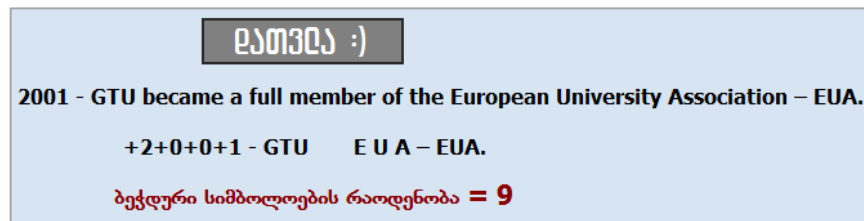
```

private void button1_Click(object sender, EventArgs e) {
    string str = "GTU was awarded accreditation by the National Center"
                + "for Educational Accreditation.";

    label1.Text = str;
    label2.Text = "";
    label3.Text = "";
    int count = 0;
    for (int i = 0; i < str.Length; i++)
    {
        if (str[i] >= 'A' && str[i] <= 'Z') count++;
        else
            if (str[i] >= 'a' && str[i] <= 'z')
                str = str.Remove(i--, 1);
            else
                if (str[i] >= '0' && str[i] <= '9')
                    str = str.Insert(i++, "+");
    }
    label2.Text = str;
}

```

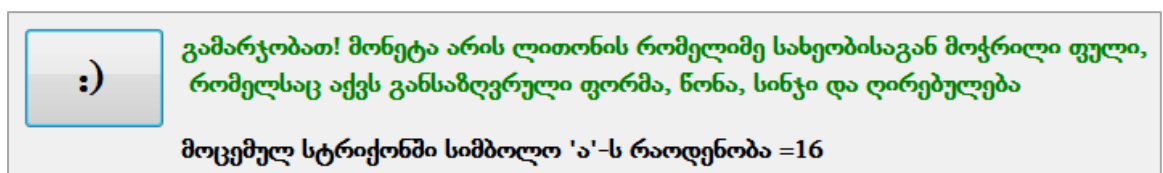
```
label3.Text="ბეჭდური სიმბოლოების რაოდენობა = " + count.ToString();
}
```



ნახ. 108. ამოცანა 16.22.-ის შესრულების შედეგი

ამოცანა 16.23. კლასი String. მოცემულია სტრიქონული ტიპის ცვლადი, რომელიც წინასწარ არის განსაზღვრული. დავწეროთ პროგრამული კოდი, რომელიც მოცემულ სტრიქონში დაითვლის სიმბოლო „ა“-ს რაოდენობას (იხ. ნახ. 109.).

```
private void button1_Click(object sender, EventArgs e){
    string str = "გამარჯობა! მონეტა არის ლითონის რომელიმე სახეობისაგან
    მოჭრილი ფული,\n" + " რომელსაც აქვს განსაზღვრული ფორმა, წონა,
    სინჯი და ღირებულება";
    label1.Text = str;
    int count = 0;
    for (int i = 0; i < str.Length; i++)
    {
        if (str[i] == 'ა')
        {
            count++;
        }
    }
    label2.Text = "მოცემულ სტრიქონში სიმბოლო 'ა'-ს რაოდენობა =" +
        count.ToString();
}
```

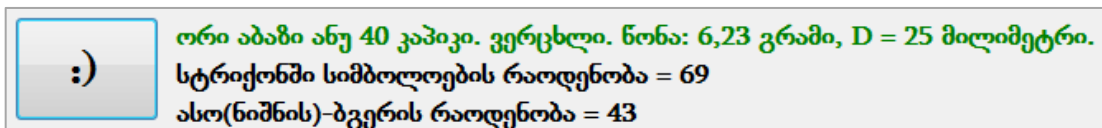


ნახ. 109. ამოცანა 16.23.-ის შესრულების შედეგი

ამოცანა 16.24. კლასი String. მოცემულია სტრიქონული ტიპის ცვლადი, რომელიც წინასწარ არის განსაზღვრული. დავწეროთ პროგრამული კოდი,

რომელიც დაადგენს მოცემული სტრიქონის სიგრძეს და დაითვლის სტრიქონში ასო-ბგერის რაოდენობას (იხ. ნახ. 110.).

```
private void button1_Click(object sender, EventArgs e)
{
    label2.Text = ""; label3.Text = "";
    string MyString =
        "ორი აბაზი ანუ 40 კაპიკი. ვერცხლი. წონა: 6,23 გრამი, D = 25 მილიმეტრი.";
    label1.Text=MyString;
    label2.Text="სტრიქონში სიმბოლოების რაოდენობა = "+MyString.Length;
    char[] LettersArray = MyString.ToCharArray();
    int count = 0;
    for (int i = 0; i < MyString.Length; ++i)
        if (Char.IsLetter(LettersArray[i])) ++count;
    label3.Text = "ასო(ნიშნის)-ბგერის რაოდენობა = " + count.ToString();
}
```



ნახ. 110. ამოცანა 16.24.-ის შესრულების შედეგი

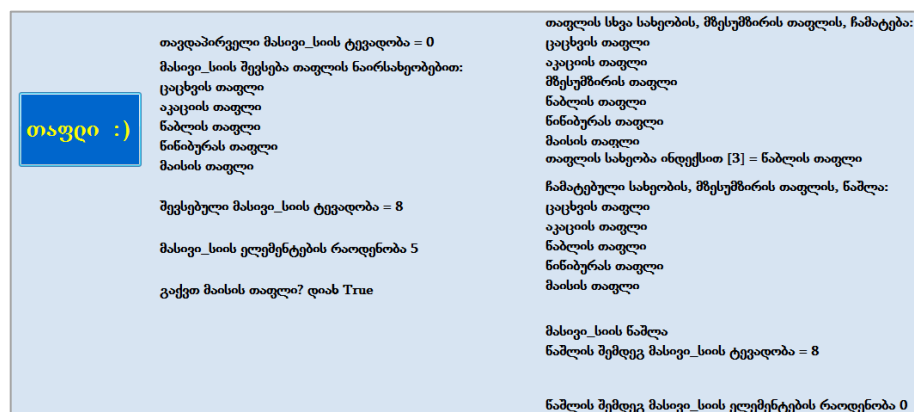
ამოცანა 16.25. მასივი_სია (იხ. ნახ. 111.).

```
private void button1_Click(object sender, EventArgs e){
    label2.Text = ""; label6.Text = "";
    List<string> tafli = new List<string>();
    label1.Text="\ითავდაპირველი მასივი_სიის ტევადობა= "+tafli.Capacity;
    label2.Text = "მასივი_სიის შევსება თაფლის ნაირსახეობებით:\n";
    tafli.Add("ცაცხვის თაფლი");
    tafli.Add("აკაციის თაფლი");
    tafli.Add("წაბლის თაფლი");
    tafli.Add("წიწიბურას თაფლი");
    tafli.Add("მაისის თაფლი");
    foreach (string tafli1 in tafli)
        { label2.Text += tafli1 + '\n'; }
    label3.Text = "\nშევსებული მასივი_სიის ტევადობა = "+ tafli.Capacity;
```

```

label4.Text = "\მასივი_სიის ელემენტების რაოდენობა " + tafli.Count;
bool element;
for (int i = 0; i < tafli.Count; i++) {
    if (tafli[i].Contains("მაისი"))
    { element = true;
      label5.Text = "\იგაქვთ მაისის თაფლი? დიახ " + element; }
    else
    { element = false;
      label5.Text = "\იგაქვთ მაისის თაფლი? არა " + element; }
}
label6.Text="თაფლის სხვა სახეობის, მზესუმზირის თაფლის, ჩამატება: \n";
tafli.Insert(2, "მზესუმზირის თაფლი");
foreach (string tafli2 in tafli)
{ label6.Text += tafli2 + '\n'; }
label7.Text = "თაფლის სახეობა ინდექსით [3] = " + tafli[3];
label8.Text = "ჩამატებული სახეობის, მზესუმზირის თაფლის, წაშლა: \n";
tafli.Remove("მზესუმზირის თაფლი");
foreach (string tafli3 in tafli)
{ label8.Text += tafli3 + '\n'; }
tafli.Clear();
label9.Text = "\მასივი_სიის წაშლა";
label9.Text+="\იწაშლის შემდეგ მასივი_სიის ტევადობა = " + tafli.Capacity;
label10.Text="\იწაშლის შემდეგ მასივი_სიის ელემენტების რაოდენობა " +
tafli.Count;
}

```



ნახ. 111. ამოცანა 16.25.-ის შესრულების შედეგი

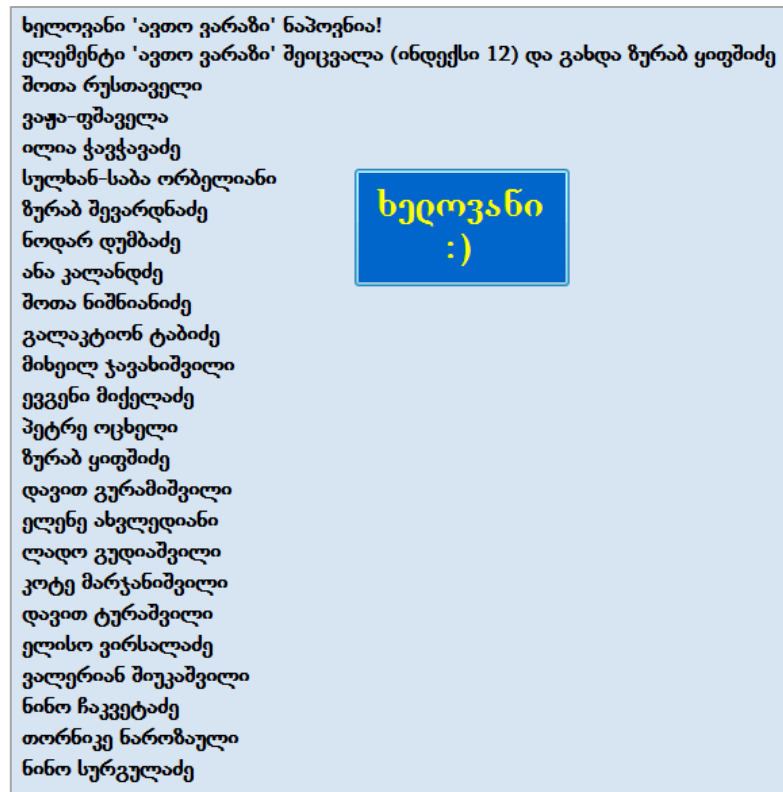
ამოცანა 16.26. კიდევ ერთი მაგალითი მასივი_სიის გამოყენებით (იხ. ნახ. 112.).

```
private void button1_Click(object sender, EventArgs e) {
    label1.Text = ""; label2.Text = "";
    List<string> xelovaniList = new List<string>();
    xelovaniList.Add("შოთა რუსთაველი");
    xelovaniList.Add("ვაჟა-ფშაველა");
    xelovaniList.Add("ილია ჭავჭავაძე");
    xelovaniList.Add("სულხან-საბა ორბელიანი");
    xelovaniList.Add("ზურაბ შევარდნაძე");
    xelovaniList.Add("ნოდარ დუმბაძე");
    xelovaniList.Add("ანა კალანდძე");
    xelovaniList.Add("შოთა ნიშნიანძე");
    xelovaniList.Add("გალაკტიონ ტაბიძე");
    xelovaniList.Add("მიხეილ ჯავახიშვილი");
    xelovaniList.Add("ევგენი მიქელაძე");
    xelovaniList.Add("პეტრე ოცხელი");
    xelovaniList.Add("ავთო ვარაზი");
    xelovaniList.Add("დავით გურამიშვილი");
    xelovaniList.Add("ელენე ახვლედიანი");
    xelovaniList.Add("ლადო გუდიაშვილი");
    xelovaniList.Add("კოტე მარჯანიშვილი");
    xelovaniList.Add("დავით ტურაშვილი");
    xelovaniList.Add("ელისო ვირსალაძე");
    xelovaniList.Add("ვალერიან შიუკაშვილი");
    xelovaniList.Add("ნინო ჩაკვეტაძე");
    xelovaniList.Add("თორნიკე ნაროზაული");
    xelovaniList.Add("ნინო სურგულაძე");
    if (xelovaniList.Contains("ავთო ვარაზი")) // მოვებზნოთ ელემენტი.
    { label1.Text = "\nიხელოვანი 'ავთო ვარაზი' ნაპოვნია!"; }
    // მოვებზნოთ მასივის ელემენტის ინდექსი და შევცვალოთ ახალი ელემენტით.
    int idx = xelovaniList.IndexOf("ავთო ვარაზი");
```

```

if (idx >= 0) { xelovaniList[idx] = "ზურაბ ყიფშიძე"; }
label2.Text = "ელემენტი 'ავთო ვარაზი' შეიცვალა (ინდექსი " + idx +
") და გახდა " + xelovaniList[idx];
foreach (var author in xelovaniList)
{
    label2.Text += '\n' + author;
}
}

```



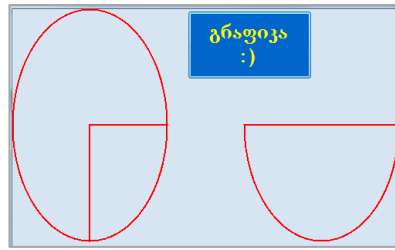
ნახ. 112. ამოცანა 16.26.-ის შესრულების შედეგი

ამოცანა 16.27. ელიფსის, სექტორის დახატვა (იხ. ნახ. 113.).

```

private void button1_Click(object sender, EventArgs e){
    Graphics SaxataviPurceli = CreateGraphics();
    Pen Kalami = new Pen(Color.Red, 2);
    Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
    SaxataviPurceli.DrawEllipse(Kalami, Otxkutxedi);
    SaxataviPurceli.DrawPie(Kalami, Otxkutxedi, 0, 90);
    Rectangle seqtori = new Rectangle(300, 0, 200, 300);
    SaxataviPurceli.DrawPie(Kalami, seqtori, 0, 180);
}

```



ნახ. 113. ამოცანა 16.27.-ის შესრულების შედეგი

ამოცანა 16.28. პროგრამაში ხდება სიბრტყეზე ხაზისა და ცილინდრის გამოსახულების ასახვა SetPixel() მეთოდის გამოყენებით (იხ. ნახ. 114.).

```
private void button1_Click(object sender, EventArgs e) {
    Pen pen = new Pen(Brushes.Yellow);
    Graphics SaxataviPurceli = CreateGraphics();
    SaxataviPurceli.DrawEllipse(pen, 10, 10, 80, 30);
    SaxataviPurceli.FillEllipse(Brushes.Yellow, 10, 50, 80, 30);
    SaxataviPurceli.DrawLine(pen, 10, 25, 10, 65);
    SaxataviPurceli.DrawLine(pen, 90, 25, 90, 65);
}

private void button2_Click(object sender, EventArgs e) {
    Graphics SaxataviPurceli = CreateGraphics();
    Bitmap Gamosaxuleba1 = new Bitmap(90, 90); // სიგანე = სიმაღლე = 90
    // ოთხკუთხედის ხატვა
    for (int x = 0; x < Gamosaxuleba1.Height; x++)
        for (int y = 0; y < Gamosaxuleba1.Width; y++)
            Gamosaxuleba1.SetPixel(x, y, Color.Yellow);
    // ლურჯი ხაზის ხატვა
    for (int x = 0; x < Gamosaxuleba1.Height; x++)
        Gamosaxuleba1.SetPixel(x, x, Color.Blue);
    // გამოსახულების გამოტანა
    SaxataviPurceli.DrawImage(Gamosaxuleba1, 300, 0, 100, 100);
}
```



ნახ. 114. ამოცანა 16.28.-ის შესრულების შედეგი

ამოცანა 11.29. პროგრამაში ხდება გამოსახულების ასახვა დიალოგურ ფანჯრაში არჩეული ფაილიდან (იხ. ნახ. 115.)

```
private void button1_Click(object sender, EventArgs e) {  
    Graphics SaxataviPurceli = CreateGraphics();  
    OpenFileDialog Surati_Failidan = new OpenFileDialog();  
    Bitmap Gamosaxuleba;  
    if (Surati_Failidan.ShowDialog() == DialogResult.OK) {  
        Gamosaxuleba = new Bitmap(Surati_Failidan.FileName);  
        SaxataviPurceli.DrawImage(Gamosaxuleba, 5, 5, 270, 250);  
    }  
}
```



ნახ. 115. ამოცანა 16.28.-ის შესრულების შედეგი

დამოუკიდებელი სამუშაო

1. მოცემულია სტრიქონი „საქართველო“. მიიღეთ სტრიქონი „რთველი“
2. მოცემულია სტრიქონი „საქართველოს ტექნიკური უნივერსიტეტი“. მიიღეთ სტრიქონი „მე ვსწავლობ ტექნიკურ უნივერსიტეტში“
3. მოცემულია სტრიქონი „ობიექტზე ორიენტირებული დაპროგრამება“. დაითვალოთ სიმბოლოების რაოდენობა.
4. მოცემულია სტრიქონი: „ეს არის ტესტი“. სტრიქონში წაშალოთ 6 სიმბოლო დაწყებული მე-2 პოზიციიდან.
5. მოცემულია სტრიქონი: „ASP.NET-ის პირველი ვერსია გამოვიდა 2002 წელს .NET Framework 1.0 ვერსიასთან ერთად“. დაწერე კოდი, რომელიც მოცემულ სტრიქონში ყველა ციფრს შეცვლის „რ“ სიმბოლოთი.
6. რა შედეგი დაიბეჭდება მოცემული კოდის შესრულების შემდეგ:

ა. 168; ბ. 123; გ. 12345;

```
string s = "123", s1 = "45";
```

```
s = String.Concat(s, s1);
```

7. რა შედეგი დაიბეჭდება მოცემული კოდის შესრულების შემდეგ:

ა. ორი; ბ. ერთი ორი სამი;

```
string[] x = { "ერთი ", "ორი ", "სამი " };
```

```
string s = String.Concat(x);
```

8. რა შედეგი დაიბეჭდება მოცემული კოდის შესრულების შემდეგ:

ა. გრამი; ბ. ენაა; გ. C#;

```
string s = "C# დაპროგრამების ენაა";
```

```
string s2 = s.Substring(0, 3);
```

9. რა შედეგი დაიბეჭდება მოცემული კოდის შესრულების შემდეგ:

ა. 2; ბ. 12AB345; გ. AB;

```
string s = "12345";
```

```
string s2 = s.Insert(2, "AB");
```

