Georgian Technical University

Roman Samkharadze, Lia Gachechiladze,
Marina Kurdadze, Mirian Kalabegishvili

A collection of tests on the subject

"Algorithmization Fundamentals and Programming Elements"

Georgian Technical University

Roman Samkharadze, Lia Gachechiladze,
Marina Kurdadze, Mirian Kalabegishvili

A collection of tests on the subject

"Algorithmization Fundamentals and Programming Elements"

Approved:
As a tutorial by the editorial board of
"IT-Consulting Scientific Center" of GTU.
Protocol # 1, 3.04.2023

Tbilisi
2023

UDC 004.65

The tutorial presents tests developed for the C++ language. The tests cover the fundamentals of the C++ language, control statements, arrays, classes, encapsulation, methods, constructors, random number generator, and file handling issues. Students can be tested using the presented tests.

It is intended for students of the "Computer Science" English-language undergraduate program.

Reviewers: Professor M. Kiknadze
Associate Professor S. Khoshtaria

Edited by Prof. G. Surguladze

ISBN 978-9941-8-5445-3

Contents

Topic 1. The concept of algorithm. Elements of C ++ programming language

Data types: int, double, bool, char, string.

In all the tests in this book, the correct answer is "a".

1.1.1. A program written in C++ language begins and ends:
a. with curly braces
b. with brackets
c. with parenthesis

1.1.2. With "//" characters start:
a. a one-line comment
b. a multi-line comment
c. comments do not start with these characters

1.1.3. Between /* and */ characters are placed:
a. a multi-line comment
b. a one-line comment
c. comments do not start with these characters

1.1.4. During program execution:
a. comments are skipped
b. comments are executed
c. comments are converted to integer types

1.1.5. Declarations of variables are terminated by:
a. a semicolon ";"
b. a comma ","
c. a colon ":"

1.1.6. Operators are separated:
a. by semicolons ";"
b. by commas ","
c. by colons ":"

1.1.7. The "int" keyword starts:
a. declaration of an integer variable
b. declaration of a double variable
c. declaration of a boolean variable

1.1.8. The "double" keyword starts:
a. declaration of a double variable
b. declaration of an integer variable
c. declaration of a boolean variable

1.1.9. The "bool" keyword starts:
a. declaration of a boolean variable
b. declaration of an integer variable
c. declaration of a double variable

1.1.10. Variable names must begin with:
a. a symbol or an underscore (_)
b. a number
c. any special character other than an underscore (_)

1.1.11. Variable names must not begin with:
a. a number
b. a symbol
c. an underscore (_)

1.1.12. A variable:
a. must be declared before it can be used
b. must be declared after it is used
c. should never be declared

1.1.13. The variable:
a. should not be declared after it is used
b. should not be declared before it is used
c. should never be declared

1.1.14. Can two or more statements be placed in one line of the program?:
a. it is allowed
b. partly allowed
c. It's not allowed

1.1.15. In C++ language uppercase and lowercase characters:
a. are different
b. are not different
c. are sometimes different

1.1.16. Which code fragment contains the error:
a. { int Numbers; numbers = 51; }
b. { int numbers; numbers = 51; }
c. { int Numbers; Numbers = 51; }

1.1.17. Which code fragment does not containtains the error:
a. { double number; number = 51.5; }
b. { double Number; number = 51.5; }
c. { double number; Number = 51.5; }

1.1.18. C++ program files have:
a. .cpp extension
b. .exe extension
c. .cs extension

1.1.19. The "char" type declares:
a. a character
b. an integer
c. a boolean variable

1.2.20. The "float" type declares:
a. single precision fraction
b. double precision fraction
c. a boolean variable

1.2.21. The "long" type declares:
a. a long integer
b. a fraction
c. a boolean variable

1.2.22. The "bool" type variable takes:
a. true and false values
b. only true values
c. only false values

1.2.23. A variable of type "int" occupies:
a. 4 bytes
b. 8 bytes
c. 1 byte

1.2.24. A variable of type "long" occupies:
a. 4 bytes
b. 1 byte
c. 8 bytes

1.2.25. A variable of type "char" occupies:
a. 1 byte
b. 2 bytes
c. 4 bytes

1.2.26. A variable of type "double" occupies:
a. 8 bytes
b. 4 bytes
c. 16 bytes

1.2.27. A variable of type "float" occupies:
a. 4 bytes
b. 16 bytes
c. 8 bytes

1.2.28. In signed numbers, the leading bit:
a. is used to indicate the sign
b. is included in the number
c. is not used to indicate the sign

1.2.29. In unsigned numbers, the leading bit:
a. is included in the number
b. is used to indicate the sign
c. is not included in the number

1.2.30. The absolute value of unsigned numbers is greater than the value of signed numbers:
a. two times
b. four times
c. three times

1.2.31. Floating point types are:
a. float and double
b. int and char
c. char and string

1.2.32. Floating point types are not:
a. char and int
b. float
c. double

1.2.33. What values does a bool type variable take:
a. true and false
b. -1 and 0
c. -1 and 1

1.2.34. What is the difference between signed and unsigned numbers:
a. In signed numbers, the leftmost bit is given to the sign of the number
b. In non-sign numbers, the leftmost bit is assigned to the sign number
c. In signed numbers, the most right bit is given to the sign of the number

Variables. Constants. Assignment Operator.

1.2.1. A variable is:
a. a named area of memory that is assigned a value
b. a constant that has a name

c. only a double data type

1.2.2. Initializing a variable is:
a. assigning a value to a variable when it is declared
b. assigning a value to a variable without declaring it
c. assigning a value to a variable after it is declared

1.2.3. Dynamic variable initialization is done:
a. anywhere in the program
b. only at the beginning of the program
c. ობლყ before the array is declared

1.2.4. Dynamic variable initialization is done:
a. during program execution
b. after completion of program execution
c. before program execution

1.2.5. What is the syntax of the assignment statement:
a. variable = expression;
b. expression = variable;
c. variable + expression;

2.2.6. What will be the value of the number2 variable after execution of the statement: int number1 = number2 = number3 = 25; :
a. 25
b. 52
c. 0

2.2.7. Is it possible to assign a value to multiple variables at once using the assignment operator:
a. Yes
b. No
c. Partially

Programming simple formulas. Priorities of arithmetic operations.

1.3.1. There are the following statements in C++ language:
a. Arithmetic, logic, comparison, and bitwise
b. Only logic and bitwise
c. Only arithmetic and comparison

1.3.2. What is an operator:
a. An operator is a symbol that tells the compiler which operation to perform
b. An operator is an array that tells the compiler which operation to perform
c. An operator is an object that tells the compiler which operation to perform

1.3.3. Arithmetic operators are used:
a. with integer and fractional numbers
b. for symbols
c. for strings

1.3.4. When we use the division operator on integers, the result will be:
a. an integer
b. a fraction
c. a symbol

1.3.5. When we apply the division operator to fractional numbers, then the result will be:
a. a fraction
b. an integer
c. a string

1.3.6. When a fraction is divided by an integer number, the result will be:
a. a fraction
b. an integer
c. a symbol

1.3.7. When an integer number is divided by a fraction, the result will be:
a. a fraction
b. an integer
c. logical value

1.3.8. Which are the arithmetic operators:
a. +, -, *, /, %, ++, --
b. <, >, !=
c. =, >, !=

1.3.9. Which operator is used to get the remainder:
a. %
b. /
c. *

1.3.10. The result of calculating the expression (2 + 3) % 2 will be:
a. 1
b. 0
c. 2

1.3.11. Which are the comparison operators:
a. <, >, !=
b. +, -, *
c. /, %, ++, --

1.3.12. Which of the * and + operators has the higher priority:

a. *

b. +

c. They have equal priority

1.3.13. Which of the - and / operators has the highest priority:

a. /

b. -

c. They have equal priority

1.3.14. {
int number1 = 5, number2 = 3, number3 = 3, number4 = 4, number5 = 1,result;
result = number1 - number2 / number3 + number4 * number5;
}
As a result of executing this code, the "result" variable will be assigned a value:

a. 8

b. 5.7

c. 3

1.3.15. { int number1 = 1, number2 = 3, number3 = 2, number4 = 5, number5 = 4, number6 = 6;
double result = (number1 + number2) / number3 + (number4 - number5) * number6; }
As a result of executing this code, the "result" variable will be assigned a value:

a. 8

b. 5

c. 3

1.3.16. { int number_1 = 1, number_2 = 3, number_3 = 2, number_4 = 5, number_5 = 4,
number_6 = 3, result;
result = (number_1 - number_2) * number_3 + (number_4 + number_5) / number_6; }
As a result of executing this code, the "result" variable will be assigned a value:

a. -1

b. 0

c. 1

1.3.17. { int number_1 = 1, number_2 = 3, number_3 = 2, number_4 = 8, number_5 = 4, result;
result = number_1 + number_2 * number_3 - number_4 / number_5; }
As a result of executing this code, the "result" variable will be assigned a value:

a. 5

b. 4

c. 6

1.3.18. { int number_1 = 1, number_2 = 2, number_3 = 8, number_4 = 2, result;
result = number_1 + number_2 * number_3 - number_4; }
As a result of executing this code, the "result" variable will be assigned a value:

a. 15

b. 16
c. 14

1.3.19. { number_1 = 1, number_2 = 16, number_3 = 8, number_4 = 2, result;
result = number_1 + number_2 / number_3 - number_4; }
As a result of executing this code, the "result" variable will be assigned a value:
a. 1
b. 2
c. 0

1.3.20. Which reasoning is correct:
a. The "/" operator has a higher priority than the "-" operator
b. The "+" operator has a higher priority than the "%" operator
c. The "-" operator has a higher priority than the "+" operator

cin and cout functions. Input-output of different types of data.

1.4.1. cin function is used:
a. to input data
b. to output data
c. Sometimes for input and sometimes for output

1.4.2. cout function is used:
a. to output data
b. to input data
c. Sometimes for input and sometimes for output

1.4.3. Which is the correct code:
a. { int number_1, number_2;
cin >> number_1 >> number_2; }
b. { int number_1, number_2;
cin << number_1 << number_2; }
c. { int number_1, number_2;
cin >> number_1 << number_2; }

1.4.4. Which is the correct code:
a. { int number_1 = 5, number_2 = 10;
cout << number_1 << number_2; }
b. { int number_1 = 5, number_2 = 10;
cout >> number_1 << number_2; }
c. { int number_1 = 5, number_2 = 10;
cout >> number_1 >> number_2; }

Topic 2. Standard Mathematical Functions of the C ++ Language

Standard mathematical functions of C ++ language: Sin(x), Cos(x), Log(x), Exp(x), Abs(x), Tan(x), Sqrt(x), Pow(x, y).

2.1.1. The floor(x) function returns:

a. the largest integral value less than or equal to the specified number
b. the smallest integral value greater than or equal to the specified number
c. the largest and smallest integral value less than or greater than the specified number

2.1.2. The ceil(x) function returns:

a. the smallest integral value greater than or equal to the specified number
b. the largest and smallest integral value less than or greater than the specified number
c. the largest integral value less than or equal to the specified number

2.1.3. The sqrt(x) function returns:

a. the square root of its argument
b. the cube root of its argument
c. the exponential value of its argument

2.1.4. y = round(5.5). After execution of this assignment, the y variable gets the value:

a. 6
b. 5
c. 4

2.1.5. The max(x,y) function returns:

a. the maximum between its two arguments
b. the square of its first argument
c. the minimum between its two arguments

2.1.6. The min(x,y) function returns:

a. the minimum between its two arguments
b. the maximum between its two arguments
c. the square of its first argument

2.1.7. The abs(x) function returns:

a. the absolute value of his argument
b. negative meaning of his argument
c. the square of its first argument

2.1.8. The pow(x,y) function returns:

a. takes the first argument to the power specified by the second argument
b. takes the second argument to the power specified by the first argument
c. returns the square of its argument

2.1.9. The log(x) function returns:

a. the logarithm of its argument
b. the negative value of its argument
c. returns the square of its argument

2.1.10. The sin(x) function returns:
a. the sine of its argument
b. the logarithm of its argument
c. the negative value of its argument

2.1.11. The cos(x) function returns:
a. the cosine of its argument
b. the sine of its argument
c. the negative value of its argument

2.1.12. The tan(x) function returns:
a. the tangent of its argument
b. the cosine of its argument
c. the sine of its argument

2.1.13. The exp(x) function:
a. raises the number "e" to the specified power
b. calculates the cosine of its argument
c. calculates the negative value of its argument

2.1.14. Which expression will calculate the value of the given formula:

$$y = \frac{\sqrt{x^3 + e^x}}{\sin x}$$

a. y = Math.Sqrt(Math.Pow(x,3) + Math.Exp(x)) / Math.Sin(x);
b. y = Math.Sqrt((Math.Pow(x,3) + Math.Exp(x)) / Math.Sin(x));
c. y = (Math.Pow(x,3) + Math.Exp(x)) / Math.Sin(x);

2.1.15. { double y = floor(5.7); } As a result of this assignment, the value of the variable "y" will be:
a. 5
b. 7
c. 6

2.1.16. { double y = floor(5.3); } As a result of this assignment, the value of the variable "y" will be:
a. 5
b. 3
c. 4

2.1.17. { double y = ceil(5.8); } As a result of this assignment, the value of the variable "y" will be:

a. 6
b. 8
c. 4

2.1.18. { double y = ceil(5.2); } As a result of this assignment, the value of the variable "y" will be:

a. 6
b. 2
c. 4

2.1.19. { double y = round(5.4); } As a result of this assignment, the value of the variable "y" will be:

a. 5
b. 6
c. 4

2.1.20. { double y = abs(5); } As a result of this assignment, the value of the variable "y" will be:

a. 5
b. 6
c. 4

2.1.21. { double y = abs(-5); } As a result of this assignment, the value of the variable "y" will be:

a. 5
b. 6
c. 4

2.1.22. { double y = pow(3,2); }As a result of this assignment, the value of the variable "y" will be:

a. 9
b. 8
c. 3

2.1.23. { double y = pow(2,3); } As a result of this assignment, the value of the variable "y" will be:

a. 8
b. 9
c. 3

Increment and decrement operations.

