ლ. ავთანდილაშვილი

კომპიუტერული დაპროექტება

IV ნაწილი

(დაპროგრამების ენა C++)

თბილისი 2006

საქართველოს ტექნიკური უნივერსიტეტი

ლ. ავთანდილაშვილი

კომპიუტერული დაპროექტება

IV ნაწილი

(დაპროგრამების ენა C++)

დამტკიცებულია სახელმძღვანელოდ სტუ-ს
სარედაქციო-საგამომცემლო საბჭოს მიერ

თბილისი 2006

წიგნში განხილულია საქართველოს ტექნიკური უნივერსიტეტის 1201 სპეციალობის - ”მანქანათმშენებლობის ტექნოლოგია” და 1201.06, 1201.05 სპეციალიზაციების შესაბამისად ”მანქანათსაშენი წარმოების ტექნოლოგიური პროცესების ოპტიმიზაცია და კომპიუტერული დაპროექტება”, ”სამედიცინო და პრეციზიული მიკროიარაღების კონსტრუირება და წარმოება” სტუდენტთათვის. იგი აგრეთვე დიდ დახმარებას გაუწევს მაგისტრანტებს, ასპირანტებს და იმ პირებს ვინ დაინტერესებულია პროგრამირების ალგორითმული ენის C++–ის შესწავლით.

რეცენზენტები: პროფ. თ. გეგეჭკორი

პროფ. თ. მჭედლიშვილი

ISBN 99940-40-71-5
ISBN 99940-35-53-3

შ ე ს ა ვ ა ლ ი

კომპიუტერზე დავალების მომზადებისა და ამოხსნის პროცედურა საკმაოდ რთული და შრომატევადი პროცესია, რომელიც შედგება შემდეგი ეტაპებისაგან:

1. ამოცანის დასმა;
2. ამოცანის მათემატიკური ფორმულირება, ანუ მატემატიკური მოდელი;
3. მონაცემთა ბაზის შექმნა, ან საწყისი მონაცემების მომზადება;
4. ამოცანის ამოხსნის ალგორითმის დამუშავება;
5. პროგრამის დაწერა რომელიმე ალგორითმულ ენაზე (ჩვენს შემთხვევაში C++);
6. პროგრამის შეტანა, პროგრამის ატლადკა და ტესტირება;
7. ამოცანის ამოხსნა და შედეგების დამუშავება.

ჩვენს შემთხვევაში იგულისხმება რომ ამოცანა დასმულია და მათემატიკური მოდელიც შექმნილია, საჭიროა შეუდგეთ ალგორითმის დამუშავებას და პროგრამის დაწერას. ალგორითმი არის ცვლადების რიცხვით მნიშვნელობებზე არითმეტიკული და ლოგიკური მოქმედებების მიმდევრობა. ალგორითმის აღწერის თვალსაჩინო ხერხია ბლოკსქემა. ამ დროს ალგორითმი წარმოდგენილია იმ ბლოკების მიმდევრობით, რომლებიც განსაზღვრავენ ფუნქციებს და მათ შორის კავშირს.

წიგნის პროგრამული უზრუნველყოფა წარმოადგენს Windows–ის გრაფიკულ ინტერფეისს და C++ - ის კომპილიატორის რომელიმე (ბოლო თუ არა საშუალო) ვერსიას. წიგნით სარგებლობის დროს იგულისხმება რომ მომხმარებელს ექნება რაიმე კომპიუტერული განათლება მაგ. საქაღალდის შექმნა, კატალოგით სარგებლობა, ფაილების ჩატვირთვა, ახალი ფაილის შექმნა, შენახვა და გახსნა. თქვენი პროგრამის შენახვა, გადუბლება და მოდიფიკაცია.

თ ა ვ ი I

C++ პროგრამის ზოგადი სტრუქტურა

ნებისმიერი C++ პროგრამა წარმოადგენს main პროგრამულ ბლოკს, რომელსაც ფუნქცია ეწოდება. C++ პროგრამა შეიძლება შეიცავდეს ერთზე მეტ ფუნქციას, რომელთაგან ერთერთი აუცილებლად უნდა იყოს main ფუნქცია. სიტყვა main-ს თანსდევს გახსნილი და დახურული მრგვალი ფრჩხილები, რაც მიუთითებს იმაზე, რომ main-ი წარმოადგენს პროგრამულ ბლოკს, რომელსაც ფუნქცია ეწოდება. ყოველი ფუნქციის ტანი უნდა იწყებოდეს { (მარცხენა ფიგურული ფრჩხილით) და მთავრდებოდეს } (მარჯვენა ფიგურული ფრჩხილით). სქემატურად C++ პროგრამას აქვს შემდეგი სახე

```
main( )
   {
   . . . . . .
   }
function ( )
   {
   . . . . . .
   }
```

პროგრამული ბლოკი შედგება აღწერილობებისაგან და ოპერატორებისაგან. ყოველი ოპერატორი მთავრდება წერტილ მძიმით. ოპერატორი შეიძლება იყოს მარტივი ან შედგენილი. შედგენილი ოპერატორი წარმოადგენს ფიგურულ ფრჩხილებში ჩაწერილი ოპერატორების ნებისმიერ ერთობლიობას.

C++-ის ალფავიტი

C++-ის ალფავიტი შეიცავს ასოებს, ციფრებს და შემდეგ სპეციალურ სიმბოლოებს:

1. ლათინური ალფავიტის დიდ და პატარა ასოებს: A, B, C, D, I, F, G,...,Z a, b, c, d,...,z
2. არაბულ ციფრებს : 0, 1, 2, 3, . . , 9
3. პუნქტუაციის ნიშნებს და სპეციალურ სიმბოლოებს, რომელთა რიცხვს მიეკუთვნება :

სიმბოლო	დასახელება	სიმბოლო	დასახელება
,	მძიმე	#	ნომერი
.	წერტილი	%	პროცენტი
;	წერტილმძიმე	&	ამპერსანდი
:	ორი წერტილი	^	ლოგიკური არა
*	ვარსკვლავი	+	პლუსი
?	კითხვის ნიშანი	-	მინუსი
'	აპოსტროფი	=	ტოლობა
!	ძახილის ნიშანი	"	ბრჭყალები
|	ვერტიკალური ხაზი	(	მრგვალი ფრჩხილი
/	წილადური ხაზი	)	მრგვალი ფრჩხილი
\	უარყოფითი ხაზი	{	ფიგურული ფრჩხილი
`	ტილდა	}	ფიგურული ფრჩხილი
[	მარცხ. კვად. ფრჩხილი	<	ნაკლებობა
]	მარჯ. კვად. ფრჩხილი	>	მეტობა

4. გამოყოფის სიმბოლოებს რომელთა რიცხვს მიეკუთვნება: ხარვეზი, ტაბულაციის სიმბოლო, ახალ სტრიქონზე გადასვლა, და ა.შ.
5. სამართ მიმდევრობებს - სპეციალურ სიმბოლოთა კომბინაციებს რომლებიც იქმნება უკუ წილადური ხაზის (\ ) და ლათინური ასოებისა და ციფრების

საშუალებით, გამოიყენება შეტანისა და გამოტანის ოპერაციებში. ცხრილში მოყვანილია ზოგიერთი მათგანი :

მართვადი მიმდევრები	დასახელება
\t	ჰორიზონტალური ტაბულაცია
\n	ახალ სტრიქონზე გადასვლა
\v	ვერტიკალური ტაბულაცია
\z	სტრიქონის თავში დაბრუნება
\f	ფორმატის გადაყვანა
\“	ბრჭყალები
\’	აპოსტროფი
\o	ნულ-სიმბოლო

job	can’t
x23_c	c-15
_bob	5d
my_dog	my dog

პროგრამირების ენების ოპერატორებში და ბრძანებებში გამოიყენება გარკვეული რაოდენობა დარეზერვირებული სიტყვები, რომლებიც ცნობილია C++ ენის კომპილატორისათვის და მათ გასაღები სიტყვები ეწოდებათ. გასაღები სიტყვებში გამოყენებულია მხოლოდ პატარა ასოები. დაუშვებელია გასაღები სიტყვების გამოყენება „მომხმარებლის იდენტიფიკატორებად“. გასაღები სიტყვათა რიცხვს მიეკუთვნება მაგალითად cout, cin, if, while და ა.შ.

1. 1. მუდმივები და ცვლადები

ენის მარტივ კონსტრუქციას მიეკუთვნება რიცხვები, მუდმივები, ცვლადები, სტანდარტული ფუნქციები და გამოსახულებები. პროგრამა გადაამუშავებს მონაცემებს, რომლებმაც შეიძლება მიიღონ რიცხვითი, ლოგიკური და სტრიქონული მნიშვნელობები.ენაში გამოიყენება შემდეგი ტიპის მონაცემები: მთელები (INTEGER), ნამდვილი (REAL), ლოგიკური (BOOLEAN) და სიმბოლური (CHAR).

რიცხვები შეიძლება იყოს მთელი და ნამდვილი ტიპის.

მთელი ტიპის რიცხვს არა აქვს ათწილადი ნწილი. მაგ. 42, -6, 786.

ნამდვილი ტიპის რიცხვები ჩაიწერება ფიქსირებული წერტილით გამოყოფილი მთელი და ათწილადი ნაწილის სახით. მაგ. –4.85, 1.64, -0,25.

ლოგიკური ტიპის ცვლადები ღებულობენ მხოლოდ ორ მნიშვნელობას ჭეშმარიტი (TRUE) და მცდარი (FALSE).

სიმბოლური ტიპის ცვლადები ღებულობენ ერთი სტრიქონის (ლიტერის) მნიშვნელობას.

ლიტერი შეიძლება შეიცავდეს ასოებისა და ციფრების დალაგებულ მიმდევრობას.

მუდმივა პროგრამაში წარმოიდგინება უცვლელი მნიშვნელობით, რომელსაც მიეკუთვნება შემდეგი ტიპის მონაცემები: მთელი, ნამდვილი, ლოგიკური, სიმბოლური ან სტრიქონული. მაგ. 1001, -44, 26.85, -0.5E-5, TRUE, ′C′, ′8′ ′ფაქტორიალი=′.

ცვლადები გამოიყენება პროგრამაში ისეთი მნიშვნელობის ჩასაწერად, რომელიც ღებულობს განსხვავებული ტიპის მქონე სხვადასხვა მნიშვნელობებს.

არსებობს ორგვარი ცვლადი, მარტივი ცვლადი და ცვლადი ინდექსით. ყოველ ცვლადს ან მუდმივას აქვს სახელი-იდენტიფიკატორი, რომელიც არის ასოებისა და ციფრების მიმდევრობა. მაგ. X, Y, REZ, REZ2, SUMMA, და ა. შ.

C++ ენაში ცვლადი არის მეხსიერებაში ინფორმაციის დამახსოვრებისათვის გამოყოფილი მისამართი-ადგილი. ცვლადის გამოცხადებისას ხდება ერთი ან რამოდენიმე ასეთი მისამართის (უჯრედის) რეზერვირება, მასში ჩვენს მიერ მითითებული ტიპის მონაცემის მნიშვნელობის განსათავსებლად. C++ ენაში საჭიროა პროგრამაში გამოყენებული ყველა ცვლადის გამოცხადება მათი ტიპის მითითებით. C++-ში გამოიყენება შემდეგი ტიპის მუდმივები და ცვლადები: bool, int , long int , float, double, long double, char. ზოგადი სახით ცვლადის გამოცხადების ჩაწერის სტრუქტურაა:

ტიპის სახელი იდენტიფიკატორი [,იდენტიფიკატორი];

ამრიგად, ცვლადის შემოტანის დროს უნდა იყოს მითითებული მონაცემის ტიპი და ერთი ან მეტი ცვლადის იდენტიფიკატორი.

ცვლადების ძირითადი ტიპები მოყვანილია ცხრილში:

ტიპი	ზომა	მნიშვნელობა
bool	1 byte	true ან false
short int	2 byte	-32 768 - დან 32768 -მდე
long int	4 byte	-2 147 483 648 - დან 2 147 483 648 -მდე
int (16 bit)	2 byte	-32 768 - დან 32768 -მდე
int (32bit)	4 byte	-2 147 483 648 - დან 2 147 483 648 -მდე
char	1 byte	256 სიმბოლური სიდიდე
float	4 byte	1.2e-38 - დან 3.4e38 - მდე
double	8 byte	2.2e-308 - დან 1.8e308 - მდე

float-ის 4 ბაიტიდან 1 ბიტი ნიშნისათვის არის გამოყოფილი, 8 ბიტი ექსპონენტისათვის, 23 ბიტი კი მანტისაა.

double-ის 8 ბაიტიდან 1 ბიტი ნიშნისათვის არის გამოყოფილი, 11 ბიტი ექპონენტისათვის, 52 ბიტი კი მანტისაა.

არსებობენ მონაცემთა წარმოებული ტიპები: მასივები, მიმთითებლები, სტრუქტურები, გაერთიანებები და ა.შ., რომლებსაც შემდგომში განვიხილავთ.

ცვლადების გამოცხადების მაგალითები : float a,b; int k;

ცვლადების გამოცხდადებისას დასაშვებია მათი ინიციალიზაცია- მათთვის ნიშვნელობების მინიჭება მაგ. float f=3.19 ; 0

სიმბოლური ტიპის ცვლადები

სიმბოლური (char) ტიპის ცვლადის სიდიდე ერთი ბაიტია. ასეთი ტიპის ცვლადების რიცხვით მნიშვნელობათა დიაპაზონია: 0-255, რაც განსაზღვრავს ყველა ასოს, ციფრს და პუნქტუაციის ნიშნებს. ეს რიცხვითი მნიშვნელობები განკუთვნილია კომპიუტერში სიმბოლოთა კოდირებისათვის. მაგ. “a” ასოს კოდი არის სიდიდე 97. ყველა დიდი-პატარა ასოები (იგულისხმება ლათინური ანბანი), ციფრები, პუნქტუაციის ნიშნები მოთავსებულია კოდური ცხრილის 1-128 ინტერვალში. დამატებითი 128-255 უჯრედები რეზერვირებულია სხვადასხვა ფსევდოგრაფიკული, დასავლეთევროპული, ზოგიერთი მათემატიკური თუ სხვა ნიშნებისათვის.

1. 2 მ უ დ მ ი ვ ე ბ ი

მუდმივა არის სიდიდე, რომელიც ინარჩუნებს თავის მნიშვნელობას უცვლელად, პროგრამის შესრულების პროცესში. მუდმივას ინიციალიზაცია აუცილებლად უნდა მოხდეს

მისი შექმნის მომენტში. C++-ში განიხილება 3 ტიპის მუდმივა-კონსტანტა : რიცხვითი კონსტანტა (მთელი და მცოცწერტილიანი)
სიმბოლური კონსტანტა.
სტრიქონული (ლიტერული) კონსტანტა.

მთელი ტიპის მუდმივა

მთელი ტიპის მუდმივას წარმოადგენენ ათობითი, რვაობითი და თექვსმეტობითი რიცხვები. ათობითი წარმოადგენს ციფრების მიმდევრობას, რომელსაც შეიძლება ჰქონდეს ნიშანი (პირველი ციფრი ნულისაგან განსხვავებულია). რვაობითი აუცილებლად იწყება 0 და შემდეგ 1 ან რამოდენიმე რვაობითი ციფრი (0,1,2,3,4,5,6,7). ათობითი კონსტანტას ტიპია int ან long. რვაობითი კონსტანტას ენიჭება ტიპი int, unsigned int, long ან unsigned long. იმისათვის, რომ ნებისმიერი მთელი კონსტანტა იყოს long ტიპის საკმარისია მას მიეწეროს სიმბოლო l ან L.
მაგ. const int x=17;

მცოცავმძიმიანი მუდმივა (კონსტანტა)

მცოცავმძიმიანი კონსტანტა შეიძლება წარმოდგენილი იქნეს ბუნებრივი, ან ექსპონენციალური ფორმით:
ბუნებრივი ფორმის გამოყენებისას რიცხვი ჩაიწერება მთელი და წილადი ნაწილების სახით, რომლებიც ერთმანეთისაგან ათობითი წერტილით არიან გამოყოფილი. მაგ. –18.539, 235.44, 0.0057
ექსპონენციალური ფორმით რიცხვის წარმოდგენისას იგი ჩაიწერება რიცხვის მანტისისა და რიგის მითითებით. მაგ. 7.8324E+002; -9.4753E+001; 2.1567E+000; 3.9168E-002
მაგ. const float y=9.274;
const float z=-6.285E+002

სიმბოლური ტიპის კონსტანტა

სიმბოლური ტიპის კონსტანტა წარმოადგენს აპოსტროფებში მოთავსებულ სიმბოლოს მაგ ‘x’. სიმბოლური კონსტანტას რიცხვითი ექვივალენტი int ტიპისაა. მართვადი მიმდევრობები განიხილება, როგორც სიმბოლური კონსტანტები.
მაგ. const char simb=’f’;

სტრიქონული (ლიტერული) კონსტანტა

სტრიქონული კონსტანტა წარმოადგენს ბრჭყალებში ჩასმულ სიმბოლოთა მიმდევრობას. მაგ “ჩვენ ვსწავლობთ C++“. მისი ტიპია char[ ], რაც იმაზე მიუთითებს, რომ სტრიქონი განიხილება როგორც სიმბოლოთა მასივი. სტრიქონის ბოლოში კომპილატორის მიერ ემატება ნულოვანი მართვადი სიმბოლო ‘\0’, როგორც ნიშანი სტრიქონის დამთავრებისა, და იგი არ წარმოადგენს თვით სტრიქონის ნაწილს. თითოეული სიმბოლო იკავებს ერთ ბაიტს მეხსიერებაში და ისინი განლაგდებიან ერთმანეთის გვერდით. ამრიგად მასივის ელემენტი ერთით მეტია სიმბოლურ კონსტანტაზე. მაგ. ტექსტი: ჩვენ ვსწავლობთ C++. მეხსიერებაში ასე განლაგდება:

ჩ	ვ	ე	ნ		ვ	ს	წ	ა	ვ	ლ	ო	ბ	თ		C	+	+	\0

მის განსაზღვრას კი ექნება შემდეგი სახე:
მაგ. const char b[]= “ჩვენ ვსწავლობთ C++”;

თვლადი მონაცემები

გარდა ზემოთჩამოთვლილი მონაცემთა სტანდარტული ტიპებისა C++- ში არსებობს მონაცემთა ე.წ. თვლადი ტიპი. ასეთი ტიპის გამოცხადება იწყება გასაღები სიტყვა enum-ით რომელსაც მოსდევს ტიპის სახელი და შემდეგ ფიგურულ ფრჩხილებში მნიშვნელობათა სია. სიაში მოთავსებული ყველა იდენტიფიკატორი უნდა იყოს უნიკალური. შეთანხმებით, რიგით პირველი იდენტიფიკატორის რიცხვითი მნიშვნელობაა 0, მეორე იდენტიფიკატორის 1 და ა.შ. ამრიგად სიის იდენტიფიკატორთა რიცხვითი მნიშვნელობები მთელი რიცხვებია.

enum ტიპის სახელი { მუდმივების იდენტიფიკატორების ჩამონათვალი };

მაგ. enum color { RED,BLUE,GREEN,WHITE,BLACK}
აქ RED<BLUE<GREEN და ა.შ. – მნიშვნელობების ჩამონათვალია.

1. 3 მათემატიკური საბიბლიოთეკო (სტანდარტული) ფუნქციები

C++-ში გამოიყენება 23 სხვადასხვა მათემატიკური საბიბლიოთეკო ფუნქცია, რომელთა გამოყენებით შესაძლებელია გარკვეული მათემატიკური გამოთვლების შესრულება. საზოგადოდ ფუნქციის გამოძახება (მიმართვა მის შესრულებაზე) ხორციელდება მისი სახელის ჩაწერით, რომლის შემდეგ მრგვალ ფრჩხილებში ჩაწერილი უნდა იქნეს არგუმენტი. ცხრილში მოყვანილია უფრო ხშირად გამოსაყენებელი მათემატიკური საბიბლიოთეკო ფუნქციები :

ფუნქცია	აღწერა	მაგალითი
1	2	3
sqrt(x)	კვადრატული ფესვი x დან	sqrt(900.0)=30.0
exp(x)	ექსპონენციალური ფუნქცია e^x	exp(1.0)=2.718282
log(x)	x-ის ნატურალური ლოგარითმი	log(2.718282)=1.0
log10(x)	x-ის ათობითი ლოგარითმი	log10(100)=2.0
fabs(x)	x-ის აბსოლუტური მნიშვნელობა	fabs(-5)=5
ceil(x)	x-ის დამრგვალება უმცირეს მთელამდე, რომელიც არ არის x-ზე ნაკლები	ceil(9.2)=10.0 ceil(-9.8)= -9.0
floor(x)	x-ის დამრგვალება უდიდეს მთელამდე, რომელიც არ არის x-ზე მეტი	ceil(9.2)=9.0 ceil(-9.8)= -10.0
pow(x,y)	x ხარისხად y	pow(9.0,0.5)=3.0
fmod(x,y)	ნამდვილი ტიპის რიცხვის სახით მიღებული x/y განაყოფის ნაშთი	fmod(13657,2.333)=1.992
sin(x)	რადიანებში	
cos(x)	რადიანებში	
tan(x)	რადიანებში	
atan(x)	arctg(x) რადიანებში	

ო პ ე რ ა ც ი ე ბ ი

ოპერაციის სახეობა კომპიუტერს მიეთითება ოპერაციის სიმბოლოს საშუალებით. ოპერანდი წარმოადგენს ოპერაციაში მონაწილე ცვლადს, კონსტანტას, ლიტერას ან გამოსახულებას. ყოველი ოპერანდი გარკვეული ტიპისაა. ოპერაციაში მონაწილე ოპერანდების რაოდენობის მიხედვით განასხვავებენ უნარულ (ერთი ოპერანდით), ბინარულ (ორი ოპერანდით) და ტერნარულ (სამი ოპერანდით) ოპერაციებს.

ძირითადი უნარული ოპერაციები	აღნიშვნა
უნარული მინუსი	-
ლოგიკური უარყოფა	!
ირიბი დამისამართება	*
მისამართის გამოთვლა	&
უნარული პლიუსი	+
გაზრდა (ინკრემენტი)	++
შემცირება (დეკრემენტი)	--
ზომის განსაზღვრა	sizeof

ძირითადი ბინარული ოპერაციები :

ოპერაციის ნიშანი	ოპერაციის დასახელება	ოპერაციის ჯგუფი
*	გამრავლება	მულტიპლიკაციური
/	გაყოფა	
%	მოდულით გაყოფა	
+	შეკრება	ადიტიური
-	გამოკლება	
<	ნაკლებია	თანაფარდობათა ოპერაციები
<=	ნაკლებია ან ტოლი	
>	მეტია	
>=	მეტია ან ტოლი	
==	ტოლობა	
!=	არ არის ტოლი	
&&	ლოგიკური “და “	ლოგიკური
‖	ლოგიკური “ან “	ლოგიკური

1.4. ოპერატორები

C++ პროგრამა წარმოადგენს გარკვეული თანმიმდევრობით ჩაწერილი, ამოცანის ამოხსნისათვის საჭირო ოპერატორების ერთობლიობას. ოპერატორი მთავრდება წერტილმძიმით. ოპერატორის ჩაწერისას ხარვეზი, ტაბულაცია, ახალ სტრიქონზე გადასვლა იქნება იგნორირებული კომპილატორის მიერ. ოპერატორები შეიძლება გაერთიანდნენ შედგენილ ოპერატორად, რომელიც იწყება და მთავრდება ფიგურული ფრჩხილით. შედგენილი ოპერატორი, კომპილატორის მიერ განიხილება, როგორც ერთი მთლიანი შესასრულებელი ოპერატორი.

მინიჭების ოპერატორი

მინიჭების ოპერატორი გამოიყენება ცვლადისათვის რაიმე მნიშვნელობის მისანიჭებლად. ზოგადი სახით იგი ჩაიწერება შემდეგნაირად B=A, სადაც A- არის არითმეტიკული ან ლოგიკური გამოსახულება, სიმბოლური ტიპის მუდმივა ან სტრიქონი, ხოლო B- კი არის ცვლადი, რომელსაც ენიჭება A-ს მნიშვნელობა.

C++ -ში გამოიყენება აგრეთვე მინიჭების რამოდენიმე დამატებითი ოპერაცია, რომელთა გამოყენება ამცირებს მინიჭების ოპერატორის ჩაწერის სიგრძეს.
საზოგადოდ, ნებისმიერი ოპერატორი, რომელის სახეა :

ცვლადი = ცვლადი ოპერაცია გამოსახულება;

შეიძლება ჩაწერილი იქნეს შემდეგნაირად:

ცვლადი ოპერაცია = გამოსახულება;

მაგ. a=a+b შეიძლება ჩაიწეროს a+=b

ინკრემენტის და დეკრემენტის ოპერაციები

C++ გამოიყენება ინკრემენტის უნარული ოპერაცია ++ (ერთით გადიდება) და დეკრემენტის უნარული ოპერაცია – (ერთით შემცირება). მაგალითად, თუ რაიმე x ცვლადის მნიშვნელობა ერთით უნდა გადიდდეს x=x+1 და x+=1 ოპერაციების ნაცვლად , უმჯობესია ინკრემენტის ++ ოპერაციის გამოყენება. მაგ. a++ ან ++a.

თუ ინკრემენტის ან დეკრემენტის ოპერაცია განთავსებულია ცვლადის წინ (მაგ. ++a ან --b), ინკრემენტის ან დეკრემენტის ჩაწერის ასეთ ფორმას პრეფიქსული ეწოდება. თუ ინკრემენტის ან დეკრემენტის ოპერაცია ჩაწერილია ცვლადის შემდეგ (მაგ. a++ ან b--) – მას პოსტფიქსური ეწოდება.

პრეფიქსური ფორმის შემთხვევაში ჯერ ხორციელდება ცვლადის მნიშვნელობის ერთით გაზრდა და შემდეგ მისი ახლად მიღებული მნიშვნელობის გამოყენება იმ გამოსახულების გამოსათვლელად, რომელშიც იგი არის ჩაწერილი. პოსტფიქსური ფორმის შემთხვევაში გამოსახულებაში ჯერ გამოიყენება ცვლადის მიმდინარე მნიშვნელობა, ხოლო შემდეგ შეიცვლება ცვლადის მნიშვნელობა. თუ ინკრემენტის ან დეკრემენტის ოპერაცია გამოიყენება დამოუკიდებელი ოპერატორის სახით, მაგალითად ++b; c--; , მაშინ სულერთია თუ რომელი ფორმა იქნება გამოყენებული (პოსტფიქსური თუ პრეფიქსური).

არითმეტიკული ოპერაციები:

C++-ის მათემატიკური ოპერაციებია: შეკრება(+), გამოკლება(-), გამრავლება (*), გაყოფა (/) და მოდულით გაყოფა (%).
მაგ. 21 % 4 =1; 5/3=1; 5.0/3.0=1.66667.

ოპერაციათა პრიორიტეტები.

რანგი	დასახელება	ოპერატორი
1	გამოსახულება	:: () [ ] Π
2	უნარული	- “ ! * & ++ -- sizeof ტიპების დაყვანა
3	მულტიპლიკატიური	* / %
4	ადიტიური	+ -
5	უტოლობა	< > <= >=
6	ტოლობა	= !=
7	ლოგიკური “და“	&&
8	ლოგიკური “ან“	!!

ოპერაცია % გამოიყენება მხოლოდ მთელი ტიპის მონაცემებზე.

1. 5. გ ა მ ო ს ა ხ უ ლ ე ბ ე ბ ი

C++ გამოიყენება არითმეტიკული და ლოგიკური გამოსახულებები. არითმეტიკული გამოსახულება წარმოადგენს არითმეტიკული ოპერაციათა ნიშნებით დაკავშირებულ ოპერანდებს (მუდმივებს, ცვლადებს, ფუნქციებს).

მოვიყვანოთ არითმეტიკული გამოსახულებების ჩაწერის მაგალითები.

1. $a^7+e^{\lg t}$ – pow(a,7)+exp(log10(t));

2. b^a – pow(b,a);
3. $(|a+c|)^{1/5}$ – pow(fabs(a+c), 1.0/5.0);

ლოგიკური გამოსახულებები განხილული იქნება მოგვიანებით.

მონაცემების შეტანის და გამოტანის ოპერატორები

C++ -ში შეტანა-გამოტანის ოპერაციების შესასრულებლად პროგრამაში საჭიროა პროცესორის დირექტივის # include<iostream.h> გამოყენება რაც პროგრამას მიუერთებს ე.წ. სასათაურო ფაილს <iostream.h> რითიც ხელმისაწვდომი ხდება შეტანა-გამოტანის ოპერაციებისათვის საჭირო ისეთი ობიექტები, როგორიცაა cin, cout, cerr რომლებიც შეესაბამება კლავიატურიდან შეტანის სტანდარტულ ნაკადს, ეკრანზე გამოტანის სტანდარტულ ნაკადს და შეცდომების შესახებ შეტყობინებების სტანდარტული ნაკადის გამოტანას. ნაკადი წარმოადგენს ბაიტების მიმდევრობას

ფაილი <iomanip.h> შეიცავს ობიექტებს რომლებიც საჭიროა ფორმატირებული შეტანა-გამოტანის განხორციელებისათვის ე.წ. ნაკადის პარამეტრიზებული მანიპულატორების გამოყენებით.

ნაკადში განთავსების ოპერაცია

ოპერაცია << წარმოადგენს "ნაკადში განთავსების ოპერაციას". აღნიშნული ოპერაცია გამოიყენება cout სტანდარტული ნაკადის ობიექტისათვის, რომელიც "მიბმულია" გამოტანის სტანდარტულ მოწყობილობასთან (დისპლეის ეკრანთან). ზოგადი სახით იგი ჩაიწერება:

cout<<გამოსახულება<<გამოსახულება<< ... ;

მაგ. cout<< abc; cout<<k<<" "<<l<<endl<<m<<" hello!"<<endl;
მაგალითისათვის მოვიყვანოთ პროგრამა რომლითაც განხორციელდება "ჩვენ ვსწავლობთ C++!" ტექსტის გამოტანა დისპლეის ეკრანზე.

```
# include<iostream.h>
  main ()
  {
   cout<<" ჩვენ ვსწავლობთ C++ ! \n";
  return 0;
  }
```

ტექსტის ეკრანზე გამოტანის შემდეგ \n მმართველი მიმდევრობით ხორციელდება კურსორის ახალ სტრიქონზე გადაყვანა. იგივეს განხორციელება შეიძლება endl (end line – სტრიქონის ბოლო) ნაკადის მანიპულატორის გამოყენებით.

```
# include<iostream.h>
  main ()
  {
   cout<<" ჩვენ ";
     cout<<" ვსწავლობთ C++ ! "<<endl;
  return 0;
    }
```

ერთი cout ობიექტისათვის შეიძლება მრავალჯერ << - "ნაკადში განთავსების" ოპერაციის გამოყენება. ინფორმაციის კლავიატურიდან კომპიუტერის მეხსიერებაში შესატანად გამოიყენება cin ობიექტი და ნაკადიდან გამოტანის (მიღების) ოპერაცია >>. მაგალითად ოპერატორი cin >>x>>y ; უზრუნველყოფს კლავიატურიდან x და y სიდიდეების მნიშვნელობების შეტანას მეხსიერებაში.

1. 6. ცვლადის გამოცხადება და შეტანა-გამოტანის ოპერატორების გამოყენება

```
// პროგრამის მარტივი მაგალითი 1.
#include <iostream>        // დირექტივა პროგრამასთან დამხმარე პროგრამის მისაერთებლად
using namespace std;     // სტანდარტული ბიბლიოთეკის კოდების გამოყენება
int main()                    // ფუნქცია
{
  int apples, oranges;        // ორი ცვლადის apples და oranges გამოცხადება
  int fruit;                  // ... მესამე ცვლადის fruit გამოცხადება
  apples = 5; oranges = 6;    // ცვლადებზე მნიშვნელობების მინიჭება
  fruit = apples + oranges;   //  ჯამის გამოთვლა
  cout << endl;               //  გამოტანა ახალი ხაზიდან
  cout << "Oranges are not the only fruit... " << endl
     << "- and we have " << fruit << " fruits in all.";
    cout << endl;             //  ახალ ხაზზე გადასვლა
  return 0;                   //  ფუნქციიდან გამოსვლა
}

// ვარჯიში ცვლადის გამოტანაზე მაგალითი 2.
#include <iostream>  // დირექტივა პროგრამასთან დამხმარე პროგრამის მისაერთებლად
using namespace std; // სტანდარტული ბიბლიოთეკის კოდების გამოყენება
int main()        // ფუნქცია
{
  int num1 = 1234, num2 = 5678; //  ცვლადების გამოცხადება და მნიშვნელობების მინიჭება
  cout << endl;                  //  ახალ ხაზზე გადასვლა
  cout << num1 << num2;          //ორი ცვლადის გამოტანა ერთმანეთის მიმდევრობით
  cout << endl;                  //  ახალ ხაზზე გადასვლა და დამთავრება
  return 0;                     //  ფუნქციიდან გამოსვლა
}
```

1. 7. წრფივი სტრუქტურის ალგორითმის პროგრამირება

შევადგინოთ პროგრამა, რომლითაც გამოითვლება

$$Y = (X * \sqrt{X} + COSX) : (A + X)$$

გამოსახულების მნიშვნელობა A და X ცვლადების მოცემული მნიშვნელობებისათვის.

```
//ჩვენი პირველი პროგრამა
#include<iostream.h> // დირექტივა სათაო პროგრამასთან მისაერთებლად
```

```
#include<math.h>   // დირექტივა მათემატიკური ფუნქციებით სარგებლობისას
 main()
{
float s,z,x;  // ნამდვილი ტიპის ცვლადების გამოცხადება
cout<<"Enter the value of X: ";
cin>>x;  // ცვლადის მნიშვნელობის  შეტანა კლავიატურიდან
z=sqrt(fabs(pow(x,3)))+
pow(fabs(sin(x)),(2.0/3));  // ფორმულის ჩაწერა
s=sqrt(z*z+2);
cout<<"s="<<s<<endl; //ეკრანზე გამოიტანება სიმბოლოთა სტრიქონი და შემდეგ ცვლადის
                          გამოთვლილი მნიშვნელობა
  return 0; // ფუნქციიდან  გამოსვლა
 }
```

ახლა ზემოდმოყვანილი პროგრამა გადავწეროთ ისეთნაირად რომ, პროგრამის შესრულების შედეგად მიღებული ამონახსნი ჩაიწეროს იმ ფაილის ბოლოში, რომელშიც იმყოფება თვით პროგრამა ე. ი. ამონახსნი უნდა მიებეს პროგრამის ტექსტს. ამისათვის, პროგრამაში საჭიროა რამოდენიმე ოპერატორის დამატება და პროგრამას ექნება შემდეგი სახე:

```
#include<iostream.h> //
#include<math.h>  // დირექტივა მათემატიკური ფუნქციებით სარგებლობისას
#include<fstream.h> //  სათავო ფაილის დირექტივა, რომელიც შეიცავს ფაილებთან მუშაობისათვის
                        საჭირო პროგრამებს
 main()
{
ofstream res("a.cpp", ios::app); //პროგრამაში იქმნება ფაილი მომხმარებლის მიერ დარქმეული
                                    სახელით მაგ. res
float s,z,x;    // ნამდვილი ტიპის ცვლადების გამოცხადება
cout<<"Enter the value of X: ";
cin>>x;
z=sqrt(fabs(pow(x,3)))+
pow(fabs(sin(x)),(2.0/3));
s=sqrt(z*z+2);
cout<<"s="<<s<<endl;
res<<"s="<<s<<endl; //ფაილში ინფორმაციის (გამოთვლილი პასუხის ) ჩაწერა
        return 0; //ფუნქციიდან გამოსვლა
      }
```

შენიშვნა ფაილი სახელით res (სახელი შერჩეულია პროგრამისტის მიერ), იარსებებს მხოლოდ პროგრამის შესრულების პროცესში და იგი დაკავშირებულია დისკზე არსებულ a.cpp ფაილთან, სადაც დამახსოვრებულია პროგრამის ტექსტი, რომელიც ios::app მანიპულატორით განსაზღვრულია როგორც მასში ინფორმაციის დამატებისათვის განკუთვნილი ფაილი. res ფაილში ინფორმაციის შეტანისას იგი ფაქტობრივად დაემატება a.cpp ფაილს ბოლოში. res<<"s="<<s<<endl; ოპერატორით ხორციელდება ფაილში ინფორმაციის ჩაწერა.