10. რომელი მეთოდის გამოყენება ხდება ქვემოთ მოცემულ კოდში? რა დაიბეჭდება პროგრამის შესრულების შედეგად და რატომ. პროგრამას ახსნა-განმარტება დაურთეთ კომენტარის სახით:

```
string text = "მზიანი დღეა";
```

```
text = text.Replace("მზიანი", "ნისლიანი");
```

```
label1.Text = text;
```

```
text = text.Replace("ი", "");
```

```
label2.Text = text;
```

11. რა შედეგი დაიბეჭდება მოცემული კოდის შესრულების შემდეგ:

ა. 123; ბ. 129; გ. 123333;

```
string s = "123429";
```

```
int i = 2, k = 3;
```

```
string s1 = s.Remove(i, k);
```

12. რა შედეგი დაიბეჭდება მოცემული კოდის შესრულების შემდეგ და რატომ? განმარტეთ თითოეული სტრიქონი და კომენტარის სახით პროგრამულ კოდს დაუმატეთ.

```

string s1 = "hello world";
char ch = 'o';
int indexOfChar = s1.IndexOf(ch);
label1.Text = indexOfChar.ToString();
string subString = "wor";
int indexOfSubstring = s1.IndexOf(subString);
label2.Text = indexOfSubstring.ToString();

```

13. რა შედეგი დაიბეჭდება label1 კომპონენტში მოცემული კოდის შესრულების შემდეგ: 14? 18? 13?

```

string striqoni1 = "ცაცხვის \n თაფლი \n";
label1.Text = striqoni1.Length.ToString() + '\n';

```

14. რა იქნება str1 ცვლადის მნიშვნელობა მოცემული კოდის შესრულების შემდეგ:

```

string str1, str2 = "ფუტკრის თაფლი", str3 = "ფიჭიანი ან გამოწურული";
str1 = str2.Insert(7, str3);
label1.Text = str1;

```

15. რა იქნება str1 ცვლადის მნიშვნელობა მოცემული კოდის შესრულების შემდეგ?

```

string str1, str2 = "ფუტკრის ფიჭიანი ან გამოწურული თაფლი";
str1 = str2.Remove(16, 14);
label1.Text = str1;

```

16. რა იქნება index1 ცვლადის მნიშვნელობა მოცემული კოდის შესრულების შემდეგ? 11? 12 თუ 13?

```

char[] simboloebis_masivi = { 'ნ', 'მ' };
string striqoni1 = "ფუტკრის ფიჭიანი ან გამოწურული თაფლი";
int index1 = striqoni1.IndexOfAny(simboloebis_masivi);
label1.Text = index1.ToString();

```

17. რა იქნება index1 ცვლადის მნიშვნელობა მოცემული კოდის შესრულების შემდეგ: 11? 21 თუ 15?

```

char[] simboloebis_masivi = { 'ნ', 'მ' };
string striqoni1 = "ფუტკრის ფიჭიანი ან გამოწურული თაფლი";

```

```
int index1 = stripon1.LastIndexOfAny(simboloebis_masivi);
```

```
label1.Text = index1.ToString();
```

18. მოცემულია სტრიქონი:

```
string str = "მეცნიერთა აზრით, თავლი აუმჯობესებს მეხსიერებას და  
ამშვიდებს ნერვებს";
```

დაწერეთ პროგრამული კოდი, რომელის შესრულების შემდეგ მოცემული სტრიქონიდან მივიღებთ ახალ სტრიქონს: "თავლი აუმჯობესებს მეხსიერებას".

19. მოცემულია სტრიქონი:

```
string str= "გოზინაყის მომზადებისას თავლი სასარგებლო თვისებებს კარგავს";
```

დაწერეთ პროგრამული კოდი, რომელიც

ა) დაითვლის სტრიქონის სიმბოლოების რაოდენობას;

ბ) დაითვლის სტრიქონში 'ა' სიმბოლოს რაოდენობას;

20. მოცემულია სტრიქონი:

```
string str1 = "ცაცხვის თავლი ძალიან სურნელოვანია";
```

მოცემულ სტრიქონში მე-14 პოზიციიდან წაშაღეთ 7 სიმბოლო. მიღებული შედეგი დაბეჭდეთ **label** კომპონენტში.

21. მოცემულია სტრიქონი:

```
string surati = "ტილოზე დაბეჭდილი სურათები. ზომა 140/85-120 ლარი, ზომა  
125/60-100 ლარი";
```

მოცემულ სტრიქონში ყველა ციფრი შეცვაღეთ 'x' სიმბოლოთი. მიღებული შედეგი დაბეჭდეთ **label** კომპონენტში.

22. მოცემულია string ტიპის მასივი:

```
string [] tafli = { "თავლი წარმოადგენს ნატურალურ პროდუქტს",  
"რომლის შემადგენლობაშიც შედის მრავალრიცხოვანი ფერმენტები",  
"მიკროელემენტები, ვიტამინები და სხვა სასარგებლო ნივთიერებები" };
```

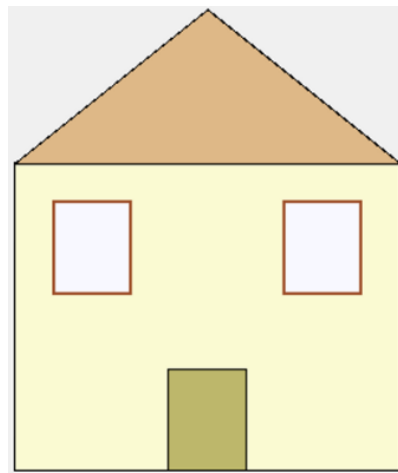
დაწერეთ პროგრამული კოდი, რომელიც მასივში დაითვლის ხმოვნების (ა, ი, ო, უ, ე) რაოდენობას. მიღებული შედეგი დაბეჭდეთ **label** კომპონენტში.

23. მოცემულია შემდეგი სახის ტექსტი: „ლაპარაკისა და დაწერილის ხმამალა კითხვისას თხრობითი წინადადების ბოლოს ხმას ვუდაბლებთ და ვჭერდებით. წერის დროს თხრობითი წინადადების ბოლოს წერტილს ვწერთ.

ქართული ენა თავისი ხასიათით თხრობითი ფორმისაა.“ დაწეროთ პროგრამული კოდი, სადაც სიტყვა „თხრობითი“ შევცვალოთ სიტყვით „მარტივი“.

24. მოცემულია სტრიქონი, რომელიც შეიცავს მომხმარებელთა სახელებს, გამოყოფილს მძიმით: "Login1, LOgin2, login3, loGin4". მოცემული სტრიქონი დავყოთ ქვესტრიქონების მასივად (თითოეული მომხმარებლის სახელი უნდა იყოს ცალ-ცალკე) და ზედა რეგისტრის სიმბოლოები გადავიყვანოთ ქვედა რეგისტრში და პირიქით.

25. დაწერეთ ისეთი პროგრამული კოდი რომლის მიხედვითაც დაიხატება სახლი და გეომეტრიული ფიგურები ქვემოთ მოცემული ნიმუშის მიხედვით ან სხვა სახის გამოსახულება საკუთარი შეხედულებისამებრ (იხ. ნახ. 116.).



ნახ. 116. გეომეტრიული ფიგურის ნიმუში.

გამოსახულება C#.net ენაზე შესრულებულია სტუ იმს ფაკულტეტის N108732

ჯგუფის კურსდამთავრებულის ვასილ ცინაძის მიერ

26. ქვემოთ მოცემულ ცხრილში აღწერილია კლასი String-ის მეთოდები. დააკვირდით თითოეულ პროგრამულ კოდს და „შედეგის“ ველში ჩაწერეთ შესრულების შედეგი და მოკლე ახსნა-განმარტება (იხ. ცხრილი 15).

ცხრილი 15. კლასი String-ის მეთოდების განხილვა

მეთოდი	პროგრამული კოდი	შედეგი
Concat	<pre>string s1 = "hello"; string s2 = "world"; string s3 = s1 + " " + s2; string s4 = String.Concat(s3, "!!!"); label1.Text = s4;</pre>	

Join	<pre> string s5 = "apple"; string s6 = "a day"; string s7 = "keeps"; string s8 = "a doctor"; string s9 = "away"; string[] values = new string [] { s5, s6, s7, s8, s9 }; String s10 = String.Join(" ", values); label1.Text = s10; </pre>	
Compare	<pre> string s1 = "hello"; string s2 = "world"; int result = String.Compare(s1, s2); if (result < 0) { label1.Text="s1 სტრიქონი იმყოფება s2 სტრიქონის წინ"; } else if (result > 0) { label1.Text="s1 სტრიქონი იმყოფება s2 სტრიქონის შემდეგ"; } else { label1.Text ="s1 და s2 სტრიქონები თანაბარია"; } //ლათინური ანბანის მიხედვით h-ს //მოსდევს w </pre>	
IndexOf IndexOfAny y LastIndexOf Of LastIndexOfAny	<pre> string s = "It is false"; int i = s.IndexOf('f'); label1.Text=i.ToString(); string s1 = "do { for("; char[] x = { '{', '(' }; int i1 = s1.IndexOfAny(x); label2.Text = i1.ToString(); string sss = "It is false for"; int i2 = sss.LastIndexOf('f'); label3.Text = i2.ToString(); string s2 = "do{ for("; </pre>	

	<pre>char[] x2 = { '{', '(' }; int iii = s2.LastIndexOfAny(x2); label4.Text = iii.ToString();</pre>	
Split	<pre>string text = "დილა მშვიდობისა, საქართველოვ!"; string [] words = text.Split(new char[]{' '}, StringSplitOptions.RemoveEmptyEntries); foreach (string s in words) label1.Text += s;</pre>	
Trim	<pre>string text = " hello world "; text = text.Trim(); text = text.Trim(new char[] { 'd', 'h' }); label1.Text = text;</pre>	
Substring	<pre>string text = "დილა მშვიდობისა, საქართველოვ!"; text = text.Substring(4); label1.Text = text; text = text.Substring(0, text.Length - 11); label2.Text = text;</pre>	
Remove	<pre>string text = "დილა მშვიდობისა, საქართველოვ!!!"; int ind = text.Length-3; text = text.Remove(ind); label1.Text = text; text = text.Remove(0, 5); label2.Text = text;</pre>	
Insert	<pre>string text = "მზიანი დღეა "; string subString = "მშვენიერი "; text = text.Insert(0, subString); label1.Text = text;</pre>	
Replace	<pre>string s = "123422"; string s1 = s.Replace('2', '5'); label1.Text=s1;</pre>	

	<pre> string s2 = "abc abc"; string s3 = s2.Replace("abc", "alfa"); label2.Text = s3; string s4 = "child_child_child"; string s5 = s4.Replace("child", "children"); label3.Text = s5; </pre>	
ToLower ToUpper	<pre> string hello = "Hello world!"; // hello world! label1.Text = hello.ToLower(); // HELLO WORLD! label2.Text = hello.ToUpper(); </pre>	

17

Windows პროგრამირება. მართვის ვიზუალური და არავიზუალური ელემენტები: Button, Label, TextBox, RadioButton, CheckBox, ListBox და ComboBox, TabControl, dateTimePicker, Web-Browser, Timer და სხვა მართვის ელემენტები.

Windows პროგრამირება. კომპანია Microsoft-მა 1990 წლების ბოლოს შეიმუშავა C# ენა. C# ენის ავტორია დანიელი პროგრამისტი ანდერს ჰელსბერგი. დაპროგრამების C# ენა მჭიდროდაა დაკავშირებული C, C++ და Java ენებთან. ის აგებულია C++ ენაში განსაზღვრულ ობიექტურ მოდელზე, C ენიდან აღებულია სინტაქსი, ოპერატორები და საკვანძო სიტყვები, Java-დან კი აღებულია შუალედური ენის გამოყენება [4].

C# ენის ერთ-ერთი მნიშვნელოვანი სიახლეა კომპონენტების ჩადგმული სახით უზრუნველყოფა. ფაქტობრივად C# ენა შექმნილია, როგორც კომპონენტებზე ორიენტირებული ენა, რომელიც მოიცავს ისეთ ელემენტებს, როგორიცაა თვისებები, მეთოდები და მოვლენები. ყველაზე მნიშვნელოვანი სიახლე C# ენაში არის მრავალენობრივ გარემოში მისი მუშაობის უნარი [4].

მართვის ელემენტი Button. ბრძანებითი ღილაკი. პროგრამების უმეტესობა იწერება სწორედ ამ ღილაკზე.

Label და LinkLabel მართვის ელემენტები. მონიშვნის და მიმართვის ღილაკი. მიმართვა შესაძლებელია როგორც ინტერნეტ-გვერდზე, ასევე დოკუმენტსა ან საქაღალდეზე:

```
namespace WindowsFormsApp22
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void linkLabel1_LinkClicked(object sender,
        LinkLabelLinkClickedEventArgs e)
```

```

{ string path1 = @"http://gtu.ge";
System.Diagnostics.Process.Start(path1);
this.linkLabel1.LinkVisited = true; }
private void linkLabel2_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e) {
string path2 = @"C:\Windows\System32\calc.exe";
if (System.IO.File.Exists(path2))
{
    System.Diagnostics.Process.Start(path2);
    this.linkLabel1.LinkVisited = true;
}
else
{
    MessageBox.Show("ასეთი ფაილი არ არსებობს!!!");
}
}
}
}

```

TextBox მართვის ელემენტი. ღილაკი გამოიყენება ინფორმაციის შესატანად.

RichTextBox მართვის ელემენტი. დიდი მოცულობის ტექსტის წარმოდგენა.

RadioButton მართვის ელემენტი. გადამრთველი ღილაკი ბურთულით. თუ გვექნება რამდენიმე გადამრთველი, შესაძლებელი იქნება, რომ მოვნიშნოთ მხოლოდ ერთი გადამრთველი.

CheckBox მართვის ელემენტი. გადამრთველი ღილაკი ალმით. შესაძლებელია მოვნიშნოთ ან ერთი ან ორი ან სამი გადამრთველი ერთდროულად ან ამოვრთოთ.

ListBox და **CheckedListBox** მართვის ელემენტები. სია და სია მონიშვნით.

ComboBox მართვის ელემენტი. ჩამოშლადი სია.

განვიხილოთ რამდენიმე ამოცანა.

ამოცანა 17.1. შევადგინოთ პროგრამა, რომლის საშუალებითაც შეგვიძლია ავირჩიოთ ცნობილი ქართველი მწერლის ან პოეტის სახელი, გვარი და ნაწარმოების დასახელება. არჩევის შედეგად ფორმაზე გამოჩნდეს ნაწარმოების შესაბამისი ფასი.

ფორმაზე განვათავსოთ სამი label და ორი comboBox კომპონენტები.

კომპონენტების გაფორმება. კომპონენტების გაფორმებისათვის Properties ფანჯარაში ავირჩიოთ შესაბამისი მახასიათებლები:

კომპონენტი label1: Text -> ქართველი მწერლები; Font -> Sylfaen, Font Style -> Bold, Size ->10; TextAlign -> MiddleCenter;

კომპონენტი label2: Text -> წიგნის დასახელება; Font -> Sylfaen, Font Style -> Bold, Size ->10; TextAlign -> MiddleCenter;

კომპონენტი label3: Text -> ფასი; Font -> Sylfaen, Font Style -> Bold, Size ->12; ForeColor -> Custom -> Maroon; TextAlign -> MiddleCenter;

კომპონენტი comboBox1: Font -> Sylfaen, Font Style -> Bold, Size ->10; Items -> ავირჩიოთ  -> შევიტანოთ მწერლის სახელი და გვარი: დავით კლდიაშვილი, გურამ დოჩანაშვილი, გოდერძი ჩოხელი, თორნიკე წაროზაული, ანა კალანდაძე.

კომპონენტი comboBox2: Font -> Sylfaen, Font Style -> Bold, Size ->10;

პროგრამული კოდი დაიწერება სამ ეტაპად:

პირველი ეტაპი: გამოვაცხადოთ სამი გლობალური ცვლადი.

```
int mwerali, nawarmoebi; double fasi;
```

მეორე ეტაპი: პროგრამული კოდის ერთი ნაწილი განვათავსოთ კომპონენტის comboBox1-ის მოვლენაზე SelectedIndexChanged.