2.2.1. The increment operator (++):

a. increments the value of its operand by one
b. increases the value of its operand by two
c. decrements the value of its operand by one

2.2.2. The decrements operator (--):
a. decrements the value of its operand by one
b. decrements the value of its operand by two
c. increases the value of its operand by one

2.2.3. What is the function of the ++ operator:
a. increments the value of its operand by one
b. increases the value of its operand by two
c. decrements the value of its operand by one

2.2.4. What is the function of the -- operator:
a. decrements the value of its operand by one
b. decrements the value of its operand by two
c. increases the value of its operand by one

2.2.5. When the increment operator (++) is specified before the operand, then we have:
a. prefix increment
b. postfix increment
c. prefix decrement

2.2.6. When the increment operator (++) is specified after the operand, then we have:
a. postfix increment
b. prefix decrement
c. prefix increment

2.2.7. When the decrement operator (++) is specified before the operand, then we have:
a. prefix decrement
b. postfix increment
c. prefix increment

2.2.8. When the decrement operator (++) is specified after the operand, then we have:
a. postfix decrement
b. postfix increment
c. prefix increment

2.2.9. In the case of prefix increment:
a. the value of the operand is incremented first, and then applied to the expression
b. the operand is first used in the expression, and then its value is incremented
c. the value of the operand is decremented first and then applied to the expression

2.2.10. In the case of postfix increment:

a. the use of the operand in the expression will be performed first, and then its value will be increased
b. the value of the operand will be incremented first, and then it will be used in the expression
c. the value of the operand will be reduced first, and then it will be used in the expression

2.2.11. In the case of prefix decrement:
a. the value of the operand will be decremented first, and then it will be used in the expression
b. the value of the operand will be incremented first, and then it will be used in the image in the expression
c. theusese of the operand in the expression will be performed first, and then its value will be incremented

2.2.12. In the case of postfix decrement:
a. the use of the operand in the expression will be performed first, and then its value will be decremented
b. the value of the operand will be incremented first, and then its use in the expression
c. the value of the operand will be decremented first, and then its use in the expression

2.2.13. { int number = 5, result;
result = ++number; }
As a result of the execution of this code, the "result" variable will get a value:
a. 6
b. 4
c. 5

2.2.14. { int number = 5, result;
result = number++; }
As a result of the execution of this code, the "result" variable will get a value:
a. 5
a. 4
c. 6

2.2.15. { int number = 5, result;
result = --number; }
As a result of the execution of this code, the "result" variable will get a value:
a. 4
b. 5
c. 6

2.2.16. { int number = 5, result;
result = number--; }
As a result of the execution of this code, the "result" variable will get a value:
a. 5

b. 4
c. 6

2.2.17. { int number = 5, result;
result = ++number; }
As a result of the execution of this code, the "result" variable will get a value:
a. 6
b. 4
c. 5

2.2.18. { int number = 5, result;
result = number++; }
As a result of the execution of this code, the "result" variable will get a value:
a. 6
b. 4
c. 5

2.2.19. { int number = 5, result;
result = --number; }
As a result of the execution of this code, the "result" variable will get a value:
a. 4
b. 5
c. 6

2.2.20. { int number = 5, result;
result = number--; }
As a result of the execution of this code, the "result" variable will get a value:
a. 4
b. 5
c. 6

2.2.21. { int result, number_1 = 10, number_2 = 20;
result = ++number_1 - number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. -9
b. -10
c. 9

2.2.22. { int result, number_1 = 10, number_2 = 20;
int result = ++number_1 + number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. 31
b. 30
c. 32

2.2.23. { int result, number_1 = 10, number_2 = 20;
int result = ++number_1 + ++number_2; }
As a result of the execution of this code, the "result" variable will get a value:

a. 32
b. 31
c. 30

2.2.24. { int result, number_1 = 10, number_2 = 20;
result = ++number_1 + number_2++; }
As a result of the execution of this code, the "result" variable will get a value:

a. 31
b. 30
c. 32

2.2.25. { result, number_1 = 10, number_2 = 20;
int result = ++number_1 - number_2++; }
As a result of the execution of this code, the "result" variable will get a value:

a. -9
b. -10
c. 30

2.2.26. { int result, number_1 = 10, number_2 = 20;
result = ++number_1 - ++number_2; }
As a result of the execution of this code, the "result" variable will get a value:

a. -10
b. 10
c. 30

2.2.27. { int result, number_1 = 10, number_2 = 20;
result = ++number_1 - --number_2; }
As a result of the execution of this code, the "result" variable will get a value:

a. -8
b. 10
c. -10

2.2.28. { int result, number_1 = 10, number_2 = 20;
result = ++number_1 - number_2--; }
As a result of the execution of this code, the "result" variable will get a value:

a. -9
b. 30
c. 9

2.2.29. { int result, number_1 = 10, number_2 = 20;
result = ++number_1 + --number_2; }
As a result of the execution of this code, the "result" variable will get a value:

a. 30
b. 10
c. -10

2.2.30. { int result, number_1 = 10, number_2 = 20;
result = ++number_1 + number_2--; }
As a result of the execution of this code, the "result" variable will get a value:
a. 31
b. 30
c. 10

2.2.31. { int result, number_1 = 10, number_2 = 20;
result = number_1++ - ++number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. -11
b. 30
c. -10

2.2.32. { int result, number_1 = 10, number_2 = 20;
result = number_1++ - number_2++; }
As a result of the execution of this code, the "result" variable will get a value:
a. -10
b. 10
c. 30

2.2.33. { int result, number_1 = 10, number_2 = 20;
result = number_1++ + number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. 30
b. 31
c. 32

2.2.34. { int result, number_1 = 10, number_2 = 20;
result = number_1++ - number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. -10
b. 10
c. 30

2.2.35. { int result, number_1 = 10, number_2 = 20;
result = number_1++ + number_2--; }
As a result of the execution of this code, the "result" variable will get a value:
a. 30
b. 31
c. 32

2.2.36. { int result, number_1 = 10, number_2 = 20;
result = number_1++ + --number_2;/ }
As a result of the execution of this code, the "result" variable will get a value:
a. 29
b. 30
c. 28

2.2.37. { int result, number_1 = 10, number_2 = 20;
result = number_1++ + ++number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. 31
b. 30
c. 32

2.2.38. { int result, number_1 = 10, number_2 = 20;
result = number_1++ + number_2++; }
As a result of the execution of this code, the "result" variable will get a value:
a. 30
b. 31
c. 32

2.2.39. { int result, number_1 = 10, number_2 = 20;
result = number_1++ - --number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. -9
b. 10
c. -10

2.2.40. { int result, number_1 = 10, number_2 = 20;
result = number_1++ - number_2--; }
As a result of the execution of this code, the "result" variable will get a value:
a. -10
b. 10
c. 9

2.2.41. { int result, number_1 = 10, number_2 = 20;
result = --number_1 - number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. -11
b. 11
c. 30

2.2.42. { int result, number_1 = 10, number_2 = 20;
result = --number_1 + number_2; }

As a result of the execution of this code, the "result" variable will get a value:

a. 29
b. 30
c. 28

2.2.43. { int result, number_1 = 10, number_2 = 20;
result = --number_1 + ++number_2; }
As a result of the execution of this code, the "result" variable will get a value:

a. 30
b. 29
c. 31

2.2.44. { int result, number_1 = 10, number_2 = 20;
result = --number_1 + number_2++; }
As a result of the execution of this code, the "result" variable will get a value:

a. 29
b. 28
c. 30

2.2.45. { int result, number_1 = 10, number_2 = 20;
result = --number_1 - number_2++; }
As a result of the execution of this code, the "result" variable will get a value:

a. -11
b. 11
c. 30

2.2.46. { int result, number_1 = 10, number_2 = 20;
result = --number_1 - ++number_2; }
As a result of the execution of this code, the "result" variable will get a value:

a. -12
b. 30
c. 12

2.2.47. { int result, number_1 = 10, number_2 = 20;
result = --number_1 - --number_2; }
As a result of the execution of this code, the "result" variable will get a value:

a. -10
b. 10
c. 30

2.2.48. { int result, number_1 = 10, number_2 = 20;
result = --number_1 - number_2--; }
As a result of the execution of this code, the "result" variable will get a value:

a. -11
b. 11

c. 30

2.2.49. { int result, number_1 = 10, number_2 = 20;
result = --number_1 + --number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. 28
b. 29
c. 30

2.2.50. { int result, number_1 = 10, number_2 = 20;
result = --number_1 + number_2--; }
As a result of the execution of this code, the "result" variable will get a value:
a. 29
b. 30
c. 28

2.2.51. { int result, number_1 = 10, number_2 = 20;
result = number_1-- - ++number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. -11
b. 11
c. 30

2.2.52. { int result, number_1 = 10, number_2 = 20;
result = number_1-- - number_2++; }
As a result of the execution of this code, the "result" variable will get a value:
a. -10
b. 30
c. 10

2.2.53. { int result, number_1 = 10, number_2 = 20;
result = number_1-- + number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. 30
b. 31
c. 29

2.2.54. { int result, number_1 = 10, number_2 = 20;
result = number_1-- - number_2; }
As a result of the execution of this code, the "result" variable will get a value:
a. -10
b. 10
c. 30

2.2.55. { int result, number_1 = 10, number_2 = 20;

result = number_1-- + number_2--; }

As a result of the execution of this code, the "result" variable will get a value:

a. 30
b. -10
c. 10

2.2.56. { int result, number_1 = 10, number_2 = 20;
result = number_1-- + --number_2; }

As a result of the execution of this code, the "result" variable will get a value:

a. 29
b. 28
c. 30

2.2.57. { int result, number_1 = 10, number_2 = 20;
result = number_1-- + ++number_2; }

As a result of the execution of this code, the "result" variable will get a value:

a. 31
b. 30
c. -10

2.2.58. { int result, number_1 = 10, number_2 = 20;
result = number_1-- + number_2++; }

As a result of the execution of this code, the "result" variable will get a value:

a. 30
b. -10
c. 10

2.2.59. { int result, number_1 = 10, number_2 = 20;
result = number_1-- - --number_2; }

As a result of the execution of this code, the "result" variable will get a value:

a. -9
b. -10
c. 10

2.2.60. { int result, number_1 = 10, number_2 = 20;
result = number_1-- - number_2--; }

As a result of the execution of this code, the "result" variable will get a value:

a. -10
b. 9
c. 10

Topic 3. Logic and comparison. Bitwise operations

Comparison statements (>, <, ==, >=, <=, !=).

3.1.1. Comparison operators return values of type:

a. bool
b. char
c. int

3.1.2. Values of type bool can be compared:

a. for equality or inequality
b. for greater than or less than
c. for equality or greater than

3.1.3.
```
{       int number1 = 5, number2 = 10;
        bool result;
        result = number1 > number2;
}
```
As a result of executing this code, the "result" variable will be assigned a value:

a. false
b. true
c. 5

3.1.4.
```
{
        int number1 = 5, number2 = 10;
        bool result;
        result = number1 < number2;
}
```
As a result of executing this code, the "result" variable will be assigned a value:

a. true
b. false
c. 5

3.1.5.
```
{
        int number1 = 5, number2 = 10;
        bool result;
        result = number1 == number2;
}
```
As a result of executing this code, the "result" variable will be assigned a value:

a. false
b. 5
c. true

3.1.6.
```
{
        int number1 = 5, number2 = 10;
        bool result;
```

```
        result = number1 != number2;
    }
```

As a result of executing this code, the "result" variable will be assigned a value:

a. true
b. 5
c. false

3.1.7.
```
{
        int number1 = 5, number2 = 10;
        bool result;
        result = number1 >= number2;
    }
```

As a result of executing this code, the "result" variable will be assigned a value:

a. false
b. 5
c. true

3.1.8.
```
{
        int number1 = 5, number2 = 10;
        bool result;
        result = number1 <= number2;
    }
```

As a result of executing this code, the "result" variable will be assigned a value:

a. true
b. 5
c. false

3.1.9. What is the result of executing comparison operators:

a. true or false
b. -1 or 0
c. -1 or 1

3.1.10. Which are the comparison operators:

a. ==, !=, >, <, <=, >=
b. +, -, *
c. /, %, ++

Logic statements (||, &&, !). Their priorities.

3.2.1. What type should the operands of logic operators have:

a. bool
b. int
c. short

3.2.2. Which are the logic operators:

a. &, |, ^, &&, ||, !
b. ==, !=, >
c. -, /, *

3.2.3. The & operator returns true when:
a. both of its operands are true
b. one of its operands is false
c. both of its operands are false

3.2.4. The | operator returns true when:
a. one of its operands is true
b. both of its operands are false
c. both of its operands are 0

3.2.5. The ! operator returns true when:
a. its operand is false
b. both of its operands are false
c. its operand is true

3.2.6. The ^ operator returns true when:
a. both of its operands have different values
b. both of its operands have the same value
c. one of its operands is true

3.2.7. The & operator returns false when:
a. one of its operands is false
b. both of its operands are true
c. one of its operands is true

3.2.8. The | operator returns false when:
a. both of its operands are false
b. one of its operands is true
c. one of its operands is false

3.2.9. The ! operator returns false when:
a. its operand is true
b. its operand is false
c. both of its operands are false

3.2.10. The ^ operator returns false when:
a. both of its operands have the same value
b. both of its operands have different values
c. one of its operands is true

3.2.11. {

```
        bool b1 = true, b2 = false, result;
        result = b1 & b2;
    }
```

As a result of executing this code, the "result" variable will be assigned a value:

a. false
b. true
c. 5

3.2.12.
```
    {
        bool b1 = true, b2 = false, result;
        result = b1 | b2;
    }
```

As a result of executing this code, the "result" variable will be assigned a value:

a. true
b. false
c. 5

3.2.13.
```
    {
        bool b1 = true, b2 = false, result;
        result = b1 ^ b2;
    }
```

As a result of executing this code, the "result" variable will be assigned a value:

a. true
b. false
c. 5

3.2.14.
```
    {
        bool b1 = true, result;
        result = !b1;
    }
```

As a result of executing this code, the "result" variable will be assigned a value:

a. false
b. 5
c. true

3.2.15.
```
    {
        bool b1 = true, b2 = false, b3 = true, b4 = false, result;
        result = !b1 || b2 && !b3 || b4;
    }
```

As a result of executing this code, the "result" variable will be assigned a value:

a. false
b. 5
c. true

3.2.16.
```
    {
```

```
        bool b1 = true, b2 = false, b3 = true, result;
        result = b1 || b2 && b3;
    }
```

As a result of executing this code, the "result" variable will be assigned a value:

a. true
b. 5
c. false

3.2.17. Which is the fast logic operator:

a. &&
b. &
c. ||

3.2.18. The fast logic operator is:

a. ||
b. !
c. &

3.2.19. If the first operand of the && operator is false, then the result will be:

a. false
b. true
c. 5

3.2.20. If the first operand of the || operator is true, then the result will be:

a. true
b. false
c. 5

Bitwise statements (&, |, ^, ~). Shift statements (<<, >>)

3.3.1. The operands of bitwise operators are:

a. integers
b. strings
c. fractional numbers

3.3.2. Which operations are called bitwise:

a. Bitwise operations are used to check, set, and shift bits of integers
b. Operations used to add fractional numbers are called bitwise operations
c. Operations used to subtract integers are called bitwise operations

3.3.3. Which operators are bitwise:

a. &, |, ^, ~
b. +, -, *
c. &, |, %, /

3.3.4. { int b1 = 10, b2 = 12, result;
result = b1 & b2; }
As a result of executing this code, the "result" variable will be assigned:
a. 8
b. 22
c. 10