1.8. რიცხვების ფორმატირებული სახით გამოტანა

ალგორითმულ ენა C++ -ში ფორმატირებული სახით ინფორმაციის გამოტანისათვის გამოიყენება ნაკადების მანიპულატორები და ფუნქციები. მათი გამოყენებით შეიძლება

მითითება ველის საჭირო სიდიდის, რიცხვითი ინფორმაციის გამოტანის სიზუსტის, ნამდვილი რიცხვის წარმოდგენის ფორმის და ა.შ.

მცოცმძიმიანი რიცხვებისათვის გამოტანის სიზუსტის დაწესება (precision, setprecision)

მცოცმძიმიანი რიცხვების გამოტანა შეთანხმების თანახმად ხორციელდება 6 ნიშნის სიზუსტით. საჭიროების შემთხვევაში შესაძლებელია განსხვავებული სიზუსტის დაწესება, რაც ხორციელდება ნაკადის მანიპულატორის setprecision ან cout.precision ფუნქციის გამოყენებით. რომელიმე მათგანის გამოყენებით დაწესებული სიზუსტე მიმდინარე პროგრამაში მოქმედებს ყველა მომდევნო გამოტანის ოპერაციებზე, მანამ სანამ არ განხორციელდება განსხვავებული სიზუსტის დაწესება. სიზუსტის დაწესება ხდება cout.precision(k) ფუნქციით ან setprecision(k) მანიპულატორით, სადაც k – მთელი ტიპის რიცხვია ან ცვლადი. თუ რიცხვის გამოტანა ხდება ბუნებრივი ფორმით, მაშინ k-ს მნიშვნელობა განსაზღვრავს რიცხვის წილად ნაწილში თანრიგების რაოდენობას, ხოლო თუ რიცხვი გამოიტანება მეცნიერული ფორმით, მაშინ მანტისის თანრიგების რაოდენობას. k=0-ის შემთხვევაში აღსდგება 6 ნიშნიანი სიზუსტე.

float-ის ტიპის მონაცემისათვის k-ს მაქსიმალური მნიშვნელობაა 6, ხოლო double ან long double ტიპის მონაცემისათვის კი 15.

ველის სიგანის მითითება

ველის სიგანის მითითება, რომელშიც იქნება გამოტანილი ინფორმაცია (ცვლადის მნიშვნელობა ან რაიმე სტრიქონი) ხდება setw (k) მანიპულატორით , ან cout.width (k), სადაც k- ველის სიგრძეა. უნდა აღინიშნოს, რომ მათი მოქმედება ვრცელდება მხოლოდ ერთ ცვლადზე რომელიც ჩაწერილია width ფუნქცია-ელემენტის ან setw (k) მანიპულატორის შემდეგ.

```
// ვარჯიში ფორმატით გამოტანაზე მაგალითი 3.
#include <iostream>
#include <iomanip> // დირექტივა ფორმატით შეტანა-გამოტანის დროს
using namespace std; // სტანდარტული ბიბლიოთეკის კოდების გამოყენება
int main()
{
  int num1 = 1234, num2 = 5678; //ცვლადების გამოცხადება და მნიშვნელობების მინიჭება
  cout << endl;                          //ახალ ხაზზე გადასვლა
  cout << setw(6) << num1 << setw(6) << num2;  //ორი ცვლადის გამოტანა დაშორებებით
  cout << endl;                          // ახალ ხაზზე გადასვლა
  return 0;                              // პროგრამიდან გამოსვლა
}
```

1. 9. ფორმატირების ალმები

ფორმატირების ესა თუ ის სახეობა, რომელიც სრულდება შეტანა-გამოტანა ოპერაციების პროცესში მიეთითება ფორმატირების ალმებით, რაც ხორციელდება setf, unsetf და flags ფუნქცია-ელემენტებით.

ალამი showpoint ყენდება იმისათვის, რომ საჭიროების შემთხვევაში, მცოცმძიმიანი რიცხვი რომელსაც მთელი მნიშვნელობა აქვს, იქნეს გამოტანილი ათობითი წერტილის და ნულების

აუცილებელი მითითებით. თუ გამოტანა ხორციელდება მონიტორის ეკრანზე მაშინ იგი ჩაიწერება პროგრამაში cout.setf(ios::showpoint) სახით, ხოლო თუ გამოტანა ხორციელდება ფაილში მაშინ კი res.setf(ios::showpoint). აქ res ფაილის სახელია, ნულების რაოდენობა, რომელიც განთავსდება რიცხვის წილად ნაწილში განისაზღვრება პროგრამაში მითითებული სიზუსტით.

ფარდობითი ოპერაციები

C++ პროგრამირების ენაში ისევე როგორც სხვა ალგორითმულ ენებში გამოიყენება მონაცემების ტიპი bool, რომელსაც აქვს ორი შესაძლო მნიშვნელობა: FALSE (მცდარი) და TRUE (ჭეშმარიტი). კომპიუტერში FALSE -ს შესაბამისი რიცხვითი მნიშვნელობა არის 0, ხოლო TRUE –ს 1. bool ტიპის გამოყენება ფარდობით ოპერაციებში აუცილებელია, რადგან ყოველი ფარდობითი ოპერაცია გამოითვლება როგორც ტიპი, რომლის მნიშვნელობა შეიძლება იყოს მცდარი(0) ან ჭეშმარიტი(1). ფარდობითი ოპერაციები მოცემულია ცხრილში:

სახელი	ოპერატორი	მაგალითი	გამოთვლის შედეგი
ტოლობა	==	100==50	false
		50==50	true
არ არის ტოლი	!=	100!=50	true
		50!=50	false
მეტია	>	100>50	true
		50>50	false
მეტია ან ტოლი	>=	100>50	true
		50>=50	true
ნაკლებია	<	100<50	false
		50<50	false
ნაკლებია ან ტოლი	<=	100<50	false
		50<=50	true

ლოგიკური გამოსახულება

ლოგიკური გამოსახულება შეიძლება იყოს მარტივი ან რთული. მარტივი ლოგიკური გამოსახულება წარმოადგენს ფარდობითი ოპერაციის ნიშნით დაკავშირებულ ორ არითმეტიკულ გამოსახულებას. მაგ. რთული ლოგიკური გამოსახულება წარმოადგენს ორ ან ორზე მეტ მარტივ ლოგიკურ გამოსახულებას დაკავშირებულს ლოგიკური ოპერაციის ნიშნით. (გამონაკლისს წარმოადგენს "არა" ოპერაცია, რომელიც წარმოადგენს უნარულ ოპერაციას). იმის და მიუხედავათ, თუ რა სახის არის ლოგიკური გამოსახულება, მისი მნიშვნელობა შეიძლება იყოს "ჭეშმარიტი" ან "მცდარი".

ლოგიკური ოპერაციები

ოპერატორი	სიმბოლო	მაგალითი
ლოგიკური "და"	&&	a>b && c!=d
ლოგიკური "ან"	||	k==m || n<v
ლოგიკური "არა"	!	!a

ლოგიკური ოპერაცია “და”-ს მნიშვნელობა არის “ჭეშმარიტი” იმ შემთხვევაში, თუ ჭეშმარიტია მასში შემავალი ყველა მარტივი ლოგიკური გამოსახულებების მნიშვნელობები. მაგ. ((x==5)&&(y==5)) ამ გამოსახულებაში მთლიანი გამოსახულების მნიშვნელობა ჭეშმარიტია თუ x=5 და y=5. თუ ერთი მათგანი არ არის 5 ტოლი, მთლიანი ლოგიკური გამოსახულების მნიშვნელობა იქნება მცდარი.

ლოგიკური ოპერაცია “ან”-ს მნიშვნელობა არის “მცდარი” მხოლოდ მაშინ, თუ არის მცდარი მასში შემავალი ყველა მარტივი ლოგიკური გამოსახულებების მნიშვნელობები. თუ ერთი მაინც მარტივი ლოგოკური გამოსახულების მნიშვნელობა არის ჭეშმარიტი, მაშინ მთლიანი ლოგიკური გამოსახულების მნიშვნელობა იქნება ჭეშმარიტი. მაგ. ((x==5)||(y==5)) ჭეშმარიტია თუ ან x ან y უდრის 5.

ლოგიკური “უარყოფა” ჭეშმარიტია როდესაც არგუმენტის მნიშვნელობაა ″მცდარი″ და პირიქით, მცდარია მაშინ, როდესაც არგუმენტის მნიშვნელობა არის ″ჭეშმარიტი″. ზემოაღნიშნულ ორ ოპერანდიან მაგალითებში კომპილატორი აღარ გამოითვლის მეორე ოპერანდის მნიშვნელობას, თუ პირველი ნაწილით შესაძლებელია სწორი დასკვნის გაკეთება.

1.10. მართვის პირობითი გადაცემის ოპერატორი (სტრუქტურა if....)

მართვის პირობითი გადაცემის ოპერატორი გამოიყენება განშტოების განხორციელებისათვის განშტოებადი სტრუქტურის ალგორითმების დაპროგრამებისას. არსებობს if ოპერატორის ორი ფორმა: if . . . ოპერატორი და if ... else ოპერატორი

if ოპერატორი ზოგადი სახით ჩაიწერება შემდეგნაირად:

if (ლოგიკური გამოსახულება) ოპერატორი1;

თუ ლოგიკური გამოსახულების მნიშვნელობა არის “ჭეშმარიტი”, მაშინ შესრულდება ოპერატორი1, და პროგრამის შესრულება გაგრძელდება მომდევნო ოპერატორიდან, ხოლო თუ ლოგიკური გამოსახულების მნიშვნელობა არის “მცდარი”, მაშინ არ შესრულდება ოპერატორი1, და პროგრამის შესრულება გაგრძელდება მომდევნო ოპერატორიდან. ე.ი ოპერატორი1 ან შესრულდება ან არა, რაც დამოკიდებულია ფრჩხილებში ჩაწერილი ლოგიკური გამოსახულების მნიშვნელობაზე. თვით ოპერატორი1 შეიძლება იყოს მარტივი, დამთავრებული წერტილ მძიმით ან შედგენილი (ფიგურულ ფრჩხილებში მოთავსებული ერთზე მეტი ოპერატორი).

მაგ. if (a<b) cout<<b;
if (2*x-5>c) {a=c+x; b=c-x:}

if . . . else ... სტრუქტურა ზოგადი სახით ჩაიწერება შემდეგნაირად:

if (ლოგიკური გამოსახულება) ოპერატორი1; else ოპერატორი2;

თუ ლოგიკური გამოსახულების მნიშვნელობა არის “ჭეშმარიტი” შესრულდება ოპერატორი1, რის შემდეგ გაგრძელდება პროგრამის შესრულება მომდევნო ოპერატორიდან. წინააღმდეგ შემთხვევაში შესრულდება ოპერატორი2. ორივე შემთხვევაში პროგრამის შესრულება გაგრძელდება მომდევნო ოპერატორიდან.

მაგ. if (x>z) y=x; else {y=z; k=k+0.35;}

შესაძლებელია ჩადგმული if ოპერატორის არსებობაც:

if (გამოსახულება 1) { if (გამოსახულება 2) ოპერატორი1; else
{ if (გამოსახულება 3) ოპერატორი 2; else { და ა.შ.

მაგ. int a=2, b=7, c=3;

```
if (a>b)
   {
    if (b<c)   c=b;
    }
else
    c=a;
   cout<<c;
```

C++-სს გააჩნია კიდევ ერთი პირობითი ოპერაცია - **? :** , რომელიც if … else სტრუქტურის მაგვარია. აქვს სამი ოპერანდი. პირველი ოპერანდი წარმოადგენს ლოგიკურ გამოსახულებას (პირობას); მეორე შეიცავს პირობითი გამოსახულების მნიშვნელობას ან ოპერატორს, რომელიც უნდა შესრულდეს თუ პირობა – ჭეშმარიტია, მესამე კი – პირობითი გამოსახულების მნიშვნელობას ან ოპერატორს, თუ პირობა – მცდარია.
ამ ოპერაციის გამოყენება შეიძლება:

1. დამოუკიდებელი ოპერატორის სახით მაგალითად:

a<b ? d=b:d=a; შედეგად d ცვლადს მიენიჭება a და b შორის უდიდესის მნიშვნელობა.

2. რაიმე სხვა ოპერატორის ნაწილის სახით მაგალითად:

value=(x>=7 ? 25 : -12); ამ ოპერატორით ცვლადს value –ს მიენიჭება მნიშვნელობა 25, თუ x>=7, ხოლო თუ x<7, მაშინ მიენიჭება მნიშვნელობა –12.

cout<<(a!=b ? a+b : a-b);

თ ა ვ ი 2

განშტოებადი სტრუქტურის ალგორითმის პროგრამირება

შევადგინოთ პროგრამა რომლითაც გამოითვლება Z ცვლადის მნიშვნელობა შემდეგი პირობების გათვალისწინებით

$$Z=\begin{cases} x \cdot \cos x, & \text{თუ } 2{,}5 \le x < 7 \\ 1+e^{x}, & \text{თუ } 7 \le x < 11 \\ x+3\sin x, & \text{წინააღმდეგ შემთხვევაში} \end{cases}$$

სადაც x – მოცემული რიცხვია.

ამოცანის ამოხსნის ალგორითმი იქნება შემდეგი :

თავდაპირველად, კომპიუტერის მეხსიერებაში საჭიროა x ცვლადის მნიშვნელობის შეტანა. შემდეგ შემოწმებული უნდა იქნეს პირობა 2,5≤ x <7, რომლის შესრულების შემთხვევაში საჭიროა z= x · cosx გამოთვლა, წინააღმდეგ შემთხვევაში უნდა შემოწმდეს პირობა 7≤x<11 და თუ სრულდება იგი გამოითვალოს Z= 1+ e^{x} , ხოლო თუ არ სრულდება მაშინ გამოთვალოს Z= x+3sinx გამოსახულების მნიშვნელობა. დასასრულ განხორციელდეს Z-ის მნიშვნელობის გამოტანა.

```
#include<iostream.h>
#include<math.h>
main()
{
float x,z;  //ნამდვილი ტიპის ცვლადების გამოცხადება
cout<<"Enter the value of x:"<<endl; // დაბეჭდე “შეიტანე ცვლადი “ და ახალ ხაზზე გადადი
cin>>x;   // ცვლადის შეტანა კლავიატურით
 if (x>2.5 && x<7) z=x*cos(x); else
  {if (x>=7 && x<11) z=1+e^x; else z=x+3*sin(x) ;}  // პირობის შემოწმება
cout<< z ;   // z –ის მნიშვნელობის ბეჭდვა
 return 0; // ფუნქციიდან გამოსვლა
 }
```

2.1 ციკლური სტრუქტურის ალგორითმების პროგრამირება

მრავალი ამოცანის ამოხსნისას საჭირო ხდება ოპერატორების გარკვეული ნაწილის გამოთვლების მრავალჯერ განმეორება ცვლადების სხვადასხვა მნიშვნელობისათვის. ასეთი სახის პროცესს ციკლური პროცესი ეწოდება. ციკლური პროცესის დასაპროგრამებლად C++ ალგორითმულ ენაში, გამოიყენება როგორც ციკლის ოპერატორიები while, do...while და for, ისე პირობითი ოპერატორი if

```
// შეტანილი სიმბოლოს შემოწმება if ოპერატორის დახმარებით მაგალითი 1.
#include <iostream>
using namespace std;
int main()
{
```

```
  char letter = 0;            // შეტანილს ინახავს letter უჯრაში
  cout << endl           //ახალ სტრიქონზე გადავა და დაბეჭდავს
      << "Enter a letter: "; // “ შეიტანეთ მნიშვნელობა “
  cin >> letter;              //  კლავიატურით შეგაქვთ მნიშვნელობა
  if(letter >= 'A')           // თუ შენახული ცვლადი > ან = 'A'
    if(letter <= 'Z')         // თუ შენახული ცვლადი < ან = 'Z'
    {
      cout << endl    // გადასვლა ახალ სტრიქონზე და ბეჭდვა
          << "You entered a capital letter."          //  “თქვენ შეიტანეთ მთავრული ასო”
          << endl;
      return 0;
    }
  if(letter >= 'a')           // თუ შენახული ცვლადი > ან = 'a'
    if(letter <= 'z')         // თუ შენახული ცვლადი < ან = 'z'
    {
      cout << endl   // გადასვლა ახალ სტრიქონზე და ბეჭდვა
          << "You entered a small letter."           // “თქვენ შეიტანეთ პატარა ასო ‘
          << endl;
      return 0;
    }
  cout << endl << "You did not enter a letter." << endl;  //  “თქვენ არ შეგიტანიათ ასო”
  return 0;   // ფუნქციიდან გამოსვლა
}

// შეტანილი სიმბოლოს შემოწმება if ოპერატორის დახმარებით მაგალითი 2.
#include <iostream>
using namespace std;
int main()
{
  char letter = 0;            // შეტანილს ინახავს letter უჯრაში
  cout << endl
      << "Enter a character: ";   // ბეჭდვა “ შეიტანე სიმბოლო “
  cin >> letter;  // კლავიატურით შეტანა
  // ანბანის მიხედვით შემოწმება
  if(((letter>='A')&&(letter<='Z')) ||   // თუ შეტანილია  A-დან Z-მდე  ან
    ((letter>='a')&&(letter<='z')))   // a-dan  z-mde
    cout << endl
        << "You entered a letter." << endl;  // თქვენ შეიტანეთ ასო
    else
        cout << endl
             <<"You didn't enter a letter."<< endl; // თქვენ არ შეგიტანიათ ასო
  return 0;
```

```
}

// EX2_04.CPP
// if ოპერატორის გამოყენება გამოტანისთვის
#include <iostream>
using namespace std;
int main()
{
  int nCakes = 1;          // ითვლის ნამცხვრების რაოდენობას
  cout << endl
      << "We have " << nCakes << " cake"    // გაქვთ 1 ნამცხვარი
      << ((nCakes>1) ? "s." : ".")       // თუ nCakes>1 მაშინ დაწერს “cakes.” თუ არა და “cake.”
      << endl;
  ++nCakes; //გაზრდის nCakes რაოდენობას
  cout << endl  // ახალ ხაზზე გადასვლა და ბეჭდვა
      << "We have " << nCakes << " cake"    // გაქვთ 2 ნამცხვარი
      << ((nCakes>1) ? "s." : ".")
      << endl;
  return 0;
}
```

2. 2. განმეორებადი სტრუქტურა while

განმეორებადი სტრუქტურა while ზოგადი სახით ჩაიწერება შემდეგნაირად:

```
while (ლოგიკური გამოსახულება)
      ოპერატორი (მარტივი ან შედგენილი);
```

განმეორებადი სტრუქტურა while -ს შესრულების დროს, ციკლის ყოველი განმეორებისას გამოითვლება ფრჩხილებში ჩაწერილი ლოგიკური გამოსახულების მნიშვნელობა და მოწმდება “ჭეშმარიტია” იგი თუ “მცდარი”. თუ ლოგიკური გამოსახულების მნიშვნელობა “ჭეშმარიტია” შესრულდება ციკლის ტანი ე. ი. მოხდება ციკლის განმეორება, ხოლო თუ “მცდარია” განხორციელდება ციკლიდან გამოსვლა, ე.ი. მართვა გადაეცემა ციკლის ტანის შემდეგ ჩაწერილ პირველივე ოპერატორს. ამ შემთხვევაში ციკლის ტანი წარმოადგენს ერთ ოპერატორს, სახელდობრ მარტივს ან შედგენილს.

შევადგინოთ პროგრამა, რომელიც დაბეჭდავს მოცემულ [a ;b] მონაკვეთზე, რაიმე h ბიჯით გამოთვლილი ფუნქციის დადებით მნიშვნელობებს.

```
#include<iostream.h>
#include<math.h>

main ()

{
   float a,b,h,x,y;  //ცვლადების გამოცხადება
 cin>>a>>b>>h;  //ცვლადების მნიშვნელობების შეტანა კლავიატურიდან
 x=a;  // x ცვლადის ინიციალიზაცია
 while (x<=b)   // ციკლის სათაური, სადაც ხდება პირობის შემოწმება
{
```

```
 y=(exp(x)+log(fabs(x)))*sin(x); //  ფუნქციის მნიშვნელობის გამოთვლა ცვლადის მიმდინარე
                                    მნიშვნელობისათვის
 if (y>0) cout << “x=“<<x
 <<“   y=“<<y<<endl; //თუ ცვლადის მნიშვნელობა დადებითია, ეკრანზე გამოიტანება არგუმენტის
                       და ფუნქციის მნიშვნელობები
 x+=h ; //  ცვლადის მნიშვნელობის გაზრდა მოცემული ბიჯით და ციკლის გამეორება
}
 return 0 ;
}
```

// EX2_12.CPP

```
// while ციკლის გამოყენება
#include <iostream>
using namespace std;
int main()
{
   double value = 0.0; // შეტანილს ინახავს value უჯრაში
   double sum = 0.0;   // მნიშვნელობების ჯამი ინახება  sum  უჯრაში
   int i = 0;          // ითვლის მნიშვნელობების რაოდენობას
   char indicator = 'y';   // გააგრძელოს თუ არა?
   while(indicator == 'y') ; // გაგრძელდეს ციკლი მანამ სანამ თქვენ შეგაქვთ y
   {
     cout << endl
          << "Enter a value: ";
     cin >> value;        // წაიკითხე მნიშვნელობა
     ++i;                 // გაზარდე i
     sum += value;        // მიმდინარე შენატანი დაუმატე sum-ს
     cout << endl
      << "Do you want to enter another value  // კიდევ გსურთ მნიშვნელობის შეტანა (თუ არა დააჭირე n-ს)?
                          (enter n to end)? ";
     cin >> indicator;             // წაიკითხე ინდიკატორი
   }
   cout << endl
     << "The average of the " << i
     << " values you entered is " << sum/i << "." // გამოიტანს შეტანილი რიცხვების საშუალო არითმეტიკულს
     << endl;
   return 0;
}
```

2. 3. განმეორებადი სტრუქტურა do...while.

do . . . …while-ს ტიპის ციკლს შემდეგი სტრუქტურა აქვს:

```
do
ოპერატორი (მარტივი ან შედგენილი)
while (ლოგიკური გამოსახულება);
```

ციკლი do . . . …while სრულდება შემდეგნაირად: ციკლის შესრულება იწყება ციკლის მოქმედების არეში შემავალი ოპერატორებით (სრულდება ციკლის ტანი), შემდეგ გამოითვლება ლოგიკური გამოსახულების მნიშვნელობა (მოწმდება ციკლის დამთავრების პირობა), და თუ იგი არის "ჭეშმარიტი" ხორციელდება ციკლის განმეორება; წინააღმდეგ შემთხვევაში – გამოსვლა ციკლიდან. რეკომენდებულია ციკლის ტანის ფიგურულ ფრჩხილებში მოთავსება იმის და მიუხედავათ, თუ რამდენ ოპერატორს შეიცავს ციკლის ტანი.

ციკლური სტრუქტურის ალგორითმის პროგრამირება განმეორებადი სტრუქტურა do . . .while-ს გამოყენებით

შევადგინოთ პროგრამა რომლითაც გამოითვლება $y = (e^x + \ln|x|) \cdot \cos(x)$ ფუნქციის უარყოფითი მნიშვნელობების ჯამი, რომელიც გამოთვლილია [a,b] მონაკვეთზე მოცემული h ბიჯით. თუ აღმოჩნდა, რომ ფუნქციის არცერთი მნიშვნელობა არ არის უარყოფითი, უნდა დაიბეჭდოს შესაბამისი ტექსტი.

```
#include <iostream.h>
#include<math.h>

main ( )
{
float a,b,h,x,y,s=0; // ცვლადების გამოცხადება და s უჯრის გაწმენდა
cout<<"შეიტანეთ  a,b,h-ის მნიშვნელობები : "<<endl; //ახალ ხაზზე გადასვლა
cin>>a>>b>>h;    // ცვლადის მნიშვნელობების შეტანა კლავიატურიდან
x=a;   //x ცვლადზე a-ს მნიშვნელობის მინიჭება
while b-a>0.00001
{
y=(exp(x)+log(fabs(x)))*cos(x);
if (y<0) s+=y;
x+=h;  // x ცვლადს ემატება ბიჯი და მოწმდება ციკლის დამთავრების პირობა x<=b, თუ
          "ჭეშმარიტია" ციკლი გამეორდება
}
if (s !=0) cout<<"s="<<s; else   //ჯამის მნიშვნელობის გამოტანა
cout<<"ფუნქციის ყველა მნიშვნელობა არაუარყოფითია"<<endl;
  return 0;

}
```

ცვლადების გამოცხადებისას ხორციელდება s ცვლადის, (რომელმაც უნდა მიიღოს ჯამის მნიშვნელობა) ინიციალიზაცია s=0. საწყისი მონაცემების შეტანის შემდეგ და x=a; ოპერატორის შესრულების შემდეგ იწყება ციკლის შესრულება, რომელშიც გამოითვლება y ფუნქციის მნიშვნელობა და თუ იგი აღმოჩნდა უარყოფითი, მისი მნიშვნელობა ემატება s ცვლადს. შემდეგ x ცვლადის მნიშვნელობა იცვლება h ბიჯით და მხოლოდ ამის შემდეგ მოწმდება ციკლის დამთავრების პირობა x<=b რომლის შესრულების შემთხვევაში გან-მეორდება ციკლი. ციკლის დამთავრების შემდეგ დაიბეჭდება ან მიღებული ჯამის მნიშვნე-ლობა, ან შესაბამისი შეტყობინება, თუკი ფუნქციის არცერთი მნიშვნელობა არ აღმოჩნდა უარყოფითი.

2.4. განმეორებადი სტრუქტურა for

for ოპერატორორის ზოგადი სტრუქტურაა:
for (გამოსახულება 1; გამოსახულება 2; გამოსახულება 3)

ციკლის ტანი.

სტრუქტურა for სრულდება შემდეგი მიმდევრობით:
გამოსახულება 1-ით ხორციელდება ციკლის ცვლადის ინიციალიზება, ე. ი. ენიჭება საწყისი მნიშვნელობა. ციკლის ყოველი განმეორებისას გამოსახულება 3 ახდენს ციკლის ცვლადის მნიშვნელობის შეცვლას, რის შემდეგ გამოსახულება 2 ამოწმებს ციკლის გამეორების პირობას, რომლის შესრულების შემთხვევაში მეორდება ციკლის ტანის შესრულება; წინააღმდეგ შემთხვევაში ხორციელდება გამოსვლა ციკლიდან.

მაგ. for (int j=x; j< 4*x+y; j+=y/x)
for(int j =99; j>=0; j - =11)

```
// EX2_08.CPP
// ცვლადების შეკრება ციკლის for ოპერატორის გამოყენებით
#include <iostream>
using namespace std;
int main()
{
  int i = 0, sum = 0;  // მთელი ტიპის ცვლადების i და sum –ის ინიციალიზაცია (გაწმენდა)
  const int max = 10;  // ცვლადი max მუდმივაა
  for(i = 1; i <= max; i++)    // ციკლის სათაური
    sum += i;                  // დაუმატე i-ს მიმდინარე მნიშვნელობა sum-ს
  cout << endl  //ახალ სტრიქონზე გადასვლა
       << "sum = " << sum      //sum ცვლადის მნიშვნელობის გამოტანა
       << endl       //ახალ სტრიქონზე გადასვლა
       << "i = " << i          // მთვლელის გამოტანა
       << endl;
  return 0;
}
```

შეადგინეთ პროგრამა რომელიც $y = (e^{x^2} + \sqrt[3]{x}) \cdot \sin(x)$ ფუნქციის [a;b] მონაკვეთზე, h ბიჯით გამოთვლილი მნიშვნელობებიდან მონახავს უდიდესს.

```
#include<iostream.h>
#include<math.h>
main ()

{
  float a,b,h,x,y,r=-1e6;
cout<<"შეიტანეთ a,b,h –ს მნიშვნელობები:";
cin>>a>>b>>h;
for (x=a; x<=b; x+=h)
{
y=(exp(pow(x,2))+pow(x,1.0/3))*sin(x);
```

```
if (y>r) r=y;
}
cout<< « Y ფუნქციის მაქსიმალური მნიშვნელობაა »<<r<<endl ;
return 0 ;
}
```

შენიშვნა : ნამდვილ რიცხვებზე მოქმედების შესრულების შედეგი წარმოადგენს მიახლოებით მნიშვნელობას (ადგილი აქვს დამრგვალების ცდომილებას). ამიტომ, როდესაც ციკლის ცვლადს წარმოადგენს ნამდვილი რიცხვი და ამასთან b-a მონაკვეთი h სიდიდის ჯერადია, შეიძლება აღმოჩნდეს რომ ბოლო გამეორების წინ x=b ნაცვლად აღმოჩნდება რომ x=b+0.000…1 რის გამოც არ განხორციელდება ციკლის ბოლო გამეორების შესრულება. ამიტომ ამ ფაქტის თავიდან აცილების მიზნით უმჯობესია x≤b პირობის შემოწმების ნაცვლად გამოყენებული იქნეს b-x ≤eps პირობა, სადაც eps–ს მნიშვნელობა შერჩეული უნდა იქნეს h მნიშვნელობის გათვალისწინებით.

თ ა ვ ი 3

ჩ ა დ გ მ უ ლ ი ც ი კ ლ ე ბ ი

ციკლის ტანში შესაძლებელია ჩადგმული იყოს სხვა ციკლი. ასეთი სახის ციკლს ჩადგმული ციკლი, რთული ციკლი ან ციკლი ციკლში ეწოდება. შიგა ციკლი სრულდება სრულად გარე ციკლის ყოველი გავლისას.

ამ ტიპის ციკლის უმარტივეს მაგალითს წარმოადგენს ორი ცვლადის ფუნქციის მნიშვნელობების გამოთვლის ალგორითმი.

შევადგინოთ პროგრამა, რომლითაც გამოითვლება $Y=x^2 \ln z+\sqrt{xz}$ ფუნქციის მნიშვნელობები, თუ x იცვლება [a;b] მონაკვეთზე h_x ბიჯით, ხოლო z ცვლადი [c;d] მონაკვეთზე h_z ბიჯით.

```
#include<iostream.h>
#include<math.h>
main ()
{
float a,b,c,d,hx,hz,x,y,eps=0.00001;
cin>>a>>b>>c>>d>>hx>>hz ;

for (x=a ; fabs(b-a)<=eps ; x+=hx)
for(z=c; fabs(c-d)<=eps; z+=hz)
{
y=pow(x,2)*log(z) + sqrt(x*z);
cout<< "x="<<x<< "\tz="<<z
<<"\ty= "<<y<<endl ;
}
return 0 ;
}
```

შიგა ციკლის ტანი

გარე ციკლის ტანი

გარე ციკლის ცვლადია x, ხოლო შიგა ციკლისა კი z. გარე ციკლის პირველი მნიშვნელობისათვის მთლიანად სრულდება შიგა ციკლი, ფიქსირდება ცვლადის მნიშვნელობა და მართვა გადაეცემა გარე ციკლის დასაწყისს. შეიცვლება გარე ციკლის ცვლადის მნიშვნელობა შესაბამისი ბიჯით და გამეორდება ისევ შიგა ციკლი ცვლადის ყველა მნიშვნელობისათვის და ა. შ.

3. 1. ოპერატორები break და continue

ოპერატორები break და continue-ს საშუალებით ხდება for, while, do… while და switch (იხილე ქვემოთ) სტრუქტურებში ოპერატორების შესრულების ბუნებრივი თანმიმდევრობის შეცვლა. break ოპერატორის შესრულებისას ხორციელდება სტრუქტურიდან გამოსვლა და მართვა გადაეცემა პროგრამის მიმდინარე სტრუქტურის შემდეგ ჩაწერილ პირველივე ოპერატორს. ამრიგად ოპერატორი break-ის დანიშნულებაა, ციკლის შესრულების დროზე ადრე შეწყვეტა, ან switch სტრუქტურის დარჩენილი ნაწილის გამოტოვება.

ოპერატორი continue იწვევს for, while და do/while სტრუქტურებში პროგრამის ტანის დარჩენილი ნაწილის გამოტოვებას და მართვის გადაცემა ხდება ციკლის მომდევნო გამეორების შესრულებაზე. while და do/while სტრუქტურებში ხორციელდება გადასვლა ციკლის დამთავრების პირობის შემოწმებაზე. სტრუქტურა for-ში წარმოებს გადასვლა ციკლის ცვლადის მნიშვნელობის შეცვლის გამოსახულებაზე, რის შემდეგ მოწმდება ციკლის დამთავრების პირობა.

მაგ.1.

```
. . . . . .
for (int x=-5 ; x<=2 ; x++) {
if (x==0) continue;
cout<<x<<” “; }
. . . . . .
```

მაგ. 2

```
. . . . . .
for (int x=-5 ; x<=2 ; x++) {
if (x==0) break;
cout<<x<<” “; }
. . . . . .
```

3. 2. ოპერატორი switch

ოპერატორი switch-ის მეშვეობით, if და if. . .else. . . ამორჩევის სტრუქტურებისაგან განსხვავებით, შესაძლებელია განხორციელებული იქნეს მრავალჯერადი ამორჩევები.

```
switch (სელექტორი) {
 case ჭდე 1: ოპერატორი 1; break;
  case ჭდე 2: ოპერატორი 2; break;
case ჭდე 3: ოპერატორი 3; break;
. . . . . .
 case ჭდე n: ოპერატორი n; break;
default: ოპერატორი; }
```

სელექტორი - C++ ენაში დაშვებული, ნებისმიერი მთელი ტიპის გამოსახულებაა, ოპერატორი კი ნებისმიერი ოპერატორია – მარტივი ან შედგენილი.

switch სტრუქტურა სრულდება შემდეგნაირად : გამოითვლება სელექტორის მნიშვნელობა და შემდეგ შესრულებისათვის ამოირჩევა ის ოპერატორი რომლის ჭდე (ჭდე i :) სელექტორის მნიშვნელობის ტოლია. თუ აღმოჩნდა რომ სელექტორის მნიშვნელობა არ ემთხვევა არცერთ ჭდეს, მაშინ განხორციელდება გადასვლა default : მართვა გადაეცემა ჭდეს, რომელიც არ წარმოადგენს აუცილებელს, ან გაიცემა შესაბამისი შეტყობინება და შესრულდება ალგორითმით გათვალისწინებული მოქმედება. ყოველი case-ს სტრუქტურის გამოთვლა გრძელდება break ოპერატორის შეხვედრამდე ან switch-ის დასრულებამდე.