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    mwerali = comboBox1.SelectedIndex;
    switch (mwerali)
    {
        case 0:
        {
            comboBox2.Text = "";
            comboBox2.Items.Clear();
            comboBox2.Items.Add("დარისპანის გასაჭირი");
            comboBox2.Items.Add("სამანიშვილის დედინაცვალი");
            comboBox2.Items.Add("ჩემი ცხოვრების გზაზე");
            break;
        }
        case 1:
```

```

{
    comboBox2.Text = "";
    comboBox2.Items.Clear();
    comboBox2.Items.Add("სამოსელი პირველი");
    comboBox2.Items.Add("თავფარავნელი ჭაბუკი");
    comboBox2.Items.Add("კეჭერაძეები - ცალ-ცალკე და ერთად");
    break;
}
case 2:
{
    comboBox2.Text = "";
    comboBox2.Items.Clear();
    comboBox2.Items.Add("ადამიანთა სედა");
    comboBox2.Items.Add("უმისამართო წერილი");
    comboBox2.Items.Add("ვიცოდი, აუცილებლად იქნებოდა აღდგომა");
    break;
}
case 3:
{
    comboBox2.Text = "";
    comboBox2.Items.Clear();
    comboBox2.Items.Add("ბავშვობის მონატრება");
    comboBox2.Items.Add("სულ რომ არაფერი ...");
    comboBox2.Items.Add("უკანასკნელი მეკვლე");
    break;
}
case 4:
{
    comboBox2.Text = "";
    comboBox2.Items.Clear();
    comboBox2.Items.Add("თუთა");
    comboBox2.Items.Add("საქართველოო ღამაზო");
    comboBox2.Items.Add("ყაყაჩოსა სიწითლითა");

```

```

        break;
    }
}
}

```

მესამე ეტაპი: პროგრამული კოდის მეორე ნაწილი განვითავსოთ კომპონენტის comboBox2-ის მოვლენაზე SelectedIndexChanged

```

private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
{
    nawarmoebi = comboBox2.SelectedIndex;
    if (nawarmoebi == 0 && mwerali == 0) fasi = 50.00;
    if (nawarmoebi == 1 && mwerali == 0) fasi = 45.30;
    if (nawarmoebi == 2 && mwerali == 0) fasi = 47.70;
    if (nawarmoebi == 0 && mwerali == 1) fasi = 30.00;
    if (nawarmoebi == 1 && mwerali == 1) fasi = 25.30;
    if (nawarmoebi == 2 && mwerali == 1) fasi = 17.70;
    if (nawarmoebi == 0 && mwerali == 2) fasi = 19.00;
    if (nawarmoebi == 1 && mwerali == 2) fasi = 55.30;
    if (nawarmoebi == 2 && mwerali == 2) fasi = 7.70;
    if (nawarmoebi == 0 && mwerali == 3) fasi = 29.00;
    if (nawarmoebi == 1 && mwerali == 3) fasi = 5.30;
    if (nawarmoebi == 2 && mwerali == 3) fasi = 12.70;
    if (nawarmoebi == 0 && mwerali == 4) fasi = 18.00;
    if (nawarmoebi == 1 && mwerali == 4) fasi = 23.30;
    if (nawarmoebi == 2 && mwerali == 4) fasi = 8.90;
    label3.Text = "ნაწარმოების ფასია: " + fasi.ToString() + " ლარი";
}

```

ამოცანა 17.2. შევადგინოთ პროგრამა, რომლის საშუალებითაც შევძლებთ ფოტოსურათის ღირებულების განსაზღვრას. ფორმაზე განვითავსოთ სამი label, სამი checkBox, ერთი textBox და ერთი button კომპონენტები (იხ. ნახ. 117.).

კომპონენტების გაფორმება.

კომპონენტი label1: Text -> ზომბა; Font -> Sylfaen, Font Style -> Bold, Size ->10; TextAlign -> MiddleCenter;

კომპონენტი label2: Text -> რაოდენობა; Font -> Sylfaen, Font Style -> Bold, Size ->10; TextAlign -> MiddleCenter;

კომპონენტი **checkBox1**: Text -> 9x12; Font -> Tahoma, Font Style -> Bold, Size ->12;

კომპონენტი **checkBox2**: Text -> 12x15; Font -> Tahoma, Font Style -> Bold, Size ->12;

კომპონენტი **checkBox3**: Text -> 18x24; Font -> Tahoma, Font Style -> Bold, Size ->12;

კომპონენტი **textBox1**: Enabled -> False;

კომპონენტი **button1**: Text -> ღირებულება; Font -> ARArchy, Size ->23, style = Boldlabel3;

კომპონენტი **label3**: Text -> ფასი; Font -> Sylfaen, Font Style -> Bold, Size ->12; ForeColor -> Custom -> Maroon; TextAlign -> MiddleCenter;

პროგრამული კოდი დაიწერება ოთხ ეტაპად:

პირველი ეტაპი: პროგრამული კოდი განვითავსოთ button1 კომპონენტის Click მოვლენაზე.

```
private void button1_Click(object sender, EventArgs e)
{
    double f1 = 0.0, f2 = 0.0, f3 = 0.0, MMM;
    int raodenoba = 0;
    if (checkBox1.Checked)
    {
        f1 = 0.50;
        raodenoba = int.Parse(textBox1.Text);
    }
    if (checkBox2.Checked)
    {
        f2 = 0.90;
        raodenoba = int.Parse(textBox1.Text);
    }
    if (checkBox3.Checked)
    {
        f3 = 1.30;
        raodenoba = int.Parse(textBox1.Text);
    }
    MMM = f1 * raodenoba + f2 * raodenoba + f3 * raodenoba;
    label3.Text = "შეკვეთის ფასი: " + MMM.ToString() + " ლარი";
}
```

მეორე ეტაპი: პროგრამული კოდი განვითავსოთ checkBox1-ის მოვლენაზე CheckedChanged.

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox1.Checked)
        textBox1.Enabled = true;
    else
    {
        textBox1.Enabled = false;
        textBox1.Text = "";
    }
}
```

```

    }
}

```

მესამე ეტაპი: პროგრამული კოდი განვათავსოთ checkBox2-ის მოვლენაზე CheckedChanged.

```

private void checkBox2_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox2.Checked)
        textBox1.Enabled = true;
    else
    {
        textBox1.Enabled = false;
        textBox1.Text = "";
    }
}

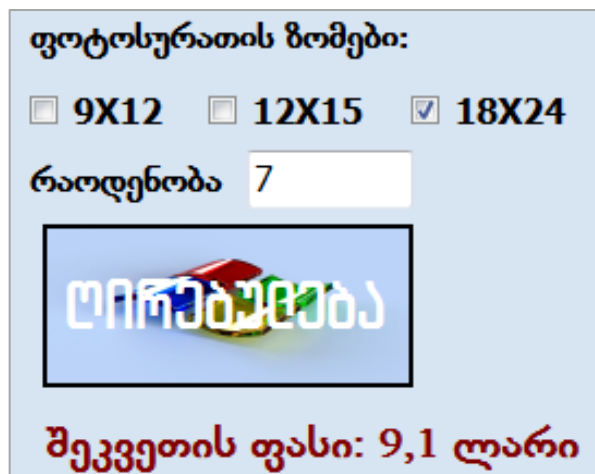
```

მეოთხე ეტაპი: პროგრამული კოდი განვათავსოთ checkBox3-ის მოვლენაზე CheckedChanged.

```

private void checkBox3_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox3.Checked)
        textBox1.Enabled = true;
    else
    {
        textBox1.Enabled = false;
        textBox1.Text = "";
    }
}

```



ნახ. 117. ფოტოსურათის ღირებულების გამოთვლა.

ამოცანა 17.3. შევადგინოთ პროგრამა, რომლის საშუალებითაც შევძლებთ ტრიგონომეტრიული ფუნქციების ჯამისა და სხვაობის მნიშვნელობების გამოთვლას. ფორმაზე განვათავსოთ ერთი comboBox, ერთი button, ორი textBox და ხუთი label კომპონენტები.

კომპონენტი comboBox1-ის თვისება Items შევიტანოთ შემდეგი ჩამონათვალი: $\sin(a+b)$; $\cos(a+b)$; $\operatorname{tg}(a+b)$; $\sin(a-b)$; $\cos(a-b)$; $\operatorname{tg}(a-b)$; (იხ. ნახ. 118. ა.ბ.).

პროგრამული კოდი მივაბათ comboBox1 კომპონენტის მოვლენას SelectedIndexChanged.

```
private void button1_Click(object sender, EventArgs e){
    double a, b, x = 0;
    string SSS = "";
    a = double.Parse(textBox1.Text);
    b = double.Parse(textBox2.Text);
    a = a / 27.123;
    b = b / 27.123;
    if (comboBox1.SelectedIndex == 0){
        x = (Math.Sin(a) * Math.Cos(b)) + (Math.Cos(a) * Math.Sin(b));
        SSS = "sin(a + b) = sin(a) * cos(b) + cos(a) * sin(b)";
    }
    if (comboBox1.SelectedIndex == 1) {
        x = (Math.Cos(a) * Math.Cos(b)) - (Math.Sin(a) * Math.Sin(b));
        SSS = "cos(a + b) = cos(a) * cos(b) - sin(a) * sin(b)";
    }
    if (comboBox1.SelectedIndex == 2) {
        x = (Math.Tan(a) * Math.Tan(b)) / (1 - Math.Tan(a) * Math.Tan(b));
        SSS = "tg(a + b) = tg(a) + tg(b) / 1 - tg(a) * tg(b)";
    }
    if (comboBox1.SelectedIndex == 3) {
        x = (Math.Sin(a) * Math.Cos(b)) - (Math.Cos(a) * Math.Sin(b));
        SSS = "sin(a - b) = sin(a) * cos(b) - cos(a) * sin(b)";
    }
    if (comboBox1.SelectedIndex == 4) {
        x = (Math.Cos(a) * Math.Cos(b)) + (Math.Sin(a) * Math.Sin(b));
```



```

        SSS = "cos(a - b) = cos(a) * cos(b) + sin(a) * sin(b)";
    }
    if (comboBox1.SelectedIndex == 5) {
x = (Math.Tan(a) * Math.Tan(b)) / (1 + Math.Tan(a) * Math.Tan(b));
        SSS = "tg(a - b) = tg(a) - tg(b) / 1 + tg(a) * tg(b)";
    }
    label3.Text = SSS + " = " + Math.Round(x, 2).ToString();
}

```

ტრიგონომეტრიული ფუნქციების
ჯამისა და სხვაობის გამოთვლა

A 2 

B 1,5

ფუნქციის ტიპი:
cos(a+b)

cos(a + b) = cos(a) * cos(b) - sin(a) * sin(b) = 0,99

ა)

ტრიგონომეტრიული ფუნქციების
ჯამისა და სხვაობის გამოთვლა

A 2 

B 1,5

ფუნქციის ტიპი:
sin(a-b)

sin(a - b) = sin(a) * cos(b) - cos(a) * sin(b) = 0,02

ბ)

ნახ. 118. ტრიგონომეტრიული ფუნქციების გამოთვლის შესრულების შედეგის:

ა) ერთი ვარიანტი; ბ) მეორე ვარიანტი

ამოცანა 17.4. გეომეტრიული ფიგურის დახატვა კოორდინატების გამოყენებით. ფორმაზე განვითავსოთ ოთხი label, ორი button, ოთხი textBox და ერთი panel კომპონენტები (იხ. ნახ. 119. ა.ბ.). პროგრამული კოდი მივაბათ კომპონენტების button1-ის და button2-ის Click მოვლენაზე.

```

int x, y, width, height;
private void button2_Click(object sender, EventArgs e)
{
    x = int.Parse(textBox1.Text);
    y = int.Parse(textBox2.Text);
    width = int.Parse(textBox3.Text);
    height = int.Parse(textBox4.Text);
    Pen shavi_pen = new Pen(Color.Black,3);
    Rectangle martk = new Rectangle(x, y, width, height);
    Graphics grafika = panel1.CreateGraphics();
    grafika.DrawRectangle(shavi_pen, martk);
}

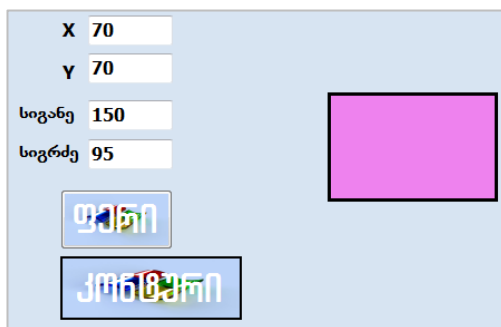
private void button1_Click(object sender, EventArgs e)

```

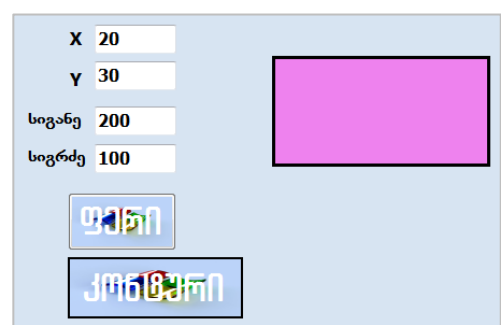
```

{
    x = int.Parse(textBox1.Text);
    y = int.Parse(textBox2.Text);
    width = int.Parse(textBox3.Text);
    height = int.Parse(textBox4.Text);
    SolidBrush witeli_brush = new SolidBrush(Color.Violet);
    Rectangle martk = new Rectangle(x, y, width, height);
    Graphics grafika = panel1.CreateGraphics();
    grafika.FillRectangle(witeli_brush, martk);
}

```



ა)



ბ)

ნახ. 119. გეომეტრიული ფიგურის დახატვის შესრულების შედეგის:

ა) ერთი ვარიანტი; ბ) მეორე ვარიანტი

ამოცანა 17.5. მომხმარებლის მიერ შექმნილი მარტივი ნახატი. ფორმაზე განვითარდეს ერთი panel და ერთი button კომპონენტები.

პროგრამული კოდი მივაბათ კომპონენტის button1-ის Click მოვლენაზე.

```

int indeqsi = 0;
private void button1_Click(object sender, EventArgs e){
    SolidBrush y_brush = new SolidBrush(Color.Yellow);
    Pen shavi_pen = new Pen(Color.Black, 3);
    Pen mwvane_pen = new Pen(Color.Green, 3);
    Pen lurji_pen = new Pen(Color.Blue, 3);
    Graphics grafika = panel1.CreateGraphics();
    Rectangle rect;
    if (indeqsi == 0)
        panel1.BackColor = Color.Green;
    if (indeqsi == 1)
    {

```

```

        rect = new Rectangle(100, 110, 70, 100);
        grafika.DrawRectangle(shavi_pen, rect);
    }
    if (indeqsi == 2)
    {
        grafika.DrawLine(shavi_pen, 100, 110, 135, 60);
        grafika.DrawLine(mwvane_pen, 135, 60, 170, 110);
    }
    if (indeqsi == 3)
    {
        rect = new Rectangle(120, 120, 30, 30);
        grafika.DrawRectangle(shavi_pen, rect);
    }
    if (indeqsi == 4)
    {
        rect = new Rectangle(100, 160, 30, 50);
        grafika.DrawRectangle(shavi_pen, rect);
    }
    if (indeqsi == 5)
    {
        grafika.FillEllipse(y_brush, 10, 10, 40, 40);
        grafika.DrawRectangle(lurji_pen, 10, 10, 40, 40);
    }
    ++indeqsi;
}

```

ამოცანა 17.6. კომპონენტი progressBar (იხ. ნახ. 120 და ნახ. 121). ფორმაზე განვითავსოთ ერთი label, ერთი progressBar, სამი button და ერთი timer კომპონენტები. კომპონენტების გაფორმებისათვის Properties ფანჯარაში ავირჩიოთ შესაბამისი მახასიათებლები:

label1 კომპონენტისათვის:

- Text ->
- Font -> Times New Roman, Font Style -> Bold, Size ->12
- TextAlign -> MiddleCenter

progressBar1 კომპონენტისათვის:

- Value->0
- Maximum->100
- Minimum->0

button1 კომპონენტისათვის:

- Text -> დაწყება
- Font -> Sylfaen, Font Style -> Bold, Size ->12
- TextAlign -> MiddleCenter

button2 კომპონენტისათვის:

- Text -> გაჩერება
- Font -> Sylfaen, Font Style -> Bold, Size ->12
- TextAlign -> MiddleCenter

button3 კომპონენტისათვის:

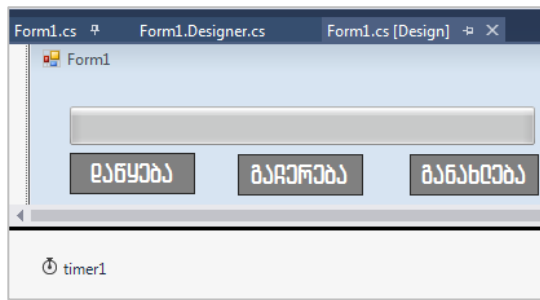
- Text -> განახლება
- Font -> Sylfaen, Font Style -> Bold, Size ->12
- TextAlign -> MiddleCenter

timer1 კომპონენტისათვის:

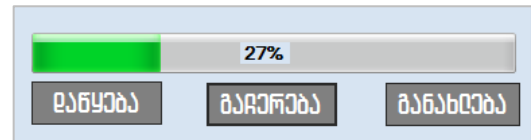
- Enabled -> False
- Interval ->100

პროგრამული კოდი განვითავსოთ timer1, button1, button2 და button3 კომპონენტების შესაბამის მოვლენაზე (Events).

```
private void timer1_Tick(object sender, EventArgs e) {  
    progressBar1.Increment(1);  
    label1.Text = progressBar1.Value + "%"; }  
private void button1_Click(object sender, EventArgs e) {  
    timer1.Start(); }  
private void button2_Click(object sender, EventArgs e){  
    timer1.Stop(); }  
private void button3_Click(object sender, EventArgs e)  
{    timer1.Start();  
        progressBar1.Value = 0;  
}
```



ნახ. 120. ამოცანა 17.6.-ის ფორმა/დიზაინი



ნახ. 121. ამოცანა 17.6.-ის შესრულების შედეგი

ამოცანა 17.7. შეძენილი პროდუქციის რაოდენობის დათვლა (იხ. ნახ. 122.). ფორმაზე განვთავსოთ ოთხი checkBox, სამი label, ერთი progressBar და ერთი panel კომპონენტები. პროგრამული კოდი დაიწერება ხუთ ეტაპად. პროგრამული კოდი მივაბათ კომპონენტის checkBox1, checkBox2, checkBox3, checkBox4 კომპონენტების შესაბამის CheckedChanged მოვლენაზე და ფორმის Form1-ის მოვლენაზე Load.

პირველი ეტაპი. კომპონენტის checkBox1 მოვლენაზე CheckedChanged-ზე განვთავსოთ შემდეგი კოდი:

```
private void checkBox1_CheckedChanged(object sender, EventArgs e){
    label3.Text = "Made in Italy: Price 12$\n";
    p += 12;
    progressBar1.Value = p;
    label1.Text = "Total Price " + Convert.ToString(p) + " $";
}
```

მეორე ეტაპი. კომპონენტის checkBox2 მოვლენაზე CheckedChanged-ზე განვთავსოთ შემდეგი კოდი:

```
private void checkBox2_CheckedChanged(object sender, EventArgs e){
    label3.Text += "Made in Italy + Price 23$\n";
    p += 23;
    progressBar1.Value = p;
    label1.Text = "Total Price " + Convert.ToString(p) + " $";
}
```

მესამე ეტაპი. კომპონენტის checkBox3 მოვლენაზე CheckedChanged-ზე განვთავსოთ შემდეგი კოდი:

```
private void checkBox3_CheckedChanged(object sender, EventArgs e){
    label3.Text += "Made in Italy + Price 21$\n";
```

```

p += 21;
progressBar1.Value = p;
label11.Text = "Total Price " + Convert.ToString(p) + " $";
}

```

მეოთხე ეტაპი. კომპონენტის checkBox4 მოვლენაზე CheckedException-ზე განვითავსოთ შემდეგი კოდი:

```

private void checkBox4_CheckedChanged(object sender, EventArgs e) {
    label3.Text += "Made in Italy + Price 17$\n";
    p += 17;
    progressBar1.Value = p;
    label11.Text = "Total Price " + Convert.ToString(p) + " $";
}

```

მეხუთე ეტაპი. გამოვაცხადოთ გლობალური ცვლადი `int p = 0`; ამასთანავე ფორმის Form1-ის მოვლენა Load-ზე განვითავსოთ შემდეგი კოდი:

```

private void Form1_Load(object sender, EventArgs e){
    checkBox1.BackColor = Color.Transparent;
    checkBox2.BackColor = Color.Transparent;
    checkBox3.BackColor = Color.Transparent;
    checkBox4.BackColor = Color.Transparent;
    label11.BackColor = Color.Transparent;
    label12.BackColor = Color.Transparent;
    label13.BackColor = Color.Transparent;
}

```



ნახ. 122. ამოცანა 17.7.-ის შესრულების შედეგი.