3.3.5. { int b1 = 10, b2 = 12, result;
result = b1 | b2; }
As a result of executing this code, the "result" variable will be assigned:
a. 14
b. 22
c. 12

3.3.6. If both operands of the & bitwise operator are 1, then the result is:
a. 1
b. 0
c. -1

3.3.7. If both operands of the | bitwise operator are 1, then the result is:
a. 1
b. 0
c. -1

3.3.8. If one of the operands of the & bitwise operator is 0, then the result will be:
a. 0
b. -1
c. 1

3.3.9. If one of the operands of the | bitwise operator is 1, then the result will be:
a. 1
b. 0
c. -1

3.3.10. If the operand of the ~ bitwise operator is 0, then the result will be:
a. 1
b. 0
c. -1

3.3.11. If the operand of the ~ bitwise operator is 1, then the result will be:
a. -1
b. 0
c. 1

3.3.12. If both operands of the ^ operator are the same, then the result is:

a. 0
ð 1
c. -1

3.3.13. If the operands of the ^ operator are different, then the result is:
a. 1
b. 0
c. -1

3.3.14. When the number is shifted to the left:
a. the leading (left) digits are lost
b. the lower (right) digits are lost
c. No digits are lost

3.3.15. When the number is shifted to the right:
a. the lower (right) digits are lost
b. the leading (left) digits are lost
c. No digits are lost

3.3.16. When the number is shifted to the right:
a. The leading (left) digits are filled with zeros
b. The leading (left) digits are filled with ones
c. The lower digits (right) are filled with zeros

3.3.17. When the number is shifted to the left:
a. Lower digits (right) are filled with zeros
b. Lower digits (right) are filled with ones
c. The leading digits are filled with zeros

3.3.18. When the number is shifted to the right:
a. The sign digit is saved
b. The sign digit is filled with one
c. The sign digit is filled with zero

3.3.19. { int result = 0, number_1 = 5;
result = number_1 << 1; } As a result of executing this code, the "result" variable is assigned a value:
a. 10
b. 5
c. 0

3.3.20. { int result = 0, number_1 = 5;
result = number_1 >> 1; } As a result of executing this code, the "result" variable is assigned a value:
a. 2

b. 1
c. 5

3.3.21. { int result = 0, number_1 = 7;
result = number_1 >> 1; } As a result of executing this code, the "result" variable is assigned a value:
a. 3
b. 7
c. 1

3.3.22. { int result = 0, number_1 = 7;
result = number_1 << 1; } As a result of executing this code, the "result" variable is assigned a value:
a. 14
b. 7
c. 1

Topic 4. Simple and complex branched structure algorithms and their programming. Program block. Ternary statement if…else statement. Embedded if… else statement.

4.1.1. What values does the condition in the "if" statement return:

a. true or false
b. bool or 0
c. -1 or 1

4.1.2. { int number = 5, result;
if (number == 5) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 10
b. 25
c. 15

4.1.3. { int result, number_1 = 5;
if (number_1 != 5) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 25
b. 10
c. 15

4.1.4. { int result, number_1 = 7;
if (number_1 != 5) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 10
b. 25
c. 15

4.1.5. { int result, number_1 = 5;
if (number_1 == 8) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 25
b. 10
c. 15

4.1.6. { int result, number_1 = 5;
if (number_1 >= 5) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 10
b. 25
c. 15

4.1.7. { int result, number_1 = 5;

if (number_1 >= 8) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 25
b. 10
c. 15

4.1.8. { int result, number_1 = 5;
if (number_1 <= 5) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 10
b. 25
c. 15

4.1.9. { int result, number_1 = 15;
if (number_1 <= 9) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 25
b. 10
c. 15

4.1.10. { int result, number_1 = 5;
if (number_1 < 5) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 25
b. 10
c. 15

4.1.11. { int result, number_1 = 5;
if (number_1 < 6) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 10
b. 25
c. 15

4.1.12. { int result, number_1 = 5;
if (number_1 > 5) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 25
b. 10
c. 15

4.1.13. { int result, number_1 = 5;
if (number_1 > 3) result = 10;
else result = 25; } As a result of executing this code, the "result" variable is assigned a value:

a. 10

b. 25
c. 15

4.1.14. The condition of the "if" statement is the expression that returns:
a. a logical type
b. an integer type
c. char type

4.1.15. A nested is an "if" statement:
a. that is placed inside another "if" statement
b. that is not placed inside another "if" statement
c. that is placed inside another "for" statement

4.1.16. The "else" part of a nested "if" statement:
a. belongs to the nearest "if" statement
b. does not belongs to the nearest "if" statement
c. belongs to the nearest "for" statement

4.1.17. { int number1 = 5, number2 = 10, result;
if (number1 > 5) result = 50;
else if (number2 <= 10) result = 100; }
ɔð As a result of executing this code, the "result" variable is assigned a value:
a. 100
b. 50
c. 150

4.1.18. { int result, number_1 = 5, number_2 = 10;
if (number_1 > 5) result = 50;
else if (number_2 <= 5) result = 100;
else result = 0; } As a result of executing this code, the "result" variable is assigned a value:
a. 0
b. 100
c. 10

4.1.19. { int result, number_1 = 5, number_2 = 10;
if (number_1 > 4) result = 50;
else if (number_2 < 5) result = 100;
else result = 0; } As a result of executing this code, the "result" variable is assigned a value:
a. 50
b. 100
c. 10

4.1.20. { int result, number_1 = 5, number_2 = 10;
if (number_1 > 4) result = 50;

else if (number_2 >= 15) result = 100;
else result = 0; } As a result of executing this code, the "result" variable is assigned a value:

a. 50
b. 100
c. 10

4.1.21. { int result, number_1 = 5, number_2 = 10;
if (number_1 != 5) result = 50;
else if (number_2 < 5) result = 100;
else result = 0; } As a result of executing this code, the "result" variable is assigned a value:

a. 0
b. 100
c. 10

4.1.22. { int result, number_1 = 5, number_2 = 10;
if (number_1 != 5) result = 50;
else if (number_2 < 15) result = 100;
else result = 0; } As a result of executing this code, the "result" variable is assigned a value:

a. 100
b. 50
c. 10

4.1.23. { int result, number_1 = 5, number_2 = 10;
if (number_1 == 5) result = 50;
else if (number_2 != 15) result = 100;
else result = 0; } As a result of executing this code, the "result" variable is assigned a value:

a. 50
b. 100
c. 10

4.1.24. { int result, number_1 = 5, number_2 = 10;
if (number_1 >= 6) result = 50;
else if (number_2 != 15) result = 100;
else result = 0; } As a result of executing this code, the "result" variable is assigned a value:

a. 100
b. 50
c. 10

4.1.25. { int result, number_1 = 5, number_2 = 10;
if (number_1 < 5) result = 50;
else if (number_2 >= 10) result = 100;

else result = 0; } As a result of executing this code, the "result" variable is assigned a value:

a. 100
b. 50
c. 10

Multi-branch statement - switch-case.

4.2.1. What type can the "switch" statement expression have:

a. char, byte, int
b. char, short, double
c. float, byte, int

4.2.2. The constants of the "case" branches of the "switch" statement:

a. must have a type compatible with the expression type
b. must not have a type compatible with the expression type
c. must sometimes have a type compatible with the expression type, sometimes not

4.2.3. The constants of the "case" branches of the "switch" statement can have the same value:

a. It's not allowed
b. It's allowed
c. Sometimes it is allowed, sometimes not

4.2.4. When the operators of the "default" branch of the "switch" statement will be executed:

a. When none of the constants of the "case" branches match the value of the "switch" expression
b. When one of the constants of the "case" branch match to the value of the "switch" expression
c. When all the constants of the "case" branches match to the value of the "switch" expression

4.2.5. The "break" operator of the "switch" statement transfers control:

a. to the statement placed after the "switch" statement
b. to the next "case" branch
c. to the "default" branch

4.2.6. Is it necessary to specify the "default" branch in the "switch" statement:

a. Yes
b. No
c. Partially

4.2.7. What is the function of the "break" operator in the "switch" statement:

a. It transfers control to the statement placed after the "switch" statement
b. It transfers control to the statement in the middle of the "switch" statement
c. It transfers control to the statement placed before the "switch" statement

4.2.8. { int number_1 = 2, result;

switch (number_1)
{
case 1: result = 10; break;
case 2: result = 20; break;
case 3: result = 30; break;
} } As a result of executing this code, the "result" variable is assigned a value:

a. 20
b. 10
c. 30

4.2.9. { int number_1 = 2, result;
switch (number_1++)
{
case 1: result = 10; break;
case 2: result = 20; break;
case 3: result = 30; break;
} } As a result of executing this code, the "result" variable is assigned a value:

a. 20
b. 30
c. 10

4.2.10. { int number_1 = 2, result;
switch (++number_1)
{
case 1: result = 10; break;
case 2: result = 20; break;
case 3: result = 30; break;
} } As a result of executing this code, the "result" variable is assigned a value:

a. 30
b. 10
c. 20

4.2.11. { int number_1 = 2, result;
switch (number_1--)
{
case 1: result = 10; break;
case 2: result = 20; break;
case 3: result = 30; break;
} } As a result of executing this code, the "result" variable is assigned a value:

a. 20
b. 30
c. 10

4.2.12. { int number_1 = 2, result;
switch (--number_1)

{
case 1: result = 10; break;
case 2: result = 20; break;
case 3: result = 30; break;
} } As a result of executing this code, the "result" variable is assigned a value:

a. 10
b. 30
c. 20

Program block. Program block. Visibility area. Local and global variables.

4.3.1. A local is a variable declared:

a. inside a function
b. outside a function
c. only in an if statement

4.3.2. A global is a variable declared:

a. outside a function
b. inside a function
c. in a for statement

4.3.3. A variable declared inside the scope of visibility:

a. is inaccessible in a code defined outside that scope
b. is accessible in a code defined outside that scope
c. is accessible only in "if" statement

4.3.4. The program block is placed:

a. in braces
b. in parenthesis
c. in brackets

4.3.5. Several statements should be placed in one block when:

a. it is necessary to establish a logical relationship between them
b. it is necessary to establish a physical relationship between them
c. it is necessary to establish a digital relationship between them

4.3.6. If the variable is declared at the beginning of the function, then:

a. it will be possible to access it within the entire code
b. it will be possible to access it only within the "if" statement
c. it will be possible to access it before it declaration

4.3.7. A variable defined in the visibility area when exiting the visibility area:

a. loses its value
b. keeps its value
c. doubles its value

4.3.8. A lifetime of a variable:
a. is limited by its visibility area
b. is not limited by its visibility area
c. is limited by the bounds of the array

4.3.9. A visibility area:
a. can be nested
b. cannot be nested
c. is limited by the bounds of the array

4.3.10. Objects and variables declared in the outer visibility area:
a. will be visible in the inner visibility area
b. will not be visible in the inner visibility area
c. is limited by the bounds of the array

4.3.11. Objects and variables declared in the inner visibility area:
a. will not be visible in the outer visibility area
b. will be visible in the outer visibility area
c. is limited by the bounds of the array

4.3.12. A global variable:
a. is valid inside all functions
b. is not valid inside all functions
c. s sometimes valid sometimes not inside a function

4.3.13. ფუნქციას პარამეტრები ჩართული არის თუ არა ამავე ფუნქციის ხილვადობის უბანში:
a. No
b. Yes
c. Partially

4.3.14. Whether a variable declared in a visibility area is visible or invisible in code defined outside that area:
a. It is invisible
b. It is visible
c. It is partially visible

4.3.15. Whether a new visibility area is created when a block of code is created:
a. Yes
b. No
c. Partially

4.3.16. Whether a variable can be accessed before it is declared:
a. No

b. Yes
c. Sometimes

4.3.17. Whether a variable retains its value when it exits the scope:
a. No
b. Yes
c. Sometimes

4.3.18. Whether a variable declared inside a function retains its value between calls to that function:
a. No
b. Yes
c. Partially

Ternary statement.

4.4.1. How many operands does the ternary statement have:
a. Three
b. One
c. Two

4.4.2. The first operand of the ternary statement returns:
a. logical value
b. integer value
c. a string

4.4.3. If the condition part of the ternary statement returns true, then:
a. The value of the second operand is returned
b. The value of the third operand is returned
c. Nothing is returned

4.4.4. If the condition part of the ternary statement returns false, then:
a. The value of the third operand is returned
b. The value of the second operand is returned
c. Nothing is returned

4.4.5. { int result, number1 = 10, number2 = 20;
result = number1 > number2 ? number1 : number2; } As a result of executing this code, the "result" variable is assigned a value:
a. 20
b. 10
c. 0

4.4.6. { int result, number1 = 10, number2 = 20;
result = number1 < number2 ? number1 : number2; } As a result of executing this code,

the "result" variable is assigned a value:
a. 10
b. 20
c. 0

4.4.7. { int result, number1 = 10, number2 = 20;
result = number1 >= number2 ? number1 : number2; } As a result of executing this code, the "result" variable is assigned a value:
a. 20
b. 10
c. 0

4.4.8. { int result, number1 = 10, number2 = 20;
result = number1 <= number2 ? number1 : number2; } As a result of executing this code, the "result" variable is assigned a value:
a. 10
b. 20
c. 0

4.4.9. { int result, number1 = 10, number2 = 20;
result = number1 == number2 ? number1 : number2; } As a result of executing this code, the "result" variable is assigned a value:
a. 20
b. 10
c. 0

4.4.10. { int result, number1 = 10, number2 = 20;
result = number1 != number2 ? number1 : number2; } As a result of executing this code, the "result" variable is assigned a value:
a. 10
b. 20
c. 0

Topic 5. The loop. The for, while and do-while statements. The break and continue statements. Compound statements

The for statement.