შემთხვევა case "A": case "B": – რამოდენიმე ჭდის მიმდევრობითი ჩამონათვალი ნიშნავს რომ თუ სელექტორის მნიშვნელობა ემთხვევა მითითებული მნიშვნელობებიდან ნებისმიერს, უნდა შესრულდეს ერთი და იგივე მოქმედება ანუ ერთი და იგივე ოპერატორი.

1.
```
#include<iostream.h>
main() {
int k;
cout<<"k=";
cin>>k;
switch (k) {
case 1: case 3: case 5:
 cout<<"k კენტია"<<endl;
break;
case 2:  cout<<"k=2"<<endl;
break;
case 4:  cout<<"k=4"<<endl;
break;
```

2.
```
#include<iostream.h>
main()  {
int k;
cout<<"k=";
cin>>k;
switch (k) {
case 1: case 3: case 5:
 cout<<"k კენტია"<<endl;
break;
case 2:  cout<<"k=2"<<endl;
break;
 case 4:  cout<<"k=4"<<endl;
 }
```

```
default : cout<<"k<1 ან k>5"
<<endl;
}
return 0;
}
```

```
return 0;
}
```

პროგრამის პირველ ვარიანტში k ცვლადის შეტანილი მნიშვნელობის მიხედვით ეკრანზე გამოიტანება ან k კენტია, ან k=2, ან k=4 ან შეტყობინება, რომ $k \notin [1;5]$ მონაკვეთს.

პროგრამის მეორე ვარიანტში კი $k \notin [1;5]$ შემთხვევაში არ გამოიტანება შესაბამისი შეტყობინება.

3. 3. მაგალითები ციკლის ოპერატორების გამოყენებით

```
// EX2_07.CPP
// ციკლის შექმნა if და goto ოპერატორის გამოყენებით
#include <iostream>
using namespace std;
int main()
{
  int i = 0, sum = 0;  // ცვლადების გამოცხადება და ინიციალიზაცია
  const int max = 10;
  i = 1;
loop:                  // ციკლის დასაწყისი
  sum += i;            // დაუმატე i-ს მიმდინარე მნიშვნელობა sum-ს
  if(++i <= max)   // პირობის შემოწმება
    goto loop;         //  ციკლის სათაურზე გადასვლა სანამ i <= 11
  cout << endl    //ახალ ხაზზე გადასვლა და ბეჭდვა
       << "sum = " << sum
       << endl
       << "i = " << i // მთვლელის ბეჭდვა
       << endl;
  return 0;
}
```

```
// EX2_08.CPP
// ცვლადების შეკრება ციკლის for ოპერატორის გამოყენებით
#include <iostream>
using namespace std;
int main()
{
  int i = 0, sum = 0;
  const int max = 10;
  for(i = 1; i <= max; i++)   // ციკლის სათაური
    sum += i;                 // დაუმატე i-ს მიმდინარე მნიშვნელობა sum-ს
  cout << endl
       << "sum = " << sum
       << endl
       << "i = " << i
       << endl;
  return 0;
}
```

```
// EX2_09.CPP
// მრავლობითი ციკლის გამოყენება
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  long i = 0, power = 0;
  const int max = 10;
  for(i = 0, power = 1; i <= max; i++, power += power)     // ციკლის გამოცხადება
    cout << endl
         << setw(10) << i << setw(10) << power;
  cout << endl;
  return 0;
}

// EX2_10.CPP
// შეუზღუდავი for ციკლის გამოყენება
#include <iostream>
using namespace std;
int main()
{
  double value = 0.0; // შეტანილ მნიშვნელობას შეინახავს value-ში
  double sum = 0.0;   // მნიშვნელობები გროვდება sum -ში
  int i = 0;          // ითვლის მნიშვნელობების რაოდენობას
  char indicator = 'n';    // გააგრძელოს თუ არა?
  for(;;)                  // შეუზღუდავი ციკლი
  {
    cout << endl
         << "Enter a value: ";
    cin >> value;   // წაიკითხე მნიშვნელობა
    ++i;                 // i იზრდება 1-ით
     sum += value;         // დაუმატე მიმდინარე შენატანი sum-ს
    cout << endl
         << "Do you want to enter another value
                              (enter n to end)? ";
    cin >> indicator;     // წაკითხვის ინდიკატორი
    if ((indicator == 'n') || (indicator == 'N'))
      break;             // გამოდი ციკლიდან
  }
  cout << endl
       << "The average of the " << i
       << " values you entered is " << sum/i << "."
       << endl;
  return 0;
```

```
}

// EX2_11.CPP
// ASCII კოდების გამოტანა ანბანის მიხედვით
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  for(char capital='A', small='a'; capital<='Z'; capital++, small++)
    cout << endl
        << "\t" << capital                          // გამოიტანე დიდი ასოები
        << hex << setw(10) << static_cast<int>(capital) // გამოიტანე დიდი ასოები თექვსმეტობით სისტემაში
        << dec << setw(10) << static_cast<int>(capital) // გამოიტანე დიდი ასოები ათობით სისტემაში
        << "    " << small                          // გამოიტანე პატარა ასოები
        << hex << setw(10) << static_cast<int>(small)   // გამოიტანე პატარა ასოები თექვსმეტობით სისტემაში
        << dec << setw(10) << static_cast<int>(small);  // გამოიტანე პატარა ასოები ათობით სისტემაში
  cout << endl;
  return 0;
}

// EX2_12.CPP
// while ციკლის გამოყენება
#include <iostream>
using namespace std;
int main()
{
  double value = 0.0; // შეტანილს ინახავს value უჯრაში
  double sum = 0.0;   // მნიშვნელობების ჯამი ინახება sum  უჯრაში
  int i = 0;          // ითვლის მნიშვნელობების რაოდენობას
  char indicator = 'y';   // გააგრძელოს თო არა?
  while(indicator == 'y') // გაგრძელდეს ციკლი მანამ სანამ თქვენ შეგაქვთ y-ის მნიშვნელობა
  {
    cout << endl
        << "Enter a value: ";
    cin >> value;       // წაიკითხე მნიშვნელობა
    ++i;                // გაზარდე i
    sum += value;       // მიმდინარე შენატანი დაუმატე sum-ს
    cout << endl
      << "Do you want to enter another value   // კიდევ გსურთ მნიშვნელობის შეტანა (თუ არა დააჭირე n-ს)?
                        (enter n to end)? ";
    cin >> indicator;           // შეიტანე ინდიკატორი
  }
  cout << endl
    << "The average of the " << i
```

```
    << " values you entered is " << sum/i << "." // გამოიტანს შეტანილი რიცხვების საშუალო
არითმეტიკულს
    << endl;
  return 0;
}

// EX2_13.CPP
// ციკლის გამოყენება ფაქტორიალის გამოსათვლელად
#include <iostream>
using namespace std;
int main()
{
  char indicator = 'n';
  long value = 0,              // შეტანილს ინახავს value უჯრაში
       factorial = 0;
  do
  {
    cout << endl
         << "Enter an integer value: ";  // შეიტანეთ მთელი რიცხვი
    cin >> value;
    factorial = 1;
    for(int i = 2; i<=value; i++)          // ციკლის დასაწყისი
      factorial *= i;
    cout << "Factorial " << value << " is " << factorial;   // გამოიტანს შეტანილი რიცხვის ფაქტორიალს
    cout << endl
         << "Do you want to enter another value           //  კიდევ გსურთ რიცხვის შეტანა?
                                (y or n)? ";
    cin >> indicator;
  }
 while((indicator=='y') || (indicator=='Y'));
  return 0;
}

// EX2_14.CPP
// ციკლის გამოყენება გამრავლების ტაბულის გამოსატანად
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  const int size = 12;         // ტაბულის ზომა
  int i = 0, j = 0;            // ციკლის დამთვლელები
  cout << endl                 // გამოიტანე ტაბულის სახელი
       << size << " by " << size
       << " Multiplication Table" << endl << endl;
  cout << endl << "    |";
  for(i=1; i<=size; i++)       // ციკლი სვეტების სათაურის გამოსატანად
```

```
    cout << setw(3) << i << "  ";
  cout << endl;              // ახალი ხაზი ხაზის ქვეშ გასასმელად
  for(i=0; i<=size; i++)
    cout << "_____";         // სათაურებს გაუსვი ქვეშ ხაზი
  for(i=1; i<=size; i++)      // გარეთა ციკლი ხაზებისათვის
  {
    cout << endl
         << setw(3) << i << " |";  // გამოიტანე ხაზების ჭდე
    // შიგა ციკლი ნამრავლის გამოსატანად
    for(j=1; j<=size; j++)
      cout << setw(3) << i*j << "  ";
    // შიგა ციკლის დასასრული
  }                          // გარეთა ციკლის დასასრული
  cout << endl;
  return 0;
}
```

თ ა ვ ი 4

მ ა ს ი ვ ე ბ ი

მასივი არის მეხსიერების მიმდევრობით განლაგებული უჯრედების ერთობლიობა, რომლებიც განკუთვნილია ერთი და იგივე ცვლადის სხვადასხვა მნიშვნელობების დასამახსოვრებლად. მასივის ამა თუ იმ უჯრედზე მიმართვისათვის საჭიროა მასივის სახელის და კვადრატულ ფრჩხილებში ჩაწერილი მასივის ცალკეული ელემენტის პოზიციის ნომრის მითითება. ყოველი მასივის რიგით პირველი ელემენტი – ნულოვანი ელემენტია. ამრიგად, c მასივის პირველ ელემენტზე მიუთითებენ – c[0], მეორეზე – c[1] და ზოგადად რიგით n-ურ ელემენტზე c[n-1]. ფრჩხილებში მითითებულ პოზიციის ნომერს ინდექსი ეწოდება. ინდექსად შეიძლება მთელი ტიპის რიცხვის ან გამოსახულების გამოყენება. მასივის ელემენტების ინდექსაცია იწყება 0-დან.

4. 1. მასივების გამოცხადება

მასივის გამოცხადებისას უნდა მიეთითოს სახელი, ტიპი და ელემენტების რაოდენობა. ამ ინფორმაციის საფუძველზე კომპილატორი გამოყოფს მეხსიერების შესაბამის არეს. მაგალითად, int a[25]; წარმოადგენს მთელი ტიპის 25 ელემენტიან a მასივს. ერთი გამოცხადებით შეიძლება ერთზე მეტი ერთი და იგივე ტიპის მასივის გამოცხადება. მაგ. float b[15], c[10] გამოცხადებით გამოიყოფა ორი b[15] და c[10] მასივი, განკუთვნილი ნამდვილი ტიპის მონაცემებისათვის.

მასივებს შეიძლება ჰქონდეთ ერთზე მეტი ინდექსი. მრავალგანზომილებიანი მასივებიდან უფრო მეტად გამოიყენება ორგანზომილებიანი მასივი ანუ მატრიცა, რომელიც შეიცავს ინფორმაციას სტრიქონებში და სვეტებში. მატრიცის გამოცხადებისას საჭიროა სტრიქონების და სვეტების რაოდენობის ცალცალკე მითითება. მაგალითად c[10][10] გამოცხადებით გამოიყოფა მთელ რიცხვთა c კვადრატული მატრიცა, რომლის როგორც სტრიქონების ასევე სვეტების რაოდენობა 10 ტოლია. მატრიცის კონკრეტული ელემენტის განსაზღვრისათვის საჭიროა ორი ინდექსის მითითება, რომელთაგან პირველი წარმოადგენს სტრიქონის ნომერს, ხოლო მეორე სვეტის ნომერს, რომელთა გადაკვეთაზეც იმყოფება ელემენტი. კიდევ ერთხელ უნდა აღინიშნოს, რომ მასივში პოზიციების ნუმერაცია იწყება 0-დან. ამრიგად c[0][0] ელემენტი წარმოადგენს c მატრიცის რიგით პირველი სტრიქონის და პირველი სვეტის გადაკვეთაზე მყოფ ელემენტს. a[5] კი წარმოადგენს ერთგანზომილებიანი a მასივის იმ ელემენტს, რომლის ინდექსი 5 ტოლია.

4. 2. მასივის ელემენტების შეტანა

დავუშვათ, რომ შესატანია 15 ელემენტიანი მთელ რიცხვთა ერთგანზომილებიანი a მასივის ელემენტების მნიშვნელობები. პროგრამის ფრაგმენტს ექნება შემდეგი სახე:

```
. . . . . .
int a[15]; int i;   // გამოცხადება a მასივისა და i ციკლის მთვლელის
for (int i=0; i<15; i++)  // ციკლის სათაური
cin>>a[i];  // მასივის ელემენტების შეტანა კლავიატურიდან
   . . . . . .
```

ორგანზომილებიანი მასივის შემთხვევაში პროგრამის ფრაგმენტს კი ექნება შემდეგი სახე:

```
. . . . . .
float a[10][10]; int i,j; // გამოცხადება  a მასივის, i და  j ციკლის მთვლელების
for (i=0; i<10; i++)   // გარე ციკლის სათაური
```

```
for (j=0; j<10; j++)  // შიგა ციკლის სათაური
cin>>a[i][j];    // მასივის ელემენტების შეტანა კლავიატურიდან
. . . . . .
```

მასივის განსაზღვრა შეიძლება მასივის გამოცხადებისას, მისი ელემენტების ინიციალიზაციით, რისთვისაც ფიგურულ ფრჩხილებში უნდა მიეთითოს ერთმანეთისაგან მძიმით გამოყოფილი მასივის ელემენტების მნიშვნელობები.
მაგ. int a[4]={8,3,12,5};

თუ საწყისი მნიშვნელობების მითითებული რაოდენობა მასივის ელემენტებზე ნაკლებია, მაშინ დანარჩენი ელემენტები მიიღებენ 0-ის ტოლ მნიშვნელობებს.

მაგალითად int a[4]={8}; აქ a[0]=8; a[1]=0; a[2]=0; a[3]=0. ამრიგად, a მასივის გამოცხადებისას მისი განულებისათვის საკმარისია int a[4]={0}; ჩაწერა.
შენიშვნა უნდა აღინიშნოს, რომ მასივის გამოცხადებისას არ ხორციელდება მისი ავტომატურად განულება. თუ მითითებულია მასივის ელემენტებზე მეტი რაოდენობის მნიშვნელობები ადგილი ექნება სინტაქსურ შეცდომას.

შესაძლებელია ორგანზომილებიანი მასივის ინიციალიზაცია მისი გამოცხადებისას ისევე როგორც ეს კეთდება ერთგანზომილებიანი მასივის შემთხვევაში. მაგალითად ორგანზომილებიანი მთელ რიცხვთა a[2][2] მასივის ინიციალიზაციისათვის ჩაიწერება int a[2][2]={{1,3},{4,6}}; მონაცემები ჯგუფდება სტრიქონების მიხედვით, რომელიც თავსდება ფიგურულ ფრჩხილებში. ამრიგად, a[0][0]=1; a[0][1]=3; a[1][0]=4; a[1][1]=6;

თუ მასივის განსაზღვრისას არ არის მითითებული მასივის განზომილება, იგი ავტომატურად განისაზღვრება მასივის მითითებული ელემენტების რაოდენობით. მაგალითად int b[]={9,2,6,15,23,4,7,25}; განსაზღვრით გამოიყოფა int b[8] მასივი, რომლის ელემენტები მიიღებენ მითითებულ მნიშვნელობებს.

4.3 მასივის ელემენტების გამოტანა

ერთგანზომილებიანი მასივის ელემენტების გამოტანა ხორციელდება ისეთივე სქემით როგორც ხორციელდება მასივის ელემენტების შეტანა. მაგალითად

```
. . . . . .
int a[15]; int i;  // გამოცხადება a მასივისა და i ციკლის მთვლელის
for (int i=0; i<15; i++)  // ციკლის სათაური
 cout<<a[i]<<" ";  // მასივის ელემენტების გამოტანა
  . . . . . .
```

განსხვავებულად შეტანისაგან ხორციელდება ორგანზომილებიანი მასივის ელემენტების გამოტანა იმ შემთხვევაში თუ სასურველია რომ მას ჰქონდეს ცხრილის (მატრიცის) სახე. ამ შემთხვევაში მატრიცის ყოველი სტრიქონის გამოტანის შემდეგ უნდა განხორციელდეს კურსორის ახალ სტრიქონზე გადატანა რათა ციკლის მომდევნო განმეორებისას მატრიცის მომდევნო სტრიქონის ელემენტები გამოტანილი იქნენ ახალ სტრიქონზე.

```
float a[10][10]; int i,j;  //ორგანზომილებიანი მასივის გამოცხადება
. . . . . .
for (i=0; i<10; i++)  //გარე ციკლის სათაური
{
for (j=0; j<10; j++)  //შიგა ციკლის სათაური
cout<<a[i][j];  //მასივის ელემენტების გამოტანა
cout<<endl;  // ახალ ხაზზე გადასვლა
}
```

4.4. ერთგანზომილებიანი მასივების დამუშავების მაგალითები

მაგალითი 1. მოცემულია $C=\{c_1,c_2,\ldots,c_n\}$ $n\le25$ ვექტორი. შევადგინოთ პროგრამა, რომლითაც გამოითვლება მოცემული ვექტორის უდიდესი ელემენტი.

```
#include <iostream.h>
main()
{
 float c[25], max; int n;  // ნამდვილი ტიპის ელემენტებიანი  c  მასივის და max ცვლადის
გამოცხადება,
 cout<<"შეიტანეთ C მასივის ელემენტების რაოდენობა->"; //მითითებული ტექსტის გამოტანა
 cin>>n;  // n ცვლადის მნიშვნელობის შეტანა კლავიატურიდან
 for (int I=0; I<n; I++)  // ციკლის სათაური
 cin>>c[I]; // მასივის i  –ური ელემენტის შეტანა კლავიატურიდან
 max=c[0];  // max ცვლადზე  c  მასივის პირველი ელემენტის მინიჭება
 for (I=1; I<n; I++)  // ციკლის სათაური
 if (c[I]>max) max=c[I]; //  უდიდესი ელემენტის ასარჩევი პირობის შემოწმება
 cout<<endl << " max= "<<max; //  ახალ ხაზზე გადასვლა და max ცვლადის მნიშვნელობის გამოტანა
 return 0 ; //  ფუნქციიდან  გამოსვლა
}
```

მაგალითი 2. მოცემულია $B=\{b_1,b_2,\ldots,b_{10}\}$ ვექტორი. მიიღეთ $C=\{c_1,c_2,\ldots,c_k\}$, $k\le10$ ვექტორი, რომლის კომპონენტები წარმოადგენენ B ვექტორის უარყოფით კომპონენტებს.

```
#include <iostream.h>
main()
{
 float b[10], c[10]; int k=0,n;
 cout<<"შეიტანეთ C მასივის ელემენტების რაოდენობა ->";
 cin>>n;
 cout<<endl<<"შეიტანეთ C მასივის ელემენტები->";
   for (int i=0; i<n; i++)
 cin>>b[i];
 for (i=0; i<n; i++)
 if (b[i] <0) { c[k]=b[i];k++;}
 cout<<endl ;
 for (i=0; i<k; i++)
 cout<<c[i];

 return 0 ;
}
```

// მაგალითი 3.

```
// ბენზინის კილომეტრაჟის გაანგარიშება
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
   const int MAX = 20;          // მნიშვნელობის მაქსიმალური მნიშვნელობა
   double gas[ MAX ];           // ბენზინის მოცულობა გალონებში
```

```
  long miles[ MAX ];           // ოდომეტრის მნიშვნელობა
  int count = 0;               // ციკლის მთვლელი
  char indicator = 'y';                // შეიტანე ინდიკატორი
  while( (indicator == 'y' || indicator == 'Y') && count < MAX )
  {
    cout << endl
         << "Enter gas quantity: ";
    cin >> gas[count];         // წაიკითხე ბენზინის მოცულობა
    cout << "Enter odometer reading: ";
    cin >> miles[count];       // წაიკითხე ოდომეტრის ჩვენება
    ++count;
    cout << "Do you want to enter another(y or n)? ";
    cin >> indicator;
  }
  if(count <= 1)
  {
    cout << endl
         << "Sorry - at least two readings are necessary.";
    return 0;
  }
  // პასუხის გამოტანა
  for(int i=1; i < count; i++)
    cout << endl
         << setw(2) << i << "."
         << "Gas purchased = " << gas[ i] << " gallons"
         << " resulted in "
         << (miles[i] - miles[i-1])/gas[i]
         << " miles per gallon.";
  cout << endl;
  return 0;
}
```

მაგალითი 4.

```
// მასივის ინიციალიზაციის დემონსტრირება
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  int value[5] = { 1, 2, 3 };
  int Junk [5];
  cout << endl;
  for(int i=0; i<5; i++)
    cout << setw(12) << value[i];
  cout << endl;
  for(i=0; i<5; i++)
    cout << setw(12) << Junk[i];
  cout << endl;
  return 0;
}
```

```
// მაგალითი 5.
// სტრიქონიში ასოების რაოდენობის დათვლა
#include <iostream>
using namespace std;
int main()
{
  const int MAX = 80;          // მასივის მაქსიმალური ზომა
  char buffer[MAX];            // ბუფერის შეტანა
  int count = 0;               // ასოების დამთვლელი
  cout << "Enter a string of less than 80 characters:\n";
  cin.getline(buffer, MAX, '\n'); // წაიკითხე სტრიქონი სანამ \n
  while(buffer[count]!= '\0')
    // გაზარდე count იმდენით რამდენი ასოცაა მიმდინარე სტრიქონში
     count++;
  cout << endl
      << "The string \"" << buffer
      << "\" has " << count << " characters.";
  cout << endl;
  return 0;
}
```

4.5 ორგანზომილებიანი მასივების (მატრიცების) დამუშავების მაგალითები

მაგალითი 1. დავუშვათ რომ მოცემულია a[4][5] მთელ რიცხვთა მატრიცა. ვიპოვოთ მოცემული მატრიცაში დადებითი ელემენტების რაოდენობა.

```
#include <iostream.h>
main()
{
int a[4][5], s=0,i,j;
for ( i=0; i<4; i++)
{
cout<<"შეიტანეთ "<<i<<"–სტრიქონის ელემენტები : " ;
for ( j=0 ; j<5 ; j++)
cin>>a[i][j];
cout<<endl;
}
for (i=0; i<4; i++)
for (j=0; j<5; j++)
if (a[i][j]>0) s++;
cout<<"დადებითი ელემენტების რაოდენობაა  "<<s<<endl;
return 0 ;
}
```

მაგალითი 2. მოცემულია a[4][4] მატრიცა. შეუცვალეთ ადგილი მოცემული მატრიცის პირველ და უდიდესი ელემენტის შემცველ სტრიქონებს. დაბეჭდეთ საწყისი და მიღებული მატრიცები. გამოიტანეთ შედეგები ფორმატირებული სახით (ფიქსირებული წერტილით, მეასედის სიზუსტით).

```
#include <iostream.h>
#include <iomanip.h>
```

```
main()
{
float a[4][4], r; int i,j,k=0;

for ( i=0; i<4; i++)
{
cout<<"შეიტანეთ "<<i<<"-სტრიქონის ელემენტები : " ;
for ( j=0 ; j<4 ; j++)
cin>>a[i][j];
cout<<endl;
}
 cout.setf(ios::fixed,ios::floatfield);
cout<<"საწყისი მატრიცა :"<<endl;
   for (i=0; i<4; i++)
       { for (j=0; j<4; j++)
         cout<<setw(10) <<setprecision(2)<<a[i][j];
        cout<<endl; }

   r=a[0][0] ;
   for (i=0; i<4; i++)
   for (j=0; j<4; j++)
   if (a[i][j]>r) { r=a[i][j]; k=i;}

  for (j=0; j<4; j++)
  {   r=a[0][j];
  a[0][j]=a[k][j];
  a[k][j]=r; }

  cout<<endl<<"მიღებული მატრიცა :"<<endl;
   for (i=0; i<4; i++)

    {  for (j=0; j<4; j++)
       cout<<setw(10)<<a[i][j];
      cout<<endl; }
    return 0;
}
```

მაგალითი 3.

```
// სტრიქონის შენახვა მასივში
#include <iostream>
using namespace std;
int main()
{
   char stars[6][80] = { "Robert Redford",
                         "Hopalong Cassidy",
                         "Lassie",
                         "Slim Pickens",
                         "Boris Karloff",
                         "Oliver Hardy"
                       };
   int dice = 0;
   cout << endl
        << " Pick a lucky star!"
```

```
      << " Enter a number between 1 and 6: ";
  cin >> dice;
  if(dice >= 1 && dice <= 6)       // შეამოწმე შენატანის მართებულობა
    cout << endl                   // გამოტანა
       << "Your lucky star is " << stars[dice-1];
  else
    cout << endl                   // გამოიტანე ახალი ხაზიდან
       << "Sorry, you haven't got a lucky star.";  // “არასწორი შენატანი”

  cout << endl;
  return 0;
}
```

თ ა ვ ი 5

ფ უ ნ ქ ც ი ე ბ ი

C++ ფუნქციები იძლევა დიდი პროგრამების მცირე ზომის მოდულებად დაყოფის საშუალებას რომლებიც შეესაბამებიან ცალკეულ ქვეამოცანებს. დიდი პროგრამების ასეთი სახის დაყოფას აქვს მრავალი უპირატესობა, როგორიცაა მაგალითად : პროგრამების წაკითხვის გაადვილება ; შედგენილი პროგრამების გამართვის და მათში ცვლილებების შეტანის გამარტივება ; ადვილდება სამუშაოს დაყოფა რამოდენიმე პროგრამისტს შორის, რაც საბოლოოდ ამცირებს მთლიანი ამოცანის დაპროგრამების და გამართვის დროს, და ა.შ. ფუნქცია გარკვეული ამოცანის გადასაწყვეტად განსაზღვრული აღწერილობების და ოპერატორების ერთობლიობაა.

ნებისმიერი C++ პროგრამა შეიცავს ერთ ფუნქციას მაინც, და ეს არის main() ფუნქცია. ნებისმიერი პროგრამის შესრულებისას, უპირველეს ყოვლისა ხდება main() ფუნქციის გამოძახება და მისი შესრულება. კონკრეტული ამოცანის ამოხსნის პროგრამა გარდა main() ფუნქციისა, შეიძლება შეიცავდეს სხვა ფუნქციებს. ამ შემთხვევაში ამოცანის ამოხსნის მთლიანი პროგრამა შედგება მთავარი პროგრამისაგან (main() ფუნქცია) და სხვა ფუნ-ქციებისაგან. ყოველ ფუნქციას გააჩნია სახელი. ამ სახელით ხდება მისი გამოძახება შესრულებისათვის. ფუნქციის ტანის შესრულების შემდეგ ხორციელდება დაბრუნება main() ფუნქციაში და გრძელდება მისი შესრულება დაწყებული იმ ოპერატორიდან, რომელიც ჩაწერილია ფუნქციის გამოძახების ოპერატორის შემდეგ. ამოცანის ამოხსნის პროგრამაში ფუნქციები შეიძლება იქნენ განთავსებულნი ნებისმიერი თანმიმდევრობით.

5.1 ფუნქციის გამოცხადება და განსაზღვრა

პროგრამაში მომხმარებლის მიერ განსაზღვრული ფუნქციის გამოყენება მოითხოვს, რომ პირველად აუცილებლად უნდა მოხდეს ფუნქციის გამოცხადება და შემდეგ ფუნქციის განსაზღვრა. ფუნქციის გამოცხადება ხდება ფუნქციის პროტოტიპის საშუალებით. ფუნქციის პროტოტიპი კომპილატორს ამცნობს ფუნქციის სახელს, დასაბრუნებელი სიდიდის და ფუნქციის პარამეტრების ტიპებს. პარამეტრები ფუნქციისათვის წარმოადგენენ ე.წ. შესასვლელ სიდიდეებს ანუ არგუმენტებს. ამრიგად ფუნქციის პროტოტიპში გარდა დასაბრუნებელი სიდიდის ტიპისა მითითებული უნდა იქნეს აგრეთვე, ერთმანეთისაგან მძიმით გამოყოფილი, არგუმენტების ტიპებიც (მათი სახელების მითითება არ არის აუცილებელი). იმ შეთხვევაში, თუ ფუნქციისათვის არ არის საჭირო არგუმენტების მითითება, მაშინ ფრჩხილებში უნდა ჩაიწეროს სიტყვა (void), ან ფრჩხილები უნდა დარჩეს ცარიელი. ამ ინფორმაციას კომპილატორი იყენებს მეხსიერებაში ფუნქციისათვის ადგილის "გამოსაყოფად". ფუნქციის აღწერა (განსაზღვრა) კი ამცნობს კომპილატორს თუ რა მოქმედებების შესრულების შედეგად გამოითვლება ფუნქცია.

ფუნქციის პროტოტიპი

ფუნქციის პროტოტიპი მიუთითებს კომპილატორს თუ რა ტიპის მონაცემებს აბრუნებს ფუნქცია, რა რაოდენობის და რა ტიპის პარამეტრებია საჭირო ფუნქციის მუშაობისათვის, და აგრეთვე მითითებული პარამეტრების თანამიმდევრობა. ფუნქციის აღწერის სინტაქსისი არის შემდეგი :

დასაბრუნე- ბელი სიდი- დის ტიპი	ფუნქციის (პარამეტრების გამოცხადება); სახელი

თუ ფუნქციამ არ უნდა დააბრუნოს არავითარი მნიშვნელობა მაშინ დასაბრუნებელ ტიპად უნდა მიეთითოს void.

ქვემოთ მოყვანილია ფუნქციის პროტოტიპის ჩაწერის მაგალითი:

```
int vect_max( int [], int );
```

C++ ში ფართოდ გამოიყენება ბიბლიოთეკური ფუნქციები რომელთა ჩართვაც პროგრამაში სრულდება პრეპროცესორის # include დირექტივით.

ფუნქციის განსაზღვრა

ფუნქციის განსაზღვრა შეიცავს ფუნქციის სათაურს და ტანს. ფუნქციის სათაური აუცილებლად უნდა შეესაბამებოდეს ფუნქციის პროტოტიპს, დასაბრუნებელი სიდიდის ტიპის, პარამეტრების რაოდენობის, მათი ტიპების და აგრეთვე პარამეტრების სიაში მათი თანმიმდევრობის თვალსაზრისით, მაგრამ პროტოტიპისაგან განსხვავებით ფუნქციის განსაზღვრა აუცილებლად უნდა შეიცავდეს პარამეტრების სახელებს და გარდა ამისა იგი არ მთავრდება წერტილმძიმით. ფუნქციის ტანი შეიცავს ფიგურულ ფრჩხილებში მოთავსებულ ოპერატორთა ერთობლიობას. ქვემოთ ნაჩვენებია ფუნქციის სათაური და ტანი.

```
int     Area      (int  length,  int  width)
{
   ოპერატორები
   . . . . . .
       return  length *  width;
}
```

არსებობს ფუნქციიდან main() ფუნქციაში მართვის დაბრუნების 3 ხერხი. თუ ფუნქციამ არ უნდა დააბრუნოს შედეგი, მაშინ მართვა გადაეცემა მთავარ ფუნქციას ან ფიგურული } ფრჩხილის მიღწევისას ან return ; ოპერატორის შესრულებისას. თუკი ფუნქცია აბრუნებს შედეგს, მაშინ

return გამოსახულება; ოპერატორის შესრულებით.

განვიხილოთ მომხმარებლის მიერ განსაზღვრული ფუნქციის შედგენის მაგალითები.

1. შევადგინოთ პროგრამა, რომელიც ფუნქცია square -ს გამოყენებით გამოითვლის 1-დან 10-დე მთელი რიცხვების კვადრატებს.

```
#include<iostream.h>
int square(int);
main()
{
for (int x=1; x<=10; x++)
    cout <<square(x) <<" ";
   cout<<endl;
return 0;
```

სტრიქონი int square(int); წარმოადგენს square ფუნქციის პროტოტიპს. square-ის წინ ჩაწერილი int მიითებს კომპილატორს რომ ფუნქციის შესრულების შედეგად მიიღება მთელი ტიპის რიცხვი. მრგვალ ფრჩხილებში ჩაწერილი ინფორმა-ციიდან ჩანს რომ ფუნქციას აქვს მხოლოდ ერთი, მთელი ტიპის პარამეტრი, რომლითაც მიეთითება კომპილატორს რომ square

```
}                                   ფუნქციის გამოძახებისას მას უნდა
// ფუნქციის აღწერა                 გადაეცეს ერთი მთელი ტიპის მნიშვნელობა.
{                                   ფუნქციაზე მიმართვა ხორციელ დება cout
int square (int y)                  ოპერატორიდან, რომელიც სრულდება ციკლში, და
                                    square ფუნქციას პარამეტრის სახით
return y*y;                         მიმდევრობით გადაეცემა x ცვლადის 1-დან 10-დე
                                    მნიშვნელობები
}
```

5. 2 ლოკალური ცვლადები

ფუნქციის პარამეტრები ფორმალური პარამეტრებია, რომლებიც მოთავსებული არიან ფუნქციის ტანში და ფუნქციის გამოძახების შემდეგ ხდება მათში ფაქტიური პარამეტრების გადაცემა.

ფუნქციის გამოსათვლელად პარამეტრების გარდა ფუნქციის ტანში შეიძლება საჭირო იქნეს აგრეთვე გარკვეული რაოდენობის დამატებითი ცვლადების გამოყენება, რომელთა გამოცხადება უნდა მოხდეს ფუნქციის ტანში, ვინაიდან ისინი მხოლოდ აქ არიან საჭირო. ფუნქციის ყოველი გამოძახებისას ასეთ ცვლადებს მეხსიერებაში გამოეყოფათ უჯრედები, და აღნიშნული ცვლადები ხელმისაწვდომი არიან (“არსებობენ “) მხოლოდ და მხოლოდ ფუნქციის ტანის შესრულების პროცესში. ასეთ ცვლადებს ლოკალურს უწოდებენ. ლოკალური ცვლადი მხოლოდ ფუნქციისათვის არის ცნობილი. ის უცნობია თვით გამომძახებელი ფუნქციისათვისაც კი. შესაძლებელია ორ ფუნქციაში ერთი და იგივე სახელის ცვლადები იყოს გამოყენებული. კომპილატორი მათ როგორც ორ განსხვავებულ ცვლადს ისე განიხილავს.