ამოცანა 17.8. გამოსახულების არჩევა კომპონენტის comboBox-ის გამოყენებით (იხ. ნახ. 123 და ნახ. 124). ფორმაზე განვათავსოთ კომპონენტები: ერთი label, ერთი comboBox და ერთი pictureBox.

პროექტის მთავარი საქაღალდის ქვესაქაღალდეში Debug შეინახეთ ოთხი გამოსახულება სახელებით: 1.jpg, 2.jpg, 3.jpg, 4.jpg ან გამოსახულებები შეარჩიეთ საკუთარი შეხედულებისამებრ.

გამოსახულებების შენახვის გზა/მისამართი კი ქვემოთ არის მოცემული:

C:\Users\user\source\repos\WindowsFormsApp19\WindowsFormsApp19\bin\Debug\

კომპონენტების გაფორმებისათვის Properties ფანჯარაში ავირჩიოთ შესაბამისი მახასიათებლები:

კომპონენტისათვის label1:

- Text -> label1
- Font -> Times New Roman, Font Style -> Bold, Size ->12
- TextAlign -> MiddleCenter

კომპონენტისათვის comboBox1:

- Text -> ჩამოშალე 😊
- Font -> Sylfaen, Font Style -> Bold, Size ->12
- Items -> 1, 2, 3, 4

კომპონენტისათვის pictureBox1:

- BackColor -> Custom -> Gray
- SizeMode -> StretchImage

პროგრამული კოდი განვათავსოთ კომპონენტის comboBox1-ის მოვლენაზე SelectedIndexChanged-ზე.

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e){
    switch (comboBox1.SelectedIndex)
    {
        case 0: pictureBox1.Image = Image.FromFile("1.jpg");
        label1.Text = " გამოსახულება N1 - computer science"; break;
        case 1: pictureBox1.Image = Image.FromFile("2.jpg");
        label1.Text = " გამოსახულება N2 - computer science"; break;
        case 2: pictureBox1.Image = Image.FromFile("3.jpg");
```

```

label1.Text = " გამოსახულება N3 - computer science"; break;
case 3: pictureBox1.Image = Image.FromFile("4.jpg");
label1.Text = " გამოსახულება N4 - computer science"; break;
}
}

```



ნახ. 123. ამოცანა 17.8.-ის
ფორმა/დიზაინი



ნახ. 124. ამოცანა 17.8.-ის შესრულების
შედეგი

ამოცანა 17.9. ერთიდან ასამდე ჩაფიქრებული რიცხვის ამოცნობა (პროგრამა ამოცნობის შესახებ). გამოვიყენოთ შემთხვევითი რიცხვების გენერირება (იხ. ნახ. 125. ა.ბ.გ.).

ფორმაზე განვათავსოთ ორი label, ერთი textBox და ერთი button კომპონენტები. კომპონენტების გაფორმებისათვის Properties ფანჯარაში ავირჩიოთ შესაბამისი მახასიათებლები:

კომპონენტი label1:

- Text -> შეიტანეთ რიცხვი
- Font -> Sylfaen, Font Style -> Bold, Size ->12
- TextAlign -> MiddleCenter

კომპონენტი label2:

- Text -> შედეგი
- Font -> Sylfaen, Font Style -> Bold, Size ->10
- TextAlign -> MiddleCenter

კომპონენტი button1:

- Text -> შემოწმება
- Font -> Sylfaen, Font Style -> Bold, Size ->12
- TextAlign -> MiddleCenter

კომპონენტი textBox1:

- Text ->
- Font -> Sylfaen, Font Style -> Bold, Size ->12
- TextAlign -> MiddleCenter

პროგრამული კოდი განვითავსთ Form1 ფორმის (ფორმაც ითვლება კომპონენტად) მოვლენაზე Load, კომპონენტის button1-ის მოვლენაზე Click, კომპონენტის textBox1-ის მოვლენაზე KeyPress და კომპონენტის textBox1-ის მოვლენაზე TextChanged.

```
გამოვაცხადოთ გლობალური ცვლადები: int chafiqrebuli, varaudi;
private void textBox1_TextChanged(object sender, EventArgs e) {
    if (textBox1.Text.Length == 0)
        button1.Enabled = false;
    else button1.Enabled = true;
}
private void textBox1_Click(object sender, EventArgs e)
{
    textBox1.Text = "";
}
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar >= '0' && e.KeyChar <= '9')
        e.Handled = false;
    else
        e.Handled = true;
}
private void Form1_Load(object sender, EventArgs e)
{
    Random rand = new Random();
    varaudi = rand.Next(100);
}
private void button6_Click(object sender, EventArgs e)
{
    chafiqrebuli = int.Parse(textBox1.Text);
}
```

```

        if (varaudi < chafiqrebuli)
            label1.Text = "შეტანილი რიცხვი დიდია.\იშეიტანეთ მასზე  

            ნაკლები მნიშვნელობა.";
        else if (varaudi > chafiqrebuli)
            label1.Text = "შეტანილი რიცხვი მცირეა.\იშეიტანეთ მასზე  

            მეტი მნიშვნელობა.";
        else label1.Text = "პასუხი სწორია.";
    }

```

ჩაფიქრებული რიცხვი 78 შეშინება :) შეტანილი რიცხვი დიდია. შეიტანეთ მასზე ნაკლები მნიშვნელობა.	ჩაფიქრებული რიცხვი 27 შეშინება :) შეტანილი რიცხვი მცირეა. შეიტანეთ მასზე მეტი მნიშვნელობა.	ჩაფიქრებული რიცხვი 43 შეშინება :) პასუხი სწორია.

ა)

ბ)

გ)

ნახ. 125. ამოცანა 17.9.-ის შესრულების შედეგის: ა) ერთი ვარიანტი;

ბ) მეორე ვარიანტი; გ) მესამე ვარიანტი

ამოცანა 17.10. შემთხვევითი რიცხვების გენერირება მითითებულ დიაპაზონში (იხ. ნახ. 126.). ფორმაზე განვითავსოთ სამი label, ორი numericUpDown და ერთი button კომპონენტები.

პროგრამული კოდი განვითავსოთ კომპონენტის button1-ის მოვლენა Click-ზე.

```

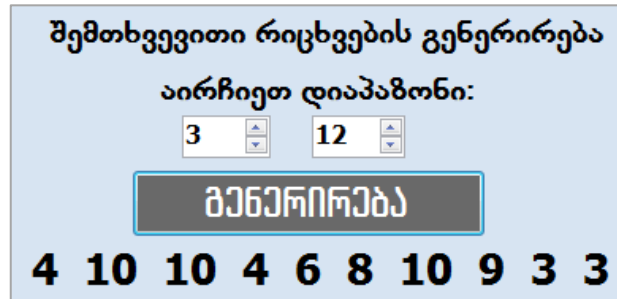
private void button1_Click(object sender, EventArgs e) {
    label3.Text = "";
    System.Random shemtxveviTi_ricxvebi = new Random();
    int sawyisi, saboloo, zzz;
    sawyisi = Convert.ToInt32(numericUpDown1.Value);
    saboloo = Convert.ToInt32(numericUpDown2.Value);
    for (int i=0; i<10; ++i)
    {
        if (sawyisi < saboloo)
        {
            zzz = shemtxveviTi_ricxvebi.Next(sawyisi, saboloo);
        }
        else
        {

```

```

        zzz = shemtxveviTi_ricxvebi.Next(saboloo, sawyisi);
    }
    label3.Text += zzz.ToString() + " ";
}
}

```



ნახ. 126. ამოცანა 17.10.-ის შესრულების შედეგი.

ამოცანა 17.11. გამოსახულების ზომის შეცვლა (იხ. ნახ. 127, ნახ. 128). ფორმაზე განვათავსოთ ერთი label, ორი numericUpDown და ერთი pictureBox კომპონენტები. კომპონენტების გაფორმებისათვის Properties ფანჯარაში ავირჩიოთ შესაბამისი მახასიათებლები:

კომპონენტი label1: Text -> სურათის ზომის შეცვლა; Font -> Sylfaen, Font Style -> Bold, Size ->12; TextAlign -> MiddleCenter

კომპონენტი numericUpDown1: Maximum -> 100; Minimum -> 0; Value -> 0;

კომპონენტი numericUpDown2: Maximum -> 100; Minimum -> 0; Value -> 0;

კომპონენტი pictureBox1:

Image -> Import -> ავირჩიოთ სურათი;

Size -> 100, 100;

SizeMode -> StretchImage;

პროგრამული კოდი განვათავსოთ Form1 კომპონენტის Load და numericUpDown კომპონენტების ValueChanged მოვლენებზე (Events).

1. კომპონენტის Form1-ის მოვლენა Load-ზე გვექნება შემდეგი პროგრამული ფრაგმენტი:

```

private void Form1_Load(object sender, EventArgs e){
    numericUpDown1.Maximum = 500;
    numericUpDown1.Minimum = 100;
    numericUpDown1.Increment = 10;
    numericUpDown2.Maximum = 500;
}

```

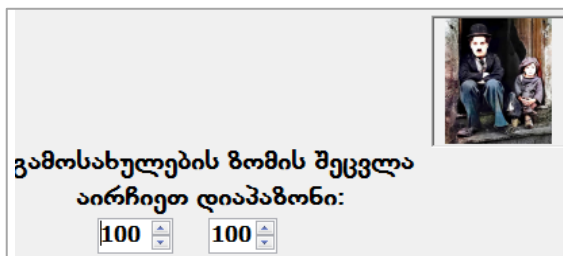
```
numericUpDown2.Minimum = 100;
numericUpDown2.Increment = 10; }
```

2. კომპონენტის numericUpDown1-ის მოვლენა ValueChanged-ზე გვექნება შემდეგი პროგრამული ფრაგმენტი:

```
private void numericUpDown1_ValueChanged_1(object sender, EventArgs e){
    pictureBox2.Height = (int)numericUpDown1.Value; }
```

3. კომპონენტის numericUpDown2-ის მოვლენა ValueChanged-ზე გვექნება შემდეგი პროგრამული ფრაგმენტი:

```
private void numericUpDown2_ValueChanged_1(object sender, EventArgs e){
    pictureBox2.Width = (int)numericUpDown2.Value;
}
```



ნახ. 127. ამოცანა 17.11.-ის
ფორმა/დიზაინი



ნახ. 128. ამოცანა 17.11.-ის
შესრულების შედეგი

ამოცანა 17.12. პროექტთან მუშაობის ხანგრძლივობის ვადის განსაზღვრა (იხ. ნახ. 129.). ფორმაზე განვათავსოთ ექვსი label, სამი numericUpDown და ერთი button კომპონენტები. კომპონენტების გაფორმებისათვის Properties ფანჯარაში ავირჩიოთ შესაბამისი მახასიათებლები:

label1 კომპონენტისათვის: Text -> ; Font -> Sylfaen, Font Style -> Bold, Size ->12; ForeColor -> Custom ->192, 0, 0; TextAlign -> MiddleCenter;

label2 კომპონენტისათვის: Text -> დღე; Font -> Sylfaen, Font Style -> Bold, Size ->10; TextAlign -> MiddleCenter;

label3 კომპონენტისათვის: Text -> თვე; Font -> Sylfaen, Font Style -> Bold, Size ->10; TextAlign -> MiddleCenter;

label4 კომპონენტისათვის: Text -> წელი; Font -> Sylfaen, Font Style -> Bold, Size ->10; TextAlign -> MiddleCenter;

label5 კომპონენტისათვის: Text -> შედეგი; Font -> Sylfaen, Font Style -> Bold, Size ->12;
ForeColor -> Custom ->192, 0, 0; TextAlign -> MiddleCenter;

label6 კომპონენტისათვის: Text -> აირჩიეთ პროექტის ხანგრძლივობის ვადა; Font -> Sylfaen, Font Style -> Bold, Size ->12; TextAlign -> MiddleCenter;

numericUpDown1 კომპონენტისათვის: Maximum -> 31; Minimum -> 0; Value -> 0;

numericUpDown2 კომპონენტისათვის: Maximum -> 12; Minimum -> 0; Value -> 0;

numericUpDown3 კომპონენტისათვის: Maximum -> 100; Minimum -> 0; Value -> 0;

button1 კომპონენტისათვის: Text -> დააჭირეთ; Font -> Sylfaen, Font Style -> Bold, Size ->12; TextAlign -> MiddleCenter;

პროგრამული კოდი დავწეროთ ქვემოთ მოცემული მითითებების შესაბამისად:

1. თავდაპირველად გამოვაცხადოთ გლობალური ცვლადები:

```
DateTime dawyebis_dro, damtavrebis_dro;
```

2. პროგრამული კოდის ერთი ნაწილი განვათავსოთ კომპონენტის Form1-ის მოვლენა Load-ზე:

```
private void Form1_Load(object sender, EventArgs e) {  
    label4.Text = " პროექტის დაწყების თარიღი: " +  
    DateTime.Today.ToString("dd MMMM yyyy") + " წელი";  
}
```

3. პროგრამული კოდის მეორე ნაწილი განვათავსოთ კომპონენტ button1-ის მოვლენა Click-ზე:

```
private void button1_Click(object sender, EventArgs e){  
    dawyebis_dro = DateTime.Today;  
    damtavrebis_dro =  
    dawyebis_dro.AddDays((int)numericUpDown1.Value);  
    damtavrebis_dro =  
    damtavrebis_dro.AddMonths((int)numericUpDown2.Value);  
    damtavrebis_dro=damtavrebis_dro.AddYears((int)numericUpDown3.Value);  
    label6.Text = "-----\n";  
    label6.Text += "პროექტის დასრულების თარიღი: " +  
    damtavrebis_dro.ToString("dd MMMM yyyy") + " წელი";  
}
```

აირჩიეთ პროექტზე მუშაობის ხანგრძლივობა

დღე თვე წელი
8 3 3

პროექტის დაწყების თარიღი: 19 სექტემბერი 2023 წელი

პროექტის დასრულების თარიღი: 27 დეკემბერი 2026 წელი

ნახ. 129. ამოცანა 17.12.-ის შესრულების შედეგი

ამოცანა 17.13. ფერების ტალღის სიგრძის დიაპაზონის წარმოდგენა (იხ. ნახ. 130.). ფორმაზე განვითავსოთ სამი label, ერთი textBox და ერთი button კომპონენტები. კომპონენტი textBox-ზე დავაწესოთ კონტროლი. საკმაოდ საინტერესო ინფორმაციას ფერთა სპექტრსა და მოდელებზე შეგიძლიათ გაეცნოთ ვებგვერდზე:

<https://intermedia.ge/%E1%83%A1%E1%83%A2%E1%83%90%E1%83%A2%E1%83%98%E1%83%90/64369%E1%83%A4%E1%83%94%E1%83%A0%E1%83%98%E1%83%93%E1%83%90%E1%83%A4%E1%83%94%E1%83%A0%E1%83%98%E1%83%A1%E1%83%A1%E1%83%98%E1%83%A6%E1%83%A0%E1%83%9B%E1%83%94-%E1%83%A4%E1%83%94%E1%83%A0%E1%83%97%E1%83%90-%E1%83%9B%E1%83%9D%E1%83%93%E1%83%94%E1%83%9A%E1%83%94%E1%83%91%E1%83%98-Photoshop-CS-4-5/16511/>

პროგრამული კოდი განვითავსოთ ქვემოთ მოცემული სქემის მიხედვით:

1. პროგრამული კოდი განვითავსოთ კომპონენტი button1-ის მოვლენა Click-ზე

```
private void button1_Click(object sender, EventArgs e) {
```

```
    try {  
        int diapazon = int.Parse(textBox1.Text);  
        if (diapazon > 380 && diapazon <= 440)  
        {  
            label3.Text = "იასამნისფერი";  
            this.BackColor = System.Drawing.Color.BlueViolet;  
            label3.ForeColor = Color.Black;  
        }  
        else if (diapazon > 440 && diapazon <= 485)  
        {  
            label3.Text = "ლურჯი";  
            this.BackColor = System.Drawing.Color.Blue;  
            label3.ForeColor = Color.Black;  
        }  
    }
```

```

else if (diapazon > 485 && diapazon <= 500)
{
    label3.Text = "ცისფერი";
    this.BackColor = System.Drawing.Color.LightBlue;
    label3.ForeColor = Color.Black;
}
else if (diapazon > 500 && diapazon <= 565)
{
    label3.Text = "მწვანე";
    this.BackColor = System.Drawing.Color.Green;
    label3.ForeColor = Color.Black;
}
else if (diapazon > 565 && diapazon <= 590)
{
    label3.Text = "ყვითელი";
    this.BackColor = System.Drawing.Color.Yellow;
    label3.ForeColor = Color.Black;
}
else if (diapazon > 590 && diapazon <= 625)
{
    label3.Text = "ნარინჯისფერი";
    this.BackColor = System.Drawing.Color.Orange;
    label3.ForeColor = Color.Black;
}
else if (diapazon > 625 && diapazon <= 740)
{
    label3.Text = "წითელი";
    this.BackColor = System.Drawing.Color.Red;
    label3.ForeColor = Color.Black;
}
else
{
    label3.Text = "ფერის ხილული სპექტრი \იდიამაზონის გარეთაა";
    label3.ForeColor = Color.Red; this.BackColor = Color.White;
}

```

```

        label3.TextAlign = ContentAlignment.MiddleCenter;
    }
}
catch (Exception)
{
    MessageBox.Show("შეცდომა!!! შეიტანეთ რიცხვი");
}
}

```

2. პროგრამული კოდი განვითავსოთ კომპონენტ textBox1-ის მოვლენა KeyPress-ზე:

```

private void textBox1_KeyPress(object sender, KeyPressEventArgs e){
    if ((e.KeyChar >= '0') && (e.KeyChar <= '9')) return;
    if (Char.IsControl(e.KeyChar))
        if (e.KeyChar == (char)Keys.Back)
        {
            return;
        }
    if (Char.IsControl(e.KeyChar))
        if (e.KeyChar == (char)Keys.Enter)
        {
            button1.Focus();
            return;
        }
    e.Handled = true;
}