5.1.1. What is the "for" operator used for:

a. To execute a single statement or block of statement multiple times
b. for branch organization
c. To break program execution

5.1.2. What is the purpose of the "initialize" section of the "for" statement:

a. In it, the control variable is assigned an initial value
b. In it, the control variable is not assigned an initial value
c. "Condition" section is checked

5.1.3. What is the purpose of the "condition" section of the "for" statement:

a. The expression specified in it is checked. If it returns "true", then the loop body is executed, otherwise, the loop interrupts
b. The expression specified in it is checked. If it returns "false", then the loop body is executed, otherwise, the loop interrupts
c. In it, the control variable is assigned an initial value

5.1.4. What is the purpose of the "iteration" section of the "for" statement:

a. This is an expression that defines the amount (step) by which the value of the control variable changes before each iteration of the loop
b. In it, the control variable is assigned an initial value
c. The expression specified in it is checked. If it returns "true", then the loop body is executed, otherwise, the loop interrupts

5.1.5. The initializer of the "for" operator is an expression that:

a. Is calculated before the beginning of the loop
b. Is calculated when the loop interrupts
c. It will never be calculated

5.1.6. In the initializer of the "for" statement:

a. the control variable of the loop is initialized
b. a global variable is declared
c. a static variable is initialized

5.1.7. The condition of the "for" statement is a logical expression that returns:

a. "true" or "false" value
b. "true" or "int" value
c. "double" or "false" value

5.1.8. The condition of the "for" statement is calculated:

a. before each iteration of the loop

b. at the beginning of the program
c. after each iteration of the loop

5.1.9. The "for" loop repeats as long as the condition evaluates to:
a. true
b. false
c. int

5.1.10. The "for"loop interrupts, when the condition returns:
a. false
b. int
c. true

5.1.11. The iterator of the "for" loop is an expression that:
a. changes the value of a control variable
b. not changes the value of a control variable
c. only decrements the value of the control variable

5.1.12. Whether or not the control variable of the loop can decrement its value:
a. Yes
b. No
c. Partially

5.1.13. The body of the "for" loop:
a. may not be performed
b. It must be performed
c. Must be performed twice

5.1.14. To specify an initializer in the "for" statement:
a. it is not necessary
b. is required when declaring an array
c. it is necessary

5.1.15. Using an iterator in the "for" statement:
a. it is not necessary
b. is required when declaring an array
c. it is necessary

5.1.16. Using condition in the "for" statement:
a. it is not necessary
b. is required when declaring an array
c. it is necessary

5.1.17. Using a body in the "for" statement:
a. it is not necessary

b. is required when declaring an array
c. it is necessary

5.1.18. In "for" statement it is allowed:
a. to specify several control variables
b. to specify only one control variable
c. to specify only two control variables

5.1.19. Which reasoning is correct:
a. To interrupt the "for" loop, we can use the "break" statement
b. To interrupt the "for" loop, we can not use the "break" statement
c. To interrupt the "for" loop, we can use the "case" statement

5.1.20. Whether it is allowed to declare a control variable in the initializer of the "for" statement:
a. Yes
b. Partially
c. No

5.1.21. A variable declared in the initializer of the "for" statement:
a. can only be used in the body of this loop
b. can be used outside the body of this loop
c. can be used inside any function

5.1.22. If a single statement is specified in the loop body, then:
a. curly braces are not required
b. the parentheses must be specified
c. curly braces are required

5.1.23. { int number_1, result = 0;
for (number_1 = 1; number_1 <= 3; number_1++)
result += number_1; } As a result of executing this code, the "result" variable will be assigned:
a. 6
b. 1
c. 2

5.1.24. { int number_1, result = 0;
for (number_1 = 1; number_1 <= 3; number_1++)
result += number_1; } As a result of executing this code, the "result" variable will be assigned:
a. 3
b. 1
c. 2

5.1.25. { int number_1, result = 0;

for (number_1 = 1; number_1 <= 5; number_1 += 2)
result += number_1; } As a result of executing this code, the "result" variable will be assigned:

a. 9
b. 6
c. 2

5.1.26. { int number_1, result = 0;
for (number_1 = 0; number_1 <= 5; number_1 += 2)
result += number_1; } As a result of executing this code, the "result" variable will be assigned:

a. 6
b. 9
c. 2

5.1.27. { int number_1, result = 0;
for (number_1 = 3; number_1 >= 1; number_1--)
result += number_1; } As a result of executing this code, the "result" variable will be assigned:

a. 6
b. 1
c. 2

5.1.28. { int number_1, result = 0;
for (number_1 = 3; number_1 >= 0; number_1--)
result += number_1; } As a result of executing this code, the "result" variable will be assigned:

a. 6
b. 1
c. 2

5.1.29. { int number_1, result = 0;
for (number_1 = 5; number_1 >= 1; number_1 -= 2)
result += number_1; } As a result of executing this code, the "result" variable will be assigned:

a. 9
b. 1
c. 3

5.1.30. { int number_1, result = 0;
for (number_1 = 5; number_1 >= 0; number_1 -= 2)
result += number_1; } As a result of executing this code, the "result" variable will be assigned:

a. 9
b. 1

c. 3

5.1.31. { int number_1 = 1, result = 0;
for (; number_1 <= 3; number_1++)
result += number_1; } As a result of executing this code, the "result" variable will be assigned:

a. 6
b. 1
c. 3

5.1.32. { int number_1, result = 0;
for (number_1 = 1; number_1 <= 3;)
result += number_1++; } As a result of executing this code, the "result" variable will be assigned:

a. 6
b. 1
c. 4

5.1.33. { int number_1 = 3, result = 0;
for (; number_1 >= 1; number_1--)
result += number_1; } As a result of executing this code, the "result" variable will be assigned:

a. 6
b. 8
c. 3

5.1.34. { int number_1, result = 0;
for (number_1 = 3; number_1 >= 1;)
result += number_1--; } As a result of executing this code, the "result" variable will be assigned:

a. 6
b. 1
c. 3

The break and continue statements.

5.2.1. What is the "break" statement used for:

a. It allows us to interrupt and exit the loop
b. It allows us to continue and exit the loop
c. It not allows us to interrupt and exit the loop

5.2.2. Can a "break" statement be used inside a "while" loop:

a. Yes
b. No
c. Partially

5.2.3. Can a "break" statement be used inside a "do-while" loop:
a. Yes
b. No
c. Partially

5.2.4. "break" statement can be used:
a. within any loop
b. only inside the "for" loop
c. only inside the "while" loop

5.2.5. "break" statement is used:
a. To interrupt the execution of the loop
b. To continue the execution of the loop
c. To start the execution of the loop

5.2.6. As a result of the execution of the "break" statement:
a. The remaining statements in the loop will not be executed.
b. The remaining statements in the loop will be executed.
c. The remaining loop statements will sometimes be executed, sometimes not.

5.2.7. If the "break" statement is used inside a nested loop, then it interrupts:
a. the execution of the inner loop
b. the execution of the outer loop
c. of both loops

5.2.8. { int result = 0, number_1;
for (number_1 = 1; number_1 < 3; number_1++)
{
result += number_1;
if (number_1 == 2) break;
} } As a result of executing this code, the "result" variable is assigned a value:
a. 3
b. 2
c. 1

5.2.9. { int result = 0, number_1;
for (number_1 = 3; number_1 > 1; number_1--)
{
result += number_1;
if (number_1 == 2) break;
} } As a result of executing this code, the "result" variable is assigned a value:
a. 5
b. 2
c. 1

5.2.10. What is the "continue" statement used for:
a. It is used to force a transition to the next iteration of the loop
b. It is used to interrupt the loop
c. It is used to interrupt the loop

5.2.11. Can the "continue" statement be used in the "while" statement:
a. Yes
b. No
c. Partially

5.2.12. Can the "continue" statement be used in the "do-while" statement:
a. Yes
b. No
c. Partially

5.2.13. As a result of the execution of the "continue" statement:
a. The remaining statements in the loop will not be executed
b. The remaining statements in the loop will be executed
c. The remaining loop statements will sometimes be executed, sometimes not

5.2.14. The "continue" statement performs:
a. a forced transition to the next iteration
b. a forced interruption of the loop
c. the transition to the next statement

5.2.15. "continue" statement can be used:
a. within any loop
b. just inside the "for" loop
c. Just inside the "while" loop

5.2.16. int result = 0, number_1;
for (number_1 = 1; number_1 < 3; number_1++)
{
result += number_1;
if (number_1 != 2) continue;
else break;
} } As a result of executing this code, the "result" variable is assigned a value:
a. 3
b. 2
c. 1

5.2.17. int result = 0, number_1;
for (number_1 = 3; number_1 > 1; number_1--)
{

```
        result += number_1;
        if (number_1 != 2) continue;
          else break;
      } } As a result of executing this code, the "result" variable is assigned a value:
```

a. 5
b. 2
c. 1

The while and do-while statements.

5.3.1. The condition of the "while" statement is a boolean expression that returns:

a. "true" or "false" value
b. "true" or "int" value
c. "double" or "int" value

5.3.2. The "while" loop is repeated while the condition returns:

a. "true" value
b. "false" value
c. "int" value

5.3.3. The "while" loop interrupts when the condition returns:

a. "false" value
b. "int" value
c. "true" value

5.3.4. The body of the "while" loop:

a. may not be executed
b. must be executed
c. must be executed once

5.3.5.
```
{ int number_1 = 1, result = 0;
  while (number_1 <= 3)
  {
    result += number_1;
    number_1++;
  } } As a result of executing this code, the "result" variable is assigned a value:
```

a. 6
b. 3
c. 2

5.3.6.
```
{ int number_1 = 3, result = 0;
  while (number_1 > 1)
  {
    result += number_1;
    number_1--;
```

} } As a result of executing this code, the "result" variable is assigned a value:

a. 5
b. 3
c. 2

5.3.7. The condition of the "do-while" statement is a boolean expression that returns:

a. "true" or "false" value
b. "true" or "int" value
c. "double" or "false" value

5.3.8. The do-while loop repeats while the condition returns:

a. "true" value
b. "int" value
c. "false" value

5.3.9. The "do-while" loop interrupts when the condition returns:

a. "false" value
b. "int" value
c. "true" value

5.3.10. The body of the "do-while" statement:

a. must be executed
b. may not be executed
c. will never be executed

5.3.11. { int number_1 = 3, result = 0;
do
{
result += number_1;
number_1--;
} while (number_1 > 1); } As a result of executing this code, the "result" variable is assigned a value:

a. 5
b. 3
c. 2

5.3.12. { int number_1 = 1, result = 0;
do
{
result += number_1;
number_1++;
} while (number_1 < 3); } As a result of executing this code, the "result" variable is assigned a value:

a. 3
b. 5

c. 2

Compound statements: +=, -=, /=, * =, %=, &=, |=, ^=.

5.4.1. What are the compound statements:

a. +=, -+, *=, /=, %=, &=, |=, ^=

b. !==, <==

c. ==, >==

5.4.2. What is the syntax of the compound operator:

a. variable **operation=** expression;

b. variable **=operation** expression;

c. variable **operation** expression;

5.4.3. What is the advantage of using the compound assignment operator over using the regular assignment operator:

a. The compound assignment operator is more compact and its use speeds up code compilation

b. The compound assignment operator is less compact and its use does not speed up code compilation

c. It has no advantage

5.4.4. sum += 25; This assignment to which of the following statements is analogous:

a. sum = sum + 25;

b. sum = sum - 25;

c. sum = sum + sum + 25;

5.4.5. sum -= 25; This assignment to which of the following statements is analogous:

a. sum = sum - 25;

b. sum = sum - sum - 25;

c. sum = 25 - sum;

5.4.6. sum *= 25; This assignment to which of the following statements is analogous:

a. sum = sum * 25;

b. sum = sum - 25;

c. sum = sum * sum * 25;

5.4.7. sum /= 25; This assignment to which of the following statements is analogous:

a. sum = sum / 25;

b. sum = sum / sum / 25;

c. sum = sum % 25;

5.4.8. sum %= 25; This assignment to which of the following statements is analogous:

a. sum = sum % 25;

b. sum = sum % sum % 25;

c. sum = sum / 25;

Topic 6. Arrays. One-dimensional arrays

One-dimensional array

6.1.1. The array is:
a. a collection of data of one type
b. a constant
c. a symbol

6.1.2. Array is used:
a. to store several variables or objects
b. to store only one variable or object
c. it is not used to store anything

6.1.3. To refer to the elements of the array, we must specify:
a. the name of the array and the index of the element
b. the index of the element
c. the name of the array

6.1.4. The indexing of the elements of the one-dimensional array begins:
a. from zero
b. from one
c. from two

6.1.5. The location of a particular element in a one-dimensional array is determined:
a. by a single index
b. by two indices
c. by three indices

6.1.6. How many bytes in memory occupy the array - int array[15]; :
a. 60
b. 15
c. 30

6.1.7. How many bytes in memory occupy the array - double array[10]; :
a. 80
b. 10
c. 40

6.1.8. How many bytes in memory occupy the array - char array[10]; :
a. 10
b. 11
c. 9

6.1.9. The index of the first element of the array is:
a. 0

b. 1
c. -1

6.1.10. The index of the third element of the array is:
a. 2
b. 3
c. -2

6.1.11. If the array consists of 5 elements, then its indices will be placed in the range:
a. 0 - 4
b. 1 - 5
c. 0 - 5

6.1.12. array[1] refers to:
a. the second element in the array
b. the first element in the array
c. the third element in the array

6.1.13. Elements of array - double array[3]; have:
a. fractional type
b. integer type
c. logical type

6.1.14. If the value of the array index exceeds the allowed limits, then:
a. the program interrupts execution
b. the program continues execution
c. the program will not interrupt and will not continue to work

6.1.15. { int result = 0;
int array[] = { 1, 2, 3 };
for (int index = 0; index < 3; index++)
result += array[index]; } As a result of executing this code, the "result" variable is assigned a value:
a. 6
b. 3
c. 1

6.1.16. { int result = 0;
int array[] = { 1, 2, 3, 4, 5 };
result = array[0] + array[2]; } As a result of executing this code, the "result" variable is assigned a value:
a. 4
b. 3
c. 1

6.1.17. { int result = 0;
int array[] = { 1, 2, 3, 4, 5 };
result = array[1] + array[3]; } As a result of executing this code, the "result" variable is assigned a value:

a. 6
b. 3
c. 1

6.1.18. { int result = 0;
int array[] = { 1, 2, 3, 4, 5 };
result = array[0] + array[4]; } As a result of executing this code, the "result" variable is assigned a value:

a. 6
b. 3
c. 1

Topic 7. Two and multidimensional arrays

Two and multidimensional array

7.1.1. How many bytes occupy the array - int array[4][5]; :

a. 80
b. 4
c. 5

7.1.2. How many bytes occupy the array - double array[4][5]; :

a. 160
b. 20
c. 80

7.1.3. How many bytes occupy the array - char array[4][5]; :

a. 20
b. 5
c. 4

7.1.4. When declared, the first index in a two-dimensional array:

a. indicates the number of rows
b. indicates the number of columns
c. indicates the sum of the rows and columns

7.1.5. When declared, the second index in a two-dimensional array:

a. indicates the number of columns
b. indicates the number of rows
c. indicates the sum of the rows and columns

7.1.6. The element at the intersection of the second row and the third column of the two-dimensional array is denoted as:

a. array[1,2]
b. array[3,2]
c. array[2,3]

7.1.7. The element at the intersection of the third row and the fourth column of the two-dimensional array is denoted as:

a. array[4,5]
b. array[3,4]
c. array[4,3]

7.1.8. x = array[2,4]; As a result of executing this assignment, the variable x is assigned:

a. the element at the intersection of the third column and the fifth row of the array
b. the element at the intersection of the second column and the fourth row of the array
c. the element at the intersection of the fourth column and the second row of the array

7.1.9. x = array[3,5]; As a result of executing this assignment, the variable x is assigned:
a. the element at the intersection of the fourth column and the sixth row of the array
b. the element at the intersection of the third column and the fifth row of the array
c. the element at the intersection of the fourth column and the second row of the array

7.1.10. The location of a particular element in a two-dimensional array is determined:
a. by two indices
b. by a single index
c. by three indices

7.1.11. { int array[2][2] = { { 1, 2 }, { 3, 4 } };
for (row = 0; row < 2; row++)
for (col = 0; col < 2; col++)
if (array[row][col] % 0 == 0)
result += array[row][col]; } As a result of executing this code, the value of the "result" variable will be:
a. 6
b. 4
c. 3

7.1.12. { int array[2][2] = { { 1, 2 }, { 3, 4 } };
for (row = 0; row < 2; row++)
for (col = 0; col < 2; col++)
if (array[row][col] % 3 == 0)
result += array[row][col]; } As a result of executing this code, the value of the "result" variable will be:
a. 3
b. 4
c. 6

7.1.13. { int array[2][2] = { { 1, 2 }, { 3, 4 } };
for (row = 0; row < 2; row++)
for (col = 0; col < 2; col++)
if (array[row][col] % 3 == 1)
result += array[row][col]; } As a result of executing this code, the value of the "result" variable will be:
a. 4
b. 6
c. 3

Topic 8. Pointers (*) and addresses. References (&)

Pointers, initialization. * and & operations. The arithmetic of the pointers. Pointers and arrays.