მთავარი ფუნქციის გამოძახების წერტილში მართვის დაბრუნების შემდეგ, ყველა ლოკალური ცვლადი მიუღწევადი ხდება. განვიხილოთ მაგალითი:

1. შევადგინოთ პროგრამა, რომელიც ჩვენს მიერ განსაზღვრული maximum ფუნქციის გამოყენებით გამოითვლის და დააბრუნებს სამ რიცხვს შორის უდიდესის მნიშვნელობას. ალგორითმი იქნება შემდეგი: უნდა შევიტანოთ 3 რიცხვი, რომლებიც შემდეგ გადაეცემათ ჩვენს მიერ განსაზღვრულ maximum ფუნქციას, რომელიც უდიდესი რიცხვის გამოთვლის შემდეგ აბრუნებს მას main( ) პროგრამაში.

```
 #include<iostream.h>
 int maximum (int, int, int);
main()                              მოყვანილ პროგრამაში maximum
{                                   ფუნქციის მიერ დაბრუნებული
 int a, b, c, r ;                   მნიშვნელობა ენიჭება r ცვლადს,
 cout<<" შეიტანეთ სამი რიცხვი :";  და მერე ხორციელდება ამ
                                    უკანასკნელის cout ოპერატორით
cin >> a>>b>>c ;                    ფუნქციის მიერ დაბრუნებულის
r=maximum(a, b, c) ;                გამოტანა. ფუნქციის აღწერაში
cout<<endl <<" მაქსიმალური          გამოყენებული max ცვლადი წარ-
ricxvia ->"<<r<<endl ;              მადგენს ლოკალურ ცვლადს.
return 0 ;                          იმავე ამოცანისათვის პროგრამის
}                                   შედგენა შეიძლება  r ცვლადის
// maximum funqciis aRwera          გამოყენების გარეშე. ამ შემთხვე-
```

```
int maximum (int x, int y, int z)
{
 int max = x;
<<maximum(a,b,c)<<endl  ;
 if (y>max)  max=y;
 if (z>max)  max=z;
 return max;
}
```

ვაში ფუნქციის გამოძახება განხორციელდება count ოპერტორის საშუალებით, რომელსაც აქვს შემდეგი სახე:

```
cout<<endl <<"უდიდესი რიცხვია->"
```

5.3. გლობალური ცვლადი

ფუნქციის გარეთ განსაზღვრული ცვლადები გლობალური ცვლადებია. მათ აქვთ გლობალური მოქმედების (დანახვის) არე და მათი გამოყენება შესაძლებელია პროგრამის ნებისმიერ ფუნქციაში თვით main() -ის ჩათვლით. შესაძლებელია გლობალურ და ლოკალურ ცვლადებს ერთი და იგივე სახელი ჰქონდეთ, ვინაიდან მეხსიერებაში ამ ცვლადებისათვის გამოყოფილი იქნება სხვადასხვა უჯრედები.
მაგალითი:

```
#include <iostream.h>
void Fun();  //ფუნქციის პროტოტიპი

int x=5, y=7; //გლობალური ცვლადები
int main();
{

  cout<<"x - ის მნიშვნელობა მთავარ ფუნქციაში:"<<x<<"\n";
    cout<<"y - ის მნიშვნელობა მთავარ ფუნქციაში :"<<y<<"\n";

  Fun();   //ფუნქციის გამოძახება
    cout<<"main ფუნქციაში დაბრუნება\n";
     cout<<" x -ის მნიშვნელობა მთავარ ფუნქციაში:"<<x<<"\n";
  cout<<" y -ის მნიშვნელობა მთავარ ფუნქციაში :"<<y<<"\n";
  return 0;
}

 void Fun ()  //ფუნქციის აღწერა
 {
   int y=10;
   cout<< "  ფუნქციაში განსაზღვრული x: "<<x<<"\n";
  cout<< "  ფუნქციაში განსაზღვრული y : "<<y<<"\n";
  }
```

კიდევ ერთხელ უნდა აღინიშნოს, რომ ლოკალური ცვლადის ხედვის არე ანუ მოქმედების არე არის ის ფუნქცია სადაც იგი არის განსაზღვრული; მაგრამ თუ ცვლადის აღწერა მოთავსებულია ბლოკში (ე.ი. ფიგურული ფრჩხილებს შიგნით), მაშინ ცვლადის მოქმედების არეს წარმოადგენს მხოლოდ და მხოლოდ ბლოკი.

```
#include <iostream.h>
void abc();
int main()
{
  int x=5;
  cout<<" x  main  ფუნქციაში ="<<x;
    abc();   //ფუნქციის გამოძახება

  cout<<"\n x  მთავარ ფუნქციაში დაბრუნების შემდეგ:"<<x;
  return 0;
}
 void abc()
 {
   int x=8;
   cout<<"  abc  ფუნქციაში   x  ლოკალური ცვლადია x="<<x<<"\n";

     {
        cout<<" x  ლოკალური ცვლადია ფუნქციის ბლოკში="<<x<<"\n";
        int x=9;
    cout<<"\n x  ბლოკის ლოკალური ცვლადია="<<x<<endl;
      }
  cout<<" x  –ის მნიშვნელობა ბლოკიდან გამოსვლის შემდეგ="
      <<x<<endl;
  }
```

ერთი ფუნქციის შიგნით აკრძალულია სხვა ფუნქციის განსაზღვრა. დასაშვებია რომელიმე ერთი ფუნქციიდან სხვა ფუნქციის გამოძახება.
განვიხილოთ კიდევ ერთი პროგრამა, რომელიც ადგილს უცვლის ორ ცვლადს.

```
#include<iostream.h>
 void swap (int , int );
main()
{
 int x=5, y=10;
 cout<<"დასაწყისში x="<<x<<"  და   y="<<y<<endl;
    swap ( x, y);
  cout<<endl<<"ფუნქციის შესრულების შემდეგ x="<<x<<"    და  y="<<y<<endl;
 return 0;
}

 void swap (int x, int y)
      {
     int temp;   // დამატებითი ცვლადი (ლოკალური)
cout<<"ფუნქციაში გადაცემული ცვლადები"
<<"  x= "<<x<<" da y="<<y<<endl;
     temp=x;
     x=y;
     y=temp;
cout<<"იგივე ცვლადები მნიშვნელობების შეცვლის შემდეგ   x="<<x<<" და y="<<y<<endl;
  }
```

ამ მაგალითში განხილული ფუნქცია არ აბრუნებს არავითარ შედეგს. გასაღები სიტყვა void კომპილატორს ეუბნება, რომ არავითარი სიდიდის დაბრუნება არ მოხდება. ფუნქციის მიერ რაიმე მნიშვნელობის დაბრუნება იმ შემთხვევაშიაც არის შესაძლებელი თუ მისი ტანი მთავრდება ოპერატორით return. ოპერატორის არგუმენტში მითითებული გამოსახულების გამოთვლილი მნიშვნელობა ბრუნდება ფუნქციის გამოძახების წერტილში.

შესაძლებელია მარტივ ფუნქციას დასაბრუნებელი სიდიდის სახით ქონდეს რამოდენიმე ალტერნატივა. ქვემოდმოყვანილ პროგრამაში გამოყენებული Double ფუნქცია, რომელიც აბრუნებს მთავარ პროგრამაში ან მიწოდებულ არგუმენტს, ან მის გაორმაგებულ მნიშვნელობას, ან –1.

```
#include <iostream.h>
int Double (int );
 main()
{
  int result=0, input;
  cout<<"შეიტანეთ რაიმე მთელი რიცხვი  10 ÷ 10000 ფარგლებში:";
    cin>>input;
    cout<<endl<<"საწყისი მნიშვნელობა:"<<input<<"\t გაორმაგებული:"
   <<result<<endl;
    result=Double(input);
    cout<<"დაბრუნება ფუნქციის გამოძახების შემდეგ"<<endl;
    cout<<"საწყისი მნიშვნელობა: "<<input<<" გაორმაგებული: "<<result<<endl;
    return 0;
}

 int Double(int original)
    {
      if (original >10 && original<=10000)
        return original*2;
      else
        return -1;
    }
```

5.4. ფუნქციები პარამეტრების ცარიელი სიით

C++ პარამეტრების ცარიელი სია განისაზღვრება ფრჩხილებში სიტყვა void –ის ჩაწერით ან ცარიელი მრგვალი ფრჩხილებით. მაგალითად ფუნქციის აღწერა void print ( ) ; მიუთითებს იმაზე, რომ print ფუნქცია არც მოითხოვს და არც აბრუნებს არავითარ მნიშვნელობას. მოვიყვანოთ პროგრამა, რომელშიც გამოყენებულია ამ ტიპის ფუნქციის როგორც გამოცხადების ისე აღწერის ორივე ხერხი.

```
#include<iostream.h>
void f1 ( );
void f2 (void);
main ( );
{
 f1( );
f2 ( );
return 0;
}

void f1( );
```

```
{
cout<<" f1 ფუნქცია არ მოითხოვს არგუმენტებს~<<endl ;
}
void f2 (void)
{
cout<< " f2 ფუნქციაც არ მოითხოვს არგუმენტებს "<<endl;
}
```

5.5 ჩადგმული ფუნქციები

ამოცანის ამოხსნის პროგრამის რეალიზება ფუნქციათა ნაკრების სახით კარგია პროგრამული უზრუნველყოფის დამუშავების თვალსაზრისით, მაგრამ ფუნქციის გამოძახება, მისი შესრულებისათვის, მოითხოვს დროს რის გამოც იზრდება პროგრამის შესარულებლად საჭირო დრო. C++ ფუნქციის გამოძახებასთან (განსაკუთრებით მცირე ზომის ფუნქციების შემთხვევაში) დაკავშირებული დროის ასეთი დანახარჯების შემცირების მიზნით C++ გათვალისწინებულია ე.წ. ჩადგმული ფუნქციები (inline), რომლებიც ამ დროითი დანახარჯების შემცირებას ემსახურება. inline სპეციფიკატორი, მოთავსებული დასაბრუნებელი ფუნქციის ტიპის წინ "ურჩევს" კომპილატორს ფუნქციის გამოძახების თავიდან აცილების მიზნით, პროგრამის შესაბამის ადგილას მოახდინოს ფუნქციის კოდის ასლის გენერაცია. შედეგად, პროგრამაში ფუნქციის ყოველი გამოძახების ადგილზე ჩაირთვება ამ ფუნქციის კოდის ასლი. კომპილატორს შეუძლია inline სპეციფიკაციას გაუკეთოს იგნორირება ყველა ფუნქციისათვის, გარდა ძალზედ პატარა ზომის ფუნქციებისა.

მაგალითი :

```
#include<iostream.h>
inline float cube(const float s) {return s*s*s;}
main ()
{
cout<< "შეიტანეთ კუბის გვერდის სიგრძე :" ;
float side ;
cin>>side ;
cout<<side<< "სიგრძის მქონე კუბის მოცულობაა : "  <<cube(side)<<endl ;
return 0;
}
```

მინიშნებები და მინიშნების ტიპის პარამეტრები

პროგრამირების ენების უმრავლესობაში არსებობს ფუნქციის გამოძახების ორი ხერხი – გამოძახება მნიშვნელობის გადაცემით და გამოძახება მინიშნებით. როდესაც არგუმენტი გადაეცემა მნიშვნელობით, ფუნქციის გამოძახებისას იქმნება არგუმენტის ასლი, და ეს უკანასკნელი გადაეცემა გამოსაძახებელ ფუნქციას. ასლის მნიშვნელობის შეცვლა გავლენას არ ახდენს მის ორიგინალზე. ზემოთ განხილულ მაგალითებში არგუმენტების გადაცემა ხორციელდებოდა მნიშვნელობის გადაცემით. ამ ხერხის ნაკლს წარმოადგენს ის გარემოება, რომ დიდი მოცულობის მონაცემების (მაგალითად მასივების) არგუმენტებად გადაცემისას, მათი ასლების შექმნა მოითხოვს დროის დამატებით დანახარჯებს.

ფუნქციის მინიშნებით გამოძახებისას, გამოძახების ოპერატორი აძლევს გამოსაძახებელ ფუნქციას განახორციელოს გადასაცემი მონაცემებისადმი პირდაპირი წვდომა და აგრეთვე საჭიროების შემთხვევაში მათი შეცვლა. მინიშნების ტიპის პარამეტრი წარმოადგენს შესაბამისი არგუმენტის ფსევდონიმს. იმ ფაქტის მითითება, რომ ფუნქციას პარამეტრი გადაეცა მინიშნებით, ხორციელდება ფუნქციის პროტოტიპში პარამეტრის ტიპის შემდეგ ამპერსანდის - & ნიშნის გამოყენებით. ასეთივე აღნიშვნა გამოიყენება პარამეტრების

ჩამონათვალში ფუნქციის სათაურში. მაგალითად, ფუნქციის სათაურში int &a გამოცხადება შეიძლება წაკითხული იქნეს შემდეგნაირად “ a წარმოადგენს int-ზე მინიშნებას”, ან უფრო გასაგებად - int &a წარმოადგენს მეხსიერების იმ უჯრედის მისამართს, რომელშიც იმყოფება a მთელი ტიპის ცვლადის მნიშვნელობა. ფუნქციის გამოძახებისას კი საკმარისია მხოლოდ ცვლადის სახელის მითითება და იგი გადაეცემა მინიშნებით. გამოსაძახებელი ფუნქციის ტანში პარამეტრის სახელის საშუალებით ცვლადის ხსენება, სინამდვილეში წარმოადგენს წვდომას იმ ფუნქციის ცვლადისადმი რომლიდანაც ხდება გამოსაძახებელ ფუნქციაზე მიმართვა და ამ ცვლადის მნიშვნელობის შეცვლა ადგილზევე (მისივე უჯრედში), უშუალოდ გამოსაძახებელი ფუნქციის მიერ. ვინაიდან ფუნქცია არ აბრუნებს შედეგს ძირითად (გამომძახებელ) ფუნქციაში, ამიტომ ფუნქციის შედეგის ტიპად გამოიყენება void ტიპი.

ფუნქციის გამოძახების აღნიშნული მეთოდი გამოიყენება აგრეთვე იმ შემთხვევებში, როდესაც ქვეამოცანის ამოხსნის ალგორითმით გათვალისწინებულია შედეგის სახით რამოდენიმე ცვლადის მნიშვნელობის მიღება, რომლებიც დაბრუნებული უნდა იქნენ ძირითად პროგრამაში მათი შემდგომი გამოყენებისათვის.

თუ ხდება ერთზე მეტი პარამეტრის გადაცემა მინიშნებით, ყოველი პარამეტრის წინ უნდა იქნეს მითითებული, როგორც მისი ტიპი აგრეთვე & ნიშანი.

მაგ. (int &, int &, float &).

მაგალითისათვის მოვიყვანოთ პროგრამა რომელიც დაბეჭდავს y= $(x^2 + lnx)\cdot sinx$ ფუნქციის უდიდესს მნიშვნელობას და მის შესაბამის არგუმენტს, თუ ფუნქციის მნიშვნელობები გამოითვლება რაიმე [a; b] მონაკვეთზე მოცემული h ბიჯით. ფუნქციის მნიშვნელობების გამოთვლის და მათ შორის უდიდესის პოვნის ქვეამოცანის პროგრამა გაფორმებულია ფუნქციის სახით.

ვინაიდან ფუნქციის შესრულების შედეგად მიიღება ორი რიცხვი – ფუნქციის მაქსიმალური მნიშვნელობა და ასევე შესაბამისი არგუმენტის მნიშვნელობა, ამიტომ მათი შესაბამისი პარამეტრების გადაცემა ხორციელდება მინიშნებით.

```
#include<iostream.h>
#include<math.h>
  void max(float, float, float, float &, float &);
main ()
{
float a,b,h, x,r;
cout<<"შეიტანეთ a,b და h  მნიშვნელობები"<<endl;
  cin>>a>>b>>h;
max (a,b,h,x,r);
 cout<<"მაქსიმალური მნიშვნელობაა "<<r<<"  არგუმენტი x= "<<x<<endl;
return 0;
}

void max(float a1,  float b1,  float h1,  float &x1,  float &y1)
{
 float x,y; y1=-1e6, eps=0.0001;
for (x=a1; fabs(x- b1)<=eps; x+=h1)
   {y=(x*x+log(x))*sin(x);
    if (y>y1)  {y1=y; x1=x;}
   }
```

5.6. პარამეტრები შეთანხმებით

საზოგადოდ ფუნქციის გამოძახებისას მას გადაეცემა ყოველი არგუმენტის კონკრეტული მნიშვნელობა. C++ შესაძლებელია იმის მითითება, რომ არგუმენტი წარმოადგენს არგუმენტს შეთანხმებით და მისთვის ამა თუ იმ მნიშვნელობის მიწერა შეთანხმებით, რისი მითითებაც ხდება ფუნქციის პროტოტიპში. თუ ფუნქციის გამოძახებისას არ არის მითითებული არგუმენტი, ამ შემთხვევაში არგუმენტის მნიშვნელობად გადაეცემა ამ არგუმენტის შესაბამისი მნიშვნელობა, რომელიც მითითებულია შეთანხმებით. ფუნქციის პარამეტრების ჩამონათვალში არგუმენტები მინიშნებით განთავსებულები უნდა იქნენ ჩამონათვალის განაპირა მარჯვენა მხარეს. თუ ხდება ფუნქციის გამოძახება, რომლის ორი ან მეტი პარამეტრი წარმოადგენენ არგუმენტებს შეთანხმებით და გამოტოვებული არგუმენტი არ წარმოადგენს განაპირა მარჯვენა პარამეტრს ჩამონათვალში, მაშინ იქნება გამოტოვებული აგრეთვე (კომპილატორი იგნორირებას გაუკეთებს) ყველა ის პარამეტრი, რომელიც ჩაწერილია გამოტოვებული პარამეტრის შემდეგ ფუნქციის პროტოტიპში. ყველა ამ არგუმენტების მნიშვნელობებად გამოიყენებს არგუმენტების მნიშვნელობებს შეთანხმებით.

მოვიყვანოთ პარალელეპიპედის მოცულობის გამოთვლის პროგრამა, რომელშიც გამოყენებულია არგუმენტები შეთანხმებით.

```
// პარალელეპიპედის მოცულობის გამოთვლა
#include <iostream.h>

inline int box(int length=1, int width=1, int height=1)

{ return length*width*height; }

main()
{
cout <<" პარალელეპიპედის მოცულობა შეთანხმებით ტოლია: "
      <<box ( )<<endl<<endl
      <<" პარალელეპიპედის მოცულობა, რომლის სიგრძეა 10, სიგანე1, სიმაღლე 1, ტოლია:"
 << box (10 )<<endl<<endl
        <<" პარალელეპიპედის მოცულობა სიგრძით 10, სიგანით 5, სიმაღლით 1 ტოლია:"
    <<  box (10,5 )<<endl<<endl
       <<" პარალელეპიპედის მოცულობა სიგრძით 10, სიგანით 5, სიმაღლით 2 ტოლია:"
 <<  box (10,5,2 )<<endl<<endl
    << endl;
return 0;
}
```

5.7. ფუნქციის გადატვირთვა

C++ -ში შესაძლებელია ერთი და იგივე სახელით რამოდენიმე ფუნქციის განსაზღვრა თუკი ამ ფუნქციებს აქვთ პარამეტრების განსხვავებული კრებული (განსხვავებული ტიპის პარამეტრები, განსხვავებული რაოდენობა პარამეტრებისა ან ორივე ერთად). ამ თავისებურებას ეწოდება ფუნქციათა გადატვირთვა. გადატვირთული ფუნქციის გამოძახებისას კომპილატორი მათ განასხვავებს მითითებული პარამეტრების მიხედვით (მათი რაოდენობით, ტიპებით და პარამეტრების ჩამონათვალში მათი რიგითობით).

```
მაგ.  int square(int x)
და    double square(double x)
```

გადატვირთული ფუნქციების შემთხვევაში დასაბრუნებელი ტიპი შეიძლება იყოს ერთი და იგივე ან განსხვავებული ტიპისა.

მაგალითისათვის მოვიყვანოთ პროგრამა, რომელშიც გადატვირთული ფუნქცია square გამოიყენება int ტიპის ცვლადის კვადრატის და აგრეთვე double ტიპის ცვლადის კვადრატის გამოსათვლელათ.

```
// გადატვირთული ფუნქციების გამოყენება
#include<iostream.h>

int square (int x) {return x*x;}
double square(double y) {return y*y;}
main ()
{
cout<< « მთელი ტიპის 7 კვადრატია = »<<square(7)<<endl ;
cout<<" double ტიპის 7.5 კვადრატია="<<square(7.5)<<endl;
return 0;
}
```

5.8. მასივის გადაცემა ფუნქციაში

C++ მასივს ფუნქციაში გადასცემს ავტომატურად, იყენებს რა ამ მიზნისათვის მოდელირებულ გამოძახებას მითითებით, რაც აძლევს გამოსაძახებელ ფუნქციას საწყისი (არგუმენტად მითითებული) მასივის ელემენტების შეცვლის საშუალებას. გამოცხადებული მასივის სახელით განისაზღვრება მასივის პირველი ელემენტის მისამართი. რადგან ფუნქციას გადაეცემა მასივის საწყისი მისამართი (მასივის სახელის სახით), ფუნქციამ იცის მასივის ადგილმდებარეობა. ამიტომ, როდესაც გამოძახებული ფუნქცია ცვლის მასივის ელემენტების მნიშვნელობებს ფუნქციის ტანში, იგი ცვლის მასივის რეალურ ელემენტებს მეხსიერების უჯრედებში.

იმისათვის, რომ ფუნქციამ შეძლოს მასივის მიღება, რომელიც მას გადაეცემა ფუნქციის გამოძახებისას, იგი უნდა იქნეს ასახული ფუნქციის პარამეტრების სიაში.

მაგალითად ფუნქცია change-ს პროტოტიპი შეიძლება ყოფილიყო ჩაწერილი შემდეგნაირად : void change (int a[], int n) ; რითიც მიეთითებოდა, რომ change ფუნქცია ელოდება პარამეტრ a-ში მთელ რიცხვთა მასივის მიღებას, და აგრეთვე მასივის განზომილების (მასივში ელემენტების რაოდენობას) პარამეტრ n-ში. ერთგანზომილებიანი მასივის (ვექტორის) ელემენტების რაოდენობის მითითება არ არის საჭირო. ზემოდ მოყვანილი ფუნქციის პროტოტიპის ჩაწერა შეიძლებოდა აგრეთვე შემდეგნაირად : void change (int [], int) ;. ორგანზომილებიანი მასივის (მატრიცის) შემთხვევაში სტრიქონების რაოდენობის მითითება არ არის საჭირო. უბრალოდ ჩაიწერება ცარიელი კვადრატული ფრჩხილები.

მაგალითები ფუნქციების გამოყენებით

1. შევადგინოთ პროგრამა რომელშიც ფუნქციის სახით გაფორმებული იქნება მატრიცაში უდიდესი ელემენტის და აგრეთვე სტრიქონის და სვეტის ნომრების მონახვა, რომელთა გადაკვეთაზეც განლაგებულია უდიდესი ელემენტი. მთავარ პროგრამაში საჭიროა მატრიცის განზომილების სტრიქონების და სვეტების რაოდენობის) და რა თქმა უნდა მისი ელემენტების შეტანა. მატრიცის მაქსიმალური განზომილებაა 20x20. ფუნქციის გამოძახების (მისი შესრულების) შემდეგ ხორციელდება, პროგრამაში ფუნქციიდან დაბრუნებული, მაქსიმალური ელემენტის, სტრიქონის და სვეტის ნომრების ეკრანზე გამოტანა.

```
#include<iostream.h>

void elmax(float [][20],int, int ,float &, int &, int &);
float a[20][20],ar;
```

```
int i,j,k,l,al,ac;
main ()
{
cout<<"შეიტანეთ k და  l  მნიშვნელობები"<<endl;
cin>>k>>l;
for (i=0;i<k;i++)
{
cout<<"შეიტანეთ  "<<i<<"  სტრიქონის ელემენტები :  "<<endl;
for (j=0;j<l;j++)
cin>>a[i][j];
}
elmax(a,k,l,ar,al,ac);
cout<<" მაქსიმალური ელემენტია :"<<ar<<endl
<<" რომელიც იმყოფება - "<<al<<" სვეტის და - "<<ac
<< " სვეტის გადაკვეთაზე" <<endl;
return 0;
}

void elmax(float d[][20], int m,int n,float &r, int &k1, int &l1)
{
  r=-1e6;
for (i=0;i<m;i++)
for (j=0;j<n;j++)
if (d[i][j]>=r) {r=d[i][j]; k1=i; l1=j;}
}
```

2. მოვიყვანოთ მაგალითი, რომელშიც void ტიპის ფუნქციების გამოყენებით ხორციელდება: მატრიცის შეტანა (ფუნქცია vvod), მატრიცის გამოტანა (ფუნქცია vivod), და შეტანილ მატრიცაში უარყოფითი ელემენტების ერთიანებით შეცვლა (ფუნქცია change).

```
#include<iostream.h>
#include<iomanip.h>

int i,j,f,id,jd;
void vivod(int [][9],int, int);
void vvod(int [][9], int,int);
void change(int [][9], int,int);
main()
{
 int a[9][9],m,n;

  cout <<"შეიტანეთ     მატრიცის განზომილება : "<<endl;
 cin>>m>>n;
cout<<"შეიტანეთ       მატრიცის ელემენტები "<<endl;
vvod (a,m,n);
cout<<"საწყისი     მატრიცა:"<<endl;
vivod(a,m,n);
change(a,m,n);
 cout<<"მიღებული       მატრიცა:"<<endl;
 vivod(a,m,n);
return 0;
}
```

```
void vvod(int d[][9],int i1, int j1)
{ for (i=0;i<i1;i++)
  for (j=0;j<j1;j++)
   cin>>d[i][j];
 }
 void vivod (int d[][9],int i1, int j1)
  { for (i=0;i<i1;i++)
  {
  for (j=0;j<j1;j++)
  { cout<<d[i][j]<<"  ";
  }
   cout<<endl;
  }
   cout<<endl;
 }
  void change(int d[][9], int i1, int j1)
  {
    for (i=0;i<i1;i++)
    for (j=0;j<j1;j++)
    {
      if (d[i][j]<0)
          d[i][j]=1;
       }
  }

// Ex4_01.cpp
//  ფუნქციის დეკლარაცია და განსაზღვრა
#include <iostream>
using namespace std;
double power(double x, int n); // ფუნქციის პროტოტიპი


int main(void)
{
   int index = 3;
   double x = 3.0;
   double y = 0.0;
   y = power(5.0, 3);          // მივანიჭოთ კონსტანტა როგორც არგუმენტი


   cout << endl
        << "5.0 cubed = " << y;
   cout << endl
        << "3.0 cubed = "
        << power(3.0, index);  // ცვლადის გამოტანა


     // ფუნქციის გამოყენება არგუმენტად
   x = power(x, power(2.0, 2.0));

   cout << endl
        << "x = " << x;
```

```
  cout << endl;
  return 0;
}


// ფუნქცია power-ის გამოსათვლელად
double power(double x, int n)
{                          // ფუნქციის ტანის დასწყისი...
  double result = 1.0;     // შედეგს result-ში
  for(int i = 1; i<=n; i++)
    result *= x;
  return result;
}                          // ...აქ მთავრდება ფუნქციის ტანი



// Ex4_02.cpp
// ფუჭი ცდა არგუმენტისთვის სახის შესაცვლელად
#include <iostream>
using namespace std;
int incr10(int num);          // ფუნქციის პროტოტიპი


int main(void)
{
  int num = 3;
  cout << endl
       << "incr10(num) = " << incr10(num)
       << endl
       << "num = " << num;
  cout << endl;
  return 0;
}


// ფუნქცია ცვლადის 10-ით გასაზრდელად
int incr10(int num)
{
  num += 10;
  return num;
}


// Ex4_04.cpp
// მასივის ჩასმა ფუნქციაში
#include <iostream>
using namespace std;


// ფუნქციის პროტოტიპი
double average(double array[], int count);


int main(void)
{
```

```
  double values[] = { 1.0, 2.0, 3.0, 4.0,
                      5.0, 6.0, 7.0, 8.0, 9.0, 10.0 };
  cout << endl
       << "Average = "
       << average(values, (sizeof values)/(sizeof values[0]));
  cout << endl;
  return 0;
}

// ფუნქცია საშუალო მნიშვნელობის გამოსათვლელად
double average(double array[], int count)
{
  double sum = 0.0;              // აქ გროვდება ჯამი
  for(int i = 0; i<count; i++)
    sum += array[i];             // იკრიბება მასივის ელემენტები
  return sum/count;              // ბრუნდება საშუალო მნიშვნელობა
}
```

5.9. რ ე კ უ რ ს ი ა

აქამდე განხილული პროგრამები შედგებოდნენ ფუნქციებისაგან, რომლებშიც ხორციელდებოდა ერთი ფუნქციიდან რაიმე სხვა ფუნქციის გამოძახება, და ა.შ. ზოგიერთ შემთხვევებში სასარგებლოა ისეთი ფუნქციების არსებობა, რომლებსაც საკუთარი თავის გამოძახება შეუძლიათ. ასეთ ფუნქციებს რეკურსიული ეწოდებათ. რეკურსიული ფუნქცია – არის ფუნქცია, რომელსაც შეუძლია საკუთარი თავის გამოძახება უშუალოდ ან ირიბად სხვა ფუნქციების მეშვეობით.

ზოგად შემთხვევაში რეკურსიული ამოცანა დაიყოფა რამოდენიმე ეტაპად. ამოცანის ამოხსნისათვის გამოიძახება რეკურსიული ფუნქცია. ამ ფუნქციამ იცის თუ როგორ უნდა იქნეს ამოხსნილი ამოცანის მხოლოდ უმარტივესი ნაწილი, რომელსაც საბაზო ან ძირითად ამოცანას უწოდებენ. თუ აღნიშნული ფუნქციის გამოძახება ხდება საბაზო ამოცანის ამოხსნისათვის ის პირდაპირ აბრუნებს შედეგს.

თუკი ფუნქცია გამოიძახება უფრო რთული ამოცანის ამოხსნისათვის , ის ყოფს ამოცანას 2 ნაწილად: ერთი ნაწილი რომლის ამოხსნა მას შეუძლია (საბაზო ამოცანა) და მეორე ნაწილი რომლის ამოხსნა მას არ შეუძლია. იმისათვის რომ რეკურსია იყოს შესაძლებელი საჭიროა რომ ეს მეორე ნაწილი "გავდეს" საწყის ამოცანას, მაგრამ იყოს უფრო მარტივი ან უფრო მცირე ზომის. ვინაიდან ეს ახალი ამოცანა საწყისი ამოცანის მსგავსია, ამ შემთხვევაში ფუნქცია იძახებს თავისივე ახალ ასლს, გადასცემს მას ახალ პარამეტრებს და იწყებს ამ მეორე ნაწილის ამოხსნას იგივე ალგორითმით. ამ პროცესს ეწოდება რეკურსიული გამოძახება ან რეკურსიის ბიჯი.

რეკურსიის ბიჯის შესრულება გრძელდება მანამ სანამ არ დამთავრდება ფუნქციის გამოთვლა. ვინაიდან ფუნქცია აგრძელებს ყოველი ახალი ქვეამოცანის დაყოფას ორ ნაწილად, რეკურსიის ბიჯის შესრულებისას შეიძლება საჭირო იყოს ამ სახის დიდი რაოდენობის რეკურსიული გამოძახებების შესრულება. რეკურსიის პროცესის დასრულებისათვის, ყოველთვის, როგორც კი ფუნქცია იძახებს თავის თავს საწყისი ამოცანის გამარტივებული ვარიანტის ამოხსნისათვის, უნდა ხდებოდეს ფორმირება სულ უფრო და უფრო მცირე ზომის ამოცანების მიმდევრობისა, რომელიც საბოლოოდ მიიყვანება საბაზო ამოცანამდე. ამ მომენტში, ფუნქცია იცნობს საბაზო ამოცანას, დააბრუნებს ფუნქციის წინა ასლის შედეგს, და დაბრუნებების მიმდევრობა იმეორებს მთელ გზას უკუ მიმართულებით, მანამ სანამ არ მიაღწევს პირველად გამოძახებას და არ დააბრუნებს შედეგს main ფუნქციაში.

რეკურსიული პროცედურის საილუსტრაციოდ განვიხილოთ ფაქტორიალის გამოთვლის რეკურსიული პროგრამა. n ნატურალური რიცხვის ფაქტორიალი n! გამოითვლება ფორმულით n!=n·(n-1)·(n-2)· . . . ·1, ამასთან ითვლება რომ 1!=1, 0!=1.
მოვიყვანოთ თერთმეტი რიცხვის 0,1,2,. . . 10 ფაქტორიალის რეკურსიული სქემით გამოთვლის პროგრამა.

```
#include<iostream.h>
#include<iomanip.h>
unsigned long factorial (unsigned long);
main ()
{
     for (int i=0; i<=10; i++)
          cout<<setw(2)<<i<<"! =  "<<factorial(i)<<endl;
                  return 0;
                }
           unsigned long factorial (unsigned long n)
       {
        if(n<=1)
         return 1;
       else   return n*factorial(n-1);
       }
```

მოყვანილ პროგრამაში რეკურსიული ფუნქცია factorial თავდაპირველად ამოწმებს ჭეშმარიტია თუ არა რეკურსიის დასრულების პირობა n<=1, რომლის ჭეშმარიტობის შემთხვევაში factorial ფუნქცია აბრუნებს შედეგად 1 და ასრულებს მუშაობას. წინააღმდეგ შემთხვევაში ოპერატორი return n*factorial(n-1) ამოცანას წარმოადგენს n-ის და factorial რეკურსიული გამოძახების ნამრავლის სახით, სადაც factorial რეკურსიული გამოძახებით გამოითვლება n-1 ფაქტორიალი, რომელიც წარმოადგენს საწყისი factorial(n) ამოცანის გამარტივებულ ამოცანას.

კიდევ ერთი მაგალითი: ფიბონაჩის რიცხვების 0, 1, 1, 2, 3, 5, 8, 13, 21,.... და ა.შ. მიღების ალგორითმი. როგორც ცნობილია ეს მიმდევრობა იწყება 0 და 1, ხოლო ყოველი მომდევნო რიცხვი არის წინა ორის ჯამი. ზოგადად $a_n=a_{n-2}+a_{n-1}$ სადაც n>2. ალგორითმს აქვს შემდეგი სახე:

განვსაზღვროთ n რომლის მნიშვნელობისათვისაც უნდა დამთავრდეს პროცესი.

გამოვიძახოთ ფუნქცია fibonnaci(n) სადაც n შეტანილი რიცხვია.

ფუნქცია fibonnaci(n) იკვლევს არგუმენტს, თუ n<3 მაშინ აბრუნებს 1, სხვა შემთხვევაში აბრუნებს fibonnaci(n-2)+fibonnaci(n-1) ე.ი აგზავნის საკუთარ თავში არგუმენტად ჯერ n-2 და გადადის მე-2 პუნქტზე და ა.შ.

```
#include <iostream.h>
unsigned long fibonnaci(unsigned long);
main()
{
unsigned long n, result;
cout<<"შეიტანეთ n ის მნიშვნელობა :";
  cin>>n;
  result=fibonnaci(n);
cout<<"ფიბონაჩის რიცხვი ("<<n<<")= "<<result<<endl;
 return 0;
}

// ფუნცია Fibonnaci
```

```
unsigned long fibonnaci(unsigned long n)
  {
   if (n==0 || n==1 )
     return n;
   else
     return fibonnaci(n-1)+fibonnaci(n-2);
  }
```

5.10. მიმთითებლები

მიმთითებელი არის ცვლადი, რომლის მნიშვნელობას წარმოადგენს მეხსიერების მისამართი. მეორეს მხრივ, მიმთითებელი შეიცავს ცვლადის მისამართს, რომელიც შეიცავს გარკვეულ მნიშვნელობას. ამრიგად, ცვლადის სახელი პირდაპირ გვაგზავნის ცვლადის მნიშვნელობასთან, ხოლო მიმთითებელი ირიბად, და ასეთი სახის დამისამართებას ირიბი დამისამართება ეწოდება. ისევე როგორც სხვა ნებისმიერი ტიპის ცვლადი, მიმთითებელიც უნდა იქნეს აღწერილი. ყოველი ცვლადი, რომელიც ცხადდება მიმთითებლად სახელის წინ უნდა შეიცავდეს * სიმბოლოს. მაგალითად int *aptr გამოაცხადებს aptr ცვლადს მიმთითებლად, რომელიც მიუთითებს მთელი ტიპის რიცხვზე. მიღებულია წესად მიმთითებლის ტიპის ცვლადის სახელის დამთავრება ptr ასოებით (რაც წარმოადგენს აბრევიატურას სიტყვა pointer-ის – მიმთითებელი). მიმთითებლის გამოცხადება შეიძლება ნებისმიერი ტიპის ცვლადზე მითითებისათვის. აუცილებელია მიმთითებლის ინიციალიზება მისი გამოცხადებისას, ან მინიჭების ოპერატორის გამოყენებით. მიმთითებელს შეუძლია მიიღოს მნიშვნელობად 0, Null ან კონკრეტული მისამართი.