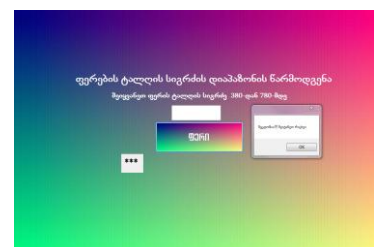
```



ა)



ბ)



გ)

ნახ. 130. ამოცანა 17.13.-ის შესრულების შედეგის: ა) ერთი ვარიანტი;

ბ) მეორე ვარიანტი; გ) მესამე ვარიანტი

ამოცანა 17.14. კატეგორია (იხ. ნახ. 131. ა.ბ.გ.).

პროგრამული კოდი:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WindowsFormsApp20
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        double summ;
        private void Form1_Load(object sender, EventArgs e)
        {
            checkBox3.Enabled = false;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            if (checkBox1.Checked && checkBox2.Checked && checkBox3.Checked &&
                checkBox4.Checked &&
                checkBox5.Checked && checkBox6.Checked && checkBox7.Checked &&
                checkBox8.Checked && checkBox9.Checked)
            {
                // თუ მომხმარებელმა შეუკვეთა ყველაფერი, ეკუთვნის 10%-იანი ფასდაკლება
                MessageBox.Show("გეკუთვნით 10%-იანი ფასდაკლება\!" + "შეკვეთის  

                ღირებულება " + (summ * 0.9).ToString("C"), "კაფე");
            }
        }
    }
}
```

```

        else
        {
            if (checkBox1.Checked || checkBox2.Checked || checkBox3.Checked ||
                checkBox4.Checked || checkBox5.Checked || checkBox6.Checked ||
                checkBox7.Checked || checkBox8.Checked || checkBox9.Checked)
                MessageBox.Show("შეკვეთის ღირებულებაა: "+summ.ToString("C"), "კაფე");
            else
                MessageBox.Show("თქვენ არაფერი შეუკეთეთ!!! ", "კაფე");
        }
    }

    private void checkBox1_CheckedChanged_1(object sender, EventArgs e){
        if (checkBox1.Checked) summ += 6.50;
        else summ -= 6.50;
        label1.Text = summ.ToString();
        label1.Refresh();
    }

    private void checkBox2_CheckedChanged_1(object sender, EventArgs e){
        if (checkBox2.Checked)
        {
            summ += 2.90;
            if (!checkBox3.Enabled) checkBox3.Enabled = true;
        }
        else
        {
            summ -= 2.90;
            if (checkBox3.Checked) checkBox3.Checked = false;
            checkBox3.Enabled = false;
        }
        label1.Text = summ.ToString();
        label1.Refresh();
    }

    private void checkBox3_CheckedChanged_1(object sender, EventArgs e){
        if (checkBox3.Checked) summ += 3.0;
        else summ -= 3.0;
    }

```

```

        label11.Text = summ.ToString();
        label11.Refresh();
    }
private void checkBox4_CheckedChanged_1(object sender, EventArgs e){
    if (checkBox4.Checked) summ += 5.0;
    else summ -= 5.0;
    label11.Text = summ.ToString();
    label11.Refresh();
}
private void checkBox5_CheckedChanged(object sender, EventArgs e){
    if (checkBox5.Checked) summ += 2.75;
    else summ -= 2.75;
    label11.Text = summ.ToString();
    label11.Refresh();
}
private void checkBox6_CheckedChanged(object sender, EventArgs e) {
    if (checkBox6.Checked) summ += 12.0;
    else summ -= 12.0;
    label11.Text = summ.ToString();
    label11.Refresh();
}
private void checkBox7_CheckedChanged(object sender, EventArgs e){
    if (checkBox7.Checked) summ += 7.5;
    else summ -= 7.5;
    label11.Text = summ.ToString();
    label11.Refresh();
}
private void checkBox8_CheckedChanged(object sender, EventArgs e){
    if (checkBox8.Checked) summ += 2.5;
    else summ -= 2.5;
    label11.Text = summ.ToString();
    label11.Refresh();
}
private void checkBox9_CheckedChanged(object sender, EventArgs e) {

```

```

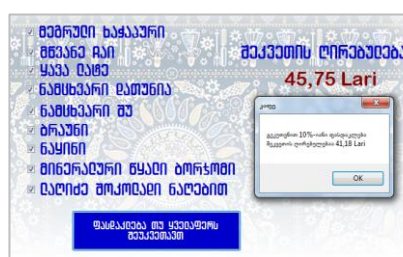
        if (checkBox9.Checked) summ += 3.35;
        else summ -= 3.35;
        label11.Text = summ.ToString();
        label11.Refresh();
    }

    private void label11_Paint_1(object sender, PaintEventArgs e) {
        label11.Text = summ.ToString("C");
    }
}
}

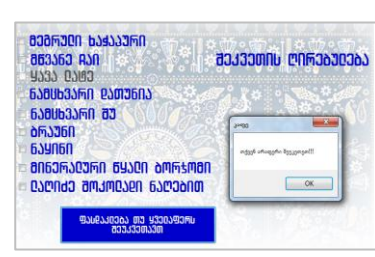
```



ა)



ბ)



გ)

ნახ. 131. ამოცანა 17.14.-ის შესრულების შედეგებს: ა) ერთი ვარიანტი;

ბ) მეორე ვარიანტი; გ) მესამე ვარიანტი

ამოცანა 17.15. ილუსტრატორი. ძეგნა მისამართის გამოძახებით (იხ. ნახ. 132. ა.ბ.).

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
namespace WindowsFormsApp21
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

```

```

    }

    private int pbh, pbw; // pictureBox1 კომპონენტის სიმაღლე და სიგრძე
        /* listBox1 კომპონენტში ახდენს ილუსტრაციების (გამოსახულებების)
        ფორმირებას aPath - ფაილის მისამართი */
    private void button1_Click(object sender, EventArgs e){
        label1.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
        label1.Text = "📁";
        FolderBrowserDialog fb = new FolderBrowserDialog();
        fb.Description = "ამოირჩიეთ საქაღალდე";
        fb.ShowNewFolderButton = false;
        if (fb.ShowDialog() == DialogResult.OK)
    if (!FillListBox1(fb.SelectedPath + "\\")) pictureBox1.Image = null;}
    private Boolean FillListBox1(string aPath) {
        // ინფორმაცია საქაღალდის შესახებ.
        DirectoryInfo di = new DirectoryInfo(aPath);
        FileInfo[] fi = di.GetFiles("*.jpg");
        listBox1.Items.Clear();
        foreach (FileInfo fc in fi)
        { listBox1.Items.Add(fc.Name); }
        label2.Text = aPath;
        if (fi.Length == 0) return false;
        else
        {
            listBox1.SelectedIndex = 0;
        }
        return true;
    }

    private void Form1_Load(object sender, EventArgs e) {
        pbh = pictureBox1.Height;
        pbw = pictureBox1.Width;
        listBox1.Sorted = true;
    }

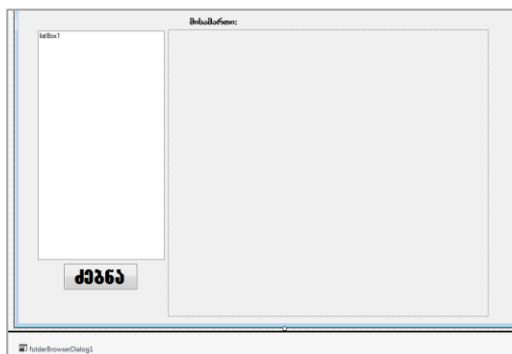
    private void listBox1_SelectedIndexChanged(object sender, EventArgs e) {
        double mh, mw;

```

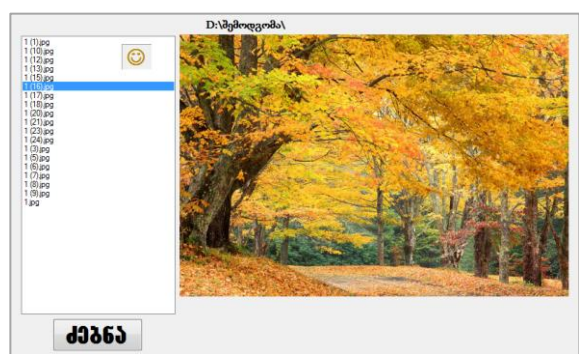
```

        pictureBox1.Image = new Bitmap(label2.Text +
        listBox1.SelectedItem.ToString());
if ((pictureBox1.Image.Width>pbw)|| (pictureBox1.Image.Height> pbh))
{
    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
    mh = (double)pbh / (double)pictureBox1.Image.Height;
    mw = (double)pbh / (double)pictureBox1.Image.Width;
    if (mh < mw)
    {
        pictureBox1.Width = Convert.ToInt32(pictureBox1.Image.Width * mh);
        pictureBox1.Height = pbh;
    }
    else
    {
        pictureBox1.Width = pbw;
        pictureBox1.Height = Convert.ToInt32(pictureBox1.Image.Height * mw);
    }
}
else
{
    pictureBox1.Width = pbw;
    pictureBox1.Height = Convert.ToInt32(pictureBox1.Image.Height * mw);
}
}
}
}
}
}
}
}

```



ა)



ბ)

ნახ. 132. ამოცანა 17.15.-ის შესრულების შედეგის:

ა) ერთი ვარიანტი; ბ) მეორე ვარიანტი

დამოუკიდებელი სამუშაო

1. გადაიყვანეთ დრო (დრო მოცემულია საათში, წუთში და წამში) წამებში. ფორმაზე განვათავსოთ სამი textBox, ხუთი label და ერთი button კომპონენტები. შეტანის კომპონენტებზე (textBox-ზე) დავაწესოთ კონტროლი ანუ დავწეროთ ისეთი კოდი, სადაც მოხდება შეცდომების გამორიცხვა.

2. listBox კომპონენტის გამოყენებით შევქმნათ რამდენიმე ქვეყნის ჩამონათვალი. რომელიმე ქვეყნის არჩევისას გამოჩნდეს მოკლე ინფორმაცია ამ ქვეყნის შესახებ: კერძოდ, ფართობი, მოსახლეობა, კულტურული ღირებულებები და დროშა.

3. მოცემულია რიცხვი. თუ რიცხვი დადებითია, იგი გადავიდეს listBox1 კომპონენტში, თუ რიცხვი უარყოფითია, მაშინ - listBox2 კომპონენტში. ფორმაზე განვათავსოთ ერთი textBox, სამი label და ორი listBox კომპონენტები. დავაწესოთ კონტროლი შეტანის კომპონენტზე.

18 ნაკადები. Thread კლასი. Thread კლასის თვისებები. ნაკადების მდგომარეობები.

პროგრამირებაში ძირითადად გავრცელებულია სინქრონული პროცესები, მაგრამ ასევე ძალზე მნიშვნელოვანია ასინქრონული პროცესების გამოყენება, როდესაც საჭიროა პროგრამის მუშაობის სიჩქარისა და რესურსების მართვა.

სინქრონული (Synchronous) პროცესები ხორციელდება თანამიმდევრულად ანუ თითოეული ოპერაცია უნდა დასრულდეს ბოლომდე და მხოლოდ მისი დასრულების შემდეგ დაიწყოს მომდევნო ოპერაციის შესრულება.

სინქრონული პროცესების მახასიათებლებია:

1. ყოველი ოპერაცია ბლოკავს შესრულებას, სანამ ეს ოპერაცია არ დასრულდება;
2. კოდის გაგება უფრო მარტივია, რადგან ყველაფერი ხდება თანამიმდევრულად;
3. თუ ერთი პროცესი დიდხანს გაგრძელდება, მაშინ მთელი პროგრამა შეიძლება „ჩამოეკიდოს“.

ასინქრონული (Asynchronous) პროცესები საშუალებას იძლევა, ოპერაციები პარალელურად შესრულდეს, რაც ზრდის პროგრამის ეფექტურობას.

ასინქრონული პროცესების მახასიათებლებია:

1. ოპერაციები არ ბლოკავენ მთავარ ნაკადს, ამიტომ სხვა პროცესები პარალელურად გრძელდება;
2. კოდის ლოგიკა საკმაოდ რთულია;
3. სწრაფი რეაქცია, რომელიც განსაკუთრებით სასარგებლოა, როცა პროგრამას გარე სერვისებთან უწევს ურთიერთობა (მაგ., API მოთხოვნები).

სინქრონული და ასინქრონული პროცესების შედარებითი ანალიზი მოცემულია მე-16 ცხრილში.

ცხრილი 16. სინქრონული და ასინქრონული პროცესების შედარებითი ცხრილი

მახასიათებელი	სინქრონული	ასინქრონული
შესრულების ტიპი	თანამიმდევრული	პარალელური
ბლოკირება	ბლოკირებადი	არაბლოკირებადი
კოდის სირთულე	მარტივი	შედარებით რთული
რესურსების გამოყენება	ნაკლებად ოპტიმიზებული	მაქსიმალურად ეფექტური
გამოყენების შემთხვევები	მცირე, მარტივი ამოცანები	სერვერები, ქსელური მოთხოვნები, დიდი სისტემები

სინქრონული პროცესები გამოიყენება, მაგალითად, ფაილების წაკითხვა/ჩაწერის დროს, მარტივი ალგორითმების დასამუშავებლად, სადაც მნიშვნელოვანია ოპერაციების რიგითობა.

ასინქრონული პროცესები გამოიყენება, მაგალითად, ვებსერვერებში, რეალურ დროში მონაცემთა დამუშავებისას, დიდ მონაცემთა ნაკადებთან მუშაობის დროს.

C# ენაში შეგვიძლია გამოვიყენოთ პარალელური (ასინქრონული) დაპროგრამება ნაკადების მხარდაჭერით. ნაკადი არის კოდის შესრულების დამოუკიდებელი გზა, რომელსაც აქვს უნარი ერთდროულად შესრულდეს სხვა ნაკადთან.

მამ ასე, გაუმარჯოს დამოუკიდებლობას ☺.

C#-ში ნაკადები (Threads) საშუალებას გაძლევს, პროგრამა დავყოთ პარალელურ ანუ ასინქრონულ პროცესებად. ეს განსაკუთრებით სასარგებლოა, როცა გსურს, რომ პროგრამის სხვადასხვა ნაწილმა ერთდროულად იმუშაოს და უფრო სწრაფად შეასრულოს დასმული ამოცანა.

როგორც აღვნიშნეთ ნაკადის შექმნისათვის გამოიყენება კლასი Thread.

ნაკადებთან სამუშაოდ უნდა გამოვიყენოთ დირექტივა: `using System.Threading;`

პროგრამული კოდი უნდა დაიწეროს კონსოლურ დანართში, ამიტომაც კოდში გვაქვს მეთოდი `Console.WriteLine()`.

Thread კლასი. Thread კლასი უზრუნველყოფენ ნაკადების მართვას. ამ კლასის გამოყენებით შეგვიძლია შევქმნათ, გავუშვათ, შევაჩეროთ და ვმართოთ ნაკადები. განვიხილოთ Thread კლასის მეთოდები:

1. `Thread.Start()` - ნაკადის გაშვება;
2. `Thread.Sleep(ms)` - ნაკადის დროებითი „დაძინება“.
3. `Thread.Join()` - ელოდება ნაკადის დასრულებას;
4. `Thread.IsAlive` - ამოწმებს, გაშვებულია თუ არა ნაკადი;
5. `Thread.Abort()` - ნაკადის იძულებით შეწყვეტა (არ არის სასურველი!);
6. `Thread.Priority` - ნაკადის პრიორიტეტის დაყენება.

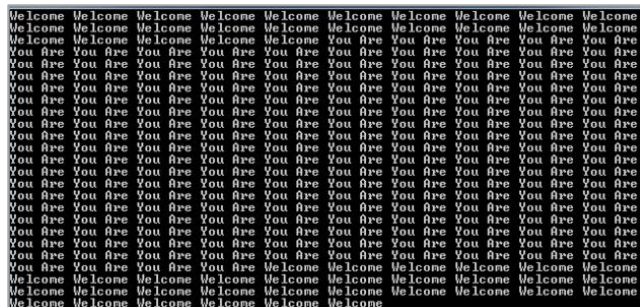
მაგალითი 1. განვიხილოთ მარტივი მაგალითი ნაკადის (კლასი Thread) გამოყენებით. მიაქცეთ ყურადღება იმას, რომ ნაკადის შექმნის დროს კონსტრუქტორს გადაეცემა მეთოდი, ხოლო ნაკადის გასაშვებად გამოიყენება მეთოდი `Start()` (იხ. ნახ. 133.).

```
using System;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
namespace ConsoleApp2
{
    class Program
    {
        static void Main(string[] args)
        {
            // მეთოდის გადაცემა.
            Thread t = new Thread(YYYY);
            // მეთოდის YYYY-ის შესრულება ახალ ნაკადში.
            t.Start();
            while (true)
            {
                // 'You Are '-ის ბეჭდვა.
                Console.Write("You Are ");
            }
            static void YYYY()
            {
                while (true)
                {
                    // 'Welcome '-ის ბეჭდვა.
                    Console.Write("Welcome ");
                }
            }
        }
    }
}

```



ნახ. 133. მაგალითი 1-ის შესრულების შედეგი. კლასი Thread.

მთავარ ნაკადში იქმნება ნაკადი t. ნაკადი t ამუშავებს იმ მეთოდს, რომელიც განუწყვეტლივ ბეჭდავს სიტყვას: Welcome. ნაკად t-სთან ერთდროულად მთავარი ნაკადი განუწყვეტლივ ბეჭდავს სიტყვას: You Are.

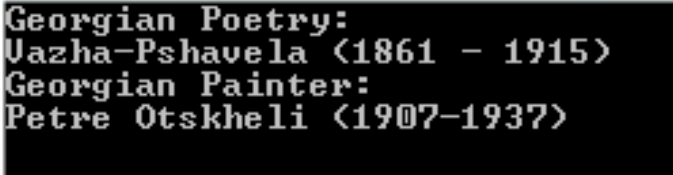
მაგალითი 2. ნაკადის გამოყენების მეორე მაგალითი (იხ. ნახ. 134.).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
namespace ConsoleApp2
{
    class Program
    {
        static void Main(string[] args)
        {
            // ნაკადის შექმნა. ნაკადს გადაეცემა ფუნქცია func_1
            Thread thread_1 = new Thread(func_1);
            // ნაკადის გაშვება.
            thread_1.Start();
            // ნაკადის შექმნა. ნაკადს გადაეცემა ფუნქცია func_2
            Thread thread_2 = new Thread(func_2);
            // ნაკადის გაშვება.
            thread_2.Start();
            Console.ReadKey();
        }
        // ფუნქცია func_1
        public static void func_1()
        {
            Console.WriteLine("Georgian Painter: ");
            Console.WriteLine("Petre Otskheli (1907-1937)");
        }
        // ფუნქცია func_2
        public static void func_2()
```

```

{
    Console.WriteLine("Georgian Poetry: ");
    Console.WriteLine("Vazha-Pshavela (1861 - 1915)");
}
}
}

```



```

Georgian Poetry:
Vazha-Pshavela (1861 - 1915)
Georgian Painter:
Petre Otskheli (1907-1937)

```

ნახ. 134. მაგალითი 2-ის შესრულების შედეგი. კლასი Thread.