8.1.1. A pointer is a variable that contains:
a. the address of another variable of any type
b. only the address of a variable of double type
c. only the address of a character

8.1.2. The pointer itself is always:
a. an integer
b. a double number
c. a byte

8.1.3. Which code is not a pointer declaration:
a. double pnumber1*;
b. double *pnumber1;
c. double* pnumber1;

8.1.4. Which code is not a pointer declaration:
a. int pnumber1*;
b. int *pnumber1;
c. int* pnumber1;

8.1.5. Which code is not a pointer declaration:
a. char psymbol1*;
b. char *psymbol;
c. char* psymbol1;

8.1.6. Which code is not a pointer declaration:
a. bool pb_1*;
b. bool *pb_1;
c. bool* pb_1;

8.1.7. { int number_1 = 25;
int* pnumber_1 = &number_1;
*pnumber_1 *= 2; } As a result of executing this code, the variable "number_1" will be assigned a value:
a. 50
b. 25
c. 20

8.1.8. { int number_1 = 25;
int* pnumber_1 = &number_1;

(*pnumber_1)++; } As a result of executing this code, the variable "number_1" will be assigned a value:

a. 26
b. 25
c. 24

8.1.9. { int result = 0, number_1 = 25;
int* pnumber_1 = &number_1;
result = -- * pnumber_1; } As a result of executing this code, the variable "number_1" will be assigned a value:

a. 24
b. 25
c. 50

8.1.10. To get the address of a variable is used:

a. the & operator
b. the + operator
c. the - operator

8.1.11. { int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1; } As a result of executing this code, the variable "number_1" will be assigned a value:

a. The pnumber_1 pointer is assigned the address of the number_1 variable
b. The pnumber_1 pointer is not assigned the address of the number_1 variable
c. The pnumber_1 pointer is sometimes assigned the address of the number_1 variable

8.1.12. { int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
cout << (*pnumber_1) << endl; } As a result of executing the code on the screen will appear:

a. 25
b. The value of the variable number_1 in hexadecimal
c. 0

8.1.13. { int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
cout << (*pnumber_1) << endl; } As a result of executing the code on the screen will appear:

a. The value of the variable number_1 in hexadecimal
b. 25
c. 0

8.1.14. { int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
cout << pnumber_1 << endl; } As a result of executing the code on the screen will appear:

a. The value of the variable number_1 in hexadecimal
b. 25
c. 0

8.1.15. { int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
cout << ((int)pnumber_1) << endl; } As a result of executing the code on the screen will appear:

a. The value of the variable number_1 in hexadecimal
b. 25
c. 0

8.1.16. { int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
cout << ((int)pnumber_1) << endl; } As a result of executing the code on the screen will appear:

a. The value of the variable number_1 in decimal
b. 25
c. The value of the variable number_1 in hexadecimal

8.1.17. { int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
cout << pnumber_1 << endl; } As a result of executing the code on the screen will appear:

a. The value of the variable number_1 in hexadecimal
b. 25
c. The value of the variable number_1 in decimal

8.1.18. { int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
cout << (*pnumber_1) << endl; } As a result of executing the code on the screen will appear:

a. The value of the variable number_1 in hexadecimal
b. 25
c. The value of the variable number_1 in decimal

8.1.19. { double number_1 = 25.75;
double* pnumber_1;

pnumber_1 = &number_1;
cout << (*pnumber_1) << endl; } As a result of executing the code on the screen will appear

a. 25.75
b. The value of the variable number_1 in hexadecimal
c. The value of the variable number_1 in decimal

8.1.20. { double number_1 = 25.75;
double* pnumber_1;
pnumber_1 = &number_1;
cout << (pnumber_1) << endl; } As a result of executing the code on the screen will appear

a. The value of the variable number_1 in hexadecimal
b. 25.75
c. The value of the variable number_1 in decimal

8.1.21. { int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
cout << &number_1 << endl; } As a result of executing the code on the screen will appear:

a. The value of the variable number_1 in hexadecimal
b. 25
c. The value of the variable number_1 in decimal

8.1.22. { double number_1 = 25.75;
double* pnumber_1;
pnumber_1 = &number_1;
cout << (&number_1) << endl; } As a result of executing the code on the screen will appear

a. The value of the variable number_1 in hexadecimal
b. 25.75
c. The value of the variable number_1 in decimal

8.1.23. { char symbol = 'a';
char* pchar;
pchar = &symbol;
cout << (*pchar) << endl; } As a result of executing the code on the screen will appear:

a. a
b. The value of the variable number_1 in decimal
c. The value of the variable "symbol" in hexadecimal

8.1.24. { char symbol = 'a';
char* pchar;
pchar = &symbol;
cout << ((int)pchar) << endl; } As a result of executing the code on the screen will appear:

a. The value of the variable number_1 in decimal
b. The value of the variable "symbol" in hexadecimal
c. a

8.1.25. Suppose the pnumber_1 pointer contains the address 1240740. Then
{ int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
pnumber_1++; } as a result of executing this code, the pointer pnumber_1 will be assigned a value:

a. 1240744
b. 1240740
c. 1240748

8.1.26. Suppose the pnumber_1 pointer contains the address 1240740. Then
{ int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
pnumber_1 += 3; } as a result of executing this code, the pointer pnumber_1 will be assigned a value:

a. 1240752
b. 1240740
c. 1240744

8.1.27. Suppose the pnumber_1 pointer contains the address 1240748. Then
{ int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
pnumber_1--; } as a result of executing this code, the pointer pnumber_1 will be assigned a value:

a. 1240744
b. 1240740
c. 1240748

8.1.28. Suppose the pnumber_1 pointer contains the address 1240752. Then
{ int number_1 = 25;
int* pnumber_1;
pnumber_1 = &number_1;
pnumber_1 -= 3; } as a result of executing this code, the pointer pnumber_1 will be assigned a value:

a. 1240740
b. 1240748
c. 1240744

8.1.29. Suppose the pnumber_1 pointer contains the address 1240740. Then
{ double number_1 = 25;
double* pnumber_1;
pnumber_1 = &number_1;

pnumber_1++; } as a result of executing this code, the pointer pnumber_1 will be assigned a value:

a. 1240748
b. 1240740
c. 1240744

8.1.30. Suppose the pnumber_1 pointer contains the address 1240740. Then

{ double number_1 = 25;
double* pnumber_1;
pnumber_1 = &number_1;
pnumber_1 += 3; } as a result of executing this code, the pointer pnumber_1 will be assigned a value:

a. 1240764
b. 1240740
c. 1240744

8.1.31. Suppose the pnumber_1 pointer contains the address 1240748. Then

{ double number_1 = 25;
double* pnumber_1;
pnumber_1 = &number_1;
pnumber_1--; } as a result of executing this code, the pointer pnumber_1 will be assigned a value:

a. 1240740
b. 1240744
c. 1240748

8.1.32. Suppose the pnumber_1 pointer contains the address 1240754. Then

{ double number_1 = 25;
double* pnumber_1;
pnumber_1 = &number_1;
pnumber_1 -= 3; } as a result of executing this code, the pointer pnumber_1 will be assigned a value:

a. 1240730
b. 1240748
c. 1240744

8.1.33. Suppose the psymbol_1 pointer contains the address 1240740. Then

{ char symbol_1 = 'a';
char* psymbol_1;
psymbol_1 = &symbol_1;
psymbol_1++; } as a result of executing this code, the pointer psymbol_1 will be assigned a value:

a. 1240741
b. 1240742
c. 1240744

8.1.34. Suppose the psymbol_1 pointer contains the address 1240740. Then
{ char symbol_1 = 'a';
char* psymbol_1;
psymbol_1 = &symbol_1;
psymbol_1 += 3; } As a result of executing this code, the pointer psymbol_1 will be assigned a value:
a. 1240743
b. 1240741
c. 1240742

8.1.35. Suppose the psymbol_1 pointer contains the address 1240748. Then
{ char symbol_1 = 'a';
char* psymbol_1;
psymbol_1 = &symbol_1;
psymbol_1--; } As a result of executing this code, the pointer psymbol_1 will be assigned a value:
a. 1240747
b. 1240744
c. 1240746

8.1.36. Suppose the psymbol_1 pointer contains the address 1240754. Then
{ char symbol_1 = 'a';
char* psymbol_1;
psymbol_1 = &symbol_1;
psymbol_1 -= 3; } As a result of executing this code, the pointer psymbol_1 will be assigned a value:
a. 1240751
b. 1240748
c. 1240744

References

8.2.1. A reference:
a. is a pseudonym for another variable
b. is a pointer
c. is an array

8.2.2. When declaring a reference:
a. we must specify the name of the variable for which it is created
b. we must specify the name of the variable for which it is not created
c. we should not specify the name of the variable for which it is created

8.2.3. For declaring a reference:
a. the & symbol is used

b. the * symbol is used
c. the & symbol is not used

8.2.4. { int number_1 = 5;
int& ref_number_1 = number_1;
ref_number_1 *= 2; } As a result of the execution of this code, number_1 will be assigned:
a. 10
b. 5
c. 0

8.2.5. { int number_1 = 5;
int& ref_number_1 = number_1;
ref_number_1++; } As a result of the execution of this code, number_1 will be assigned:
a. 6
b. 5
c. 0

8.2.6. { double number_1 = 5.5;
double& ref_number_1 = number_1;
ref_number_1 += 3.3; } As a result of the execution of this code, number_1 will be assigned:
a. 8.8
b. 3.3
c. 5.5

8.2.7. { double number_1 = 5.5;
double& ref_number_1 = number_1;
ref_number_1 -= 3.3; } As a result of the execution of this code, number_1 will be assigned:
a. 2.2
b. 3.3
c. 5.5

8.2.8. { char symbol = 'R';
char& ref_symbol = symbol;
ref_symbol = 'S'; } As a result of the execution of this code, "symbol" variable will be assigned:
a. S
b. R
c. T

8.2.9. { char symbol = 'K';
char& ref_symbol = symbol;
ref_symbol = 'G'; } As a result of the execution of this code, "symbol" variable will be assigned:

a. G
b. K
c. Y

8.2.10. { string str_1 = "Computer";
string& ref_str_1 = str_1;
ref_str_1 = "C++ Language"; } As a result of the execution of this code, "str_1" variable will be assigned:
a. C++ Language
b. Computer
c. Language

8.2.11. { string str_1 = "C# Language";
string& ref_str_1 = str_1;
ref_str_1 = "C++ Language"; } As a result of the execution of this code, "str_1" variable will be assigned:
a. C++ Language
b. Language
c. C# Language

Topic 9. Random number generator. The strings

Random number generator.

9.1.1. The srand(time(NULL)) function:

a. is used to generate random numbers
b. is not used to generate random numbers
c. is used to generate only random negative numbers

9.1.2. [a - b] = rand() % (b + 1 - a) + a - formula is used to generate random numbers in a range:

a. +a - +b
b. -a - +b
c. -a - -b

9.1.3. { result = rand() % 11 + 5; } As a result of executing this code, the "result" variable will be assigned a value from the following range:

a. +5 - +15
b. -5 - +15
c. -15 - -5

9.1.4. { result = rand() % 21 - 5; } As a result of executing this code, the "result" variable will be assigned a value from the following range:

a. -5 - +15
b. +5 - +15
c. -15 - -5

9.1.5. { result = rand() % 11 - 15; } As a result of executing this code, the "result" variable will be assigned a value from the following range:

a. -15 - -5
b. -5 - +15
c. +5 - +15

9.1.6. { result = rand() % 15 + 17; } As a result of executing this code, the "result" variable will be assigned a value from the following range:

a. +17 - +31
b. -17 - +31
c. -31 - -17

9.1.7. { result = rand() % 30 - 11; } As a result of executing this code, the "result" variable will be assigned a value from the following range:

a. -11 - +18
b. +11 - +18
c. -18 - -11

9.1.8. { result = rand() % 13 - 18; } As a result of executing this code, the "result" variable will be assigned a value from the following range:

a. -18 - -6
b. -6 - +18
c. -18 - +6

Strings: Length(), insert(), replace(), erase(), substr(), append(), find(), compare(). Compare of strings. Arrays of strings

9.2.1. The Length() function returns:
a. the number of characters in a stringt
b. the length of the string in bytes
c. the first character of a string

9.2.2. The insert() function:
a. inserts the specified string into the source string
b. replaces the substring in the source string
c. deletes the substring from the source string

9.2.3. The replace() function:
a. replaces the substring in the source string
b. deletes the substring from the source string
c. returns the substring from the source string

9.2.4. The erase() function:
a. deletes the substring from the source string
b. replaces the substring in the source string
c. returns the substring from the source string

9.2.5. The substr() function:
a. returns the substring from the source string
b. appends the specified substring into the source string
c. inserts the specified string into the source string

9.2.6. The append() function:
a. appends the specified substring into the source string
b. returns the substring from the source string
c. inserts the specified string into the source string

9.2.7. If the specified substring is found in the source string, then the function find() returns:
a. The index of the found substring
b. -1
c. 0

9.2.8. If the specified substring is not found in the source string, then the function find() returns:
a. -1
b. The index of the found substring

c. 0

9.2.9. If the source string is equal to the string being compared, then the function compare() returns:
a. 0
b. 1
c. -1

9.2.10. If the source string is greater than to the string being compared, then the function compare() returns:
a. 1
b. 0
c. -1

9.2.11. If the source string is less than to the string being compared, then the function compare() returns:
a. -1
b. 1
c. 0