მოქმედებები მიმთითებლებზე

ერთერთი ოპერაცია, რომელიც სრულდება მიმთითებელზე არის & ანუ დამისამართების ოპერაცია. იგი წარმოადგენს უნარულ ოპერაციას, რომლის შედეგს წარმოადგენს მისი ოპერანდის მისამართი. თუ მაგალითად პროგრამაში გამოცხადებულია

```
int y=5 ;
int *yptr  ;
```

მაშინ ოპერატორით yptr = &y ; y ცვლადის მისამართი მიენიჭება yptr მიმთითებელს.

ოპერაცია *, რომელსაც ირიბი ადრესაციის ოპერაცია ან განსახელების ოპერაცია ეწოდება, შედეგად იძლევა იმ ობიექტის მნიშვნელობას, რომელზეც მიუთითებს მისი ოპერანდი (მიმთითებელი). ასე მაგალითად

```
cout<<*yptr<<endl ;
```

ოპერატორით დაიბეჭდება y ცვლადის მნიშვნელობა.

მითითებული ხერხით * გამოყენებას ეწოდება მიმთითებლის განსახელება.

ფუნქციის გამოძახება მითითებით

C++- ში შესაძლებელია მიმთითებლების და ირიბი დამისამართების ოპერაციის გამოყენება მითითებით ფუნქციის გამოსაძახებლად. არგუმენტების მქონე ფუნქციების გამოძახებისას, როდესაც საჭიროა არგუმენტების შეცვლა ფუნქციის მიერ, გადაეცემათ არგუმენტების მისამართები, რისთვისაც გამოიყენება ცვლადის დამისამართების ოპერაცია &.

ფუნქციისათვის ცვლადის მისამართის გადასაცემად შეიძლება აგრეთვე ირიბი დამისამართების ოპერაციის * გამოყენება.

მოვიყვანოთ მთელი რიცხვის კუბში ახარისხების პროგრამა, რომელშიც ფუნქციას არგუმენტის სახით მისამართის გადასაცემად გამოყენებულია მიმთითებელი.

```
#include<iostream.h>
```

```
void cube(int *);
main ()
{
int number =5;
cout<<"ასახარისხებელი რიცხვია : "<<number<<endl;
cube (&number);
cout<<"მიღებული მნიშვნელობაა :"<<number<<endl;
return 0;
}
void cube(int *n)
{
 *n=*n * *n * *n;
}
```

ფუნქციაში, რომელმაც არგუმენტის სახით უნდა მიიღოს მისამართი, არგუმენტი განსაზღვრული უნდა იყოს მიმთითებელის სახით. ამიტომ არის რომ, cube ფუნქციის სათაურს აქვს სახე : void cube(int *n)

5.11. სიმბოლოების და სტრიქონების დამუშავება

სიმბოლოები წარმოადგენენ C++ პროგრამების ძირითად სტანდარტულ ბლოკებს. ყოველი პროგრამა შედგენილია სიმბოლოთა მიმდევრობებისაგან, რომლებიც კომპიუტერის მიერ აღიქმება როგორც ამოცანის ამოხსნისათვის საჭირო ინსტრუქციათა მიმდევრობა.

პროგრამა შეიძლება შეიცავდეს სიმბოლური ტიპის მუდმივებს. სიმბოლური ტიპის მუდმივა – არის მთელი მნიშვნელობა, წარმოდგენილი აპოსტროფებში ჩაწერილი სიმბოლოს სახით. სიმბოლური მუდმივას მნიშვნელობა – არის რაიმე მთელი მნიშვნელობა მანქანური სიმბოლოების კრებულიდან. ასე მაგალითად 'b' არის b სიმბოლოს მთელი მნიშვნელობა, ხოლო '\n' წარმოადგენს ახალ სტრიქონზე გადასვლის სიმბოლოს მთელ მნიშვნელობას.

სტრიქონი – არის სიმბოლოთა მიმდევრობა, რომელიც მუშავდება როგორც ერთიანი მოდული. სტრიქონი შეიძლება შეიცავდეს ასოებს, ციფრებს და სპეციალურ სიმბოლოებს, როგორიც არის +,–,*,/,< და ა.შ. ლიტერული ანუ სტრიქონის ტიპის მუდმივა ჩაიწერება ბრჭყალებში. მაგ. "380009 Botsvadze street ", "app #35 ".

C++ სტრიქონი წარმოადგენს სიმბოლოთა მასივს, რომელიც ('\0') – ნულოვანი სიმბოლოთი მთავრდება. სტრიქონი ხელმისაწვდომია სტრიქონის პირველ სიმბოლოზე მიმთითებლის მეშვეობით. სტრიქონის მნიშვნელობას წარმოადგენს მისი პირველი სიმბოლოს მისამართი. ამრიგად, შეიძლება ითქვას, რომ სტრიქონი წარმოადგენს მიმთითებელს, კერძოდ – მიმთითებელს მის პირველ სიმბოლოზე. ამ თვალსაზრისით სტრიქონი მასივის მსგავსია, რადგან მასივიც წარმოადგენს მის პირველ ელემენტზე მიმთითებელს.

სტრიქონის გამოცხადება შესაძლებელია როგორც სიმბოლოთა მასივის ან char* ტიპის ცვლადის სახით. მაგალითად თითოეული char color [ ] = "green"; ან char *colorptr ="green"; სტრიქონის ტიპის ცვლადს ანიჭებს მნიშვნელობას green. ამასთან, პირველი გამოცხადებით იქმნება 6 ელემენტიანი color მასივი, ხოლო მეორეთი – მიმთითებელი *colorptr , რომელიც მიუთითებს სადღაც მეხსიერებაში არსებულ "green" სტრიქონზე.

გამოცხადება char color [ ] = {"green"} შეიძლება ჩაწერილი იქნეს შემდეგნაირად: char color [ ] = {'g','r','e','e','n','\0'};

სტრიქონის გამოყენებისას პროგრამაში, გამოცხადებული უნდა იქნეს ამოცანისათვის საჭირო უდიდესი სიგრძის სტრიქონი, ვინაიდან თუ შესატანი სტრიქონი შეიცავს გამოცხადებულ სიგრძეზე უფრო მეტ სიმბოლოს, მაშინ შეტანილი სტრიქონის ბოლო

სიმბოლოები აღმოჩნდება სტრიქონის დასამახსოვრებლად გამოყოფილი მეხსიერების არის შემდეგ მდებარე მეხსიერების ნაწილში, და გააფუჭებს მასში არსებულ ინფორმაციას.

cin ოპერაციის გამოყენებით შეიძლება მასივისათვის სტრიქონის მინიჭება. მაგალითად, თუ გვსურს სიმბოლური ტიპის text[20] მასივისათვის სტრიქონის მინიჭება, ამის გაკეთება შეიძლება შემდეგი ოპერატორით cin>> text; სიმბოლოების შეტანა განხორციელდება მანამ, სანამ არ შეხვდება ხარვეზი, ტაბულაციის სიმბოლო, ახალ სტრიქონზე გადასვლის სიმბოლო ან სიმბოლო "ფაილის ბოლო". მეოცე პოზიცია განკუთვნილი იქნება დამამთავრებელი ნულოვანი სიმბოლოსათვის. მანიპულატორი setw გამოყენება შეიძლება იმის გარანტიისათვის, რომ text-ში შეტანილი სტრიქონი არ აღმოჩნდეს მასივზე დიდი. მაგალითად ოპერატორი cin>>setw(20)>>text; . ზოგიერთ შემთხვევებში, საჭიროა ტექსტის მთლიანი სტრიქონის (მაგალითად რაიმე წინადადების) შეტანა მასივში. ამისათვის გამოიყენება cin.getline ფუნქცია, რომელიც მოითხოვს სამი არგუმენტის მითითებას - მასივის სახელის, სტრიქონის სიგრძის და შემზღუდველი სიმბოლოსი.

მაგალითად პროგრამის ფრაგმენტი

```
char sentence [80];
cin.getline(sentence, 80,'\n');
```

გამოაცხადებს 80 სიმბოლოიან sentence მასივს, და შემდეგ განახორციელებს კლავიატურიდან მასში სიმბოლოების შეტანას. შეტანა გაგრძელდება მანამ, სანამ არ იქნება შეტანილი '\n' სიმბოლო, ან სიმბოლო "ფაილის ბოლო" ან სიმბოლოების შეტანა გაგრძელდება მანამ, სანამ არ იქნება შეტანილი გამოცხადებული მასივის სიგრძეზე ერთით ნაკლები სიმბოლო. სიმბოლო-შემზღუდველი კი უბრალოდ იქნება უკუგდებული. cin.getline ფუნქციის მესამე პარამეტრის '\n' მითითება არ არის აუცილებელი, ვინაიდან იგი მიიღება ავტომატურად. აქედან გამომდინარე, ზემოდ მოყვანილი cin.getline ფუნქციაზე მიმართვა შეგვეძლო ჩაგვეწერა შემდეგნაირად cin.getline(sentence, 80) ; .

სტრიქონებთან მუშაობის ძირითადი ბიბლიოთეკური ფუნქციები

ფუნქციის პროტოტიპი	ფუნქციის აღწერა
char *strncpy(char *s1, const char *s2)	აკოპირებს s2 სტრიქონს s1 მასივში. შედეგად მიიღება s1.
char *strncpy(char *s1, const char *s2, size_t n)	აკოპირებს არა უმეტესი n სიმბოლოსი s2-დან s1-ში
char *strncat(char *s1, const char *s2)	s1 სტრიქონს დაემატება s2 სტრიქონი. მიღება s1.
char *strncat(char *s1, const char *s2, size_t n)	s1 სტრიქონს დაემატება არა უმეტესი n სიმბოლოსი s2 სტრიქონიდან. მიიღება s1 სტრიქონი.
int strcmp(const char *s1, const char *s2)	ადარებს s1 და s2 სტრიქონებს. ფუნქცია აბრუნებს ნულზე ნაკლებ, ნულის ტოლ ან ნულზე მეტ მნიშვნელობას იმის და მიხედვით თუ s1<s2, s1=s2 ან s1>s2
int strcmp(const char *s1,const char *s2,size_t n)	ადარებს s1 სტრიქონის არაუმეტესი n სიმბოლოსი s2 სტრიქონთან. ფუნქცია აბრუნებს ნულზე ნაკლებ,ნულის ტოლ ან ნულზე მეტ მნიშვნელობას იმის და მიხედვით თუ s1<s2, s1=s2 ან s1>s2

size_t strlen(const char *s)	განსაზღვრავს s სტრიქონის სიგრძეს-სიმბოლოების რაოდენობას, ჩაწერილს დამამთავრებელი ნულოვანი სიმბოლოს წინ.

ზოგიერთ ფუნქციებს პარამეტრად აქვთ size_t ტიპის მონაცემები. აღნიშნული ტიპი განსაზღვრულია სასათაურო ფაილ stddef.h-ში, როგორც მთელი უნიშნო ტიპი, ისეთი როგორიცაა unsigned int ან unsigned long.

სტრიქონების დამუშავების მაგალითები

strcpy და strncpy ფუნქციების გამოყენება

```
#include<iostream.h>
#include<string.h>

main()
{
 char x[]="Happy birthday to you";
 char y[35], z[15];

  cout<<"სტრინგი x მასივში :"<<x<<endl;
  cout<<" სტრინგი y მასივში:"<<strcpy(y,x)<<endl;
 strncpy(z,x,14);
 z[14]='\0';
 cout<<" სტრინგი z მასივში:"<<z<<endl;
 return 0;
}
```

მოყვანილ პროგრამაში strcpy ფუნქცია გამოყენებულია x მასივში ჩაწერილი მთლიანი სტრიქონის y მასივში კოპირებისათვის, ხოლო strncpy x მასივში ჩაწერილი სტრინგის პირველი 14 სიმბოლოს y მასივში კოპირებისათვის. ნულოვანი სიმბოლოს \0 დამატება z მასივის ბოლოში გახდა საჭირო იმიტომ, რომ რადგან მესამე არგუმენტის სიგრძე ნაკლებია მეორე არგუმენტის სტრიქონის სიგრძეზე, ნულოვანი სიმბოლო არ ჩაიწერება z მასივში.

strcat და strncat ფუნქციების გამოყენება

```
#include<iostream.h>
#include<string.h>

main()
{

char x[25]= " Happy ";
char y[]="New Year ";
char z[40] ="";
cout << "x= " <<x<<endl<< "y= " <<y<<endl ;
cout << "strcat(x,y) = " <<strcat(x,y)<<endl ;
cout<< "strncat(z,x,6)= " <<strncat (z,x,6)<<endl ;
cout<< "strcat(z,x) = " <<strcat(z,x)<<endl ;
return 0 ;
}
```

ფუნქცია strcat თავის მეორე არგუმენტად მითითებულ სტრიქონს უმატებს პირველ არგუმენტად მითითებულ სტრიქონის შემცველ სიმბოლოების მასივს. მეორე არგუმენტის პირველი სიმბოლოთი იცვლება პირველი არგუმენტის სტრიქონის ბოლო – ნულოვანი სიმბოლო. ფუნქცია strncat უმატებს სიმბოლოების მითითებულ რაოდენობას მეორე სტრიქონიდან პირველში. შედეგს კი ემატება ნულოვანი სიმბოლო.

strcmp და strncmp ფუნქციების გამოყენება

```
#include<iostream.h>
#include<iomanip.h>
#include<string.h>

main ()
{
  char *x ="Happy New Year";
  char *y ="Happy New Year";
  char *z="Happy Holiday";

cout<<" x= " << x<<endl<< " y= "<<y<<endl
      << " z= " <<z<<endl<<endl<< "strcmp(x,y)= "
      <<setw(2)<<strcmp(x,y)<<endl<< "strcmp(x,z) = "
      <<setw(2) << strcmp(x,z)<<endl<<"strcmp(z,x) = "
      <<setw(2) <<strcmp(z,x)<<endl<<endl;

cout<< "strncmp(x,z,8) = " << setw(2)
<< strncmp(x,z,8)<<endl
<< "strncmp(x,z,9) = "<<setw(2);
cout<<strncmp(x,z,9)<<endl
<<"strcnmp(z,x,9) = " <<setw(2)
<< strncmp(z,x,9)<<endl;
return 0;
}
```

5.12. ს ტ რ უ ქ ტ უ რ ე ბ ი

სტრუქტურა არის შედგენილი ტიპის მონაცემი, რომელიც აგებულია სხვა ტიპების გამოყენებით. განვიხილოთ სტრუქტურის შემდეგი განსაზღვრა:

```
struct person {
 char[20] nom;
 char[10] prenom ;
int ne ;
char[30] adress ;
} ;
```

გასაღები სიტყვა struct იწყებს სტრუქტურის განსაზღვრას. იდენტიფიკატორი person-სტრუქტურის აღნიშვნაა (სახელი). სტრუქტურის აღნიშვნა გამოიყენება აღნიშნული ტიპის ცვლადი სტრუქტურების მონაცემების გამოცხადებისას. მოყვანილ მაგალითში ახალი ტიპის სახელია – person. სახელები, გამოცხადებული სტრუქტურის აღწერის ფიგურულ ფრჩხილებში, წარმოადგენენ სტრუქტურის ელემენტებს. სტრუქტურაში შემავალ ელემენტებს უნდა ჰქონდეთ განსხვავებული სახელები. სტრუქტურის განსაზღვრა მთავრდება წერტილმძიმით. სხვა ალგორითმულ ენებში ზემოთაღწერილი ტიპის სტრუქტურებს –

ჩანაწერს უწოდებენ. სტრუქტურის განსაზღვრა არ იკავებს ადგილს მეხსიერებაში ; იგი ქმნის მონაცემების ახალ ტიპს, რომელიც შემდგომში გამოიყენება ცვლადების გამოცხადებისათვის. მაგ. person a,b ;

სტრუქტურის ელემენტების წვდომისათვის გამოიყენება ელემენტების წვდომის ოპერაციები : – ოპერაცია წერტილი (.) და ოპერაცია ისარი (->). ოპერაცია წერტილი მიმართავს სტრუქტურის ელემენტს ცვლადის ობიექტის სახელით ან ობიექტზე მითითებით. მაგალითად person-ის ტიპის a სტრუქტურის ადრესს ელემენტის დასაბეჭდათ ვიყენებთ ოპერატორს cout<<a.adress ;

ოპერაცია ისარი, რომელიც შედგება მინუს ნიშნისაგან და მეტობის ნიშნისაგან, უზრუნველყობს სტრუქტურის ელემენტის წვდომას ობიექტზე მიმთითებლის საშუალებით. დავუშვათ რომ, უკვე გამოცხადებული მიმთითებელი personPtr მიუთითებს person-ის ტიპის ობიექტზე, და რომ a სტრუქტურის მისამართი უკვე არის მინიჭებული personPtr. მაშინ, personPtr მიმთითებლის მქონე a person ტიპის სტრუქტურის ადრესს ელემენტის დასაბეჭდათ ვიყენებთ ოპერატორს cout<<personPtr->adress ;

გამოსახულება personPtr->adress ; ექვივალენტურია (*personPtr).adress, რომელიც ახდენს მიმთითებლის განსახელებას, რის გამოც ხელმისაწვდომი ხდება ადრესს ელემენტი ოპერაცია წერტილის საშუალებით.

თ ა ვ ი 6

ფაილების დამუშავება

მონაცემთა დამახსოვრება ცვლადებში და მასივებში არის დროებითი. ფაილები კი განკუთვნილია დიდი მოცულობის მონაცემების მუდმივად დამახსოვრებისათვის. ფაილების დამახსოვრება ხდება გარე დამამახსოვრებელ მოწყობილობებზე ისეთზე როგორიცაა მაგნიტური დისკოები, ოპტიკური დისკოები და მაგნიტური ფირები.

ფაილი შედგება ჩანაწერებისაგან. ჩანაწერი კი წარმოადგენს რაიმე ობიექტის შესახებ სხვადასხვა ტიპის მონაცემებისაგან შედგენილ მონაცემს. ჩანაწერი შედგება ველებისაგან. ჩანაწერის თითოეული ველი განკუთვნილია ობიექტთან დაკავშირებული ამა თუ იმ მონაცემის დამახსოვრებისათვის. ფაილი შეიძლება შეიცავდეს დიდი რაოდენობის ჩანაწერებს. ფაილში არსებული საჭირო ჩანაწერის მოძებვნის გაადვილებისათვის, ჩანაწერის ერთერთი ველი შეირჩევა ჩანაწერის გასაღებად. გასაღები ახდენს იმის ინიფიცირებას, რომ მოცემული ჩანაწერი მიეკუთვნება კონკრეტულ ობიექტს (მაგ. კონკრეტულ სტუდენტს, თანამშრომელს და ა.შ.). ამიტომ ფაილის ყოველი ჩანაწერის გასაღების მნიშვნელობა აუცილებლად უნდა იყოს უნიკალური.

არსებობს ფაილში ჩანაწერების ორგანიზების სხვადასხვა ხერხი, რომელთაგან ყველაზე გავრცელებულს წარმოადგენს ფაილში ჩანაწერების ორგანიზების ხერხი, რომელსაც მიმდევრობით ფაილს უწოდებენ. ამ ტიპის ფაილებში ჩანაწერები არიან განლაგებულნი იმ თანმიმდევრობით, რომელიც შეესაბამება გასაღების ველს.

ფირმების უმრავლესობა მონაცემთა დამახსოვრებისათვის იყენებს დიდი რაოდენობის სხვადასხვა ფაილებს. დაკავშირებულ ფაილთა ჯგუფს მონაცემთა ბაზას უწოდებენ, ხოლო პროგრამათა ერთობლიობას განკუთვნილს მონაცემთა ბაზის შექმნისათვის და მართვისათვის ეწოდება – მონაცემთა ბაზის მართვის სისტემა.

6.1 ფაილები და ნაკადები

C++ ყოველი ფაილი განიხილება როგორც ბაიტების მიმდევრობითი ნაკადი. ყოველი ფაილი მთავრდება მარკერით " ფაილის ბოლო " (eof – end of file marker). ფაილის გახსნისას იქმნება ობიექტი, რომელთანაც ხდება ნაკადის დაკავშირება. ნაკადები, დაკავშირებული ამ ობიექტებთან, უზრუნველყოფენ კავშირის არხებს პროგრამასა და ცალკეულ ფაილებს ან მოწყობილობებს შორის. მაგალითად, cin ობიექტი აძლევს პროგრამას საშუალებას კლავიატურიდან მონაცემების შეტანისა, ხოლო cout ობიექტი – ეკრანზე მონაცემების გამოტანისა.

ფაილების დამუშავებისათვის C++ პროგრამაში ჩართული უნდა იქნენ შემდეგი საათაურის ფაილები: <iostream.h> და <fstream.h>. ფაილი <fstream.h> შეიცავს ნაკადების კლასების განსაზღვრებებს ifstream (ფაილიდან შეტანისათვის), ofstream (ფაილში ჩასაწერად) და fstream (ფაილების შეტანა-გამოტანისათვის). ფაილები იხსნება ამ კლასის ნაკადების ობიექტების შექმნით.

მიმდევრობითი წვდომის ფაილის შექმნა

ფაილის შექმნისას პირველ რიგში საჭიროა, კონკრეტული ამოცანიდან გამომდინარე, ფაილის სტრუქტურის განსაზღვრა.

დავუშვათ რომ ვქმნით ფაილს, რომლის ჩანაწერი შეიცავს ინფორმაციას სტუდენტების შესახებ, კერძოდ : გვარს, სახელს, ჯგუფის ნომერს და დაბადების წელს. ვინაიდან ჩამოთვლილიდან არცერთი არ გამოდგება ჩანაწერის გასაღებად, დავუმატოდ კიდევ ერთი

რეკვიზიტი – ჩათვლის წიგნაკის ნომერი, რომელიც ნამდვილად უნიკალურია. ჩანაწერის სტრუქტურას ექნება შემდეგი სახე :

```
int order_number;
char lastname[20];
char firstname[10];
int group[3];
char birthday[10];
```

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>

main()
{
  ofstream stud("student.dat", ios::out);

  if (! stud)
  {
  cerr<<"შეუძლებელია ფაილის გახსნა"<<endl;
  exit(1);
   }

cout<<"შეიტანეთ სტუდენტის მონაცემები:"<<endl;
int order_number;
char lastname[20];
char firstname[10];
int group;
char birthday[10];

while (cin>>order_number>>lastname>>firstname>>group>>birthday) {
cout<<"?  ";
stud<<order_number<<' '<<lastname<<' '<<firstname<<' '<<group<<'
'<<birthday<<endl;
}
return 0;
}
```

ზემოთ იყო აღნიშნული რომ ფაილები იხსნება ifstream, ofstream, ან fstream ნაკადების კლასის ობიექტების შექმნით. ვინაიდან მოცემული პროგრამით ფაილი უნდა გაიხსნას მასში ინფორმაციის გამოსატანად, ამიტომ იქმნება ofstream ობიექტი. ობიექტის კონსტრუქტორს გადაეცემა ორი არგუმენტი – ფაილის სახელი ხისტ დისკოზე და ფაილის გახსნის რეჟიმი. ofstream ტიპის ობიექტისათვის ფაილის გახსნის რეჟიმებია – ios::out ფაილში გამოსატანად და ios::app – არსებული ფაილის ბოლოში ჩანაწერების დასამატებლად. თუ არსებული ფაილი იხსნება ios::out რეჟიმით მასში არსებული ჩანაწერები ყოველგვარი გაფრთხილების გარეშე უკუგდებული იქნება, და ცხადია ფაილი მიუწვდომელი ხდება. გამოცხადება ofstream stud("student.dat", ios::out); ქმნის ofstream კლასის stud ობიექტს, რომელიც იხსნება გამოტანისათვის. არგუმენტები "student.dat" და ios::out გადაეცემა ofstream კლასის კონსტრუქტორს, რომელიც ხსნის ფაილს. ამით იქმნება “კავშირის ხაზი” ფაილთან. ofstream კლასის კონსტრუქტორს თუ არ მივაწოდებთ ios::out არგუმენტს იგი ავტომატურად მაინც შექმნის ფაილს განკუთვნილს გამოტანისათვის. ამრიგად ფაილის შექმნის ოპერატორი შეგვეძლო ჩაგვეწერა ofstream stud("student.dat");

იგივეს გაკეთება შეიძლება ofstream კლასის ფუნქცია-ელემენტი open-ის გამოყენებით. ამ შემთხვევაში ჩავწერთ stud.open("student.dat", ios::out);

ofstream კლასის შექმნის შემდეგ მისი გახსნის მცდელობისას პროგრამა ამოწმებს იყო თუ არა ფაილის გახსნის ოპერაცია წარმატებული. პროგრამის ფრაგმენტი

```
if (! stud)
{
cerr<<"ფაილის გახსნა შეუძლებელია"<<endl;
exit(1);
 }
```

იყენებს ios კლასის ელემენტს – გადატვირთულ ფუნქცია-ოპერაციას ოპერატორ !, იმის დასადგენად წარმატებულად გაიხსნა თუ არა ფაილი. თუ ადგილი ჰქონდა ფაილის გახსნის წარუმატებელ მცდელობას, გაიცემა შესაბამისი შეტყობინება და პროგრამიდან გამოსასვლელად გამოძახებული იქნება exit ფუნქცია.

while (cin>>order_number>>lastname>>firstname>>group>>birthday) სტრიქონით შეიტანება მონაცემთა კრებული და მოწმდება ხომ არ არის შეტანილი ნიშანი "ფაილის ბოლო", რომლის შეტანის შემთხვევაში ან არასწორი მონაცემების შეტანისას ოპერატორი while წყვეტს მუშაობას.

ოპერატორი სტუდ<<order_number<<' '<<lastname<<' '<<firstname<<' '<<group <<' ' <<birthday<<endl; უზრუნველყოფს მონაცემთა კრებულის "student.dat" ფაილში ჩაწერას. "student.dat" ფაილი წარმოადგენს ტექსტურ ფაილს და მისი შიგთავსის წაკითხვა შეიძლება ნებისმიერი ტექსტური რედაქტორით. ნიშანი "ფაილის ბოლო" შეტანის შემდეგ stud ობიექტი ნადგურდება მისი დესტრუქტორის გამოძახებით, რომელიც ხურავს student.dat ფაილს. ofstream ობიექტის დახურვა შეიძლება აგრეთვე close ფუნქცია-ელემენტის გამოყენებით stud.close(); .

6. 2. მიმდევრობითი ფაილიდან ჩანაწერების ამოკითხვა

მიმდევრობითი ფაილიდან ჩანაწერების ამოკითხვისათვის (შესატანად) პროგრამამ უნდა ამოიკითხოს არსებული ფაილიდან ("student.dat"), ეკრანზე გამოიტანოს ჩანაწერების შიგთავსი. ამოკითხვისათვის ფაილი იხსნება ifstream კლასის ობიექტის შექმნით რომელსაც გადაეცემა ფაილის სახელი და ios::in ფაილის გახსნის რეჟიმი, რაც განხორციელდება ოპერატორით ifstream stud("student.dat", ios::in); . ifstream კლასის ობიექტი ავტომატურად იხსნება შეტანის რეჟიმისათვის, და ამიტომ მეორე არგუმენტის ios::in მითითება არ არის საჭირო.

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
#include<iomanip.h>
void printline(int ,char*,char*);
main()
{
  ifstream stud("student.dat");

    if (!stud)
  {
  cerr<<"ფაილის გახსნა შეუძლებელია"<<endl;
    exit(1);
   }

int order_number;
char lastname[20];
```

```
char firstname[10];
int group;
char birthday[10];
cout<<setiosflags(ios::left)<<setw(10)<<"Order"<<setw(10)<<"Lastname"<<setw(
8)<<"Firstname"<<endl;
cout<<setiosflags(ios::left)<<setw(10)<<"Number"<<endl;
while (stud>>order_number>>lastname>>firstname>>group>>birthday) {
 printline(order_number,lastname,firstname);}
return 0;
}
void printline(int order, char *lname, char *fname)
{
cout<<setw(11)<<order<<setw(12)<<lname<<setw(15)<<fname<<endl;
}
```

პროგრამა იყენებს !stud პირობას იმის დასადგენად წარმატებით არის გახსნილი თუ არა ფაილი.

ოპერატორით - while(stud>>order_number>>lastname>>firstname>>group>>birthday)

ხორციელდება მიმდინარე ჩანაწერის ამოკითხვა ფაილიდან. ეს პროცესი გაგრძელდება მანამ, სანამ არ მიიღწევა ფაილის ბოლო. ჩანაწერების ეკრანზე გამოტანას უზრუნველყოფს printline ფუნქცია, რომელიც ნაკადის პარამეტრიზებული მანიპულატორების გამოყენებით ახდენს მათ ფორმატირებას. ცხადია რომ, ჩანაწერების ამოკითხვა შეიძლება მოვახდინოთ ჩანაწერის რომელიმე ველის მნიშვნელობის მიხედვით. ამ შემთხვევაში, მიმდინარე ჩანაწერის ამოკითხვის შემდეგ ვადარებთ ჩვენთვის საინტერესო ველის მნიშვნელობას - იმ მნიშვნელობას, რომელიც შეტანილია წინასწარ, რათა მის მიხედვით მოხდეს ჩანაწერის ამორჩევა. შედარების პირობის შესრულების შემთხვევაში მოხდება ამოკითხული ჩანაწერის ეკრანზე გამოტანა. შეიძლება იქნეს საჭირო ასეთი წესით ამოკითხული ინფორმაციის სხვა (ახალ) ფაილში დამახსოვრება, რათა შემდგომში იყოს შესაძლებელი მისი პრინტერზე ამობეჭდვა. ამისათვის უნდა შეიქმნას ახალი ფაილი ofstream ობიექტის სახით და მასში უნდა ჩაიწეროს საჭირო ინფორმაცია.

მიმდევრობით ფაილში რაიმე საჭირო ჩანაწერის მოსაძებნად პროგრამა იწყებს ჩანაწერების ამოკითხვას თავიდან და მიმდევრობით ამოიკითხავს ჩანაწერებს, მანამ სანამ არ შეხვდება საჭირო ჩანაწერს. თუ კი საჭიროა რამოდენიმე ჩანაწერის პოვნა, პროგრამამ შესაბამისად მრავალჯერ უნდა განახორციელოს ჩანაწერების ამოკითხვა ფაილის დასაწყისიდან. ამავე დროს, როგორც ifstream ასევე ofstream კლასები შეიცავენ ფუნქციაელემენტებს ფაილის პოზიციის მიმთითებლის პოზიციონირებისათვის (მომდევნო ჩანაწერის ნომერი, რომელიც უნდა იქნეს ამოკითხული ან ჩაწერილი). ასეთი ფუნქციაელემენტებად არიან seekg (პოზიციონირება ნაკადიდან ამოღებისათვის) ifstream კლასისათვის და seekp (პოზიციონირება ნაკადში განსათავსებლად) ofstream კლასისათვის. ifstream კლასის ნებისმიერ ობიექტს აქვს ე.წ. მიმთითებელი “get”, რომელიც აჩვენებს შესატანი ჩანაწერის ნომერს ფაილში; ხოლო ofstream კლასის ობიექტს – მიმთითებელი “set”, რომელიც აჩვენებს ფაილიდან გამოსატანი მიმდინარე ჩანაწერის ნომერს. ოპერატორი stud.seekg(0); დააყენებს ფაილის პოზიციის მიმთითებელს stud ფაილის დასაწყისში (ნულოვანი პოზიცია). მეორე არგუმენტი რომლის მითითებაც არის შესაძლებელი, მიუთითებს პოზიციონირების მიმართულებას. პოზიციონირების მიმართულება შეიძლება იყოს ios::beg (ხდება მითითების გარეშეც) ფაილის დასაწყისიდან პოზიციონირებისათვის; ios::cur პოზიციონირება მიმთითებლის მიმდინარე პოზიციიდან; ios::end – პოზიციონირება ფაილის ბოლოდან. მოვიყვანოთ მაგალითები:

stud.seekg(n); პოზიციონირება ფაილის n-ურ ჩანაწერზე.

stud.seekg(n,ios::cur); პოზიციონირება მიმთითებლის მიმდინარე პოზიციიდან n ჩანაწერით წინ.

stud.seekg(0, ios::end); პოზიციონირება ფაილის ბოლოში.
stud.seekg(n,ios::end); პოზიციონირება n-ურ ჩანაწერზე ფაილის ბოლოდან.

ანალოგიურად სრულდება პოზიციონირება seekp ofstream კლასის ფუნქცია-ელემენტისათვის.

ფუნქცია-ელემენტები tellg და tellp იძლევიან მიმთითებლის მიმდინარე პოზიციას ფაილში, შესაბამისად ifstream და ofstream კლასის ობიექტებისათვის.
ოპერატორი location = stud.tellg(); ანიჭებს long ტიპის location ცვლადს “get” მიმთითებლის მნიშვნელობას, ხოლო location = stud.tellp(); - “set” მიმთითებლის მნიშვნელობას.

6. 3. პირდაპირი წვდომის ფაილები

პირდაპირი წვდომის ფაილების ცალკეული ჩანაწერი მიიღწევა პირდაპირ (უშუალოდ) და ამრიგად არ არის საჭირო მისი მოძებნა სხვა ჩანაწერებს შორის. პირდაპირი წვდომის ფაილის შექმნა შეიძლება სხვადასხვა მეთოდებით. ფაილის შექმნის ერთერთი მეთოდით მოითხოვება, რომ ყველა ჩანაწერი იყოს ერთი და იგივე სიგრძის. აღნიშნულ ხერხს აქვს მთელი რიგი უპირატესობები, როგორიცაა: შესაძლებელია ჩანაწერის ჩასმა ფაილში ისე რომ არ დაზიანდეს სხვა ჩანაწერები; შესაძლებელია ჩანაწერში ცვლილებების შეტანა ან მისი წაშლა მთლიანი ფაილის ხელახალი გადაწერის გარეშე.

პირდაპირი წვდომის ფაილის შექმნა

ostream კლასის write ფუნქცია-ელემენტით მოცემულ ნაკადში გამოიტანება ბაიტების ფიქსირებული რაოდენობა, დაწყებული მეხსიერების მოცემული ადგილიდან. იმ შემთხვევაში, როდესაც ნაკადი დაკავშირებულია ფაილთან, მონაცემები ჩაიწერება ფაილ-ში, დაწყებული იმ პოზიციიდან, რომელიც მიეთითება « put » ფაილის პოზიციის მიმთითებლის მიერ. istream კლასის read ფუნქცია-ელემენტით მოცემული ნაკადიდან შეიტანება ფიქსირებული რაოდენობის ბაიტები მეხსიერების არეში, დაწყებული მითითე-ბული მისამართიდან. იმ შემთხვევაში, თუ ნაკადი დაკავშირებულია ფაილთან, მონაცემები ამოიკითხება (შეიტანება მეხსიერებაში) ფაილიდან, დაწყებული იმ პოზიციიდან, რომელიც მითითებულია « get » ფაილის პოზიციის მიმთითებლის მიერ.