კლასი Thread-ის ხშირად გამოყენებადი წევრებია:

- Thread(delegate) - კლასი Thread-ის ობიექტის შექმნა. პარამეტრის სახით გადაეცემა დელეგატი, რომელიც მიუთითებს ნაკადის შემსრულებელ მეთოდზე.
- Sleep() - აჩერებს ნაკადს მითითებული დროზე (დრო ითვლება მილიწამებში).
1 წამი = 1000 მილიწამი.

შესრულების ნაკადის მდგომარეობები. იმისათვის, რომ გავარკვიოთ, თუ რა მდგომარეობაში იმყოფება ნაკადი უნდა გამოვიყენოთ თვისება ThreadState. თვისება ThreadState-ის ზოგადი ფორმაა:

```
public ThreadState ThreadState { get; }
```

ნაკადი თავისი არსებობის მანძილზე შეიძლება რამდენიმე მდგომარეობაში იმყოფებოდეს. განვიხილოთ ნაკადის მდგომარეობები:

ThreadState. Unstarted - ნაკადს შესრულება არ დაუწყია;

ThreadState. Running - ნაკადი სრულდება;

ThreadState. Background - ნაკადი ფონურ რეჟიმში სრულდება;

ThreadState. WaitSleepJoin - ნაკადი ბლოკირებულია Wait, Sleep ან Join მეთოდის მიერ;

ThreadState. SuspendRequested - მოთხოვნილია ნაკადის დროებით შეჩერება;

ThreadState. Suspended - ნაკადი დროებით შეჩერებულია;

ThreadState. StopRequested - მოთხოვნილია ნაკადის გაჩერება;

ThreadState. Stopped - ნაკადი გაჩერებულია;

ThreadState. AbortRequested - მოთხოვნილია ნაკადის გაუქმება;

ThreadState. Aborted - ნაკადი გაუქმებულია.

შექმნის შემდეგ ნაკადს აქვს Unstarted მდგომარეობა. Start() მეთოდის გამოძახების შემდეგ ნაკადი Running მდგომარეობაში გადადის. თუ ნაკადის IsBackground თვისებას true მნიშვნელობას მივანიჭებთ, მაშინ ის ფონის რეჟიმში იმუშავებს. ნაკადი დროის ერთსა და იმავე მომენტში შეიძლება რამდენიმე მდგომარეობაში იმყოფებოდეს, მაგალითად, WaitSleepJoin და AbortRequested მდგომარეობებში.

შესრულების ნაკადის პრიორიტეტები. ნაკადების შექმნისას თითოეულ მათგანს ენიჭება პრიორიტეტი Normal. პროგრამის გამშვები მეთოდი Main(), არის ერთადერთი პრიორიტეტული ანუ მთავარი მეთოდი, რომელიც ავტომატურად იქმნება CLR-ის (Common Language Runtime) და ოპერაციული სისტემის მიერ. არსებობს პრიორიტეტების ხუთი დონე:

- Lowest - უდაბლესი
- BelowNormal - ნორმალურზე დაბალი
- Normal - ნორმალური
- AboveNormal - ნორმალურზე მაღალი
- Highest - უმაღლესი

თითოეულ ნაკადს აქვს თვისება IsBackground. თვისებამ შეიძლება მიიღოს მნიშვნელობა true ან false. თუ ნაკადის ობიექტის თვისება IsBackground არის true, მაშინ ნაკადი არის ფონურ რეჟიმში. თუ ნაკადის ობიექტის თვისება IsBackground არის false-ს, მაშინ ნაკადი პრიორიტეტულია.

ჩუმათობის პრინციპის მიხედვით, ობიექტებისთვის თვისება IsBackground არის false მდგომარეობაში, ამიტომ ყველა ნაკადი, რომელიც ადრე შევქმენით, პრიორიტეტული იყო. თუ გვინდა ფონური ნაკადი შევქმნათ, მაშინ ამ ნაკადის ობიექტის IsBackground თვისებას უნდა მივანიჭოთ მნიშვნელობა true. ფონური ნაკადი ითვლება მეორეხარისხოვნად.

მაგალითი 3. ფონური ნაკადის შექმნის დემონსტრირება (იხ. ნახ. 135.).

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;
```

```

using System.Threading.Tasks;
using System.Threading;
namespace ConsoleApp2
{
    // კლასი მთავარი მეთოდით.
    class Program
    {
        // ფონურ რეჟიმში გაშვებული მეთოდი.
        static void shesruleba()
        {
            Console.WriteLine("fonuri nakadis gashveba ***");
            int k = 1;
            while (true) // უსასრულო ციკლი.
            {
                Console.WriteLine("fonuri nakadi - " + k);
                k++;
                Thread.Sleep(2100); // ნაკადის შესრულების დაყოვნება.
            }
        }
        static void Main(string[] args)
        {
            Console.WriteLine("mtavari nakadis gashveba *****");
            Thread ttt = new Thread(shesruleba);
            ttt.IsBackground = true; // ფონური ნაკადი.
            ttt.Start(); // ფონური ნაკადის გაშვება.
            for (char s = 'A'; s <= 'F'; s++)
            {
                Console.WriteLine("mtavari nakadi - " + s);
                // მთავარი ნაკადის შესრულების დაყოვნება.
                Thread.Sleep(1000);
            }
            Console.WriteLine("mtavari nakadis shesruleba damtavrda!
By!!!");
            Console.ReadLine();
        }
    }
}

```

```
}
}
```

```
mtavari nakadis gashveba *****
mtavari nakadi - A
fonuri nakadis gashveba ***
fonuri nakadi - 1
mtavari nakadi - B
mtavari nakadi - C
fonuri nakadi - 2
mtavari nakadi - D
mtavari nakadi - E
fonuri nakadi - 3
mtavari nakadi - F
mtavari nakadis shesruleba damtavrda! By!!!
fonuri nakadi - 4
fonuri nakadi - 5
fonuri nakadi - 6
fonuri nakadi - 7
fonuri nakadi - 8
```

ნახ. 135. მაგალითი 3-ის შესრულების შედეგი. კლასი Thread.

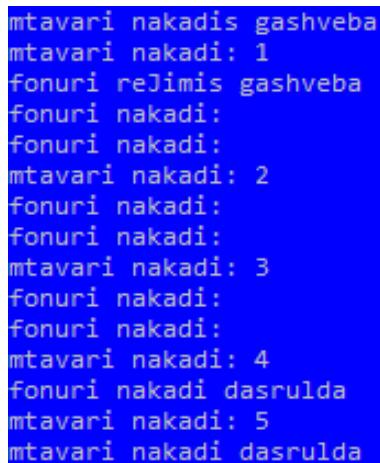
მაგალითი 4. ფონური ნაკადის შექმნის მეორე ნიმუში სტატიკური მეთოდის გამოყენებით (იხ. ნახ. 136.).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
namespace ConsoleApp2
{ // კლასი მთავარი მეთოდით
    class Program
    {
        // ფონურ რეჟიმში გაშვებული მეთოდი
        static void shesruleba()
        {
            Console.WriteLine("fonuri reJimis gashveba ");
            for (int k = 0; k <= 5; k++)
            {
                Console.WriteLine("fonuri nakadi: ", (char)('A' + k));
                // დაყოვნება 1 წამი.
                Thread.Sleep(1000);
            }
        }
    }
}
```

```

        Console.WriteLine("fonuri nakadi dasrulda");
    }
    static void Main(string[] args)
    {
        Console.WriteLine("mtavari nakadis gashveba");
        Thread mt = new Thread(shesruleba);
        // ფონური ნაკადის გაშვება.
        mt.Start();
        for (int k = 1; k <= 5; k++)
        {
            Console.WriteLine("mtavari nakadi: " + k);
            // დაყოვნება 2 წამი.
            Thread.Sleep(2000);
        }
        Console.WriteLine("mtavari nakadi dasrulda ");
        // დაყოვნება.
        Console.ReadLine();
    }
}

```



```

mtavari nakadis gashveba
mtavari nakadi: 1
fonuri nakadi:
fonuri nakadi:
mtavari nakadi: 2
fonuri nakadi:
fonuri nakadi:
mtavari nakadi: 3
fonuri nakadi:
fonuri nakadi:
mtavari nakadi: 4
fonuri nakadi:
fonuri nakadi:
mtavari nakadi: 5
fonuri nakadi:
fonuri nakadi dasrulda

```

ნახ. 136. მაგალითი 4-ის შესრულების შედეგი. კლასი Thread.

მაგალითი 5. ფონური ნაკადის შექმნის მესამე ნიმუში ჩვეულებრივი მეთოდის გამოყენებით (იხ. ნახ. 137.).

```

using System;
using System.Collections.Generic;

```



```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
namespace ConsoleApp2
{
    // კლასი მთავარი მეთოდით.
    class MyClass
    {
        // შეტყობინების რაოდენობა.
        private int count;
        // დაყოვნება შეტყობინებებს შორის.
        private int time;
        private string text;
        public void show()
        {
            Console.WriteLine(text + " dasawyisi ***");
            for (int i = 1; i <= count; i++)
            {
                Console.WriteLine(text + " -> " + i);
                // დაყოვნება ნაკადის შესრულების დროს.
                Thread.Sleep(time);
            }
            Console.WriteLine(text + " dasruleba ***");
        }
        public MyClass(int c, int t, string txt)
        {
            count = c;
            time = t;
            text = txt;
        }
    }
    // ძირითადი კლასი.
    class Program

```

```

{
    static void Main()
    {
        Console.WriteLine("mtavari nakadi: dasawyisi ");
        // ობიექტების შექმნა:
        MyClass objA = new MyClass(5, 2000, "nakadi A");
        MyClass objB = new MyClass(6, 1200, "nakadi B");
        MyClass objC = new MyClass(8, 1000, "nakadi C");
        // ნაკადის ობიექტის შექმნა.
        Thread A = new Thread(objA.show);
        Thread B = new Thread(objB.show);
        Thread C = new Thread(objC.show);
        // ფონური ნაკადების გაშვება:
        A.Start();
        B.Start();
        C.Start();
        Console.WriteLine("mtavari nakadi: dasruleba by ");
        // დაყოვნება.
        Console.ReadLine();
    }
}

```

```

mtavari nakadi: dasawyisi
nakadi A dasawyisi ***
nakadi A -> 1
nakadi B dasawyisi ***
nakadi B -> 1
mtavari nakadi: dasruleba by
nakadi C dasawyisi ***
nakadi C -> 1
nakadi C -> 2
nakadi B -> 2
nakadi A -> 2
nakadi C -> 3
nakadi B -> 3
nakadi C -> 4
nakadi B -> 4
nakadi A -> 3
nakadi C -> 5
nakadi B -> 5
nakadi C -> 6
nakadi A -> 4
nakadi C -> 7
nakadi B -> 6
nakadi C -> 8
nakadi B dasruleba ***
nakadi A -> 5
nakadi C dasruleba ***
nakadi A dasruleba ***

```

ნახ. 137. მაგალითი 5-ის შესრულების შედეგი. კლასი Thread.

პროგრამული კოდიდან ჩანს, რომ მთავარი ნაკადის პარამეტრები ისე არის არჩეული, რომ იგი უფრო ადრე შესრულდება, ვიდრე ფონური ნაკადი. ან არა და ვიცით, რომ ჩუმათობის პრინციპის მიხედვით, ფონურ ნაკადს აქვს პრიორიტეტი და ისინი განაგრძობენ მუშაობას მას შემდეგაც, რაც დასრულდება ძირითადი ნაკადის მუშაობა.

ახლა კი გადავიდეთ ამოცანების განხილვაზე.

ამოცანა 18.1. მიმდინარე ნაკადის შესახებ ინფორმაციის მიღება (იხ. ნახ. 138).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
namespace ConsoleApp4
{
    class Program
    {
        static void Main(string[] args)
        {
            // მიმდინარე ნაკადის მიღება.
            Thread currentThread = Thread.CurrentThread;
            //ნაკადის სახელი.
            Console.WriteLine($"nakadis saxeli - {currentThread.Name}");
            currentThread.Name = "meTodi Main";
            Console.WriteLine($"nakadis saxeli - {currentThread.Name}");
            Console.WriteLine($"gashvebulia Tu ara nakadi -
{currentThread.IsAlive}");
            Console.WriteLine($"nakadia Id -
{currentThread.ManagedThreadId}");
            Console.WriteLine($"nakadis prioriteti -
{currentThread.Priority}");
            Console.WriteLine($"nakadis statusi -
{currentThread.ThreadState}");
            Console.ReadLine();
        }
    }
}
```

```

    }
}
}

```

```

nakadis saxeli -
nakadis saxeli - meTodi Main
gashvebulia Tu ara nakadi - True
nakadia Id - 1
nakadis prioriteti - Normal
nakadis statusi - Running

```

ნახ. 138. ამოცანა 18.1. შესრულების შედეგი. კლასი Thread.

ამოცანა 18.2. Thread (delegate). ნაკადებთან მუშაობა დელეგატის გამოყენებით.

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Thread_delegati
{
    class One
    {
        public void Print()
        {
            for(int i=0; i<25; i++)
            {
                Console.Write("1");
            }
            Console.WriteLine();
        }
    }

    class Two
    {
        public void Print()
        {
            for(int i=0; i<25; i++)
            {
                Console.Write("2");
            }
            Console.WriteLine();
        }
    }
}

```

```

    }
}
class Program
{
    // მეთოდი Main() პრიორიტეტული ნაკადია და ამიტომ არსებობს
    ჩუმათობით:
    public static int Main() {
        // ძირითადი ნაკადი გასცემს ციფრს 1
        One o = new One();
        o.Print();

        // მომდევნო ნაკადი გასცემს ციფრს 2
        // კონსტრუქტორს გადაეცემა დელეგატი, რომელიც დამალულია და იღებს
        მეთოდის სახელს.
        Two T = new Two();
        Thread Th = new Thread(T.Print);
        // ნაკადის შემსრულებელი მეთოდი.
        Th.Start();
        Console.ReadKey();
        return 0;
    }
}

```

ამოცანა 18.3. მეთოდი Sleep(). ნაკადის შეჩერება მითითებული დროის მიხედვით.

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Thread_delegati
{
    class Program {
        public static int Main() {
            string str = "dila mshvidobisa!";
            // თითოეული ასო_ნიშნის გამოტანა 0.4 მილიწამის დაყოვნებით.

```

```

foreach(var i in str)
{
    Console.Write(i);
    Thread.Sleep(400);
}
Console.WriteLine();
Console.ReadKey();
return 0;
}
}

```

დამოუკიდებელი სამუშაო

დავალება 1. დაწერეთ პროგრამა, რომელშიც ორი ნაკადია. ნაკადების შევსება უნდა მოხდეს მთელი რიცხვებით. ერთი ნაკადი ახდენს მასივის შევსებას შემდეგი თანამიმდევრობით: 2^0 , 2^1 , 2^2 და ა.შ. მეორე ნაკადი კი ახდენს მასივის შევსებას შემდეგი თანამიმდევრობით: 3^0 , 3^1 , 3^2 და ა.შ.

19 LINQ ტექნოლოგია. მეთოდის სინტაქსი და ლამბდა გამოსახულება. დიდი ზომის მონაცემებთან მუშაობა. აგრეგირების ოპერატორები: Average(), Sum(), Max(), Min().

ტექნოლოგია LINQ (Language-Integrated Query) არის მარტივი ენა მონაცემთა წყაროსთან მიმართებაში მოთხოვნების დასამუშავებლად. წყარო შეიძლება იყოს ობიექტი, რომელიც ინტერფეის IEnumerable-ის რეალიზებას ახდენს, მონაცემთა ნაკრები DataSet, დოკუმენტი XML.

რა სარგებლობა მოაქვს ტექნოლოგია LINQ-ს?

მოდით განვიხილოთ მარტივი კოდი:

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "";
    string[] adamiani = { "ლუკა", "ნიკოლოზი", "ნინო", "ნათია",
                          "დავითი", "ალექსანდრე" };

    var selectedadamianebi = new List<string>();
    foreach (string adamianebi in adamiani)
    {
        if (adamianebi.ToUpper().StartsWith("ნ"))
            selectedadamianebi.Add(adamianebi);
    }
    // დახარისხება.
    selectedadamianebi.Sort();
    foreach (string adamianebi in selectedadamianebi)
        label1.Text += adamianebi + '\n';
}
```

ეს კოდი შეგვიძლია გადავაკეთოთ ტექნოლოგია LINQ-ის გამოყენებით. ტექნოლოგია LINQ-ის ოპერატორები ძალიან ჰგავს sql-მოთხოვნებს:

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    string[] adamiani = { "ლუკა", "ნიკოლოზი", "ნინო", "ნათია",
                          "დავითი", "ალექსანდრე" };
    var selectedadamianebi = from p in adamiani where
```

```
p.ToUpper().StartsWith("ბ") orderby p select p;
foreach (string adamianebi in selectedadamianebi)
    label1.Text += adamianebi + '\n';
}
```

მოთხოვნა LINQ შედგება სამი ნაწილისაგან:

1. მონაცემთა წყაროს მიღება;
2. მოთხოვნის შექმნა;
3. მოთხოვნის შესრულება.

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    //მონაცემთა წყარო.
    int[] masivi = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
    //მოთხოვნის შექმნა.
    var cxrili = from val in masivi select val;
    //მოთხოვნის შესრულება
    foreach (var i in cxrili)
    {
        label1.Text += i + " ";
    }
}
```

მეთოდის სინტაქსი და ლამბდა გამოსახულება. მოთხოვნა LINQ-ის სინტაქსი არის ორი სახის: პირველი - მოთხოვნების სინტაქსი და მეორე - მეთოდის სინტაქსი. განვიხილოთ მოთხოვნების სინტაქსის მაგალითი:

```
int[] masivi = { -11, 21, 3, 8, -14 };
var shedegi = from n in masivi
    where n > 0 select n; // 21, 3, 8
```

განვიხილოთ მეთოდის სინტაქსის მაგალითი:

```
int[] masivi = { -11, 21, 3, 8, -14 };
var res = masivi.Where(n => n > 0); // 21, 3, 8
```

მოთხოვნა LINQ-ის სტანდარტული მეთოდები რეალიზებული არიან როგორც გაფართოებული მეთოდები, რომელთა უმეტესობაც პარამეტრად იღებს ლამბდა გამოსახულებას ან დელეგატს.