9.2.12. { string str1 = "C++ is a language";
int result = str1.length(); }
As a result of executing this code, the "result" variable is assigned a value:
a. 17
b. 14
c. 16

9.2.13. { string str1 = "C++ is a popular language";
int result = str1.length(); }
As a result of executing this code, the "result" variable is assigned a value:
a. 25
b. 21
c. 24

9.2.14. { string str1 = "C++ is a language", str2 = " good", result;
result = str1.insert(8, str2); }
As a result of executing this code, the "result" variable is assigned a value:
a. C++ is a good language
b. good language C++ is a
c. is a good C++ language

9.2.15. { string str1 = "C++ is a language", str2 = " popular", result;
result = str1.insert(8, str2); }
As a result of executing this code, the "result" variable is assigned a value:
a. C++ is a popular language

b. is a language popular C++
c. popular language C++ is a

9.2.16. { string str1 = "Java is a language", str2 = "C++", result;
result = str1.replace(0, 4, str2); } As a result of executing this code, the "result" variable is assigned a value:
a. C++ is a language
b. is a language C++
c. C++ language is a

9.2.17. { string str1 = "C# is a language", str2 = "C++", result;
result = str1.replace(0, 2, str2); } As a result of executing this code, the "result" variable is assigned a value:
a. C++ is a language
b. C++ language is a
c. is a language C++

9.2.18. { string str1 = "C++ is a good language", result;
result = str1.erase(8, 5); } As a result of executing this code, the "result" variable is assigned a value:
a. C++ is a language
b. is a language C++
c. C++ language is a

9.2.19. { string str1 = "C++ is a popular language", result;
result = str1.erase(8, 8); } As a result of executing this code, the "result" variable is assigned a value:
a. C++ is a language
b. C++ language is a
c. is a language C++

9.2.20. { string str1 = "C++ is a good language", result;
result = str1.substr(4, 9); } As a result of executing this code, the "result" variable is assigned a value:
a. is a good
b. C++ is a
c. language

9.2.21. { string str1 = "C++ is a popular language", result;
result = str1.substr(4, 12); } As a result of executing this code, the "result" variable is assigned a value:
a. is a popular
b. language
c. C++ is

9.2.22. { string str1 = "C++ is a good", str2 = " language", result;
result = str1.append(str2); } As a result of executing this code, the "result" variable is assigned a value:
a. C++ is a good language
b. good language
c. C++ is a

9.2.23. { string str1 = "C++ is a popular", str2 = " language", result;
result = str1.append(str2); } As a result of executing this code, the "result" variable is assigned a value:
a. C++ is a popular language
b. C++ is a
c. a popular

9.2.24. { string str1 = "C++ is a good language", str2 = "good";
int result = str1.find(str2); } As a result of executing this code, the "result" variable is assigned a value:
a. 9
b. 8
c. 10

9.2.25. { string str1 = "C++ is a popular language", str2 = "popular";
int result = str1.find(str2); } As a result of executing this code, the "result" variable is assigned a value:
a. 9
b. 10
c. 8

9.2.26. { string str1 = "C++ is a popular language", str2 = "good";
int result = str1.find(str2); } As a result of executing this code, the "result" variable is assigned a value:
a. -1
b. 1
c. 0

9.2.27. { string str1 = "abcde", str2 = "abcde";
int result = str1.compare(str2); } As a result of executing this code, the "result" variable is assigned a value:
a. 0
b. 1
c. -1

9.2.28. { string str1 = "abcde", str2 = "abcd";
int result = str1.compare(str2); } As a result of executing this code, the "result" variable is assigned a value:

a. 1
b. 0
c. -1

9.2.29. { string str1 = "abcd", str2 = "abcde";
int result = str1.compare(str2); } As a result of executing this code, the "result" variable is assigned a value:
a. -1
b. 0
c. 1

9.2.30. { string str1 = "abcd", str2 = "fghi";
int result = str1.compare(str2); } As a result of executing this code, the "result" variable is assigned a value:
a. -1
b. 0
c. 1

9.2.31. { string str1 = "fghi", str2 = "abcde";
int result = str1.compare(str2); } As a result of executing this code, the "result" variable is assigned a value:
a. 1
b. 0
c. -1

Topic 10. The files. Work with RAM

The files.

10.1.1. In the C++ language for working with files:
a. there are ofstream and ifstream types
b. there is string type
c. there is int type

10.1.2. ofstream type is used:
a. when writing data to a file
b. when reading data to from file
c. when reading data from an array

10.1.3. ifstream type is used:
a. when reading data to from file
b. when writing data to a file
c. when writing data to an array

10.1.4. ofstream and ifstream are defined in:
a. in the "fstream" library
b. in the "string" library
c. in the "ctime" library

10.1.5. If we are going to work with files, then:
a. we should add the #include "fstream" directive to the beginning of the program file
b. we should not add the #include "fstream" directive to the beginning of the program file
c. we should add the #include "fstream" directive to the end of the program file

10.1.6. { ofstream out_file("file_1.txt"); } As a result of the execution of the code, the out_file variable is set to the file_1.txt file:
a. in writing mode
b. in read mode
c. is not connected

10.1.7. { ifstream in_file("file_1.txt"); } As a result of the execution of the code, the in_file variable is set to the file_1.txt file:
a. in read mode
b. in writing mode
c. is not connected

10.1.8. Which is the correct code:
a. ofstream out_file("D:\\data\\file_1.txt");
b. ofstream out_file("D:\data\\file_1.txt");
c. ofstream out_file("D:\\data\file_1.txt");

10.1.9. Which is the wrong code:
a. ofstream out_file("D:\data\\file_1.txt");
b. ofstream out_file("D:\\data\\file_1.txt");
c. ofstream out_file("C:\\data\\file_1.txt");

10.1.10. { ifstream in_file("file_1.txt"); } The code will be executed if the file_1.txt file:
a. exists
b. not exists
c. sometimes exists

10.1.11. { ofstream out_file("file_1.txt"); } code:
a. Will always be executed
b. Will be executed if the file_1.txt file exists
c. Will be executed if the file_1.txt file not exists

10.1.12. { int number_1 = 10, number_2 = 25, number_3, number_4;
ofstream out_file("file_1.txt");
out_file << number_1 << " " << number_2;
out_file.close(); } As a result of the execution of the code, to the file file_1.txt will be written:
a. 10 and 25
b. 25 and 10
c. 35

10.1.13. { int number_1 = 20, number_2 = 35, number_3, number_4;
ofstream out_file("file_1.txt");
out_file << number_1 << " " << number_2;
out_file.close(); } As a result of the execution of the code, to the file file_1.txt will be written:
a. 20 and 35
b. 35 and 20
c. 55

10.1.14. { int number_1 = 20, number_2 = 35, number_3, number_4;
ofstream out_file("file_1.txt");
out_file << number_1 << " " << number_2;
out_file.close(); } As a result of the execution of the code:
a. 20 and 35 will be written in the file_1.txt file
b. Numbers will be read from the file_1.txt file
c. Nothing will happen

10.1.15. { int number_1 = 20, number_2 = 35, number_3, number_4;
ofstream out_file("file_1.txt");
out_file << number_1 << " " << number_2;
out_file.close(); } As a result of the execution of the code:

a. 20 and 35 will be written in the file_1.txt file
b. Numbers will be read from the file_1.txt file
c. Double numbers will be written in the file_1.txt file

10.1.16. { int number_1 = 20, number_2 = 35;
ofstream out_file("file_1.txt");
out_file << number_1 << " " << number_2;
out_file.close();
int number_3, number_4;
ifstream in_file("file_1.txt");
in_file >> number_3 >> number_4;
in_file.close(); } As a result of the execution of this code, the number_3 and number_4 variables will be assigned the following values:
a. number_3 = 20 and number_4 = 35
b. number_3 = 20 and number_4 = 20
c. number_3 = 35 and number_4 = 20

10.1.17. { double number_1 = 20.55, number_2 = 35.55, number_3, number_4;
ofstream out_file("file_1.txt");
out_file << number_1 << " " << number_2;
out_file.close(); } As a result of the execution of this code:
a. Double numbers will be written in the file_1.txt file
b. Numbers will be read from the file_1.txt file
c. Integer numbers will be written in the file_1.txt file

10.1.18. { double number_1 = 20.55, number_2 = 35.55, number_3, number_4;
ofstream out_file("file_1.txt");
out_file << number_1 << " " << number_2;
out_file.close(); } As a result of the execution of this code:
a. 20.55 and 35.55 will be written in the file_1.txt file
b. Numbers will be read from the file_1.txt file
c. Nothing will happen

10.1.19. { double number_3, number_4;
ifstream in_file("file_1.txt");
in_file >> number_3 >> number_4;
in_file.close(); } As a result of the execution of this code:
a. fractional numbers will be read
b. integers will be read
c. fractional numbers will be recorded

10.1.20. { double number_1 = 20.55, number_2 = 35.55;
ofstream out_file("file_1.txt");
out_file << number_1 << " " << number_2;
out_file.close();

double number_3, number_4;
ifstream in_file("file_1.txt");
in_file >> number_3 >> number_4;
in_file.close(); } As a result of the execution of this code, the number_3 and number_4 variables will be assigned the following values:

a. number_3 = 20.55 and number_4 = 35.55
b. number_3 = 20.55 and number_4 = 20.55
c. number_3 = 35.55 and number_4 = 20.55

10.1.21. { int number_3, number_4;
ifstream in_file("file_1.txt");
in_file >> number_3 >> number_4;
in_file.close(); } As a result of the execution of this code:

a. integers will be read
b. fractional numbers will be read
c. fractional numbers will be recorded

10.1.22. { char symbol_1 = 'A', symbol_2 = 'd', symbol_3, symbol_4;
ofstream out_file("file_1.txt");
out_file << symbol_1 << " " << symbol_2;
out_file.close(); } As a result of the execution of this code:

a. characters will be written
b. fractional numbers will be recorded
c. fractional numbers will be read

10.1.23. { char symbol_3, symbol_4;
ifstream in_file("file_1.txt");
in_file >> symbol_3 >> symbol_4;
in_file.close(); } As a result of the execution of this code:

a. characters will be read
b. integers will be written
c. characters will be written

10.1.24. { char symbol_1 = 'A', symbol_2 = 'd';
ofstream out_file("file_1.txt");
out_file << symbol_1 << " " << symbol_2;
out_file.close(); } As a result of the execution of the code, in the file file_1.txt will be written:

a. 'A' and 'd'
b. 'd' and 'A'
c. 'A' and 'A'

10.1.25. { char symbol_1 = 'A', symbol_2 = 'd';
ofstream out_file("file_1.txt");
out_file << symbol_1 << " " << symbol_2;

out_file.close();
char symbol_3, symbol_4;
ifstream in_file("file_1.txt");
in_file >> symbol_3 >> symbol_4;
in_file.close(); } As a result of the execution of this code, the number_3 and number_4 variables will be assigned the following values:

a. symbol_3 = 'A' and symbol_4 = 'd'
b. symbol_3 = 'd' and symbol_4 = 'A'
c. symbol_3 = 'A' and symbol_4 = 'A'

10.1.26. { string str_1 = "Roman", str_2 = "Samkharadze";
ofstream out_file("file_1.txt");
out_file << str_1 << " " << str_2;
out_file.close(); } As a result of the execution of the code:

a. strings will be written
b. fractional numbers will be written
c. characters will be written

10.1.27. { string str_3, str_4;
ifstream in_file("file_1.txt");
in_file >> str_3 >> str_4;
in_file.close(); } As a result of the execution of the code:

a. strings will be read
b. characters will be read
c. characters will be written

10.1.28. { bool bool_1 = true, bool_2 = false;
ofstream out_file("file_1.txt");
out_file << bool_1 << " " << bool_2;
out_file.close(); } As a result of the execution of the code:

a. logical data will be written
b. logical data will be read
c. characters will be written

10.1.29. { bool bool_3, bool_4;
ifstream in_file("file_1.txt");
in_file >> bool_3 >> bool_4;
in_file.close(); } As a result of the execution of the code:

a. logical data will be read
b. characters will be read
c. logical data will be written

10.1.30. { bool bool_1 = true, bool_2 = false;
ofstream out_file("file_1.txt");
out_file << bool_1 << " " << bool_2;

out_file.close(); } As a result of the execution of the code, in the file file_1.txt will be written:

a. true and false
b. true and true
c. false and true

10.1.31. { bool bool_1 = true, bool_2 = false, bool_3, bool_4;
ofstream out_file("file_1.txt");
out_file << bool_1 << " " << bool_2;
out_file.close();
ifstream in_file("file_1.txt");
in_file >> bool_3 >> bool_4;
in_file.close(); } As a result of the execution of this code, the bool_1 and bool_2 variables will be assigned the following values:

a. bool_3 = true and bool_4 = false
b. bool_3 = false and bool_4 = false
c. bool_3 = false and bool_4 = true

10.1.32. { string str_1 = "C++ is ", str_2 = "a language";
ofstream out_file("file_1.txt");
out_file << str_1 << " " << str_2;
out_file.close(); } As a result of the execution of the code, in the file file_1.txt will be written:

a. "C++ is a language"
b. "a language is C++"
c. "is a language C++"

10.1.33. { string str_1 = "C++ is ", str_2 = "a language";
ofstream out_file("file_1.txt");
out_file << str_1 << " " << str_2;
out_file.close();
string str_3, str_4;
ifstream in_file("file_1.txt");
in_file >> str_3 >> str_4;
in_file.close(); } As a result of the execution of this code, the str_3 and str_4 variables will be assigned the following values:

a. str_3 = "C++ is " and str_4 = "a language"
b. str_3 = " a language " and str_4 = " C++ is "
c. str_3 = " a C++ is " and str_4 = " language"

The sizeof operation. The size() function. new and delete statements

size() function

10.2.1. The size() function returns information about:

a. the total number of elements in a one-dimensional array
b. the length of each element in a one-dimensional array
c. the number of integer elements in a one-dimensional array

10.2.2. { int array[] = {1, 2, 3};
cout << size(array) << endl; } As a result of the execution of this code will be returned:
a. 3
b. 12
c. 1

10.2.3. { double array[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
cout << size(array) << endl; } As a result of the execution of this code will be returned:
a. 5
b. 40
c. 6

10.2.4. { char array[] = { 'a', 'b', 'c', 'd' };
cout << size(array) << endl; } As a result of the execution of this code will be returned:
a. 4
b. 5
c. 3

sizeof operation

10.2.5. The sizeof operation returns the size of its operand:
a. in bytes
b. in kilobytes
c. in megabytes

10.2.6. { int number_1, result;
result = sizeof number_1; } As a result of the execution of the code, the "result" variable will be assigned:
a. 4
b. 1
c. 8

10.2.7. { int result;
double number_1;
result = sizeof number_1; } As a result of the execution of the code, the "result" variable will be assigned:
a. 8
b. 4
c. 1

10.2.8. { char symbol_1;

int result = sizeof symbol_1; } As a result of the execution of the code, the "result" variable will be assigned:

a. 1
b. 4
c. 2

10.2.9. { double array[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
int result = sizeof array; } As a result of the execution of the code, the "result" variable will be assigned:

a. 40
b. 5
c. 20

10.2.10. { int array[] = { 1, 2, 3, 4, 5 };
int result = sizeof array; } As a result of the execution of the code, the "result" variable will be assigned:

a. 20
b. 5
c. 40

10.2.11. { char array[] = { 'a', 'b', 'c', 'd', 'e' };
int result = sizeof array; } As a result of the execution of the code, the "result" variable will be assigned:

a. 5
b. 20
c. 40

10.2.12. { int array[] = { 1, 2, 3, 4, 5 };
int result = sizeof array[2]; } As a result of the execution of the code, the "result" variable will be assigned:

a. 4
b. 5
c. 20

10.2.13. { double array[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
int result = sizeof array[2]; } As a result of the execution of the code, the "result" variable will be assigned:

a. 8
b. 5
c. 40

10.2.14. { char array[] = { 'a', 'b', 'c', 'd', 'e' };
int result = sizeof array[2]; } As a result of the execution of the code, the "result" variable will be assigned:

a. 1

b. 10
c. 20

10.2.15. { int array[] = { 1, 2, 3, 4, 5 };
int result = (sizeof array) / (sizeof array[0]); } As a result of the execution of the code, the "result" variable will be assigned:
a. 5
b. 4
c. 20

10.2.16. { double array[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
int result = (sizeof array) / (sizeof array[0]); } As a result of the execution of the code, the "result" variable will be assigned:
a. 5
b. 4
c. 20

10.2.17. { char array[] = { 'a', 'b', 'c', 'd', 'e' };
int result = (sizeof array) / (sizeof array[0]); } As a result of the execution of the code, the "result" variable will be assigned:
a. 5
b. 4
c. 20

Topic 11. The functions

Functions with and without parameters. Declaration of function. The parameters and arguments. Prototype of the function. Call of function. The return value from the function.