მთელი ტიპის მაგ. number ცვლადის ფაილში ჩაწერისას, “ფაილის სახელი”<<number-ის ნაცვლად, რომელსაც 4 ბაიტიანი მთელი ცვლადისათვის შეუძლია დაბეჭდოს 1-დან 11-დე სიმბოლოები (თითოეულ სიმბოლოს ესაჭიროება მეხსიერების 1 ბაიტი), შეიძლება შემდეგი ოპერატორის გამოყენება :

« ფაილის სახელი ».write((char *) &number, sizeof(number)) ; , რომელიც ყოველთვის ჩაწერს 4 ბაიტს (4 ბაიტიანი მთელებიან კომპიუტერზე). write ფუნქცია პირველ არგუმენტად მოითხოვს char * ტიპის არგუმენტს, რის გამოც საჭიროა მისი დაყვანა (char *) . ფუნქციის მეორე არგუმენტი წარმოადგენს size_t მთელი ტიპის არგუმენტს, და ამ უკანასკნელით განისაზღვრება ჩასაწერი ბაიტების რაოდენობა. ანალოგიურად ხდება ფაილიდან მთელი ტიპის number ცვლადში 4 ბაიტის ამოკითხვა, რისთვისაც გამოიყენება istream კლასის read ფუნქცია. პირდაპირი შეღწევადობის ფაილის დამუშავების პროგრამა, როგორც წესი ფაილში წერს მთლიან სტრუქტურას (ჩანაწერს) ან კლასს. ძალზედ იშვიათია შემთხვევა, როდესაც საჭიროა ჩანაწერის მხოლოდ ერთი ველის მნიშვნელობის შეტანა.

მაგალითისათვის შეიქმნას პროგრამა რომელიც დაამუშავებს მეანაბრეთა ანგარიშებს, რომელსაც შეეძლება 100 ჩანაწერის დამახსოვრება. ჩანაწერი უნდა შეიცავდეს : ანგარიშის ნომერს (რომელიც იქნება გამოყენებული ჩანაწერის გასაღებად), მეანაბრის გვარს, სახელს და მისი ბალანსისაგან პროგრამამ უნდა შეძლოს ანგარიშის განახლება, ახალი მეანაბრის დამატება, მეანაბრის ამოშლა, და აგრეთვე ჩანაწერების გამოტანა ფორმატირებულ ტექსტურ ფაილში შემდგომში მათი პრინტერზე ამობეჭდვისათვის.

თავდაპირველად საჭიროა იმის სწავლა, თუ როგორ იქმნება კონკრეტული ამოცანისათვის განკუთვნილი ფაილი (ცარიელი). ფაილის შექმნის პროგრამას აქვს შემდეგი სახე :

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>

struct clientData {
 int acctnum;
 char lastName[15];
 char firstName[10];
 float balance;
 };

 main()
{
  ofstream outCredit("credit.dat", ios::out);

  if (! outCredit)
  {
  cerr<<"ფაილის გახსნა შეუძლებელია"<<endl;
    exit(1);
   }

 clientData blankClient = {0, "", "", 0.0};

 for (int i=1; i<=100; i++)
     outCredit.write((char *)&blankClient, sizeof(blankClient));
    return 0;
}
```

პროგრამა write ფუნქციის გამოყენებით ქმნის 100 ცარიელ struct ტიპის ჩანაწერს "credit.dat" ფაილში. ყოველი შეუვსებელი struct ტიპის ჩანაწერი შეიცავს 0 ტოლ ანგარიშის ნომერს, 2 ნულოვან (ცარიელ) სტრიქონს (string) გვარისათვის და სახელისათვის, და აგრეთვე 0.0 ბალანსის მნიშვნელობის სახით.

ოპერატორი outCredit.write((char *)&blankClient, sizeof(blankClient)); იძახებს blankClient სტრუქტურას რომლის ზომა განსაზღვრულია ofstream კლასის sizeof(blankClient) არგუმენტით.

6. 4. ჩანაწერების შეტანა (ჩაწერა) ფაილში

ფაილში ჩანაწერების შეტანის პროგრამას აქვს შემდეგი სახე:

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>

struct clientData {
 int acctNum;
 char lastName[15];
 char firstName[10];
 float balance;
 };
```

```
 main()
{
  ofstream outCredit("credit.dat", ios::ate);

  if (! outCredit)
  {
  cerr<<"ფაილის გახსნა შეუძლებელია"<<endl;
    exit(1);
   }

  cout<<"შეიტანეთ ბარათის ნომერი "
       <<"( 1-დან  100-დე, 0 – შეტანის დასასრული)  "<<endl<<"? ";
 clientData client;
   cin>>client.acctNum;

  while (client.acctNum>0 && client.acctNum<=100) {
  cout<<"შეტანეთ გვარი, სახელი, ბალანსი"<<endl<<"? ";

      cin>>client.lastName>>client.firstName>>client.balance;

   outCredit.seekp ((client.acctNum-1) *sizeof(client));
   outCredit.write((char *)&client, sizeof(client));

    cout<<"Input the account number "<<endl<<"? ";
    cin>>client.acctNum;

    }

    return 0;

    }
```

პროგრამის საშულებით ხდება ჩანაწერების შეტანა credit.dat ფაილში. მონაცემების დასამახსოვრებლად ფაილში ზუსტად განსაზღვრულ ადგილას, იგი იყენებს ostream კლასის seekp და write ფუნქციების კომბინაციას. seekp ფუნქცია აყენებს “put” მიმთითებელს ფაილის მითითებულ პოზიციაში, ხოლო write ფუნქციას გაყავს მონაცემები ფაილში. ოპერატორი outCredit.seekp ((client.acctNum-1) *sizeof(client)); აყენებს ფაილის მიმთითებელს (client.acctNum-1) *sizeof(client) განსაზღვრულ პოზიციაში. უნდა აღინიშნოს, რომ ostream კლასის outCredit ობიექტი იხსნება ფაილის გახსნის რეჟიმში ios::ate. ფაილის პოზიციის მიმთითებელი “put” თავიდან ყენდება ფაილის ბოლოში, მაგრამ მონაცემების ჩაწერა შეიძლება ფაილის ნებისმიერ ადგილას.

6. 5. ფაილში არსებული ჩანაწერების მიმდევრობითი ამოკითხვა

ხშირად საჭირო ხდება ფაილში არსებული (შევსებული) ჩანაწერების მიმდევრობითი ამოკითხვა. პროგრამამ მიმდევრობიდ უნდა ამოიკითხოს ყველა ჩანაწერი, დაწყებული თავიდან, და მასში მონაცემების არსებობის შემთხვევაში გამოიტანოს მისი შიგთავსი ეკრანზე. მოვიყვანოთ პროგრამა

```
#include<iostream.h>
#include<iomanip.h>
#include<fstream.h>
```

```
#include<stdlib.h>

struct clientData {
 int acctNum;
 char lastName[15];
 char firstName[10];
 float balance;
 };

 void outputLine(ostream &, clientData);
  main()
{
  ifstream inCredit("credit.dat", ios::in);

  if (! inCredit)
  {
  cerr<<"ფაილის გახსნა შეუძლებელია"<<endl;
    exit(1);
   }

   cout<<setiosflags(ios::left) <<setw(9)<<"Account"
        <<setw(15)<<"Lastame"<<setw(12)
        <<"Firstname"<<setiosflags(ios::right)
        <<setw(10)<<"Balance"<<endl;

 clientData client;

 inCredit.read((char *)&client, sizeof(clientData));

  while(! inCredit.eof()) {
   if (client.acctNum !=0)
   outputLine(cout, client);

   inCredit.read((char *)&client, sizeof(clientData));
   }
   return 0;
   }

   void outputLine(ostream &output, clientData c)
   {
   output<<setiosflags(ios::left) <<setw(9)<<c.acctNum
         <<setw(15)<<c.lastName<<setw(12)<<c.firstName
         <<setw(10)<<setprecision (2)
         <<setiosflags(ios::showpoint | ios::right)
         <<c.balance<<endl;
    }
```

istream კლასის read ფუნქციას შეყავს ობიექტში მითითებული ნაკადის მიმდინარე პოზიციიდან ბაიტების გარკვეული რაოდენობა. მაგალითად, ოპერატორი inCredit.read((char *)&client, sizeof(clientData)); ამოიკითხავს sizeof(clientData)-ს მიერ განსაზღვრულ ბაიტების რაოდენობას ფაილიდან, რომელიც დაკავშირებულია ifstream კლასის inCredit ობიექტთან და განათავსებს მათ client სტრუქტურაში. ფაილიდან სტრუქ-

ტურაში შეტანილი მონაცემები (მასში ინფორმაციის არსებობის შემთხვევაში) გამოიტანება ეკრანზე ფუნქცია outputLine –ს მეშვეობით.

6.6. მოთხოვნების დამუშავების პროგრამა

ქვემოთ მოყვანილი პროგრამა ახორციელებს: არსებული ანგარიშების განახლებას, ფაილში ახალი ჩანაწერების დამატებას, ჩანაწერების წაშლას და აგრეთვე ქმნის ტექსტურ ფაილს ყველა მიმდინარე ანგარიშების დასაბეჭდათ.

```
#include<iostream.h>
#include<iomanip.h>
#include<fstream.h>
#include<stdlib.h>

struct clientData {
 int acctNum;
 char lastName[15];
 char firstName[10];
 float balance;
 };
  int enterChoice(void);
   void textFile(fstream&);
  void updateRecord(fstream&);
  void newRecord(fstream&);
  void deleteRecord(fstream&);
  void outputLine(ostream &, clientData);

  main()
{

   fstream inOutCredit("credit.dat", ios::in |ios::out);
   if (! inOutCredit)
   {
   cerr<<"File can't be open"<<endl;
   exit(1);
    }
      int choice;

     while ((choice = enterChoice () ) !=5) {

     switch (choice) {
    case 1:
    textFile(inOutCredit);
    break;
    case 2:
    updateRecord(inOutCredit);
    break;
    case 3:
    newRecord(inOutCredit);
    break;
    case 4:
   deleteRecord(inOutCredit);
```

```
 break;
default:
cerr << "incorrect choice" <<endl;
break;
  }

  inOutCredit.clear ();
  }

  return 0;
  }

// Invitation de choisir la division de menu
  int enterChoice(void)

  {
  cout<<endl<<"Choisissez:" <<endl
      <<"1 - Creation du fichier formate des accounts"<<endl
      <<"    dont le nom est \"print.txt\ * pour impression"<<endl
      <<"2 - changement d'account"<<endl
      <<"3 - ajouter un account neaf"<<endl
      <<"4 - efaccer un account"<<endl
       <<"5 - fin du travaille"<<endl<<"? ";

 int menuChoice;
 cin>>menuChoice;
 return menuChoice;
 }


 // Création  du fichier du texte
 void textFile(fstream &readFromFile)
 {
 ofstream outPrintFile("print.txt", ios::out);
 if (!outPrintFile) {
 cerr <<"Fichier ne peut pas être ouvert."<<endl;
 exit (1);
 }

 outPrintFile<<setiosflags(ios::left) <<setw(9)<<"Account"
             <<setw(14)<<"Lastame"<<setw(12)
             <<"Firstname"<<setiosflags(ios::right)
             <<setw(10)<<"Balance"<<endl;

  readFromFile.seekg(0);
  clientData client;
  readFromFile.read((char *)&client, sizeof(client));

  while (!readFromFile.eof()) {
  if (client.acctNum !=0)
  outputLine(outPrintFile, client);
  readFromFile.read((char *)&client, sizeof(client));
   }
```

```
    }
//Changément de la balance du credit

   void updateRecord(fstream &updateFile)
   {
   int account;
   do {
   cout<<"Enter le numero de l'account a changer (1-100): ";
   cin>>account;
   } while (account <1 || account>100);
    clientData client;
   updateFile.seekg((account - 1) *sizeof(client));


   updateFile.read((char *)&client, sizeof(client));

   if (client.acctNum !=0) {
    outputLine(cout,client);
    cout<<endl<<"Enter depence (+) or supplement (-) : ";

    float transaction;
    cin>>transaction;
    outputLine(cout, client);
    updateFile.seekp((account -1) * sizeof(client));
    updateFile.write((char *)&client, sizeof(client));
    }
    else
    cerr<<"Account # "<<account<<"n'est pas rempli."<<endl;
     }


   //Ajouter un unregistrement neuf
   void newRecord(fstream &insertInFile)
   {
    cout<<"Enter le numero de l'account neuf (1-100) : ";
    int account;
    cin>>account;
    clientData client;
    insertInFile.seekg((account -1)+sizeof(client));
    insertInFile.read((char *)&client, sizeof(client));

    if (client.acctNum ==0) {
   cout <<"Enter nom, prenom, balance"<<endl <<"? ";
   cin>>client.lastName>>client.firstName>>client.balance;
   client.acctNum = account;
   insertInFile.seekp((account - 1) * sizeof(clientData));
   insertInFile.write((char *)&client, sizeof(clientData));
   }
   else
   cerr<<"Account # "<<account<<" deja contain l information."<<endl;
          }
   //Impréssion de l'enregistrement sur la screen
  void outputLine(ostream &output, clientData c)
```

```
   {
   output<<setiosflags(ios::left) <<setw(9)<<c.acctNum
    <<setw(15)<<c.lastName<<setw(12)<<c.firstName
    <<setw(10)<<setprecision (2)
    <<setiosflags(ios::showpoint | ios::right)
    <<c.balance<<endl;
     }


  //Enlèvement de l'enregistrement du fichier
  void deleteRecord(fstream &deleteFromFile)
  {
   cout<<"Enter le numero de l'account a effacer (1 - 100) : ";
   int account;
   cin>>account;
     clientData client;
   deleteFromFile.seekg((account -1) * sizeof(client));

    deleteFromFile.read((char *)&client, sizeof(client));
    if (client.acctNum !=0) {
    clientData blankClient = {0, "", "", 0.0};

     deleteFromFile.seekp((account -1) * sizeof(client));
     deleteFromFile.write((char *)&blankClient, sizeof(client));
     cout<<"Account # "<<account<<" est efface."<<endl;
     }
     else
     cout<<"Account # "<<account<<" est wide."<<endl;
  }
```

6.7. კლასები და სტრუქტურები

მონაცემების შედგენილი ტიპებიდან ზემოთ განხილული იყო მასივები, რომლებიც იძლევიან მხოლოდ ერთი და იგივე ტიპის ცვლადების მნიშვნელობების დაჯგუფების საშუალებას. სხვადასხვა ტიპების მონაცემების დასაჯგუფებლად C++ გამოიყენება სტრუქტურები და კლასები, რომლებიც წარმოადგენენ ობიექტზე ორიენტირებული პროგრამირების ძირითად პროგრამულ ერთეულს. C++ მუშაობისას პროგრამისტები ძირითად ყურადღებას უთმობენ მათ მიერ განსაზღვრული ტიპების შექმნას, რომლებსაც კლასები ეწოდება. ყოველი კლასი შეიცავს მონაცემებს და ფუნქციების გარკვეულ ერთობლიობას, რომლებიც ამუშავებენ ამ მონაცემებს. კლასის კომპონენტ-მონაცემებს ელემენტ-მონაცემებს უწოდებენ. კომპონენტ-ფუნქციებს კი ფუნქცია-ელემენტს უწოდებენ. საზოგადოდ მომხმარებლის მიერ განსაზღვრულ ტიპს (ე.ი.კლასს) ობიექტს უწოდებენ. C++ კლასების დამუშავების სპეციფიკის განხილვამდე, ავაგოთ მომხმარებლის მიერ განსაზღვრული ტიპი, რომელიც ეფუძნება სტრუქტურას.

სტრუქტურა წარმოადგენს შედგენილი ტიპის მონაცემს, რომელიც აგებულია მონაცემების სხვადასხვა ტიპების გამოყენებით. განვიხილოთ შემდეგი სტრუქტურის განსაზღვრა :

```
struct time {
  int hour ;
  int minute ;
  int second;
} ;
```

სტრუქტურის განსაზღვრა იწყება სიტყვა struct რომელიც წარმოადგენს გასაღებ სიტყვას. იდენტიფიკატორი time წარმოადგენს სტრუქტურის სახელს, მონაცემის ტიპის სახელს, რომელიც გამოიყენება შემდეგ ამ ტიპის ცვლადების გამოცხადებისათვის. სტრუქტურის აღწერისას გამოცხადებული ცვლადები წარმოადგენენ სტრუქტურის ელემენტებს (წევრებს) ან ველებს. სტრუქტურის აღწერა მთავრდება } სიმბოლოთი. სტრუქტურის ელემენტები შეიძლება იყოს სხვადასხვა ტიპისა. რაიმე სტრუქტურის ელემენტი არ შეიძლება იყოს იმავე სტრუქტურის ტიპისა. ასე მაგალითად time ტიპის ელემენტი არ შეიძლება წარმოადგენდეს time სტრუქტურის ელემენტს. დასაშვებია, რომ სტრუქტურის ელემენტი წარმოადგენდეს მიმთითებელს იგივე სახელის მქონე სტრუქტურაზე. ამ ტიპის სტრუქტურებს ეწოდებათ – სტრუქტურები თვითდამისამართებით. სტრუქტურის განსაზღვრისას არ ხდება მეხსიერების რაიმე არის რეზერვირება, მხოლოდ და მხოლოდ იქმნება მონაცემის ახალი ტიპი. სტრუქტურის ტიპის მონაცემების გამოცხადება ხდება ისევე როგორც სტანდარტული ტიპის ცვლადების გამოცხადება. შემდეგი აღწერით:

time timeobject, timearray[15], *timerptr;

გამოცხადებულია time-ის ტიპის timeobject ცვლადი, timearray - მასივი რომელიც შეიცავს time-ის ტიპის 15 ელემენტს, და timeptr - მიმთითებელს time-ის ტიპის ობიექტზე.

სტრუქტურის ელემენტებისადმი შეღწევა (სტრუქტურის ველების ადრესაცია)

სტრუქტურის ან კლასის ელემენტებთან შეღწევისათვის გამოიყენება – ოპერაცია წერტილი (.) და ოპერაცია ისარი (->). ოპერაცია წერტილი მიმართავს სტრუქტურის ან კლასის ელემენტს ობიექტის ცვლადის სახელით ან ობიექტზე მინიშნებით. მაგალითად worker სტრუქტურის hour ელემენტის ეკრანზე გამოსატანად გამოიყენება ოპერატორი

cout<<timeobject.hour ;

ოპერაცია ისარი, უზრუნველყოფს სტრუქტურის ან კლასის ელემენტისადმი შეღწევას ობიექტზე მიმთითებლის საშუალებით. დავუშვათ, რომ გამოცხადებული იყო მიმთითებელი timeptr რომელიც მიუთითებდა time ტიპის ობიექტზე, და რომ time სტრუქტურის მისამართი უკვე იყო მინიჭებული timeptr. მაშინ, timeptr მიმთითებლის მქონე time ობიექტის hour ელემენტის ეკრანზე გამოსატანად, შეიძლება შემდეგი ოპერატორის გამოყენება

cout<<timeptr->hour;

გამოსახულება cout<<timeptr->hour; ექვივალენტურია (*timeptr).hour გამოსახულებისა, რომელიც ახდენს მიმთითებლის განსახელებას რის შედეგადაც ოპერაცია წერტილის მეშვეობით ხდება ხელმისაწვდომი hour ელემენტი.

ქვემოდ მოყვანილია პროგრამა, რომლითაც იქმნება მომხმარებლის მიერ განსაზღვრული time სტრუქტურის ტიპი, რომელიც შეიცავს მთელი ტიპის სამ ელემენტს: hour, minute, და second. პროგრამაში განსაზღვრულია time ტიპის ერთადერთი სტრუქტურა სახელით dinnertime და გამოყენებულია ოპერაცია წერტილი სტრუქტურის ელემენტებისათვის საწყისი მნიშვნელობების მისანიჭებლად. შემდეგ ხორციელდება დროის დაბეჭდვა ე.წ სამხედრო (24 საათიან) და ე.წ. სტანდარტულ (12 საათიან) ფორმატებში. შევნიშნოთ, რომ ბეჭდვის ფუნქციები ღებულობენ მინიშნებებს time ტიპის მუდმივ სტრუქტურებზე.

```
//სტრუქტურის შექმნა, მისი ელემენტების განსაზღვრა და ბეჭდვა
#include<iostream.h>
struct time {
  int hour ;
  int minute ;
  int second;
} ;
```

```
void printmilitary (const time &);
void printstandard (const time &);

main ()
{
time dinnertime;
// ელემენტებისათვის სწორი მნიშვნელობების მიწოდება
dinnertime.hour=18;
dinnertime.minute=30;
dinnertime.second=0;

cout<<"სამხედრო დროით სადილი შედგება ";
printmilitary(dinnertime);
cout<<endl<<"რაც შეესაბამება სტანდარტული დროით ";
printstandard(dinnertime);

// ელემენტებისათვის არასწორი მნიშვნელობების მიწოდება
dinnertime.hour=35;
dinnertime.minute=83;
dinnertime.second=97;

cout<<endl<<"დრო არა სწორად შეტანილი მნიშვნელობებით: ";
printmilitary(dinnertime);
cout<<endl;
return 0;
}

//დროის ბეჭდვა ე.წ. სამხედრო ფორმატით

void printmilitary (const time &t)
{
  cout<<(t.hour<10 ? "0":" ")<<t.hour
        <<":"<< (t.minute<10 ? "0" :" ") <<t.minute
        <<":"<<(t.second<10 ? "0": " ") <<t.second;
}


//დროის ბეჭდვა ე.წ. სტანდარტული ფორმატით
void printstandard (const time &t)
{
cout<<(t.hour==0 || t.hour == 12) ? 12 : t.hour % 12
        <<":"<< (t.minute<10 ? "0" :" ") <<t.minute
        <<":"<<(t.second<10 ? "0": " ") <<t.second
         << (t.hour <12 ? "AM" : " PM ");
}
```

6. 8. კლასის მეშვეობით **time** აბსტრაქტული ტიპის მონაცემების გამოყენება

კლასები აძლევენ პროგრამისტს ისეთი ობიექტების მოდელირების საშუალებას, რომლებსაც გააჩნიათ ატრიბუტები (წარმოდგენილი მონაცემი – ელემენტების სახით) და ქცევის ვარიანტები ანუ ოპერაციები (წარმოდგენილი ფუნქცია-ელემენტების სახით). ტიპები, რომლებიც შეიცავენ მონაცემ-ელემენტებს და ფუნქცია-ელემენტებს, საზოგადოდ

განისაზღვრებიან გასაღები სიტყვა class-ის საშუალებით. კლასის განსაზღვრის შემდეგ, მისი სახელი შეიძლება გამოყენებული იქნეს ამ კლასის ობიექტების გამოცხადებისათვის.

მაგალითისათვის გავნსაზღვროთ time კლასი, რომელიც ზემოთ მოყვანილი სტრუქტურის მსგავსად შეიცავს მთელი ტიპის სამ ელემენტს:

hour, minute და second. time კლასის განსაზღვრა იწყება class გასაღები სიტყვით. კლასის განსაზღვრის ტანი მოთავსებულია ფიგურულ ფრჩხილებში { } და მთავრდება წერტილმძიმით.

```
class time {
public:
  time ( );
  void settime(int, int, int);
  void printmilitary ( );
  void printstandard ( );
private:
   int hour;
   int minute;
   int second;
};
```

public: (გახსნილი) და private: (ჩაკეტილი) ჭდეები წარმოადგენენ ელემენტებზე წვდომის სპეციფიკატორებს. ნებისმიერი მონაცემი-ელემენტები და ფუნქცია-ელემენტები გამოცხადებული სპეციფიკატორი public: -ის შემდეგ (განსაზღვრაში მომდევნო წვდომის სპეციფიკატორამდე), მისაწვდომია time ობიექტზე პროგრამის ნებისმიერი მიმართვისას.

ნებისმიერი მონაცემი-ელემენტები და ფუნქცია-ელემენტები გამოცხადებული სპეციფიკატორი private: -ის შემდეგ (განსაზღვრაში მომდევნო წვდომის სპეციფიკატორამდე), მისაწვდომია მხოლოდ და მხოლოდ ამ კლასის ფუნქციებისათვის. ზემოთ მოყვანილი time კლასის განსაზღვრა public სპეციფიკატორის შემდეგ შეიცავს ოთხ ფუნქცია-ელემენტს. ესენი არიან: time, settime, printmilitary და printstandard რომლებიც წარმოადგენენ გახსნილ ფუნქცია-ელემენტებს ან სხვანაირად კლასის მომსახურების გახსნილ ინტერფეისს. ეს ფუნქციები გამოყენებული იქნება კლასის კლიენტების მიერ (ე.ი პროგრამის იმ ნაწილების მიერ, რომლებიც მომხმარებლის როლს ასრულებენ) ამ კლასის მონაცემების დამუშავებისათვის.

კლასის განსაზღვრაში ერთერთ ფუნქცია-ელემენტს სახელად აქვს თვით კლასის სახელი. ამ ფუნქციას კლასის კონსტრუქტორი ეწოდება და წარმოადგენს სპეციალურ ფუნქცია-ელემენტს რომელიც გამოიყენება ამ კლასის ობიექტის მონაცემი-ელემენტების ინიციალიზაციისათვის. კლასის კონსტრუქტორზე მიმართვა ხდება ავტომატურად ამ კლასის ობიექტის შექმნისას.

სპეციფიკატორი private-ს შემდეგ გამოცხადებულია სამი მონაცემი-ელემენტი, რაც მიუთითებს იმაზე, რომ ისინი ხელმისაწვდომია მხოლოდ ოთხი ფუნქციისათვის (ან ამ კლასის მეგობრებისათვის) რომელთა პროტოტიპები მოყვანილია ამ კლასის განსაზღვრაში. კლასის განსაზღვრის შემდეგ, იგი შეიძლება გამოყენებული იქნეს ახალი ტიპის სახელად სხვადასხვა სიდიდეების გამოცხადებისას.

მოვიყვანოთ პროგრამა რომელშიც გამოყენებულია time კლასი. ამ პროგრამით იქმნება time ტიპის ერთადერთი t ობიექტი. ობიექტის შექმნისას, ავტომატურად ხდება time კონსტრუქტორის გამოძახება, რომელიც ნულის ტოლ მნიშვნელობებს ანიჭებს ჩაკეტილ private ნაწილში არსებულ ყველა მონაცემ-ელემენტს. შემდეგ ამ საწყისი მონაცემებისათვის სამხედრო და სტანდარტულ ფორმატებში იბეჭდება დრო, იმის დასადასტურებლად, რომ კონსტუქტორმა ნამდვილად მიანიჭა 0 - ის ტოლი მნიშვნელობები. პროგრამის მომდევნო ნაწილში ფუნქცია settime-ს საშუალებით ყენდება დრო (დასაშვები დრო), რომელიც ისევ იბეჭდება სამხედრო და სტანდარტულ ფორმატებში. ამის შემდეგ ფუნქცია settime-ს

საშუალებით კეთდება მცდელობა დროის არასწორი მნიშვნელობების დაყენებისა, რაც მოწმდება ფუნქცია settime-ს მიერ, რომელიც ასეთ შემთხვევებში აყენებს დროის 0 –ის ტოლ მნიშვნელობებს, რის შემდეგ ისევ ხორციელდება ბეჭდვა ორივე ფორმატში.

```
//კლასი time

#include<iostream.h>
//აბსტრაქტული მონაცემთა ტიპის განსაზღვრა
class time {
public:
void settime(int,int,int);
void printmilitary();
void printstandard();

private:
int hour;
int minute;
int second;
};

//time კონსტრუქტორის მიერ მონაცემ-ელემენტებისათვის საწყისი
//მნიშვნელობების მინიჭება
time::time()  {hour=minute=second=0;}

//სამხედრო დროის (24 საათიანი) სახით დროის ახალი შესატანი //მნიშვნელობების შემოწმება
სისწორეზე და მათი მინიჭება  time-სათვის
void time::settime (int h, int m, int s)
{
hour = (h>=0 && h<24) ? h :0;
minute = (m>=0 && m<60) ? m:0;
second = (s>=0 && s<60) ?s:0;
}

//დროს გამოტანა სამხედრო (24-ანი) ფორმატით
void printmilitary (const time &t)
{
  cout<<(t.hour<10 ? ”0”:” ”)<<t.hour
        <<”:”<< (t.minute<10 ? “0” :” “) <<t.minute
        <<”:”<<(t.second<10 ? “0”: “ “) <<t.second;
}

//დროის ბეჭდვა ე.წ. სტანდარტული (12 საათიანი) ფორმატით
void printstandard (const time &t)
{
cout<<(t.hour==0 || t.hour == 12) ? 12 : t.hour % 12
        <<”:”<< (t.minute<10 ? “0” :” “) <<t.minute
        <<”:”<<(t.second<10 ? “0”: “ “) <<t.second
         << (t.hour <12 ? “AM” : “ PM “);
}
// მთავარი ფუნქცია
main()
{
time t;
```

```
cout<<" სამხედრო დროის საწყისი მნიშვნელობა : ";
t.printmilitary();
cout<<endl<< "სტანდარტული დროის საწყისი მნიშვნელობა : " ;
t.printstandard() ;
t.settime(19,42,26) ;
cout<<endl<<endl<< "სამხედო დრო settime-ს შემდეგ : ";
t.printmilitary();
cout<<endl<<endl<< "სტანდარტული დრო settime-ს შემდეგ : ";
t.printstandard ();
t.settime(77,77,77) ;
cout<<endl<<endl<<"არასწორი დროის დაყენების მცდელობის შემდეგ :
    <<endl<< "სამხედრო დრო : ";
t.printmilitary();

cout<<endl<<endl<< "სტანდარტული დრო: ";
t.printstandard() ;
cout<<endl;
return 0;
}
```

კლასის მონაცემი-ელემენტებისათვის აკრძალულია საწყისი მონაცემების მიწოდება იმ კლასის ტანში სადაც ისინი გამოცხადებულნი არიან. აღნიშნული მონაცემი-ელემენტებისათვის საწყისი მნიშვნელობების მიწოდება შესაძლებელია კლასის კონსტრუქტორის მიერ, ან მათთვის საწყისი მნიშვნელობების მინიჭება შეიძლება ფუნქციების საშუალებებით.

ზემოთ მოყვანილ პროგრამაში, კლასის განსაზღვრის შემდეგ, ყოველი ფუნქცია ელემენტის განსაზღვრისას გამოყენებულია (::) ოპერაცია – მოქმედების არის ნების დართვის ბინარული ოპერაცია. კლასის განსაზღვრის და მისი ფუნქცია-ელემენტების გამოცხადების შემდეგ, ეს ფუნქცია-ელემენტები უნდა იქნენ აღწერილნი. ყოველი ფუნქცია-ელემენტი შეიძლება აღწერილი იყოს ან კლასის ტანში (კლასის ფუნქციის პროტოტიპის ჩართვის ნაცვლად) ან კლასის ტანის შემდეგ. იმ შემთხვევებში როდესაც ფუნქცია-ელემენტი აღწერილია შესაბამისი კლასის განსაზღვრის შემდეგ, უშუალოდ ფუნქციის სახელის წინ საჭიროა კლასის სახელისა და მოქმედების არის ნების დართვის ბინარული ოპერაციის (::) მითითება. ამის მითითების აუცილებლობა გამოწვეულია იმით, რომ სხვადასხვა კლასებს შეიძლება ჰქონდეთ ერთი და იგივე სახელის მქონე ელემენტები, და მოქმედების არის ნების დართვის ოპერაცია "მიაბამს" ელემენტის სახელს კლასის სახელს, რათა ცალსახად იქნენ იდენტიფიცირებულნი მოცემული კლასის ფუნქცია-ელემენტები.

მიუხედავათ იმისა, რომ კლასის განსაზღვრისას გამოცხადებული ფუნქცია-ელემენტი შეიძლება აღწერილი იქნეს ამ განსაზღვრის გარეთ, ამ ფუნქცია-ელემენტის მოქმედების არედ მაინც რჩება კლასი.

კლასების გამოყენებით მარტივდება პროგრამირების პროცესი, იმიტომ რომ კლიენტს (ანუ კლასის ობიექტის მომხმარებელს) საქმე აქვს მხოლოდ ოპერაციებთან (ქმედებებთან), რომლებიც ინკაპსულირებულნი ანუ ჩადგმულები არიან ობიექტში. ხშირ შემთხვევებში კლასები არ იქმნება "ცარიელ ადგილას". საზოგადოდ ისინი წარმოადგენენ სხვა კლასების წარმოებულებს, რომლებიც უზრუნველყოფენ ახალ კლასებს მათთვის საჭირო ოპერაციებით. ახალი კლასების შექმნას ძველი კლასების საფუძველზე უწოდებენ მემკვიდრეობით მიღებას (დამკვიდრება).

თ ა ვ ი 7

ლაბორატორიული პრაქტიკუმი

ლაბორატორიული სამუშაო 1

სამკუთხედის ფართობის გამოთვლა

სამუშაოს მიზანი: პროგრამირების სიახლეების ცოდნა, პროგრამის არქიტექტურის გაცნობა, ცვლადების გამოცხადება, პროგრამის ტესტირება და გამართვა.

დავალება დამოუკიდებელი მომზადებისათვის

აუცილებელია იცოდეთ:

1. ცვლადების აღწერა;
   - არითმეტიკული გამოსახულების ჩაწერის წესები;
   - მინიჭების (მიკუთვნების) არითმეტიკული ოპერატორი;
2. მონაცემების შეტანა-გამოტანის მარტივი ორგანიზება;
3. ამოცანის ამოხსნის პროგრამის შედგენა;
4. შემოწმების მიზნით საწყისი მონაცემების ტესტური ვარიანტის მომზადება და ხელით ან კალკულატორით გამოთვლა.

დ ა ვ ა ლ ე ბ ა

შეადგინეთ პროგრამა სამკუთხედის ფართობის გამოსათვლელად ჰერონის ფორმულით.