ლამბდა გამოსახულება. ლამბდა გამოსახულება გამოიყენება ანონიმური მეთოდების მოკლე და გამარტივებული სახით ჩასაწერად.

ლამბდა ოპერატორს აქვს შემდეგი სინტაქსი: (პარამეტრები) => გამოსახულება
ლამბდა ოპერატორის (=>) მარცხნივ იწერება შეტანის პარამეტრი ან პარამეტრები, ხოლო მარჯვნივ იწერება გამოსახულება, რომელიც იყენებს ამ პარამეტრებს.

ლამბდა გამოსახულების მაგალითი: $n \Rightarrow n * n$

ეს გამოსახულება იღებს ერთ პარამეტრს n -ს და აბრუნებს $n * n$.

```
int[] masivi = { 1, 2, 3 };
```

```
var res = masivi.Select(n => n * n); // 1, 4, 9
```

კოდში მოცემული ლამბდა გამოსახულება შეგვიძლია ჩავწეროთ შემდეგი სახითაც: $n \Rightarrow \{ \text{return } n * n; \}$ ან შემდეგი სახითაც `delegate(int n) { return n * n; }`

მონაცემთა ტიპის მიხედვით ლამბდა გამოსახულება არის დელეგატი.

აგრეგირების ოპერატორები.

LINQ მოთხოვნას აქვს აგრეგირების ოპერატორები, რომლებიც იძლევიან შედეგის ანალიზის საშუალებას. ხშირად გამოყენებადი ოპერატორებია: `Average()`, `Count()`, `Max()`, `Min()`, `Sum()`. განვიხილოთ ისინი.

1. `Average()` - გასცემს რიცხვების საშუალო არითმეტიკულს:

```
private void button1_Click(object sender, EventArgs e){  
    int[] masivi = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };  
    double mnishvneloba = masivi.Average();  
    label1.Text = mnishvneloba.ToString() + '\n'; //6  
}
```

2. `Count()` - გასცემს მონაცემების რაოდენობას, დასაბრუნებელი ტიპი არის `int`;

```
private void button1_Click(object sender, EventArgs e) {  
    int[] masivi = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };  
    double mnishvneloba = masivi.Count();  
    label1.Text = mnishvneloba.ToString() + '\n'; //11  
}
```

3. `Max()` - გასცემს მონაცემების მაქსიმალურ მნიშვნელობას;

```
private void button1_Click(object sender, EventArgs e){  
    int[] masivi = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };  
    double mnishvneloba = masivi.Max();
```

```

        label1.Text = mnishvneloba.ToString() + '\n'; //11
    }

```

4. Min() - გასცემს მონაცემების მინიმალურ მნიშვნელობას;

```

private void button1_Click(object sender, EventArgs e){
    int[] masivi = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
    double mnishvneloba = masivi.Min();
    label1.Text = mnishvneloba.ToString() + '\n'; //1
}

```

5. Sum() - გასცემს მონაცემების ჯამს;

```

private void button1_Click(object sender, EventArgs e){
    int[] masivi = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
    double mnishvneloba = masivi.Sum();
    label1.Text = mnishvneloba.ToString() + '\n'; //66
}

```

ანონიმური ტიპი var. C# ენაში ცვლადის გამოცხადება ხდება შემდეგნაირად: თავიდან უნდა მივუთითოთ ტიპი, ხოლო შემდეგ ცვლადის სახელი.

მაგალითად, string str = "Hello"; მოცემულ მაგალითში ცვლადის ტიპია - string, ცვლადის სახელია - str, მაგრამ C# ენაში ასევე შესაძლებელი გახდა კომპილატორისათვის ცვლადის ტიპის დადგენის დავალება. ამ ამოცანას ასრულებს var საკვანძო სიტყვა. ცვლადის ტიპს var-ს შეუძლია ნებისმიერი ტიპის მნიშვნელობა მიიღოს (თუმცა C# ენაში არსებობს var ტიპთან მიმართებით გარკვეული სახის შეზღუდვები). მას ასევე უწოდებენ ანონიმურ ტიპს.

ამდენად, C# ენა გვაძლევს საშუალებას, გამოვაცხადოთ არაცხადად ტიპიზირებული ცვლადი var საკვანძო სიტყვის დახმარებით.

var ტიპის ცვლადი აუცილებლად უნდა იყოს ინიციალიზირებული ე. ი. გამოცხადებისთანავე უნდა მივანიჭოთ მნიშვნელობა. var არის მკაცრად ტიპიზებული ტიპი.

მაგალითად, var x = 10; გამოსახულების გაანალიზებისას კომპილატორი თავისით დაადგენს x ცვლადის ტიპს.

ზემოთ მოყვანილი გამოსახულება var x = 10; ტოლფასია int x = 10; გამოსახულების.

```

var list = new List(); // list მიეკუთვნება List ტიპს.

```

```
var count = 7; // count მიეკუთვნება int ტიპს.  
var greeting = "გამარჯობა"; // greeting მიეკუთვნება string ტიპს.  
var varr = 'a'; // varr მიეკუთვნება char ტიპს.  
greeting = 9; // შეცდომაა, რადგან greeting არის string ტიპის.
```

როდის შეგვიძლია გამოვიყენოთ var საკვანძო სიტყვა?

var საკვანძო სიტყვა შეგვიძლია გამოვიყენოთ ანონიმური ტიპის ცვლადების ან კოლექციის გამოცხადებისას. ანონიმურ ტიპებს არ გააჩნიათ სახელი, რომლითაც შეგვეძლო მათთვის მიგვემართა.

ასევე var საკვანძო სიტყვის new სიტყვასთან გამოყენებისას შეგვიძლია შევქმნათ არაცხადი (ანონიმური) ტიპი. ეს ტიპი წარმოადგენს უსახელო object კლასიდან წარმოებულ კლასს. მაგალითად:

```
var adamiani = new { Saxeli = "Shota", Gvari = "Rustaveli" };
```

გამოსახულება ქმნის ობიექტს Saxeli და Gvari თვისებებით.

თუ ჩვენ შევქმნით სხვა ობიექტს:

```
var pexburteli = new { Saxeli = "Shota", Gvari = "Arveladze"};
```

მაშინ adamiani და pexburteli ტიპი ერთი და იგივე ტიპს გამოხატავს. ამ შემთხვევაში გამოსახულება adamiani = pexburteli; დასაშვებია; ამ ობიექტების ტიპის სახელი არ არის ცნობილი. კომპილატორი თავისით იგონებს ტიპის სახელს და მხოლოდ მას შეუძლია მისი გამოყენება.

არ არის რეკომენდებული var სიტყვის გამოყენება როგორც კლასის სახელი, რადგან ასეთ შემთხვევაში ის თავის სტანდარტულ ფუნქციას ვეღარ შეასრულებს.

მაგალითი 1. var ტიპის გამოყენების მაგალითები:

მაგალითი 1.1.

```
// int ტიპის მნიშვნელობა.  
var age = 25;  
label1.Text = age.ToString();  
// string ტიპის მნიშვნელობა.  
var name = "ანდერს ჰეილსბერგი";  
label2.Text = name;
```

მაგალითი 1.2.

```
// array ტიპის მნიშვნელობა.
```

```

var odds = new[] { 1, 3, 5, 7, 9 };
foreach (var num in odds)
{
    label1.Text += num + " ";
}

```

მაგალითი 1.3.

```

for (var x = 1; x < 10; x++)
{
    label1.Text += x + " ";
}

```

მაგალითი 1.4.

```

// var ტიპი შეგვიძლია გამოვიყენოთ using ოპერატორთან.
using (var file = new
System.IO.StreamReader(@"C:\Windows\teqsti.txt"))
{
    string line;
    while ((line = file.ReadLine()) != null)
    {
        label1.Text += line;
    }
}

```

მაგალითი 1.5.

```

label1.Text = "";
string[] authors={ "ნინო", "გიორგი", "ანა", "მარიამი", "ნიკოლოზი" };
var authorQ = from author in authors
               where author[0] == 'ნ'
               select author;
foreach (var a in authorQ)
{
    label1.Text += a + '\n';
}

```

გადავიდეთ ამოცანების განხილვაზე.

ამოცანა 19.1. მოთხოვნის სინტაქსის დემონსტრირება. მოთხოვნის სინტაქსის აუცილებლად უნდა დაწყოს from მნიშვნელობით in წყაროთი და დასრულდეს select ან group.

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    int[] masivi = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
    //მოთხოვნის შექმნა.
    var query = from val in masivi
    where (val % 2 == 0) && (val > 6)
    select val;
    //მოთხოვნის შესრულება.
    foreach (var i in query)
    {
        label1.Text += i + " "; // 8 10
    }
}
```

ამოცანა 19.2. მეთოდის სინტაქსის დემონსტრირება.

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    int[] masivi = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
    //მოთხოვნის შექმნა.
    var query = masivi.Where(val=>(val % 2==0) && (val>6)).Select(val => val);
    //მოთხოვნის შესრულება.
    foreach (var i in query)
    {
        label1.Text += i + " "; // 8 10
    }
}
```

ამოცანა 19.3. მოთხოვნა select-ის მარტივი ფორმა.

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    int[] masivi = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
    //მოთხოვნის შექმნა.
```

```

var query = from val in masivi where (val % 2 == 0) && (val > 6)
select val;

//მოთხოვნის შესრულება.
foreach (var i in query)
{
    label1.Text += i + " "; // 8 10
}
}

```

ამოცანა 19.3. მოთხოვნა select-ის ანონიმური ტიპის სახით (იხ. ნახ. 139.).

```

private void button1_Click(object sender, EventArgs e){
label1.Text = "";

//მონაცემთა წყარო მასივის სახით.
int[] masivi = new int[] { 0, 1, 2, 3, 4, 5, 6, 7 };
var query = from mnish in masivi select new //მოთხოვნის შექმნა.
    { mnish, a = 2, shedegi = mnish * 2 };

//მოთხოვნის შესრულება.
foreach (var i in query)
{
    label1.Text += i.mnish + " * " + i.a + " = " + i.shedegi + "
\n ";
}
}

```

```

0 * 2 = 0
1 * 2 = 2
2 * 2 = 4
3 * 2 = 6
4 * 2 = 8
5 * 2 = 10
6 * 2 = 12
7 * 2 = 14

```

ნახ. 139. ამოცანა 19.3.-ის შესრულების შედეგი

ამოცანა 19.4. მონაცემების დალაგება ზრდადობით (იხ. ნახ. 140.).

```

private void button1_Click(object sender, EventArgs e){
label1.Text = ""; label2.Text = "";
label3.Text = ""; label4.Text = "";

//მონაცემთა წყარო მასივების სახით.

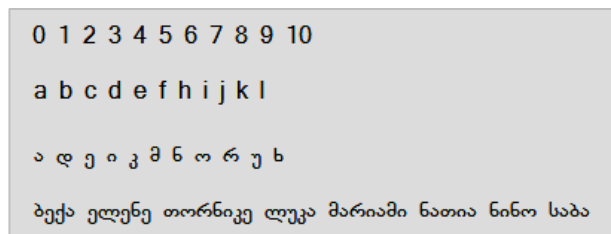
```

```

int[] masivi1 = new int[] { 7, 4, 1, 3, 8, 2, 5, 9, 0, 10, 6 };
string[] masivi2 = new string[] { "e", "j", "l", "b", "k", "d", "h",
"a", "f", "c", "i" };
string[] masivi3 = new string[] { "ნ", "რ", "კ", "დ", "ხ", "ა", "ო",
"ო", "უ", "ე", "მ" };
string[] masivi4 = new string[] { "ნინო", "ლუკა", "ელენე",
"მარიამი", "თორნიკე", "საბა", "ბექა", "ნათია" };

//მოთხოვნის შექმნა.
var query1 = from val in masivi1 orderby val select val;
//მოთხოვნის შესრულება.
foreach (var i in query1) label1.Text += i + " ";
//მოთხოვნის შექმნა.
var query2 = from val in masivi2 orderby val select val;
//მოთხოვნის შესრულება.
foreach (var i in query2) label2.Text += i + " ";
//მოთხოვნის შექმნა.
var query3 = from val in masivi3 orderby val select val;
//მოთხოვნის შესრულება.
foreach (var i in query3) label3.Text += i + " ";
//მოთხოვნის შექმნა.
var query4 = from val in masivi4 orderby val select val;
//მოთხოვნის შესრულება.
foreach (var i in query4) label4.Text += i + " ";
}

```



```

0 1 2 3 4 5 6 7 8 9 10
a b c d e f h i j k l
ა დ ე ი კ მ ნ ო რ უ ხ
ბექა ელენე თორნიკე ლუკა მარიამი ნათია ნინო საბა

```

ნახ. 140. ამოცანა 19.4.-ის შესრულების შედეგი

ამოცანა 19.5. მონაცემების დაჯგუფება პირველი სიმბოლოს მიხედვით (იხ. ნახ. 141.).

```

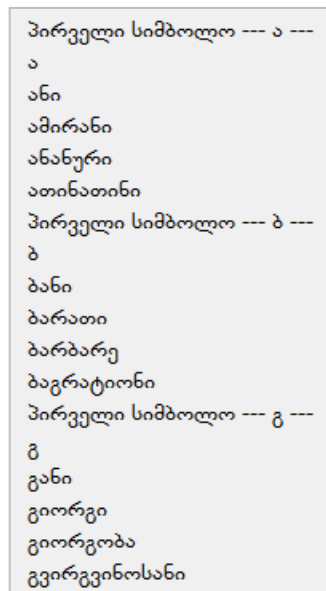
private void button1_Click(object sender, EventArgs e){
label1.Text = ""; label2.Text = "";

```

```

label3.Text = ""; label4.Text = "";
//მონაცემთა წყარო მასივის სახით.
string[] masivi = new string[] {
    "ა", "ანი", "ამირანი", "ანანური", "ათინათინი",
    "ბ", "ბანი", "ბარათი", "ბარბარე", "ბაგრატიონი",
    "გ", "განი", "გიორგი", "გიორგობა", "გვირგვინოსანი"
};
//მოთხოვნის შექმნა.
var query = from val in masivi group val by val[0] into simbolo
select new { pirvelisim = simbolo.Key, word =simbolo };
foreach (var i in query) //მოთხოვნის შესრულება.
{
label1.Text += "პირველი სიმბოლო --- " + i.pirvelisim + " ---\n";
    {
        foreach (var j in i.word) label1.Text += j + "\n";
    }
}
}

```



```

პირველი სიმბოლო --- ა ---
ა
ანი
ამირანი
ანანური
ათინათინი
პირველი სიმბოლო --- ბ ---
ბ
ბანი
ბარათი
ბარბარე
ბაგრატიონი
პირველი სიმბოლო --- გ ---
გ
განი
გიორგი
გიორგობა
გვირგვინოსანი

```

ნახ. 141. ამოცანა 19.5.-ის შესრულების შედეგი

ამოცანა 19.6. LINQ მოთხოვნის აგრეგირების ოპერატორები (იხ. ნახ. 142.).

```

private void button1_Click(object sender, EventArgs e){
    int[] numberList = new int[5] { 1, 2, 3, 4, 5 };

```



```

        listBox1.Items.Add("მასივის ელემენტები: 1, 2, 3, 4, 5");
listBox1.Items.Add("=====");
        int listTotal = numberList.Sum();
        listBox1.Items.Add("მასივის ელემენტების ჯამი = " + listTotal);
        int lowestNumber = numberList.Min();
        listBox1.Items.Add("უმცირესი = " + lowestNumber);
        int highestNumber = numberList.Max();
        listBox1.Items.Add("უდიდესი = " + highestNumber);
        double averageValue = numberList.Average();
        listBox1.Items.Add("საშუალო არითმეტიკული = " + averageValue);
        bool doesContain = numberList.Contains(3);
        listBox1.Items.Add("შეიცავს ციფრს სამიანს? = " + doesContain);
        int elementValue = numberList.ElementAt(1);
        listBox1.Items.Add("პირველ პოზიციაზე მყოფი ელემენტი = " +
elementValue);
        int firstElement = numberList.First();
        listBox1.Items.Add("მასივის პირველი ელემენტის მნიშვნელობა=" +
firstElement);
        int lastElement = numberList.Last();
        listBox1.Items.Add("მასივის ბოლო ელემენტის მნიშვნელობა = " +
lastElement);
listBox1.Items.Add("=====");
        listBox1.Items.Add("მასივის ელემენტები: 1, 1, 2, 2, 3, 4, 5, 5");
        listBox1.Items.Add("დუბლირებული ელემენტის ამოშლა");
listBox1.Items.Add("=====");
        int[] aryNums = new int[8] { 1, 1, 2, 2, 3, 4, 5, 5 };
        var distinctNums = aryNums.Distinct();
        foreach (var num in distinctNums)
        {
            listBox1.Items.Add(num.ToString() + " ");
        }
    }
}

```

```

მასივის ელემენტები: 1, 2, 3, 4, 5
=====
მასივის ელემენტების ჯამი = 15
უმცირესი = 1
უდიდესი = 5
საშუალო არითმეტიკული = 3
შეიცავს ციფრს სამიანს? = True
პირველ პოზიციაზე მყოფი ელემენტი = 2
მასივის პირველი ელემენტის მნიშვნელობა = 1
მასივის ბოლო ელემენტის მნიშვნელობა = 5
=====
მასივის ელემენტები: 1, 1, 2, 2, 3, 4, 5, 5
დუბლირებული ელემენტის ამოშლა
=====
1
2
3
4
5

```

ნახ. 142. ამოცანა 19.6.-ის შესრულების შედეგი

დამოუკიდებელი სამუშაო

1. ქვემოთ მოცემულ კოდში არის შეცდომა. აღმოაჩინეთ იგი, მოცემულობა დაწერეთ დამოუკიდებლად და გადმოიტანეთ კოდის სწორი ვარიანტი შესრულების შემდეგ.