11.1.1. A keyword as a function name:
a. cannot be used
b. can be used
c. can sometimes be used

11.1.2. The name Main() is reserved for the function from which program execution:
a. begins
b. not begins
c. sometimes begins

11.1.3. A function consists of:
a. a header and code (body)
b. a header only
c. a code (body)

11.1.4. The function header consists of:
a. the type of the result returned by the function, the name of the function, and a list of parameters
b. only the function name
c. only of the returned result type

11.1.5. When declaring a function, parameter names:
a. are separated by commas
b. are not separated by commas
c. are separated by points

11.1.6. Each parameter name:
a. must be preceded by the corresponding type
b. must not be preceded by the corresponding type
c. should sometimes be preceded by the corresponding type

11.1.7. The parameter is a variable that receives the value of an argument:
a. that is passed to a function when it is called
b. that is not passed to a function when it is called
c. that is passed to a function when it is not called

11.1.8. Definition of function in another function:
a. is not allowed
b. is allowed
c. is sometimes allowed

11.1.9. A function with a return operator can return:
a. 1 result
b. 2 result
c. 3 result

11.1.10. void Function_1()
{ cout << "C++ is a programming language" << endl; } This function:
a. does not return anything
b. returns a string
c. returns double value

11.1.11. int Function_1()
{ return 5 * 5; } This function:
a. returns an integer number
b. returns a string
c. does not return anything

11.1.12. double Function_1()
{ return 5.5 * 5.5; } This function:
a. returns a double value
b. returns a string
c. does not return anything

11.1.13. char Function_1()
{ return 'A'; } This function:
a. returns a symbol
b. returns a string
c. does not return anything

11.1.14. string Function_1()
{ return "C++"; } This function:
a. returns a string
b. returns an integer number
c. does not return anything

11.1.15. bool Function_1()
{ return true; } This function:
a. returns a logical value
b. returns a string
c. does not return anything

11.1.16. If the function has no parameters, then to call it in the main program we:
a. indicate only the name of the function
b. not indicate only the name of the function

c. sometimes indicate only the name of the function

11.1.17. A prototype is a function header that:
a. ends with a semicolon
b. not ends with a semicolon
c. ends with a comma

11.1.18. The function prototype:
a. must be specified before the Main() function
b. should not be specified before the Main() function
c. must be specified after the Main() function

11.1.19. When using a prototype, the function with the title (header) and code:
a. is defined after the main program
b. is not defined after the main program
c. is defined before the main program

11.1.20. The value passed to the function is called:
a. an argument
b. a parameter
c. a type

11.1.21. A variable declared in the parameter list of a function that receives the value of an argument is called:
a. a parameter
b. an argument
c. a type

11.1.22. A function parameter:
a. is only visible inside its own function
b. is only visible inside another function
c. is not visible only inside its own function

11.1.23. If the function has several parameters, then they:
a. are separated by commas
b. are not separated by commas
c. are separated by points

11.1.24. void Function_1(int par1)
{
  if (par1 >= 0) cout << "positive" << endl;
  else cout << "negative" << endl;
}
int main()
{ Function_1(5); } As a result of calling this function, it will be displayed on the screen:

a. positive
b. negative
c. 0

11.1.25. The execution of the function is interrupted when:
a. a closing curly brace is encountered
b. the opening curly brace is encountered
c. an array declaration is encountered

11.1.26. The execution of the function is interrupted when:
a. the "return" operator is encountered
b. the "for" operator is encountered
c. the "switch" operator is encountered

11.1.27. When executing the "return" statement:
a. control is transferred to the line of the calling program from which the function was called
b. control is not transferred to the line of the calling program from which the function was called
c. control is transferred to the line of the calling program from which the function was not called

11.1.28. When executing the "return" statement:
a. the rest of the function code will not be executed
b. the rest of the function code will be executed
c. the rest of the function code will be sometimes executed

11.1.29. int Function_1(int gverdi1, int gverdi2)
{ return gverdi1 + gverdi2; }
int main()
{ int result = Function_1(5, 10); } This function returns the result:
a. 15
b. 10
c. 5

Topic 12. Ways to transfer values to functions

The functions. Transfer values to functions through pointers. Transfer values to functions through references. Transfer one and two-dimensional arrays to functions

12.1.1. int Function_1(int array[]) { }
void main()
{ int array[] = { 1, 2, 3, 4, 5 };
Function_1(array); } In this program, when calling the function, is passed:

a. A one-dimensional array
b. The first element of the array
c. The last element of the array

12.1.2. int Function_1(int array[3][3]) { }
void main()
{ int array[3][3] = { { 1, 2, 3}, { 4, 5, 6 }, { 7, 8, 9 } };
Function_1(array); } In this program, when calling the function, is passed:

a. A two-dimensional array
b. The first element of the array
c. A one-dimensional array

12.1.3. void Function_1(int number_1) { }
int main()
{ int array[] = { 1, 2, 3, 4, 5 };
Function_1(array[1]); } In this program, when calling the function, is passed:

a. The second element of a one-dimensional array
b. The first element of a one-dimensional array
c. A one-dimensional array

12.1.4. void Function_1(int number_1) { }
int main()
{ int array[] = { 1, 2, 3, 4, 5 };
Function_1(array[3]); } In this program, when calling the function, is passed:

a. The fourth element of a one-dimensional array
b. The third element of a one-dimensional array
c. A one-dimensional array

12.1.5. void Function_1(int number_1) { }
int main()
{ int array[3][3] = { { 1, 2, 3}, { 4, 5, 6 }, { 7, 8, 9 } };
Function_1(array[2][1]); } In this program, when calling the function, is passed:

a. The element at the intersection of the third row and the second column of a two-dimensional array
b. The element at the intersection of the second row and the firts column of a two-dimensional array

c. A two-dimensional array

12.1.6. void Function_1(int number_1) { }
int main()
{ int array[3][3] = { { 1, 2, 3}, { 4, 5, 6 }, { 7, 8, 9 } };
Function_1(array[0][2]); } In this program, when calling the function, is passed:

a. The element at the intersection of the first row and the third column of a two-dimensional array

b. The element at the intersection of the first row and the second column of a two-dimensional array

c. A two-dimensional array

12.1.7. Which is an incorrect function declaration:

a.
```
void Function_3(int array[2][2]) {  };
int main()
{ int array[] = { 1, 2, 3, 4, 5 };
   Function_3(array); }
```

b.
```
void Function_2(int array[5]) {  }
int main()
{ int array[] = { 1, 2, 3, 4, 5 };
   Function_2(array); }
```

c.
```
void Function_1(int array[]) {  }
int main()
{ int array[] = { 1, 2, 3, 4, 5 };
Function_1(array); }
```

12.1.8. Which is an incorrect function declaration:

a.
```
void Function_3(int array[]) {  };
int main()
{ int array[] = { { 1, 2, 3}, { 4, 5, 6 }, { 7, 8, 9 } };
   Function_3(array); }
```

b.
```
void Function_2(int array[5]) {  }
int main()
{ int array[] = { 1, 2, 3, 4, 5 };
   Function_2(array); }
```

c.
```
void Function_1(int array[]) {  }
int main()
{ int array[] = { 1, 2, 3, 4, 5 };
Function_1(array); }
```

12.1.9. Which is the correct call of function - void Function_1(double number_1) { }:

a.
```
{ Function_1(5.5); }
```

b.
```
{ double array[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
Function_1(array); }
```

c.
```
{ double array[3][3] = { { 1.1, 2.2, 3.3}, { 4.4, 5.5, 6.6 }, { 7.7, 8.8, 9.9 } };
```

Function_1(array); }

12.1.10. Which is the correct call of function - void Function_1(double array_1[]) { } :

a. { double array_1[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
Function_1(array_1); }

b. { double array_2[2][2] = { { 1.1, 2.2 }, { 3.3, 4.4 } };
Function_1(array_2); }

c. { Function_1(5.5); }

12.1.11. When passing parameters to a function, we:

a. can use pointers and references

b. can not use pointers and references

c. can use only pointers

12.1.12. void Gacvla_1(int* par1, int* par2)
{ int temp;
temp = *par1;
*par1 = *par2;
*par2 = temp; }
int main()
{ int number_1 = 5, number_2 = 10;
Gacvla_1(&number_1, &number_2); } As a result of executing this code, the variables "number_1" and "number_2" will be assigned:

a. number_1 = 10 and number_2 = 5

b. number_1 = 5 and number_2 = 10

c. number_1 = 15 and number_2 = 10

12.1.13. void Gacvla_2(int& par1, int& par2)
{ int temp;
temp = par1;
par1 = par2;
par2 = temp; }
int main()
{ int number_1 = 5, number_2 = 10;
Gacvla_1(number_1, number_2); } As a result of executing this code, the variables "number_1" and "number_2" will be assigned:

a. number_1 = 10 and number_2 = 5

b. number_1 = 5 and number_2 = 10

c. number_1 = 15 and number_2 = 10

12.1.14. void Gacvla_1(int* par1, int* par2)
{ int temp;
temp = *par1;
*par1 = *par2;
*par2 = temp; } The parameters of this function are:

a. pointers to integers
b. references to integers
c. an array

12.1.15. void Gacvla_2(int& par1, int& par2)
{ int temp;
temp = par1;
par1 = par2;
par2 = temp; } The parameters of this function are:
a. references to integers
b. pointers to integers
c. an array

12.1.16. void Gacvla_1(double* par1, double* par2)
{ double temp;
temp = *par1;
*par1 = *par2;
*par2 = temp; } The parameters of this function are:
a. pointers to doubles
b. references to doubles
c. an array

12.1.17. void Gacvla_2(double& par1, double& par2)
{ double temp;
temp = par1;
par1 = par2;
par2 = temp; } The parameters of this function are:
a. references to doubles
b. pointers to doubles
c. an array

12.1.18. void Gacvla_1(int* par1, int* par2)
{ int temp;
temp = *par1;
*par1 = *par2;
*par2 = temp; } Which is the correct call to this function:
a. { int number_1 = 5, number_2 = 10;
Gacvla_1(&number_1, &number_2); }
b. { int number_1 = 5, number_2 = 10;
Gacvla_1(number_1, number_2); }
c. { int number_1 = 5, number_2 = 10;
Gacvla_1(*number_1, *number_2); }

12.1.19. void Gacvla_1(int& par1, int& par2)
{ int temp;

temp = par1;
par1 = par2;
par2 = temp; } Which is the correct call to this function:

a. { int number_1 = 5, number_2 = 10;
Gacvla_1(number_1, number_2); }

b. { int number_1 = 5, number_2 = 10;
Gacvla_1(&number_1, &number_2); }

c. { int number_1 = 5, number_2 = 10;
Gacvla_1(*number_1, *number_2); }

12.1.20. void Gacvla_1(int par1, int par2) { }
int main()
{ int number_1 = 5, number_2 = 10;
Gacvla_1(number_1, number_2); } When this function is called, par1 and par2 parameters will be assigned values:

a. par1 = 5 and par2 = 10

b. par1 = 10 and par2 = 5

c. par1 = 5 and par2 = 5

12.1.21. void Gacvla_1(double par1, double par2) { }
int main()
{ double number_1 = 5.5, number_2 = 10.7;
Gacvla_1(number_1, number_2); } When this function is called, par1 and par2 parameters will be assigned values:

a. par1 = 5.5 and par2 = 10.7

b. par1 = 10.7 and par2 = 5.5

c. par1 = 10.7 and par2 = 10.7

12.1.22. void Gacvla_1(int* par1, int* par2)
{ int temp;
temp = *par1;
*par1 = *par2;
*par2 = temp; } Changing parameter values in this function:

a. will change the values of the arguments

b. will not change the values of the arguments

c. will sometimes change the argument values

12.1.23. void Gacvla_2(int& par1, int& par2)
{ int temp;
temp = par1;
par1 = par2;
par2 = temp; } Changing parameter values in this function:

a. will change the values of the arguments

b. will not change the values of the arguments

c. will sometimes change the argument values

12.1.24. A function using a pointer:

a. can return multiple results
b. cannot return multiple results
c. can return only two results

12.1.25. A function using a reference:

a. can return multiple results
b. cannot return multiple results
c. can return only two results

12.1.26. int FartPerim(int par1, int par2, int* pfartobi)

```
{ *pfartobi = par1 * par2;
  return 2 * (par1 + par2); }
int main()
{ int sigrdze = 5, sigane = 10, fartobi, perimetri;
    perimetri = FartPerim(sigrdze, sigane, &fartobi); }
```

This function returns one value using the "return" operator, and another value using:

a. a pointer
b. a reference
c. an array

12.1.27. int FartPerim1(int par1, int par2, int& fart)

```
{ fart = par1 * par2;
  return 2 * (par1 + par2); }
int main()
{ int sigrdze, sigane, fartobi, perimetri;
  perimetri = FartPerim1(sigrdze, sigane, fartobi); }
```

This function returns one value using the "return" operator, and another value using:

a. a reference
b. a pointer
c. an array

Topic 13. The classes. The methods

The concept of class. Its members. Class declaration. The method. Object creation.