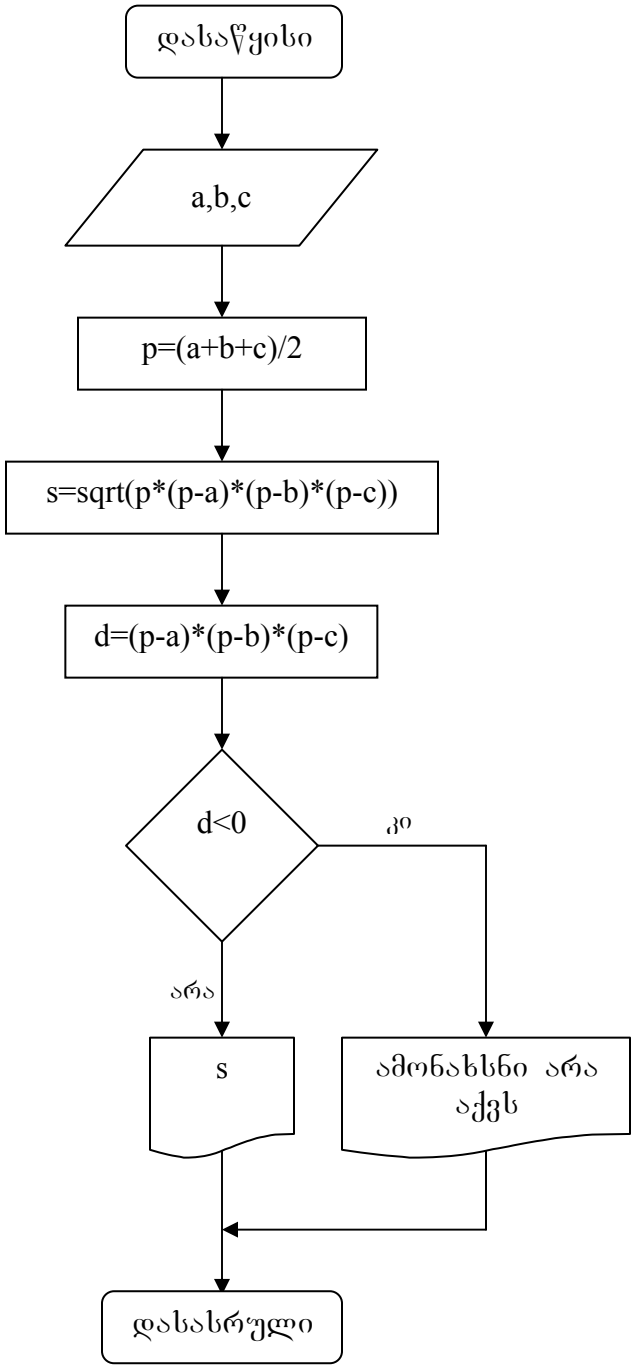
დავალების შესრულების მაგალითი

$S = \sqrt{p(p-a)(p-b)(p-c)}$, სადაც $p = (a+b+c)/2$.

```
#include <iostream>
#include <MATH.h>
using namespace std;

int main()
{
        double a,b,c;
        float p,s,d;
        cout << "a=";
                cin>>a;
                cout<<"b=";
                cin>>b;
```

```
		cout<<"c=";
		cin>>c;
		p=(a+b+c)/2;
		s=sqrt(p*(p-a)*(p-b)*(p-c));
		d=(p-a)*(p-b)*(p-c);

		if (d<0)

			cout<<"amonaxsni ara aqvs"<<endl;

			else

				cout<<"s="<<s<<endl;



	return 0;
}
```

დავალების ალგორითმი ბლოკსქემის სახით



ლაბორატორიული სამუშაო 2

წრფივი სტრუქტურის ალგორითმის დამუშავება

სამუშაოს მიზანი: წრფივი სტრუქტურის პრაქტიკული დამუშავებისა და გამოთვლითი პროცესის პროგრამირების სიახლეების ცოდნა, პროგრამის ტესტირება და გამართვა.

დავალება დამოუკიდებელი მომზადებისათვის

აუცილებელია იცოდეთ:

1. მუდმივების, ცვლადების, სტანდარტული ფუნქციების ჩაწერა;
   - არითმეტიკული გამოსახულების ჩაწერის წესები;
   - მინიჭების (მიკუთვნების) არითმეტიკული ოპერატორი;
2. მონაცემების შეტანა-გამოტანის მარტივი ორგანიზება;
3. ამოცანის ამოხსნის პროგრამის შედგენა;
4. შემოწმების მიზნით საწყისი მონაცემების ტესტური ვარიანტის მომზადება და ხელით ან კალკულატორით გამოთვლა.

დ ა ვ ა ლ ე ბ ა

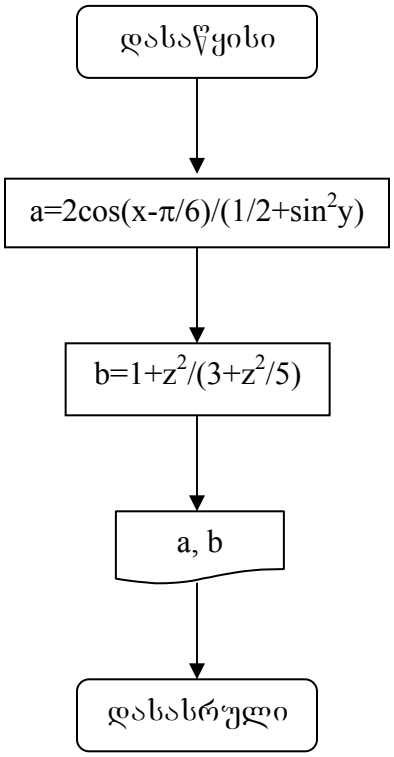
გამოთვალეთ ცვლადების მნიშვნელობები ცხრილი 5.2-ში მოცემული გამოსათვლელი ფორმულებით და საწყისი მონაცემებით. დაბეჭდეთ შესატანი საწყისი მონაცემების მნიშვნელობები და მიღებული შედეგები.

დავალების შესრულების მაგალითი

$$a = \frac{2\cos(x - \pi/6)}{1/2 + \sin^2 y}, \quad b = 1 + \frac{z^2}{3 + z^2/5}$$ სადაც x=1.426, y=-1.220, z=3.5

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
        double x=1.426;
        double y=-1.220;
        double z=3.5;
        float a,b;
        a=(2*cos(x-3.14/6))/((1/2)+(sin(y))*(sin(y)));
        b=1+(z*z)/(3+(z*z)/5);
        cout << "a="<<a<<endl;
        cout <<"b="<<b<<endl;
        return 0;
}
```

დავალების ალგორითმი ბლოკსქემის სახით



ვარიანტების ცხრილი 7. 2

დავალების ვარიანტი	გ ა მ ო ს ა თ ვ ლ ე ლ ი ფ ო რ მ უ ლ ე ბ ი	საქყისი მონაცემების მნიშვნელობები
1	$a=\dfrac{2\cos(x-\pi/6)}{1/2+\sin^2 y}$ $b=1+\dfrac{z^2}{3+z^2/5}$	x = 1,426 y = -1,220 z =3,5
2	$\gamma=\left\lvert x^{y/x}-\sqrt[3]{y/x}\right\rvert$ $\Psi=(y-x)\dfrac{y-z/(y-x)}{1+(y-x)^2}$	x = 1,825 y = 18,225 z = -3,298
3	$s=1+x+\dfrac{x^2}{2!}+\dfrac{x^3}{3!}+\dfrac{x^4}{4!}$ $\psi=x(\sin x^3+\cos^2 y)$	x = 0,335 y = 0,025
4	$y=e^{-bt}\sin(at+b)-\sqrt{\lvert bt+a\rvert}$ $s=b\sin(at^2\cos 2t)-1$	a = -0,5 b = 1,7 t = 0, 44
5	$\omega=\sqrt{(x^2+b)}-b^2\sin^3\dfrac{(x+a)}{x}$ $y=\cos^2 x^3-\dfrac{x}{\sqrt{a^2+b^2}}$	a = 1,5 b = 15,5 x = -2,9
6	$s=x^3 tg(x+b)^2+\dfrac{a}{\sqrt{x+b}}$ $Q=\dfrac{bx^2-a}{e^{ax}-1}$	a = 16,5 b = 3,4 x = 0,61
7	$R=x^2\dfrac{(x+1)}{b}-\sin^2(x+a)$ $s=\sqrt{\dfrac{xb}{a}}+\cos^2(x+b)^3$	a = 0,7 b = 0,05 x = 0,5
8	$y=\sin^3(x^2+a)^2-\sqrt{\dfrac{x}{b}}$ $z=\dfrac{x^2}{a}+\cos(x+b)^3$	a = 1,1 b = 0,004 x = 0,2

9	$f = \sqrt[3]{mtgt + \lvert c \sin t \rvert}$ $z = m\cos(bt \sin t) + c$	m = 2 c = -1 t = 1, 2 b = 0,7
0	$y = btg^2 x - \frac{a}{\sin^2(x/a)}$ $d = ae^{-\sqrt{a}} \cos(\frac{bx}{a})$	a = 3,2 b = 17,5 x = -4,8
11	$f = \ln(a + x^2) + \sin^2(x/b)$ $z = e - cx\frac{x + \sqrt{x + a}}{x - \sqrt{\lvert x - b \rvert}}$	a = 10,2 b = 9,2 x = 2,2 c = 0,5
12	$y = \frac{a^{2x} + b^{-x}\cos(a + b)x}{x + 1}$ $R = \sqrt{x^2 + b} - b^2 \sin^3(x + a)/x$	a = 0,3 b = 0,9 x = 0,61
13	$z = \sqrt{ax \sin 2x + e^{-2x}(x + b)}$ $\omega = \cos^2 x^3 - \frac{x}{\sqrt{a^2 + b^2}}$	a = 0,5 b = 3,1 x = 1,4
14	$U = \frac{a^2 x + e^{-x}\cos bx}{bx - e^{-x}\sin bx + 1}$ $f = e^{2x}\ln(a + x) - b^{3x}\ln(b - x)$	a = 0,5 b = 2,9 x = 0,3
15	$z = \frac{\sin x}{\sqrt{1 + m^2 \sin^2 x}} - cm \ln mx$ $s = e^{-ax}\sqrt{x + 1} + e^{-bx}\sqrt{x + 1,5}$	m = 0,7 c = 2,1 x = 1,7 a = 0,5 b = 1,08

ლაბორატორიული სამუშაო 3

განშტოებადი და ციკლური სტრუქტურის ალგორითმის დაპროგრამება დავალება

სამუშაოს მიზანი: განშტოებადი და ციკლური სტრუქტურის გამოთვლითი პროცესის პროგრამირება, პროგრამის ტესტირება და გამართვა.

დავალება დამოუკიდებელი მომზადებისათვის

1. პროგრამირების ენის შესაძლებლობების ცოდნა რეალიზაციისათვის:
   მართვის პირობითი დაუპირობო გადაცემა;
   განშტოებადი სტრუქტურის გამოთვლითი პროცესი;
2. დავალების შესაბამისი ალგორითმის დამუშავება;
3. დავალების ამოხსნის პროგრამის შედგენა;
4. პროგრამის ფუნქციონირების სისწორის შემოწმება.

დ ა ვ ა ლ ე ბ ა

გამოვთვალოთ ფუნქციის მნიშვნელობა ცხრილ 3.3-ში მოცემული ფორმულით და დაბეჭდეთ იგი ცვლადების სახელის მითითებით.

დავალების შესრულების მაგალითი

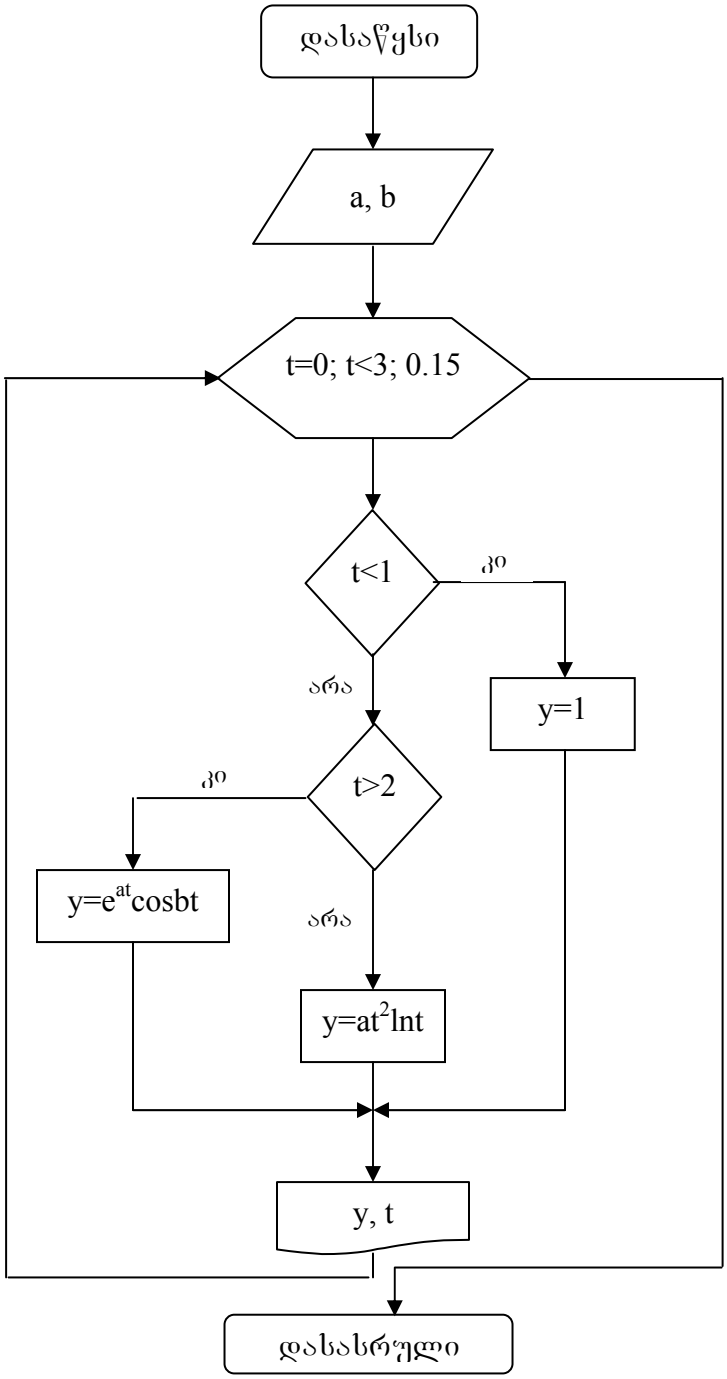
ფუნქცია პირობა

$$y=\begin{cases} a\cdot t^2\cdot \ln t & \begin{pmatrix} 1\le t \prec 2 \\ t<1 \end{pmatrix} \\ 1 & \\ e^{a\cdot t}\cdot \cos b\cdot t & \begin{pmatrix} t>2 \end{pmatrix} \end{cases}$$ სადაც a=-0.5, b=2, $t\in[0;3]$, $\Delta t=0.15$

```
#include <iostream>
# include <iomanip>
#include <math.h>
using namespace std;

int main()
{
        double a,b,e=2.71;
        float y;
        cout<<"a=";
        cin>>a;
        cout<<"b=";
        cin>>b;
        for (float t=0;t<3;t=t+0.15)
        {
```

```
                if (t<1)
                        y=1;
                else
                {
                        if (t>2)
                                y=(powf(e,a*t))*cos(b*t);
                        else
                                y=a*t*t*logf(t);
                }
                cout<<setw(28)<<"y="<<y<< setw(10)<<"t="<<t<<endl ;
        }
        return 0;
}
```

დავალების ალგორითმი ბლოკსქემის სახით



ვარიანტების ცხრილი 7. 3

დავალების ვარიანტი	ფუნქცია	პირობები	საწყისი მონაცემები	არგუმენტის ცვლილების დიაპაზონი და ბიჯი
1	$y=\begin{cases} at^2 \ln t \\ 1 \\ e^{at}\cos bt \end{cases}$	$1 \le t \le 2$ $t < 1$ $t > 2$	a = -0,5 b = 2	$t \in [0;3]$ $\Delta t = 0,15$
2	$y=\begin{cases} \pi x^2 - 7/x^2 \\ ax^3 + 7\sqrt{x} \\ \lg(x+7\sqrt{x}) \end{cases}$	x < 1,3 x = 1,3 x = 1,3	a =1,5	$x \in [0,8;2]$ $\Delta x = 0,1$
3	$\omega=\begin{cases} ax^2+bx+c \\ \frac{a}{x}+\sqrt{x^2+1} \\ (a+bx)/\sqrt{x^2+1} \end{cases}$	x < 2,8 x = 1,2 x > 1,2	a = 2,8 b = -0,3 c = 4	$x \in [1,2]$ $\Delta x = 0,05$
4	$Q=\begin{cases} \pi x^2 - 7/x^2 \\ ax^3 + 7\sqrt{x} \\ \ln(x+7\sqrt{\lvert x+a\rvert}) \end{cases}$	x < 1 x = 1 x > 1,4	a = 1,65	$x \in [0,7;2]$ $\Delta x = 0,1$
5	$y=\begin{cases} 1,5\cos^2 x \\ 1,8ax \\ (x-2)^2+6 \\ 3tgx \end{cases}$	x < 1 x = 1 1 < x < 2 x > 2	a = 2,3	$x \in [0,2;2,8]$ $\Delta x = 0,2$
6	$\omega=\begin{cases} x\sqrt[3]{x-a} \\ x\sin ax \\ e^{-ax}\cos ax \end{cases}$	x > a x = a x < a	a = 2,5	$x \in [1;5]$ $\Delta x = 0,5$
7	$Q=\begin{cases} bx - \lg bx \\ 1 \\ bx + \lg bx \end{cases}$	bx < 1 bx = 1 bx >1	b = 1,5	$x \in [0,1;1]$ $\Delta x = 0,1$
8	$y=\begin{cases} \sin x \lg x \\ \cos^2 x \end{cases}$	x > 3,5 $x \le 3,5$	–	$x \in [2;5]$ $\Delta x = 0,25$
9	$f=\begin{cases} \lg(x+1) \\ \sin^2\sqrt{\lvert ax\rvert} \end{cases}$	x > 1 $x \le 1$	a = 20,3	$x \in [0,5;2]$ $\Delta x = 0,1$

ცხრილი 7. 3 – ის გაგრძელება

დავალების ვარიანტი	ფუნქცია	პირობები	საწყისი მონაცემები	არგუმენტის ცვლილების დიაპაზონი და ბიჯი
10	$z=\begin{cases}(\ln^3 x+x^2)/\sqrt{x+t}\\ \sqrt{x+t}+1/x\\ \cos x+t\sin^2 x\end{cases}$	$x<0,5$ $x=0,5$ $x>0,5$	$t=2,2$	$x\in[0,2;2]$ $\Delta x=0,2$
11	$s=\begin{cases}\dfrac{a+b}{e^x+\cos x}\\ (a+b)/(x+1)\\ e^x+\sin x\end{cases}$	$x<2,8$ $2,8\le x<6$ $x\ge 0,5$	$a=2,6$ $b=-0,39$	$x\in[0;7]$ $\Delta x=0,5$
12	$y=\begin{cases}a\lg x+\sqrt[3]{\lvert x\rvert}\\ 2a\cos x+3x^2\end{cases}$	$x>1$ $x\le 1$	$a=0,9$	$x\in[0,8;2]$ $\Delta x=0,1$
13	$\omega=\begin{cases}\dfrac{a}{i}+bi^2+c\\ i\\ ai+bi^3\end{cases}$	$i<4$ $4\le i\le 6$ $i>6$	$a=2,1$ $b=1,8$ $c=-20,5$	$i\in[0:12]$ $\Delta i=1$
14	$z=\begin{cases}a\sin\left(\dfrac{i^2+1}{n}\right)\\ \cos\left(i+\dfrac{1}{n}\right)\end{cases}$	$\sin\dfrac{i^2+1}{n}>0$ $\sin\dfrac{i^2+1}{n}<0$	$a=0,3$ $n=10$	$i\in[1;10]$ $\Delta i=1$
15	$\omega=\begin{cases}\sqrt{at^2+b\sin t+1}\\ at+b\\ \sqrt{at^2+b\cos t+1}\end{cases}$	$t<0,1$ $t=0,1$ $t>0,1$	$a=2,5$ $b=0,4$	$t\in[-1;1]$ $\Delta t=0,2$

ლაბორატორიული სამუშაო 4

იტერაციული ციკლური სტრუქტურის პროგრამირების ალგორითმი

სამუშაოს მიზანი: იტერაციული ციკლური სტრუქტურის დამუშავება და პროგრამირება; პროგრამის ტესტირება და გამართვა.

დავალება დამოუკიდებელი მომზადებისათვის

უნდა იცოდეთ:

1. იტერაციული ციკლის ორგანიზაცია;
2. იტერაციული ცილის ორგანიზაციისათვის პროგრამირების ენის შესაძლებლობები;
3. პროგრამირების ხერხები – იტერაციის მეთოდით განტოლების ფესვების დაზუსტება, უსასრულო მწკრივის წევრების ჯამის გამოთვლა, ჯამის დაგროვება;
4. დავალების შესრულების ალგორითმის დამუშავება.

დ ა ვ ა ლ ე ბ ა

ცხრილ 3-ში მოცემული $f(x)=0$ სახის განტოლების ფესვის გამოთვლა იტერაციის მეთოდით, რომელიც მოთავსებულია $[\alpha;\beta]$ შუალედში, ε აბსოლუტური ცდომილებით.

დავალების შესრულების მაგალითი

$3\sin\sqrt{x}+0.35\cdot x-3.8=0$, შუალედი [2;3] სიზუსტე $\varepsilon=10^{-3}$

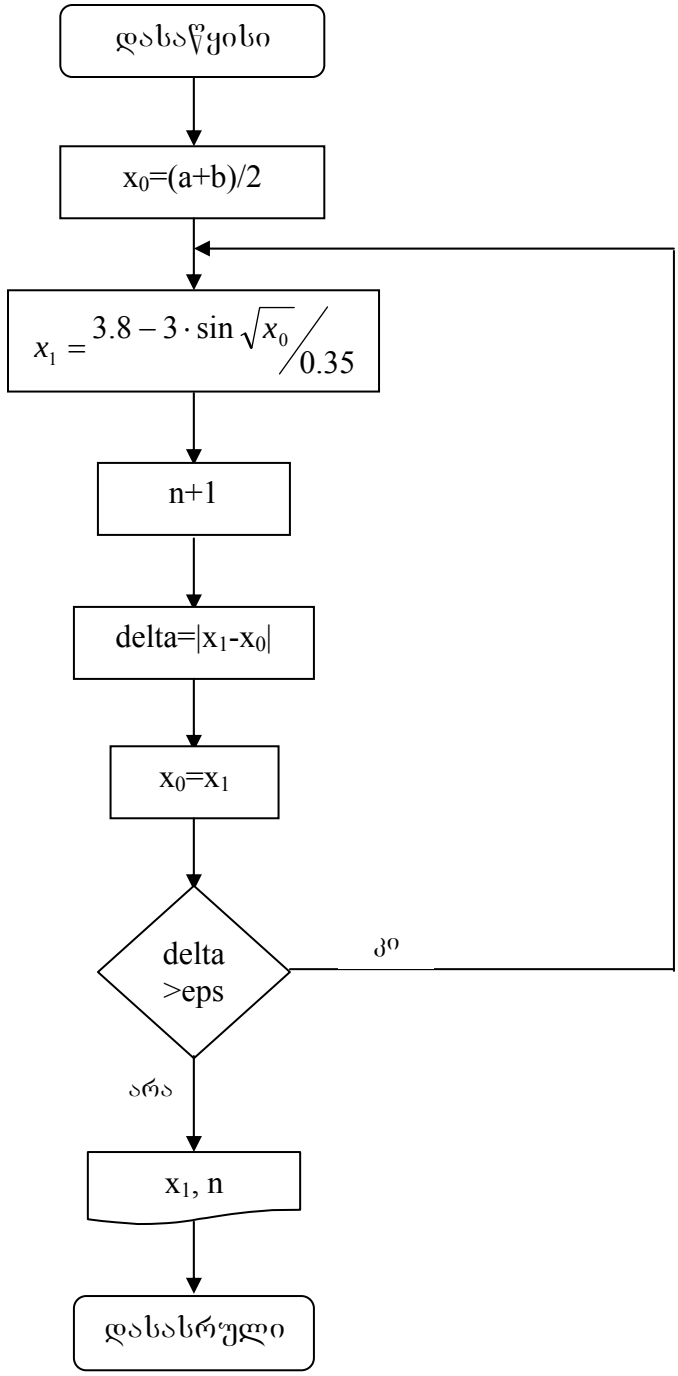
```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
        double x0,x1,eps,delta,a=0,b=1;
        eps=0.001;
        int n=0;
        x0=(a+b)/2;
loop:
        x1=(3.8-3*sin(sqrt(x0)))/0.35;
        n+=1;
```

```
        delta=fabs(x1-x0);
        x0=x1;
        if (delta>eps)
                goto loop;
        cout<<"fesvi = "<<x1<<endl;
        cout<<"iteraciis ricxvi n= "<<n<<endl;

        return 0;
}
```

ვარიანტების ცხრილი 7. 4

დავალების ვარიანტი	განტოლებები	მონაკვეთი (შუალედი)	სიზუსტე
1.	$e^{x}-e^{-x}-2=0$	[0; 1]	10^{-3}
2.	$3\cdot\sqrt{x}+0{,}35\cdot x-3{,}8=0$	[2; 3]	10^{-3}
3.	$x-2+\sin\left(\frac{1}{x}\right)=0$	[1,2; 2]	10^{-4}
4.	$1-x+\sin x-\ln(1+x)=0$	[0; 1,5]	10^{-5}
5.	$x^2-\ln(1+x)-3=0$	[2; 3]	10^{-4}
6.	$x=\frac{1}{3+\sin(3{,}6\cdot x)}=0$	[0; 0,85]	$0{,}5*10^{-4}$
7.	$\ln x-x+1{,}8=0$	[2; 3]	$0{,}5*10^{-4}$
8.	$0{,}1\cdot x^2-x\cdot\ln x=0$	x1ж 2ъ	$0{,}5*10^{-3}$
9.	$x+\cos(x^{0{,}52}+2)=0$	[0,5; 1]	10^{-3}
10.	$\sqrt{1-0{,}4\cdot x}-\arcsin x=0$	[0; 1]	10^{-3}
11.	$x^2+10\cdot x+10=0$	[0; 1]	10^{-5}
12.	$3\cdot x-4\cdot\ln x-5=0$	[2; 4]	$0{,}5*10^{-3}$
13.	$0{,}4+arctg\sqrt{x}-x=0$	[1; 2]	10^{-3}
14.	$\arccos x-\sqrt{1-0{,}3\cdot x^3}=0$	[0; 1]	$0{,}5*10^{-3}$
15.	$2\cdot x-3\cdot\ln x-3=0$	[0,5; 0,6]	10^{-3}

დავალების ალგორითმი ბლოკსქემის სახით



ლაბორატორიული სამუშაო 5

ციკლური სტრუქტურის ალგორითმის პროგრამირება ციკლის გამეორების მოცემული რიცხვით

სამუშაოს მიზანი: ციკლური სტრუქტურის ალგორითმის დამუშავება და პროგრამირება ციკლის გამეორების მოცემული რიცხვით, პროგრამის ტესტირება და გამართვა.

დავალება დამოუკიდებელი მომზადებისათვის

უნდა იცოდეთ:

1. ციკლური სტრუქტურის ალგორითმის ორგანიზაცია ციკლის გამეორების მოცემული რიცხვით;
   - ასეთი ციკლების აგებისათვის პროგრამირების ენის შესაძლებლობები;
   - პროგრამირების ხერხები – გასაზღვრული ინტეგრალის გამოთვლა;
2. დავალების ამოხსნის შესაბამისი ალგორითმის დამუშავება;
3. ამოცანის ამოხსნის პროგრამის შედგენა;
4. გამოთვლების სისწორის შესამოწმებლად განსაზღვრეთ ინტეგრალის ზუსტი მნიშვნელობის მათემატიკური გამოსახულება და ჩართეთ პროგრამაში მისი გამოთვლა.

დ ა ვ ა ლ ე ბ ა

გამოთვალეთ ინტეგრალის $z = \int_a^b f(x)dx$ მნიშვნელობა მოცემულ [a;b] შუალედში. ინტეგრალქვეშა ფუნქცია, დაყოფის წერტილთა რიცხვი n, ინტეგრირების შუალედი [a;b] და სიზუსტე ε აიღეთ ცხრილიდან 4. ზუსტი მნიშვნელობების გამოსათვლელად ისარგებლეთ ცხრილი 5-ით.

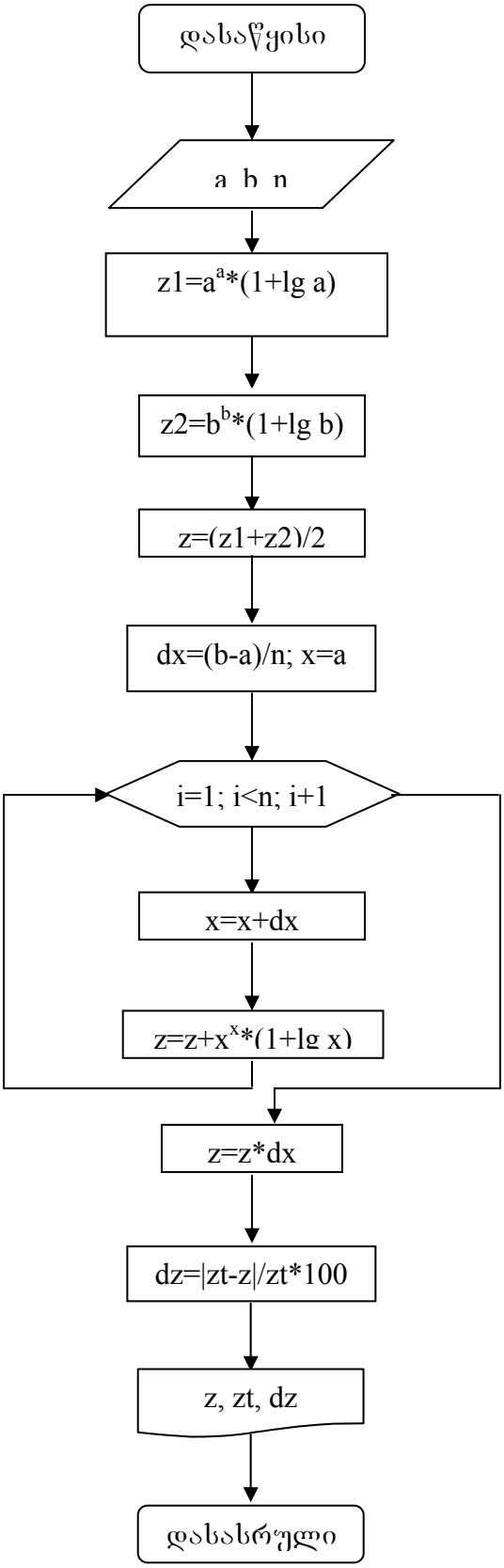
დავალების შესრულების მაგალითი

$f(x) = x^x(1 + \ln x)$, n=40, [1;3], $\varepsilon=10^{-4}$

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
        int n,i;
        float z1,a,b,z2,z,dx,x,dz,zt;
        cout<<"a=";
        cin>>a;
        cout<<"b=";
```

```
        cin>>b;
        cout<<"n=";
        cin>>n;
        z1=powf(a,a)*(1+logf(a));
        z2=powf(b,b)*(1+logf(b));
        z=(z1+z2)/2;
        dx=(b-a)/n;
        x=a;
        for (i=1;i<n;i=i+1)
        {
                x=x+dx;         z=z+powf(x,x)*(1+logf(x));
        }
        z=z*dx;
        zt=26;
        dz=fabs(zt-z)/zt*100;
        cout<<"miaxloebiti mnishvneloba z="<<z<<endl
                <<"zusti mnishvneloba zt="<<zt<<endl
                <<"cdomileba Dz="<<dz<<endl;
        return 0;
}
```

დავალების ალგორითმი ბლოკსქემის სახით



ვარიანტების ცხრილი 7.5

დავალების ვარიანტი	ინტეგრალქვეშა ფუნქცია ფ(x)	რიცხვითი ამოხსნის მეთოდი	მონაკვეტების (დაყოფის) რიცხვი	ინტეგრირების ინტერვალი [a, b]	სიზუსტე ε
1	$\frac{\ln^2 x}{x}$	ტრაპეციის	60	[1; 4]	10^{-4}
2	$\frac{1}{x^2}\sin\frac{1}{x}$	მართკუთხედების	50	[1: 2,5]	$0,5\cdot10^{-3}$
3	$x^x(1+\ln x)$	ტრაპეციის	40	[1; 3]	10^{-4}
4	$\cos x$	ტრაპეციის	60	$\left[0;\frac{\pi}{2}\right]$	10^{-4}
5	$\sin^2 x$	ტრაპეციის	60	$\left[0;\frac{\pi}{2}\right]$	$0,5\cdot10^{-3}$
6	$xe^x \sin x$	ტრაპეციის	100	[0: 1]	10^{-4}
7	$\left(\frac{\ln x}{x}\right)^2$	მართკუთხედების	50	[1: 2,5]	10^{-4}
8	$xarctgx$	ტრაპეციის	50	[0; 3]	$0,5\cdot10^{-3}$
9	$\frac{1}{\sqrt{9+x^2}}$	მართკუთხედების	100	[0; 2]	10^{-5}
10	$e^x \cos^2 x$	ტრაპეციის	60	$[0;\pi]$	10^{-4}
11	$\frac{x^3}{3+x}$	მართკუთხედების	80	[1; 2]	$0,5\cdot10^{-4}$
12	$\left(\frac{\ln x}{x}\right)^3$	ტრაპეციის	50	[1; 2]	10^{-4}
13	$x\left(\frac{e^x - e^{-x}}{2}\right)$	მართკუთხედების	50	[0; 2]	10^{-4}
14	$x^2 \sin 2x$	ტრაპეციის	100	[1; 2]	10^{-4}
15	$\frac{x}{x^4+3x^2+2}$	ტრაპეციის	50	[1; 2]	$0,5\cdot10^{-3}$

შ ე ნ ი შ ვ ნ ა : მითითებული სიზუსტე მოცემულ სამუშაოში არ გამოიყენება

პირველყოფილი ფუნქციის $F(x)=\int f(x)dx$ გამოთვლისათვის საჭირო გამოსახულება მოყვანილია 5. 7 ცხრილში.

ცხრილი 7.5[1]

დავალების ვარიანტი	პირველყოფილი ფუნქცია $F(x)=\int f(x)dx$	დავალების ვარიანტი	პირველყოფილი ფუნქცია $F(x)=\int f(x)dx$
1	$\frac{\ln^3 x}{3}$	9	$\ln\left\|x+\sqrt{x^2+9}\right\|$
2	$\cos\left(\frac{1}{x}\right)$	10	$e^x\left(\frac{1}{2}+\frac{1}{10}\cos 2x+\frac{1}{5}\sin 2x\right)$
3	x^x	11	$\frac{x^3}{3}-\frac{3}{2}x^2+9x-27\ln(x+3)$
4	$\sin x$	12	$-\frac{1}{8}\left(4\ln^3 x+6\ln^2 x+6\ln x+3\right)$
5	$\frac{x}{2}-\frac{\sin 2x}{4}$	13	$xchx-shx$
6	$\frac{e^x}{2}\left(x\sin x+(1-x)\cos x\right)$	14	$-\frac{2x^2-1}{4}\cos 2x+\frac{x}{2}\sin 2x$
7	$-\frac{1}{x}\left(\ln^2 x+2\ln x+2\right)$	15	$\frac{1}{2}\ln\left(\frac{-x^2-1}{x^2+2}\right)$
8	$\frac{arctg\left(x^2+1\right)-x}{2}$		

ლაბორატორიული სამუშაო 6

ერთგანზომილებიანი მასივების დამუშავება

სამუშაოს მიზანი: მასივებთან პრაქტიკული მუშაობის გაცნობა, მასივების შეტანა-გამოტანის თავისებურებების გათვალისწინება, ციკლური სტრუქტურის პროგრამის ორგანიზაცია პროგრამირების ხერხების გამოყენებით.

დავალება დამოუკიდებელი მომზადებისათვის

1. პროგრამირების ენის შესაბამისად მასივის განზომილების აღწერის ხერხები;
   - მასივების შეტანა-გამოტანის ხერხები, პროგრამირების კონკრეტული ენისათვის ჯამისა და ნამრავლის დაგროვების ხერხების რეალიზაცია;
   - შედეგების დამახსოვრება, მასივის ელემენტებიდან უდიდესი და უმცირესი მნიშვნელობს ელემენტების მოძებნა;
2. დავალების შესაბამისი ალგორითმის დამუშავება;
3. ამოცანის ამოხსნის პროგრამის შედგენა;
4. პროგრამის ამონახსნის შემოწმება.