```

private void button1_Click(object sender, EventArgs e) {
    int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };
    List<User> users = new List<User>()
    {
        new User { Name = "ნინო", Age = 23 },
        new User { Name = "მირიანი", Age = 43 },
        new User { Name = "ალექსანდრე", Age = 35 } };
    int sum1 = numbers.Sum();
    decimal sum2 = users.Sum(n => n.Age);
    label1.Text = sum1.ToString();
    label2.Text = sum2.ToString();
}

```

2. ქვემოთ მოცემულ კოდში არის შეცდომა. აღმოაჩინეთ იგი, მოცემულობა დაწერეთ დამოუკიდებლად და გადმოიტანეთ კოდის სწორი ვარიანტი შესრულების შემდეგ.

```

private void button1_Click(object sender, EventArgs e) {
    int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };
    List<User> users = new List<User>()
    {
        new User { Name = "Tom", Age = 23 },
        new User { Name = "Sam", Age = 43 },
        new User { Name = "Bill", Age = 35 }
    };
    int min1 = numbers.Min();
    int min2 = users.Min(n => n.Age);
    int max1 = numbers.Max();
    int max2 = users.Max(n => n.Age);
    double avr1 = numbers.Average();
    double avr2 = users.Average(n => n.Age);
}

```

3. მოცემულ კოდში არის შეცდომა. აღმოაჩინეთ იგი, მოცემულობა დაწერეთ დამოუკიდებლად და გადმოიტანეთ კოდის სწორი ვარიანტი შესრულების შემდეგ.

```

int[] nums = { 1, -2, 3, 0, -4, 5 };
var posNums = from n in nums where n > 0 select n;
foreach (int i in posNums)
    nums[1] = 99;
foreach (int i in posNums)

```

20 რთულ ობიექტებთან მუშაობა. მოთხოვნაში ახალი ობიექტის შექმნა. LINQ მეთოდები: Distinct(), All(), Take(), Skip(), First(), Intersect(), Except(), Union(), Joins() მეთოდები.

მე-20 თემას განვიხილავთ პრაქტიკული ამოცანების სახით.

მეთოდი **Distinct()**. მეთოდი აბრუნებს მიმდევრობას, მხოლოდ ერთნაირი ელემენტების გამოკლებით.

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "შედეგი: ";
    int[] mimdevroba=new int[] { 1, 12, 1, 3, 1, 7, 1, 5, 1, 16, 8, 1 };
    var query = mimdevroba.Distinct();
    foreach (var i in query)
    {
        label1.Text += i + " "; //შედეგი: 1 12 3 7 5 16 8
    }
}
```

მეთოდი **All()**. მეთოდი გვებმარება, დავადგინოთ კოლექციის ყველა ელემენტი აკმაყოფილებს ანუ შესაბამება თუ არა პირობას. თუ ყველა ელემენტი აკმაყოფილებს პირობას, მაშინ გაიცემა true.

```
private void button1_Click(object sender, EventArgs e){
    //label1.Text = "შედეგი: ";
    string[] adamianebi = { "გიო", "ანა", "ენა", "მზე" };
    //ვამოწმებთ ყველა ელემენტი შეიცავს თუ არა სამ სიმბოლოს
    bool allHas3Chars = adamianebi.All(s => s.Length == 3);
    label1.Text += allHas3Chars; // true
    // ვამოწმებთ ყველა სიტყვა იწყება თუ არა სიმბოლო "ა"-თი
    bool allStartsWitha = adamianebi.All(s => s.StartsWith("ა"));
    label2.Text += allStartsWitha; // false
}
```

მეთოდი **Take()**. მეთოდი აბრუნებს მიმდევრობის პირველ n ცალ ელემენტს.

```
private void button1_Click(object sender, EventArgs e){
```

```

label1.Text = "შედეგი: ";
int[] mimdevroba = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
                                12, 13, 14, 15, 16, 11};

var query = mimdevroba.Take(7);
foreach (var i in query)
{
    label1.Text += i + " "; //შედეგი: 1 2 3 4 5 6 7 }
}

```

მეთოდი Skip(). მეთოდი გამოტოვებს მიმდევრობის პირველ *n* ცალ ელემენტს და აბრუნებს დანარჩენს.

```

private void button1_Click(object sender, EventArgs e){
label1.Text = "შედეგი: ";
int[] mimdevroba = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
                                12, 13, 14, 15, 16 };

var query = mimdevroba.Skip(7);
foreach (var i in query)
{
    label1.Text += i + " "; //შედეგი: 8 9 10 11 12 13 14 15 16
}
}

```

მეთოდი First(). მეთოდი აბრუნებს მიმდევრობის პირველ ელემენტს.

მეთოდი Last(). მეთოდი აბრუნებს მიმდევრობის ბოლო ელემენტს.

```

private void button1_Click(object sender, EventArgs e){
label1.Text = "შედეგი: "; label2.Text = "შედეგი: ";
int[] mimdevroba = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
                                12, 13, 14, 15, 16 };

var query = mimdevroba.First();
label1.Text += query.ToString() + "\n"; //შედეგი: 1
var query1 = mimdevroba.Last();
label2.Text += query1.ToString(); //შედეგი: 16
}

```

მეთოდი Except(). მეთოდი აბრუნებს მიმდევრობებში არსებულ განსხვავებულ ელემენტებს.

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "შედეგი: ";
    int[] mim1 = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    int[] mim2 = new int[] { 1, 2, 3, 4 };
    var query = mim1.Except(mim2);
    foreach (var i in query)
    {
        label1.Text += i + " "; //შედეგი: 5 6 7 8 9 10 11 12
    }
}
```

მეთოდი Intersect(). მეთოდი აბრუნებს მიმდევრობებში არსებულ ერთნაირ ელემენტებს.

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "შედეგი: ";
    int[] mim1 = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    int[] mim2 = new int[] { 1, 2, 3, 4 };
    var query = mim1.Intersect(mim2);
    foreach (var i in query)
    {
        label1.Text += i + " "; //შედეგი: 1 2 3 4
    }
}
```

მეთოდი Union(). მონაცემების გაერთიანება. მონაცემთა გაერთიანებისათვის გამოიყენება მეთოდი Union. მეთოდი აბრუნებს ახალ გაერთიანებულ მიმდევრობას, როგორც პირველი, ისე მეორე მიმდევრობიდან, ერთნაირი ელემენტების გამეორების გარეშე.

ვარიანტი 1. მეთოდი Union().

```
private void button1_Click(object sender, EventArgs e){
    label1.Text = "შედეგი: ";
    int[] mim1 = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    int[] mim2 = new int[] { 1, 2, 3, 4 };
    var query = mim1.Union(mim2);
```

```

foreach (var i in query)
{
    label1.Text += i + " "; //შედეგი: 1 2 3 4 5 6 7 8 9 10 11 12
}
}

```

ვარიანტი 2. მეთოდი Union().

```

private void button1_Click(object sender, EventArgs e){
    label1.Text = "შედეგი: ";
    string[] soft = {"Microsoft", "Google", "Apple"};
    string[] hard = { "Apple", "IBM", "Sumsung" };
    // მონაცემების გაერთიანება
    var result = soft.Union(hard);
    foreach (string s in result)
        label1.Text += s + " "; //შედეგი: Microsoft Google Apple IBM Sumsung
}

```

ოპერატორი join. მეთოდი Join(). ობიექტების გაერთიანება LINQ ტექნოლოგიაში ნიშნავს ორი განსხვავებული ტიპის ნაკრების გაერთიანებას. ასეთი გაერთიანება შეგვიძლია განვახორციელოთ ოპერატორი join-ის ან მეთოდი Join()-ის დახმარებით, ამასთან მიმდევრობებს უნდა ჰქონდეთ რაღაცა საერთო კრიტერიუმი ან თვისება.

ამოცანა 20.1. ოპერატორი join. პროგრამულ კოდში მოცემულია ორი კოლექცია: Person და Pet. მოთხოვნა შეიცავს ოპერატორს join, რათა დავაკავშიროთ Person-ის ობიექტები Pet-ის ობიექტებთან. კოდში ობიექტები ანონიმური ტიპის წარმომადგენლები არიან და შეიცავენ ცხოველი სახელს და იმ ადამიანის სახელს, რომლის კუთვნილებაცაა ცხოველი. (იხ. ნახ. 142.)

```

class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
}

class Pet
{
    public string Name { get; set; }
}

```

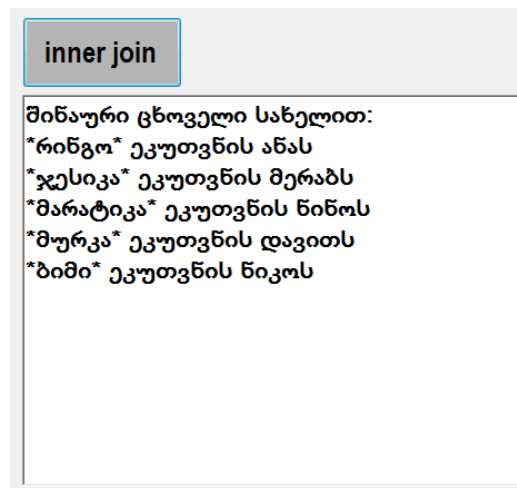
```

        public Person Owner { get; set; }
    }

    private void button1_Click(object sender, EventArgs e){
        Person ana = new Person { FirstName = "ანა", LastName = "წინიძე" };
        Person merabi = new Person { FirstName = "მერაბ", LastName =
"არჩუაძე" };
        Person nino = new Person { FirstName = "ნინო", LastName =
"მეგრელიშვილი"};
        Person daviti = new Person { FirstName = "დავით", LastName =
"გეგენავა" };
        Person niko = new Person { FirstName = "ნიკო", LastName = "ნიკოლაძე"
};
        Pet barley = new Pet { Name = "რინგო", Owner = ana };
        Pet boots = new Pet { Name = "ჯესიკა", Owner = merabi };
        Pet whiskers = new Pet { Name = "მურკა", Owner = daviti };
        Pet bluemoon = new Pet { Name = "მარტინა", Owner = nino };
        Pet daisy = new Pet { Name = "ბიმი", Owner = niko };
        List<Person> people = new List<Person> { ana, merabi, nino, daviti,
niko };
        List<Pet> pets = new List<Pet> { barley, boots, whiskers, bluemoon,
daisy};

        var query = from person in people
                     join pet in pets on person equals pet.Owner
                     select new { OwnerName = person.FirstName, PetName = pet.Name };
        richTextBox1.Text = "შინაური ცხოველი სახელით: \n";
        foreach (var ownerAndPet in query)
        {
            richTextBox1.Text += "*" + ownerAndPet.PetName + "*" კუთვნილი "
+ownerAndPet.OwnerName + "\n";
        }
    }
}

```

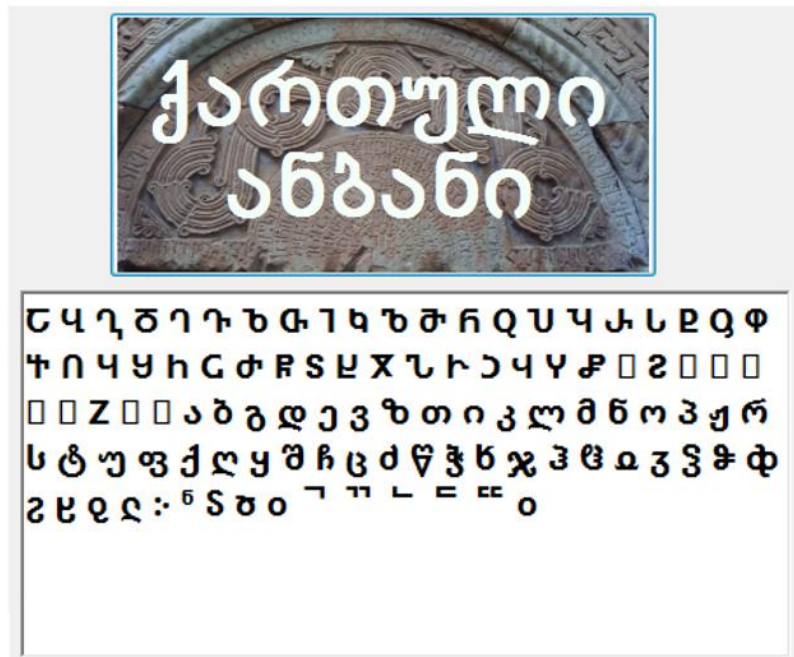



ნახ. 142. ამოცანა 20.1.-ის შესრულების შედეგი

ამოცანა 20.2. მეთოდი Join(). მეთოდის დახმარებით დავბეჭდოთ ქართული ანბანის მიმდევრობა. (იხ. ნახ. 143.)

<https://www.alanwood.net/unicode/georgian.html>

```
private void button1_Click(object sender, EventArgs e){
    int idx1 = 4256;
    int idx2 = 4351;
    var anabana = string.Join(" ", CreateAlphabet());
    richTextBox1.Text += anabana.ToString() + " ";
    List<char> CreateAlphabet()
    {
        var vals = new List<char>();
        int ana1 = idx1 + 100;
        for (int i = idx1; i <= ana1; ++i)
        {
            vals.Add(Convert.ToChar(i));
        }
        int bana2 = idx2;
        for (int i = idx2; i <= bana2; ++i)
        {
            vals.Add(Convert.ToChar(i));
        }
        return vals;
    }
}
```



ნახ. 143. ამოცანა 20.2.-ის შესრულების შედეგი

ამოცანა 20.3. მონაცემთა გაფილტვრა. ამოვირჩიოთ ისეთი მოთამაშეები, რომელთა ასაკი 30 წელზე მეტია.

```
class Player
{
    public string Name;
    public int Age;
}

private void button1_Click(object sender, EventArgs e) {
    label1.Text = "";
    List<Player> players = new List<Player>(); // სიის შექმნა
    // სიის შევსება
    players.Add(new Player() { Name = "ნინო", Age = 21 });
    players.Add(new Player() { Name = "გიორგი", Age = 39 });
    players.Add(new Player() { Name = "შალვა", Age = 25 });
    players.Add(new Player() { Name = "ამირანი", Age = 31 });
    players.Add(new Player() { Name = "ზურაბი", Age = 47 });
    players.Add(new Player() { Name = "ლუკა", Age = 29 });
    // შევარჩიოთ 30 წლის ზემოთ მოთამაშეები (Linq-ის გამოყენებით)
    IEnumerable<Player> result = from item in players
        where item.Age > 30 select item;
```

```
foreach (Player value in result) // კრანზე დალაგებული შედეგის ჩვენება
label1.Text += "სახელი = " + value.Name + " " + " ასაკი = " +
value.Age + '\n';
}
```

ამოცანა 20.4. მოთამაშეების ასაკის მიხედვით დალაგება.

```
class Player
{
    public string Name;
    public int Age;
}

private void button1_Click(object sender, EventArgs e){
    label1.Text = "";
    // სიის შექმნა.
    List<Player> players = new List<Player>();
    // სიის შევსება.
    players.Add(new Player() { Name = "ნინო", Age = 21 });
    players.Add(new Player() { Name = "გიორგი", Age = 39 });
    players.Add(new Player() { Name = "შალვა", Age = 25 });
    players.Add(new Player() { Name = "ამირანი", Age = 31 });
    players.Add(new Player() { Name = "ზურაბი", Age = 47 });
    players.Add(new Player() { Name = "ლუკა", Age = 29 });
    // სიის დალაგება ასაკის მიხედვით (Linq-ის გამოყენებით).
    IEnumerable<Player> result = from item in players
                                orderby item.Age select item;
    // კრანზე დალაგებული შედეგის ჩვენება.
    foreach (Player value in result)
        label1.Text += "სახელი = " + value.Name + " " + " ასაკი = "
        + value.Age + '\n';
}
```

ამოცანა 20.5. დათვლის მეთოდი Count(). დამოუკიდებლად დაწერეთ თუ რას ითვლის კოდი.

```
private void button1_Click(object sender, EventArgs e) {
```

```
label1.Text = "";  
int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };  
int size = (from i in numbers where i % 2 == 0 && i > 10 select  
i).Count();  
label1.Text += size;  
}
```

დათვლის მეთოდმა შეიძლება მიიღოს ლამბდა გამოსახულება.

მოცემული კოდი შესაძლებელია შევცვალოთ ლამბდა გამოსახულებით:

```
int size = numbers.Count(i => i % 2 == 0 && i > 10);
```

გამოყენებული ლიტერატურა და ინტერნეტ რესურსი

1. სურგულაძე გ. კომპიუტერული პროგრამირების მეთოდები და მეთოდოლოგიები (SP, OOP, VP, Agile, UML), ISBN 978-9941-8-1900-1. სტუ-ს „IT-კონსალტინგის სამეცნიერო ცენტრი“, 2019, -200 გვ.
2. ჯანელიძე გ., მეფარიშვილი თ. ობიექტზე ორიენტირებული დაპროგრამება, მეთოდური მითითებები ლაბორატორიული სამუშაოების შესრულებლად დაპროგრამების C++ ენაზე. ISBN: 9789941064265 გ. ჯანელიძის რედაქციით. თბ., 2014, 155 გვ.
3. ტოკაძე ლ., კაროიანი ი. მაგალითების კრებული Microsoft Visual Studio C# პროგრამული გარემოსთვის. ISBN: 978-9941-20-717-4. დამხმარე სახელმძღვანელო. საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, თბ., 2016, გვ. 67.
4. სამხარაძე რ. Visual C#.NET. ISBN: 978-9941-20-443-2 სტუ. საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, თბ., 2014, 507 გვ.
5. სურგულაძე გ. ვიზუალური დაპროგრამება C#_2010 ენის ბაზაზე. ISBN: 978-9941-20-021-2. საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, თბ., 2011. -445 გვ.
6. ბაწაშვილი გ. კომპიუტერული პროგრამირების სამყარო, თბ., 2007, გვ. 207, ISBN: 978-9941-0-0120-8
7. C# Tutorial. <https://www.tutorialspoint.com/csharp/>
8. The complete C# Tutorial. <http://csharp.net-tutorials.com/>
9. C#Tutorial. <http://www.tutorialsteacher.com/csharp/csharp-tutorials>
10. <https://samuraisxmali.wordpress.com/category/c/>
11. <https://vmakharashviliblog.wordpress.com/2017/09/23/first-blog-post/>
12. <https://www.guru99.com/best-c-sharp-books.html>

რედაქტორი ბ. ცხადაძე

გადაეცა წარმოებას 04.08.2025. ხელმოწერილია დასაბეჭდად 28.10.2025. ქაღალდის
ზომა 60X84 1/8. პირობითი ნაბეჭდი თაბახი 40. №3805.

საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, თბილისი, კოსტავას 77



Verba volant,
scripta manent