13.1.1. A class:
a. is a user-created type
b. is an array
c. is an object

13.1.2. A class:
a. Object creation mechanism
b. Array creation mechanism
c. Class creation mechanism

13.1.3. An object is:
a. an instance of a class
b. an instance of an array
c. an instance of an object

13.1.4. An object is:
a. a concrete implementation of a class
b. a concrete realization of an array
c. a concrete realization of an object

13.1.5. A class:
a. associates data with the code that manipulates it
b. does not associates data with the code that manipulates it
c. associates data with code that does not manipulate it

13.1.6. A class:
a. provides a means of accessing the members of the class
b. does not provides a means of accessing the members of the class
c. sometimes provides a means of accessing the members of the class

13.1.7. Encapsulation is:
a. the separation of access rights to members of a class
b. the separation of access rights to arrays of a class
c. the separation of access rights to variables of a class

13.1.8. Variables and functions declared inside a class:
a. are called members of that class
b. are not called members of that class
c. are called members of the other class

13.1.9. By default, variables and functions declared in the class are:

a. private
b. public
c. not private

13.1.10. To declare public variables and functions:
a. is used the public keyword
b. is used the private keyword
c. is not used the public keyword

13.1.11. To declare private variables and functions:
a. is used the private keyword
b. is used the public keyword
c. is used the int keyword

13.1.12. Access to public members of a class:
a. can be made from code defined outside of that class
b. can not be made from code defined inside that class
c. can not be made from code defined inside another class

13.1.13. Access to private members of a class:
a. can be made by members of the same class
b. can not be made by members of the same class
c. can be made by members of the other class

13.1.14. Code defined outside a class:
a. can only refer to private members of that class by using public functions of the same class
b. can only refer to private members of that class by using public functions of the other class
c. can only refer to private members of that class by using private functions of the other class

13.1.15. The right to access the closed members of the class:
a. have only members defined within the class
b. have only members defined outside the class
c. has no one

13.1.16. class Class_1
{ int number_1;
int number_2; }; In this class number_1 and number_2 are:
a. private
b. public
c. not public and not private

13.1.17. class Class_1
{ int number_1;
int number_2; } obj_2; Here obj_2 object type is:
a. Class_1

b. obj_2
c. class

13.1.18. class Class_1
{ int number_1;
int number_2; };
int main()
{ Class_1 obj_1; } Here obj_1 object type is:
a. Class_1
b. int
c. class

13.1.19. class Class_1
{ int number_1;
int number_2; };
int main()
{ Class_1 obj_1; } Here, from the object obj_1 to the member "number_1":
a. we can not access
b. we can access
c. sometimes we can access

13.1.20. class Class_1
{ int number_1;
public: int number_2; };
int main()
{ Class_1 obj_1; } Here, from the object obj_1 to the member "number_2":
a. we can access
b. we can not access
c. sometimes we can access

13.1.21. A function:
a. may or may not be a member of a class
b. must be a class member
c. must not be a class member

13.1.22. A function declared inside a class:
a. is called a method
b. is called an array
c. is called a function

13.1.23. Inside a class:
a. a function prototype can be declared
b. a function prototype can not be declared
c. must declare a function prototype

13.1.24. A function code (body):
a. can be declared both inside the class and outside the class
b. cannot be defined either inside a class or outside a class
c. must be declared only inside the class

13.1.25. If "public" is not specified, then the function:
a. is private and therefore only accessible within the class in which it is declared
b. is public and therefore only accessible within the class in which it is declared
c. is private and therefore only accessible within the class in which it is not declared

13.1.26. If we want to declare the function code outside the class, then:
a. after the name of the class we must place "::" characters and the name of the function
b. after the name of the class we should not place "::" symbols and the name of the function
c. after the name of the class we must place ":" characters and the name of the function

13.1.27. The "::" characters:
a. are called the "visibility area expansion operator"
b. are not called the "visibility area expansion operator"
c. are called the "visibility area restriction operator"

13.1.28. class Class_1
```
{ public:
   int Method_1(int); };
int Class_1::Method_1(int par_1)
{ return par_1; }
int main()
{ Class_1 obj_1;
   cout << obj_1.Method_1(5) << endl; } In this code, the method body:
```
a. is declared outside the class
b. is declared inside the class
c. is not declared

13.1.29. class Class_1
```
{ public: int Method_1(char par_1) { return par_1; } };
```
Which is the correct call of the method Method_1():
a. { Class_1 obj_1; cout << obj_1.Method_1('a') << endl; }
b. { Class_1 obj_1; cout << obj_1.Method_1() << endl; }
c. { Class_1 obj_1; cout << obj_1.Method_1("asd") << endl; }

Topic 14. The classes. The constructor

The constructor. Object creation

14.1.1. A constructor is a function whose purpose:

a. is to initialize object variables
b. is not to initialize object variables
c. is to return a value

14.1.2. A constructor:

a. is always called when an object is created
b. is not always called when an object is created
c. is never called when the object is created

14.1.3. Unlike a regular function:

a. a constructor has the same name as a class
b. a constructor does not have the same name as a class
c. a constructor has a different name than a class

14.1.4. Unlike a regular function:

a. The type of the return value is not specified in the constructor
b. The type of the return value is specified in the constructor
c. The type of the return value sometimes is specified in the constructor

14.1.5. Constructors:

a. are used to assign initial values to object variables
b. are not used to assign initial values to object variables
c. are only used to assign integer type values to object variables

14.1.6. Every class:

a. has a constructor whether it is defined or not
b. does not has a constructor whether it is defined or not
c. has no constructor if it is not defined

14.1.7. In C++ language:

a. provides a constructor that assigns all object variables to zero (this applies to variables of a normal type) or "null" value (this applies to variables of a reference type)
b. does not provides a constructor that assigns all object variables to zero (this applies to variables of a normal type) or "null" value (this applies to variables of a reference type)
c. provides a constructor that does not assigns all object variables to zero (this applies to variables of a normal type) or "null" value (this applies to variables of a reference type)

14.1.8. The default constructor:

a. assigns all object variables to null (this applies to ordinary type variables) or null value (this applies to reference type variables)

b. does not assign null (this applies to ordinary type variables) or null value (this applies to reference type variables) to all variables of the object
c. assigns all object variables to non-null (this applies to variables of the usual type)

14.1.9. The default constructor:
a. is called when no constructor is defined in the class
b. will not be called if no constructor is defined in the class
c. is called when a constructor is defined in the class

14.1.10. If a constructor:
a. is explicitly defined in a class, then it will be called when the object is created
b. is not explicitly defined in the class, then it will be called when the object is created
c. is explicitly defined in the class, then it will not be called when the object is created

14.1.11. class Class_1
{ int number_1;
public: Class_1(int par_1)
{ number_1 = par_1; } }; What is the correct form of object creation:
a. Class_1 obj_3(5)
b. Class_1 obj_2();
c. Class_1 obj_1;

14.1.12. class Class_1
{ double number_1;
public: Class_1(double par_1)
{ number_1 = par_1; } }; What is the correct form of object creation:
a. Class_1 obj_3(5.5)
b. Class_1 obj_2();
c. Class_1 obj_1;

14.1.13. class Class_1
{ double number_1;
public:
void Method_1(double par_1)
{ number_1 = par_1; } }; What is the correct form of object creation:
a. Class_1 obj_1;
b. Class_1 obj_2();
c. Class_1 obj_3(5.5);

14.1.14. A constructor to initialize variables declared in a class:
a. can use an initialization list
b. can not use an initialization list
c. must use the initialization list

14.1.15. class Class_1

{ double number_1;
public:
  Class_1(double par_1) : number_1(par_1)
  { } }; to initialize the variable "number_1":
a. the initialization list is used
b. no initialization list is used
c. an array is used

14.1.16. class Class_1
{ double number_1;
public:
  Class_1(double par_1)
  { number_1 = par_1; } }; to initialize the variable "number_1":
a. no initialization list is used
b. the initialization list is used
c. an array is used

14.1.17. class Class_1
{ double number_1;
public:
  Class_1(double par_1)
  { number_1 = par_1; } };
int main()
{ Class_1 obj_2 = 7.89; } When the constructor has one parameter, then this form of object creation:
a. is allowed
b. is not allowed
c. sometimes is allowed

Topic 15. The structures. The enumerators

The structures. Its members. Structure declaration. The method. Object creation.

15.1.1. A structure is a type that:

a. has its own members: variables and functions
b. has no members of its own: variables and functions
c. has only variables

15.1.2. By default structure members:

a. are public
b. are not public
c. are private

15.1.3. To declare the structure:

a. The "struct" keyword is used
b. The "struct" keyword is not used
c. The "class" keyword is used

15.1.4. struct Struct_1
{ double number_1; } In this declaration, the "number_1" variable:

a. is public
b. is private
c. is not public

15.1.5. struct Struct_1
{ int number_1;
public: Struct_1(int par_1)
{ number_1 = par_1; } }; What is the correct form of object creation:

a. Class_1 obj_3(5)
b. Class_1 obj_2();
c. Class_1 obj_1;

15.1.6. struct Struct_1
{ double number_1;
public: Struct_1(double par_1)
{ number_1 = par_1; } }; What is the correct form of object creation:

a. Class_1 obj_3(5.5)
b. Class_1 obj_2();
c. Class_1 obj_1;

15.1.7. struct Struct_1
{ double number_1;
public:
void Method_1(double par_1)

{ number_1 = par_1; } }; What is the correct form of object creation:

a. Class_1 obj_1;
b. Class_1 obj_2();
c. Class_1 obj_3(5.5);

15.1.8. struct Struct_1
{ double number_1;
public:
Struct_1(double par_1) : number_1(par_1)
{ } }; To initialize the variable "number_1":

a. the initialization list is used
b. no initialization list is used
c. an array is used

15.1.9. struct Struct_1
{ double number_1;
public:
Struct_1(double par_1)
{ number_1 = par_1; } }; To initialize the variable "number_1":

a. no initialization list is used
b. the initialization list is used
c. an array is used

15.1.10. struct Struct_1
{ double number_1;
public:
Struct_1(double par_1)
{ number_1 = par_1; } };
int main()
{ Class_1 obj_2 = 7.89; } When a constructor has a single parameter, this form of object creation:

a. is allowed
b. is not allowed
c. sometimes is allowed

The enumerators.

15.2.1. To declare an enumerated type:

a. the "enum" keyword is used
b. the "public" keyword is used
c. the "static" keyword is used

15.2.2. Which reasoning is correct:

a. Enumerations are sets of named integer constants that define all possible values of a variable for a given type
b. Enumerations are sets of named double constants that define all possible values of a variable for a given type

c. Enumerations are sets of named string constants that define all possible values of a variable for a given type

15.2.3. To each constant in the enumeration corresponds:
a. an integer value
b. a double value
c. a string value

15.2.4. In the enumeration, the integer value for each constant:
a. is greater than the value of the previous constant
b. is less than the value of the previous constant
c. is equal to the value of the previous constant

15.2.5. In the enumeration, the integer value for each constant:
a. is less than the value of the next constant
b. is greater than the value of the next constant
c. is equal to the value of the next constant

15.2.6. The value of the first constant in the enumeration is:
a. 0
b. -1
c. 1

15.2.7. To refer to the elements of an enumeration:
a. we must specify the name of the enumeration, two colons, and the name of the element
b. we only need to specify the enumeration name
c. we only need to specify the name of the enumeration element

15.2.8. Using an initializer in enumerations:
a. an integer value of a constant can be used
b. an integer value of a constant cannot be used
c. an integer value of a constant sometimes can be used

15.2.9. In enumerations, the value specified by the initializer:
a. must be greater than the value of the previous constant
b. must be less than the value of the previous constant
c. must be equal to the value of the previous constant

15.2.10. In enumerations, the value specified by the initializer:
a. must be less than the value of the next constant
b. must be greater than the value of the next constant
c. must be equal to the value of the next constant

15.2.11. enum WeekDay { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday }; Here the value of the "Sunday" constant is:

a. 6
b. 5
c. 7

15.2.12. enum Months : char { January = 2, February, March, April, May, June, July, August, September, October, November, December }; Here the value of the "April" constant is:
a. 5
b. 4
c. 6

15.2.13. enum WeekDay : char { Monday, Tuesday = 10, Wednesday, Thursday, Friday, Saturday, Sunday }; Here the value of the "Thursday" constant is:
a. 12
b. 11
c. 13

15.2.14. enum WeekDay : char { Monday = 20, Tuesday = 10, Wednesday, Thursday, Friday, Saturday, Sunday }; Here the value of the "Thursday" constant is:
a. 12
b. 13
c. 11

15.2.15. enum WeekDay { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday }; Here the value of the "Friday" constant is:
a. 4
b. 5
c. 6

15.2.16. enum WeekDay { Monday = 4, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday }; Here the value of the "Thursday" constant is:
a. 7
b. 6
c. 8

15.2.17. enum WeekDay : char { Monday, Tuesday, Wednesday = 5, Thursday, Friday, Saturday, Sunday }; Here the value of the "Saturday" constant is:
a. 8
b. 7
c. 9

15.2.18. enum WeekDay : char { Monday = 20, Tuesday = 10, Wednesday, Thursday, Friday, Saturday, Sunday }; Here the value of the "Thursday" constant is:
a. 12
b. 11
c. 13

Literature

1. R. Samkharadze, L. Gachechiladze. Programming in C++ language (manual). GTU "IT-Consulting Scientific Center", 2018. p. 247. ISBN 978-9941-27-493-0.
2. R. Samkharadze. Visual C++/CLI.NET (manual). Georgian Technical University. Tbilisi, "Technical University", 2010. 442 p. ISBN 978-9941-14-795-1.
3. R. Samkharadze. Visual C#.NET (manual). Georgian Technical University. Tbilisi, GTU "IT-Consulting Scientific Center", 2014. 508 p. ISBN 978-9941-20-443-2.
4. R. Samkharadze, L. Gachechiladze. SQL Server (manual). Georgian Technical University. Publishing House "Technical University". 2016. - 453 p. ISBN 978-9941-20-661-0.
5. Schildt. D. C++ Tutorial: Translation from English 2001. – 688 p.
6. E.Butow, T. Ryan. Your visual blueprint for building .NET applications.
7. Harold Davis. Visual C# .NET Programming.
8. Horton A. Visual C++ 2005. Basic course.
9. Kernighan B, Ritchie D. Language C.
10. Lippman. C++ for beginners.

Computer support l. Gachechiladze.

It is printed as presented by the authors

Transferred to production 10.03.2023. Signed for printing 10.03.2023. Paper size 60X84 1/8. Conditional printed sheet 6.63. Circulation 50 copy.

Publishing House "Technical University", Tbilisi, Kostava 77