დ ა ვ ა ლ ე ბ ა

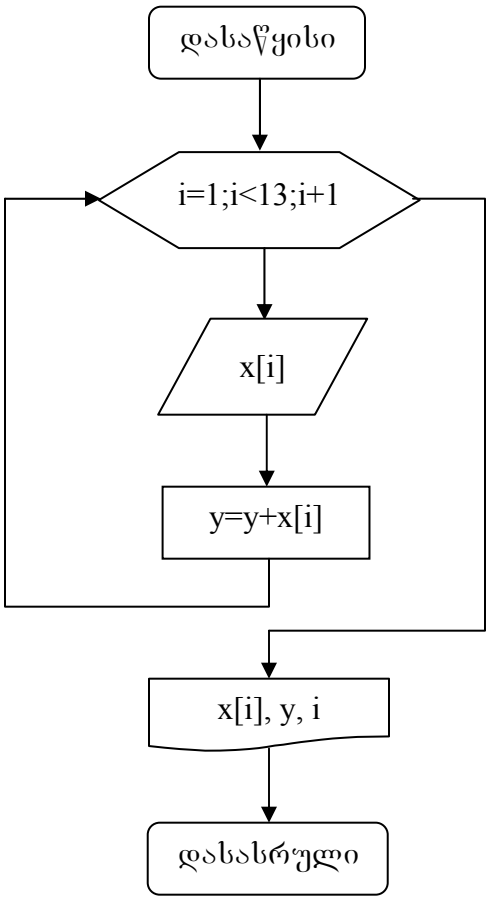
შეიმუშავეთ მასივი, გამოთვალეთ მისი ელემენტების რაოდენობა და ამ ელემენტების ჯამი.

დავალების შესრულების მაგალითი

```
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

int main()
{
        const int j=12;
        double X[j];
        double y=0;
```

```
	for (int i=1;i<13;i++)
	{
		cout<<"X("<<i<<")=";
		cin>>X[i];
		y=y+X[i];
	}
	for (i=0;i<12;i+=3)

	cout<<setw(10)<<X[i+1]<<setw(10)<<X[i+2]<<setw(10)<<X[i+3]<<endl;

	cout<<endl;
	cout<<"მასივის ელემენტების ჯამი Y="<<y<<endl
		<<"მასივის ელემენტების რაოდენობა i="<<i-1<<endl;
	return 0;
}
```

დავალების ალგორითმი ბლოკსქემის სახით



ვარიანტების ცხრილი 7. 6

დავალების ვარიანტი	მასივი	მოქმედება	პირობა და შეზღუდვა (საზღვრები)
1.	X(100)	გამოთვალეთ X მასივის ელემენტების რაოდენობა და ჯამი	$0 \le x_i \le 1$
2.	A(80)	გამოთვალეთ A მასივის ელემენტების საშუალო არითმეტიკულის მნიშვნელობა	$a_i > 0$
3.	X(70)	გადაწერეთ X მასივის ელემენტები Y მასივში და დათვალეთ ელემენტების რაოდენობა	$-1 \le x_i \le 1$
4.	B(50)	განსაზღვრეთ B მასივის მაქსიმალური ელემენტი და მისი რიგითი ნომერი	$x_i > 0$
5.	C(40)	განსაზღვრეთ C მასივის მინიმალური ელემენტი და მისი რიგითი ნომერი	$x_i < 0$
6.	D(80)	იპოვეთ D მასივის მაქსიმალური და მინიმალური ელემენტები და შეუცვალეთ ადგილები	–
7.	Y(20)	გამოთვალეთ Y მასივის ელემენტების გეომეტრიული მნიშვნელობა	–
8.	Z(30)	R მასივში მოათავსეთ Z მასივის ჯერ დადებითი, შემდეგ კი უარყოფითი ელემენტები	–
9.	N(50)	განსაზღვრეთ N მასივის სამის ჯერადი ელემენტების ჯამი	$n_i / 3 \bullet 3 = n_i$
10.	X(N)	გამოთვალეთ X მასივის ელემენტების რაოდენობა და ჯამი	$x_i > 0, N \le 30$
11.	A(N)	იპოვეთ A მასივის ელემენტების საშუალო გეომეტრიული	$a_i > 0, N < 50$
12.	X(N)	გადაწერეთ Y მასივში X მასივის დადებითი ელემენტები მიმდევრობით	$x_i > 0, N \le 40$

ცხრილი 7. 6-ის გაგრძელება

13.	X(N)	გადაწერეთ მიმდევრობით X მასივის დადებითი ელემენტები Y მასივში, ხოლო უარყოფითი ელემენტები კი Z მასივში	$N \le 40$
14.	B(K)	განსაზღვრეთ B მასივის მაქსიმალური ელემენტი და მისი რიგითი ნომერი	$x < 0, K \le 40$
15.	C(K)	განსაზღვრეთ C მასივის მინიმალური ელემენტი და მისი რიგითი ნომერი	$-1 \le x_i \le 1, K \le 20$

ლაბორატორიული სამუშაო 7

ჩადგმული ციკლების სტრუქტურით ალგორითმების პროგრამირება

სამუშაოს მიზანი: ჩადგმული ციკლების სტრუქტურის ალგორითმიზაციისა და პროგრამირების შესწავლა.

დავალება დამოუკიდებელი მომზადებისათვის

უნდა იცოდეთ:

1. ჩადგმული ციკლების სტრუქტურის გამოთვლის ორგანიზაცია;
   - პროგრამირების ენის საშუალებები ჩადგმული ციკლების სტრუქტურის ორგანიზაციისათვის;
   - პროგრამირების ხერხები მოცემული სიზუსტით განსზღვრული ინტეგრალის გამოსათვლელად;
2. დავალების ამოსახსნელად შესაბამისი ალგორითმის დამუშავება;
3. დავალების ამოსახსნელად პროგრამის შედგენა;
4. პროგრამის ამონახსნის შემოწმება.

დ ა ვ ა ლ ე ბ ა

გამოთვალეთ მოცემული ε სიზუსტით განსაზღვრული ინტეგრალის $\int_a^b f(x)dx$ მნიშვნელობა.

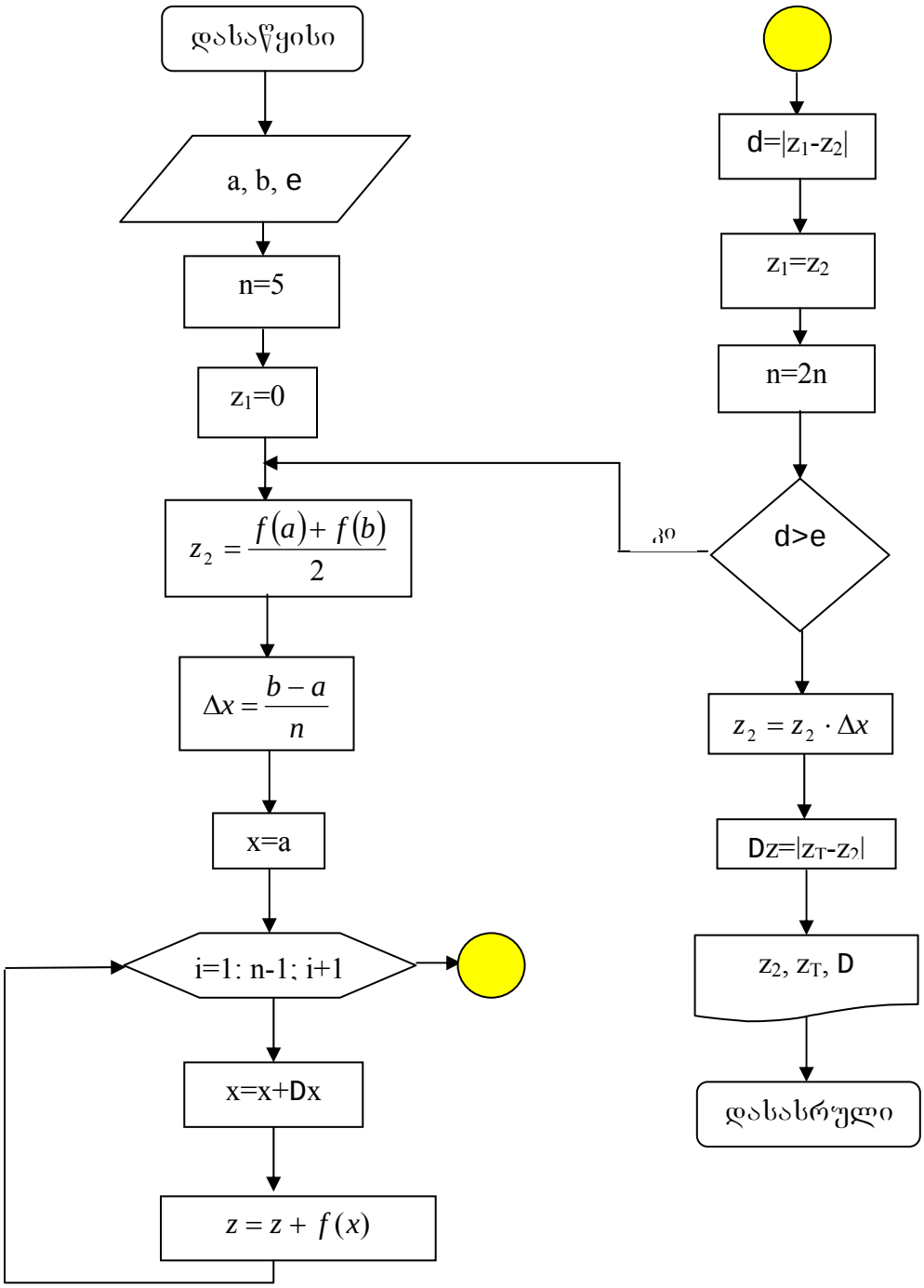
დავალების შესრულების მაგალითი

$z = \int_a^b \frac{\sin^2 x}{1 + 2 \cdot k \cdot \cos x + k^2} dx$, სიზუსტით ε; a=0; b=π; k=0.5.

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
        double a,b,eps,z3,n,z4,z2,x,z1,Dx,delta;
        cin>>a;
        cin>>b;
        cin>>eps;
        const double k=0.5,zt=3.141592/2;
```

```
        n=5;
        z1=0;
        z3=powf(sin(a),2)/(1+2*k*cos(a)+k*k);
        z4=powf(sin(b),2)/(1+2*k*cos(b)+k*k);
        z3=(z3+z4)/2;
loop:
        z2=z3;
        Dx=(b-a)/n;
        x=a;
        for(int i=1;i<n-1;i++)
        {
                x=x+Dx;
           z2=z2+powf(sin(x),2)/(1+2*k*cos(x)+k*k);
        }
        z2=z2*Dx;
        cout<<"N="<<n<<endl
                <<"Z2="<<z2<<endl;
        delta=fabs(z1-z2);
        z1=z2;
        n=2*n;
        if (delta>eps)
                goto loop;
        cout<<"miaxloebiTi mnishvneloba Z="<<z2<<endl
                <<"zusti mnishvneloba Zt="<<zt<<endl
                <<"absoluturi cdomileba"<<fabs(z2-zt)<<endl;
        return 0;
}
```

დავალების ალგორითმი ბლოკსქემის სახით



ვარიანტების ცხრილი 7. 7

დავალების ვარიანტი	ინტეგრალქვეშა ფუნქცია f(x)	რიცხვითი ამოხსნის მეთოდი	მონაკვეთების (დაყოფის) რიცხვი	ინტეგრირების ინტერვალი [a, b]	სიზუსტე ε
1	$\frac{\ln^2 x}{x}$	ტრაპეციის	60	[1; 4]	10^{-4}
2	$\frac{1}{x^2}\sin\frac{1}{x}$	მართკუთხედების	50	[1: 2,5]	$0,5\cdot10^{-3}$
3	$x^x(1+\ln x)$	ტრაპეციის	40	[1; 3]	10^{-4}
4	$\cos x$	ტრაპეციის	60	$\left[0;\frac{\pi}{2}\right]$	10^{-4}
5	$\sin^2 x$	ტრაპეციის	60	$\left[0;\frac{\pi}{2}\right]$	$0,5\cdot10^{-3}$
6	$xe^x \sin x$	ტრაპეციის	100	[0: 1]	10^{-4}
7	$\left(\frac{\ln x}{x}\right)^2$	მართკუთხედების	50	[1: 2,5]	10^{-4}
8	$xarctgx$	ტრაპეციის	50	[0; 3]	$0,5\cdot10^{-3}$
9	$\frac{1}{\sqrt{9+x^2}}$	მართკუთხედების	100	[0; 2]	10^{-5}
10	$e^x \cos^2 x$	ტრაპეციის	60	$[0;\pi]$	10^{-4}
11	$\frac{x^3}{3+x}$	მართკუთხედების	80	[1; 2]	$0,5\cdot10^{-4}$
12	$\left(\frac{\ln x}{x}\right)^3$	ტრაპეციის	50	[1; 2]	10^{-4}
13	$x\left(\frac{e^x - e^{-x}}{2}\right)$	მართკუთხედების	50	[0; 2]	10^{-4}
14	$x^2 \sin 2x$	ტრაპეციის	100	[1; 2]	10^{-4}
15	$\frac{x}{x^4+3x^2+2}$	ტრაპეციის	50	[1; 2]	$0,5\cdot10^{-3}$

შ ე ნ ი შ ვ ნ ა : მითითებული სიზუსტე მოცემულ სამუშაოში არ გამოიყენება

ლაბორატორიული სამუშაო 8

მატრიცის დამუშავება

სამუშაოს მიზანი: მატრიცების დასამუშავებლად ჩადგმული ციკლების გამოყენება, ალგორითმიზაციისა და პროგრამირების შესწავლა, ცოდნის განმტკიცება.

დავალება დამოუკიდებელი მომზადებისათვის

უნდა იცოდეთ:

1. ჩადგმული ციკლების ორგანიზაციის წესი მატრიცის ელემენტების დალაგების (დახარისხების) გათვალისწინებით;
   - ჩადგმული ციკლების სტრუქტურაში პროგრამირების ხერხების გამოყენების წესები;
   - პროგრამირების ენის შესაბამისად მატრიცის ელემენტების შეტანა-გამოტანის ხერხები;
2. დავალების შესაბამისი ალგორითმის დამუშავება;
3. ამოცანის ამოხსნის პროგრამის შედგენა;
4. საწყისი მონაცემებისათვის პროგრამის მუშაობის სისწორის შემოწმება.

დ ა ვ ა ლ ე ბ ა

დაამუშავეთ დავალებით გათვალისწინებული მატრიცა, ამობეჭდეთ შედეგები და საწყისი მატრიცა.

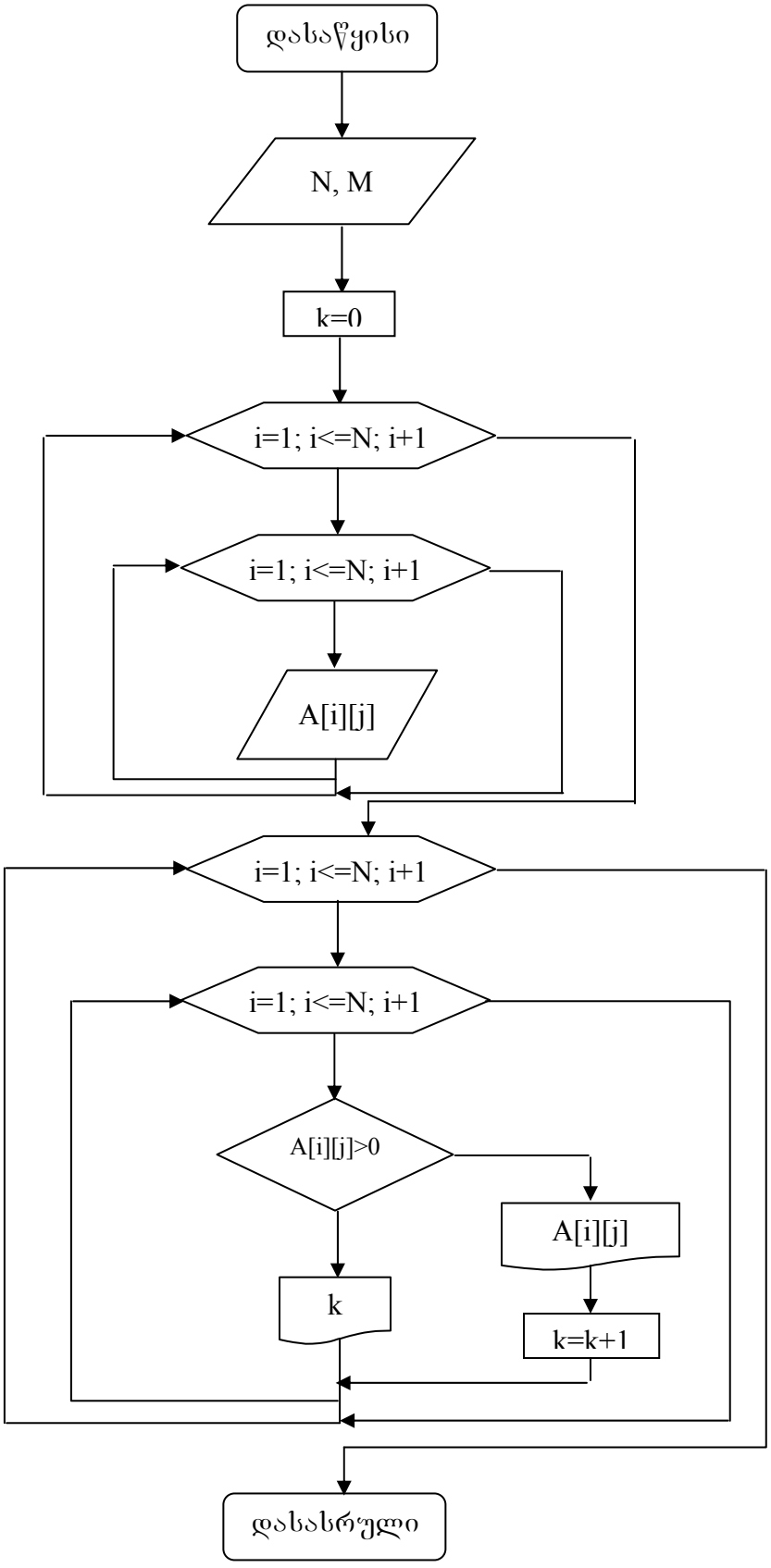
დავალების შესრულების მაგალითი

გამოთვალეთ მატრიცის დადებითი ელემენტების რაოდენობა.

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
        double A[20][10];
        int N,M,k;
        cin>>N
           >>M;
        k=0;
        for(int i=1;i<=N;i++)
        {
                for (int j=1;j<=M;j++)
                {
                        cout<<"A("<<i<<","<<j<<")=";
```

```
                cin>>A[i][j];
            }
        }
        for(i=1;i<=N;i++)
        {
            for(int j=1;j<=M;j++)
            {
                if (A[i][j]>=0)
                {
                    cout<<A[i][j]<<endl; k++;
                }
                else
                {
                    cout<<"dadebiti elementebis raodenoba k="<<k<<endl;
                }
            }
        }
        return 0;
}
```

დავალების ალგორითმი ბლოკსქემის სახით



ვარიანტების ცხრილი 7.8

დავალების ვარიანტი	მატრიცის სახელი და ზომები	მოქმედება	პირობები და საზღვრები
1.	A(10, 15)	გამოთვალეთ და დაიმახსოვრეთ მატრიცის თითოეული სვეტის დადებითი ელემენ-ტების რაოდენობა და ჯამი	$a_{ij} > 0$
2.	A(N, M)	გამოთვალეთ და დაიმახსოვრეთ მატრიცის თოთოეული სტრიქონის ელემენტების რაოდენობა და ჯამი	$N \le 20$ $M \le 15$
3.	B(N, N)	გამოთვალეთ მატრიცის მთავარი დიაგონა-ლის ზემოთ და ქვემოთ მდგომი ელე-მენტების რაოდენობა და ჯამი	$N \le 12$
4.	C(N, N)	გამოთვალეთ მთავარი დიაგონალის ზემოთ მდგომი ელემენტების რაოდენობა და ჯამი	$c_{ij} > 0$ $N > 12$
5.	D(K, K)	მატრიცის უარყოფითი ელემენტების ადგილზე ჩაწერეთ ნულები და დაბეჭდეთ ზოგადი სახით	$K \le 10$
6.	D(10, 10)	მატრიცის უარყოფითი ელემენტების ადგილზე ჩაწერეთ ნულები, ხოლო დადებითი ელემენტების ადგილზე კი ერთიანები. დაბეჭდეთ ქვედა სამკუთცა მატრიცა	
7.	F(N, M)	მოძებნეთ მატრიცის თითოეული სტრიქონის მაქსიმალური და მინიმალური ელემენტები და მათ ადგილზე ჩაწერეთ შესაბამისად სტრიქონის პირველი და ბოლო ელემენტი. დაბეჭდეთ მატრიცა.	$N \le 20$ $M \le 10$
8.	F(10, 8)	დაბეჭდეთ მთავარი ელემენტების დიაგო-ნალი და მის ზემოდ მდგომი დიაგონალი. შედეგები მოათავსეთ ორ სტრიქონში	
9.	N(10, 10)	იპოვეთ მთელრიცხვა მატრიცის თითოეული სტრიქონის იმ ელემენტების რიცხვი, რომლებიც ხუთის ჯერადია და ამოარჩიეთ უდიდესი	$\frac{n_{ij}}{5} * 5 = n_{ij}$

10.	N(10,10)	N მატრიცის დადებითი ელემენტებისაგან აწარმოეთ მატრიცი M(10, KMAX), მის სტრიქონებში ელემენტები მოათავსეთ მიმდევრობით, სადაც KMAX – მატრიცის დადებითი ელემენტების მაქსიმალური რიცხვია. გამოტოვებულ ელემენტების ადგილზე ჩაწერეთ ნულები და დაბეჭდეთ ორივე მატრიცი.	
11.	P(N, N)	მოძებნეთ მატრიცის თითოეულ სტრიქონში უდიდესი ელემენტი და ადგილი შეუცვალეთ მთავარი დიაგონალის ელემენტებთან. დაბეჭდეთ მიღებული მატრიცი.	$N \le 15$
12.	R(K, N)	მოძებნეთ მატრიცის უდიდესი და უმცირესი ელემენტი შეუცვალეთ ადგილები.	$K \le 20$ $N \le 10$
13.	S(25, 8)	მატრიცის პირველ 24 სტრიქონსა და 7 სვეტში შეიტანეთ საწყისი მონაცემები. გამოთვალეთ თითოეული სტრიქონის ელემენტების საშუალო არითმეტიკული მნიშვნელობა და ჩაწერეთ მე-8-ე სვეტში, გამოთვალეთ ასევე თითოეული სვეტის ეელემენტების საშუალო არითმეტიკული მნიშვნელობა და ჩაწერეთ 25-ე სტრიქონში. დაბეჭდეთ მიღებული მატრიცი.	
14.	T(N, M)	მოძებნეთ სტრიქონი უდიდესი და უმცირესი ელემენტების ჯამით. დაბეჭდეთ მოძებნილი სტრიქონი და მისი ელემენტების ჯამი.	$N \le 20$ $M \le 15$
15.	V(15, 10)	მატრიცის თითოეული სტრიქონის ელემენტები დაალაგეთ ზრდის მიხედვით. დაბეჭდეთ მიღებული მატრიცი.	

ლაბორატორიული სამუშაო 9

დიალოგური პროგრამის დამუშავება

სამუშაოს მიზანი: დიალოგური ტიპის ამოცანების ალგორითმიზაციისა და პროგრამირების სიახლეების, შეტანა-გამოტანის ოპერაციების თავისებურობების ცოდნა; დიალოგურ პროგრამებში მომხმარებლის შეტყობინების ანალიზისა და გადაწყვეტილების მიღების ხერხების ფლობა.

უნდა იცოდეს:

1. დიალოგური პროგრამების სახეები;
   - მომხმარებლის პასუხების ანალიზის ხერხები;
   - სიმბოლური სტრიქონების შედარების ხერხები, მათგან წვესტრიქონის გამოყოფა;სიმბოლური ინფორმაციის გარდაქმნის ხერხები;
2. დავალების შესაბამისი ალგორითმის დამუშავება;
3. დავალების ამოხსნის პროგრამის შედგენა;
4. პროგრამის ტესტური ვარიანტისა და საწყისი მონაცემების მომზადება.
5.

დ ა ვ ა ლ ე ბ ა

შეადგინეთ დიალოგური პროგრამა მოსწავლის ცოდნის შსაფასებლად

დავალების შესრულების მაგალითი

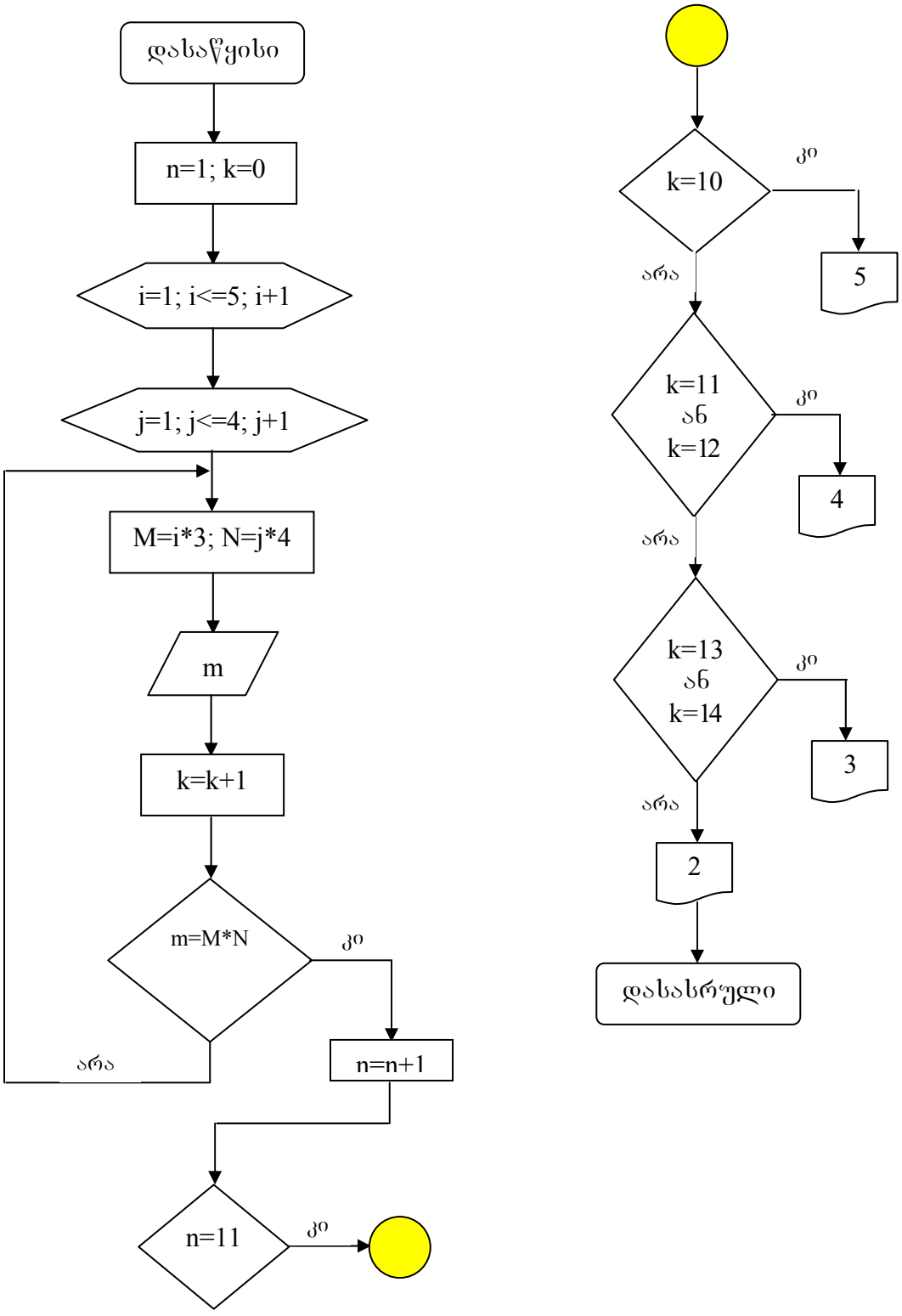
შეადგინეთ პროგრამა, რომელიც შეამოწმებს მოსწავლის ცოდნას, მთელი მთელი რიცხვების გამრავლებაზე. მთელი რიცხვებია N x M, სადაც N და M ღებულობენ მნიშვნელობებს დიაპაზონიდან (0, 20). კითხვების რაოდენობაა 10. პროგრამამ უნდა შეამოწმოს პასუხის სისწორე. თუ პასუხი სწორია გამოიტანოს შეტყობინება “სწორია” და მომდევნო შეკითხვა. თუ პასუხი სწორი არ არის , მაშინ გამოიტანოს შეკითხვა “არასწორია”, ხელახლა უპასუხოს კითხვას და დააბრუნოს იგივე შეკითხვა. მოსწავლის შეფასება მოხდეს შემდეგნაირად:

- თუ პირველივე ცდაზე სწორი პასუხი 10-ზე ნაკლები არ არის შეფასება= “5”;
- თუ პიველივე ცდაზე სწორი პასუხი 8-ზე ნაკლები არ არის შეფასება = “4’;
- თუ პიველივე ცდაზე სწორი პასუხი 6-ზე ნაკლები არ არის შეფასება = “3’;
- ყველა სხვა შემთხვევაში შეფასება =”2”

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
        int M,N,m,n,k;
        n=1;
        k=0;
        for(int i=1;i<=5;i++)
```

```
        {for(int j=1;j<=4;j++)
                {
loop:
                        M=i*3;
                        N=j*4;
                        cout<<"kitxva "<<n<<endl
                                <<M<<"*"<<N<<"=";
                        cin>>m;
                        k+=1;
                        if(m= =(M*N))
                        { n+=1;
                        }else
                                goto loop;
                        if(n= =11)
                        { if(k= =10)
                                        cout<<"5"<<endl;
                                else{
                                        if(k= =11||k= =12)
                                                cout<<"4"<<endl;
                                        else{
                                                if(k= =13||k= =14)
                                                        cout<<"3"<<endl;
                                                else
                                                        cout<<"2"<<endl;
                                        }
                                }return 0;
                        }
                }
        }return 0;
}
```

დავალების ალგორითმი ბლოკსქემის სახთ



ლაბორატორიული სამუშაო 10

მონაცემთა ფაილური სტრუქტურის დამუშავება

სამუშაოს მიზანი: მონაცემთა ფაილური სტრუქტურის ალგორითმიზაციისა და პროგრამირების სიახლეების ფლობა; ფაილის სტრუქტურის დაპროექტება, ფაილში მონაცემების შეტანა და ფაილიდან მონაცემების წაკითხვა.

უნდა იცოდეს:

1. მონაცემთა ფაილური სტრუქტურის ძირითადი ტერმინოლოგია;
   - ფაილი და მისი სტრუქტურა, ფიზიკური და ლოგიკური ჩანაწერი, ჩანაწერები ფორმატით და უფორმატოდ, შეღწევადობის მეთოდები;
   - მიმდევრობითი ფაილის დამუშავებისათვის პროგრამირების ენის შესაძლებლობები: ფაილში მონაცემების ჩაწერა, ფაილიდან მონაცემების წაკითხვა, ფაილში ჩანაწერების დამატება, ჩანაწერების შესწორება და ა. შ.
2. დავალების შესაბამისი ალგორითმის დამუშავება;
3. ამოცანის ამოხსნის პროგრამის შედგენა;
4. პროგრამის ტესტური ვარიანტისა და საწყისი მონაცემების მომზადება.

დ ა ვ ა ლ ე ბ ა

შეადგინეთ პროგრამა სტუდენტის მიერ სესიებზე ჩაბარებული საგნების შესახებ

ფაილის სტრუქტურა: ჯგუფის ნომერი,სტუდენტის გვარი, ხუთი საგნის შეფასება, ჩანაწერის რაოდენობა 25.

დავალების შესრულების მაგალითი

მიმდინარე ჩანაწერის მისაღებად პროგრამაში მოახდინეთ მოთხოვნა, ეკრანიდან ოპერატიულ მეხსიერებაში ინფორმაციის მორიგი ნაკადის შეტანაზე. ამისათვის ისარგებლეთ ალგორითმული ენის C++-ის ჩვეულებრივი შეტანა-გამოტანის ოპერატორებით. მონაცემთა მიღებული სტრიქონი ჩაწერეთ ფაილში პირველ ჩანაწერად. ამისათვის გამოიყენეთ ფაილში მონაცემების ჩაწერის ოპერატორი, რის შემდეგაც ისევ მოითხოვთ ეკრანიდან ოპერატიულ მეხსიერებაში მეორე სტრიქონის შეტანას და მოახდენთ მის ფაილში ჩაწერას. ეს პროცესი -ეკრანიდან მონაცემის შეტანა და ფაილში ჩაწერა-გააგრძელეთ მანამ, სანამ ეკრანზე არ გამოიტანება მონაცემების შეტანის დამთავრების ნიშანი. ფაილში ჩანაწერების რაოდენობის დასათვლელად გამოიყენეთ მთვლელი, რომლის მნიშვნელობა ფაილში ყოველი სტრიქონის ჩაწერის შემდეგ გაიზრდება 1-ით. ფაილის შექმნის შემდეგ, შემოწმებისათვის მოახდინეთ ფაილის ჩანაწერების წაკითხვა და მათი ეკრანზე გამოტანა.

```
#include <fstream.h>
#include <iomanip.h>

void main()
{
	char jgufi[4],gvari[20],nishani[5],ind,ind1;
	fstream myfile;
	myfile.open("nishnebi.txt",ios::out);
	cout<<"sheitanet jgufis nomeri: ";
	cin>>jgufi;
	myfile<<jgufi;
loop:
	cout<<"sheitanet studentis gvari: ";
	cin>>gvari;
	cout<<"sheitanet studentis nishnebi: "<<endl;
	for(int i=1;i<6;i++)
	{
		cout<<i<<"-e gamocdis nishani: ";
		cin>>nishani[i];
	}
	myfile<<endl<<'\t'<<gvari<<'\t'<<nishani[1]<<'\t'<<nishani[2]<<'\t'<<nishani[3]<
<'\t'<<nishani[4]<<'\t'<<nishani[5];
	cout<<"kidev gsurt monacemebis shetana (y/n)? ";
	cin>>ind;
	if(ind=='y'||ind=='Y')
	{
		cout<<"gsurt jgufis nomris shecvla (y/n)? ";
		cin>>ind1;
		if(ind1=='y'||ind1=='Y')
		{
			cout<<"sheitanet jgufis nomeri: ";
			cin>>jgufi;
			myfile<<endl<<jgufi;
		}
		goto loop;
	}
	myfile.close();
}
```

ლ ი ტ ე რ ა ტ უ რ ა

1. Ivor Horton “Begin Visual C++ 6.0” ელექტრონული ვერსია 2004წ.
2. Ivor Horton “ Introduction to Visual C++ 6.0 Standart Edition” ელექტრონული ვერსია 2004წ.
3. П. Франка “C++ Учебный курс “ “ Питер “ Санкт-Петербург, Москва – Харьков-Минск 2000г.
4. В. Е. Алексеев, А. С. Ваулин, Г. Б. “ Петрова вычислительная техника и програм-мирование “, Практикум по программированию под редакцией др-а. тех. наук. проф. А. В. Петрова. , Москва , “Высшая Школа“ 1991г.

ს ა რ ჩ ე ვ ი