

Міністерство освіти та науки України
Національний університет водного господарства та
природокористування
Кафедра автоматизації, електротехнічних та комп'ютерно-
інтегрованих технологій

04-03-418М

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з навчальної дисципліни
«Програмування. Частина 1. Програмування мовою С++»
для здобувачів вищої освіти першого (бакалаврського) рівня
за освітньо-професійною програмою «Автоматизація,
комп'ютерно-інтегровані технології та робототехніка»
спеціальності 174 «Автоматизація, комп'ютерно-інтегровані
технології та робототехніка», 141 «Електроенергетика,
електротехніка та електромеханіка»
денної та заочної форм навчання

Рекомендовано науково –
методичною радою з якості
ННІ ЕАВГ
Протокол № 3 від 26.11.2024 р.

Рівне – 2024

Методичні вказівки до виконання лабораторних робіт з навчальної дисципліни «Програмування. Частина 1. Програмування мовою С++» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка», 141 «Електроенергетика, електротехніка та електромеханіка» денної та заочної форм навчання. [Електронне видання] / Присяжнюк О. В. – Рівне : НУВГП, 2024. – 128 с.

Укладач: Присяжнюк О. В., к.т.н., доцент кафедри АЕКІТ.

Відповідальний за випуск: Древецький В. В., д.т.н., професор, завідувач кафедри АЕКІТ.

Керівник освітньої програми «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»: Христюк А. О., к.т.н., доцент кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій.

Керівник групи забезпечення спеціальності 141 «Електроенергетика, електротехніка та електромеханіка»: Василець С. В., д.т.н., професор кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій

© О. В. Присяжнюк, 2024

© НУВГП, 2024

Content

1. Programming based on linear algorithms	4
2. Development of programs with a branched structure	18
3. Developing programs with one-dimensional arrays.	39
4. Developing programs with multidimensional arrays.	52
5. Developing programs with string variables	66
6. Developing programs with file variables. Working with files	87
7. Developing programs with custom classes. Working with classes and objects	101
8. Developing programs with custom classes. Inheritance	119
8. List of recommended literature	128

№1. Programming based on linear algorithms

1.1. The purpose of the work

Getting the knowledge and skills necessary for programming using linear algorithms and learning to use them in practice in the process of developing programs in the C++ programming language.

1.2. Brief theoretical information

The concept of an algorithm. Types of algorithms

An algorithm is a system of rules for performing a computing process that necessarily leads to the solution of a certain class of problems after a finite number of operations. When writing computer programs, an algorithm describes a logical sequence of operations. Block diagrams are often used to visually represent algorithms.

Each algorithm is a list of well-defined instructions for solving a problem. Starting from an initial state, the algorithm's instructions describe the process of computations that occur through a sequence of states that ultimately culminate in an end state. The transition from one state to the next is not necessarily deterministic — some algorithms contain elements of randomness.

An algorithm is a description of the process of solving one or another task. An algorithm is a finite set of rules arranged in a certain logical order, which allows the performer to solve any specific problem from a certain class of problems of the same type.

Algorithms have a number of important properties:

Finitude. The algorithm must always terminate after completing a finite number of steps. A procedure that has the remaining characteristics of an algorithm, without possibly being finite, is called a calculation method.

Discretion. The process determined by the algorithm can be dissected (divided) into separate elementary stages (steps), each of which is called a step of the algorithmic process or algorithm.

Certainty. Each step of the algorithm must be precisely defined. Actions to be taken should be clearly and unambiguously defined for each possible case.

Input data. The algorithm has a certain number (perhaps zero) of input data, i.e., values specified before its operation or the values of which are determined

during the operation of the algorithm.

Output data. The algorithm has one or more output data, that is, values that have a fairly definite relationship with the input data.

Efficiency. An algorithm is considered efficient if all its operators are simple enough to be executed accurately in a finite amount of time using a pencil and a piece of paper.


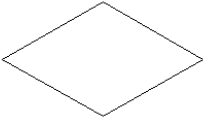


The following main ways of recording algorithms are distinguished:

verbal - the algorithm is described in natural language;

symbolic - the algorithm is described using a set of symbols;

graphic - the algorithm is described using a set of graphic images.

Table 1.1 Table of main elements of block diagrams

№. п/п	Element	Name	Meaning
1.		Calculation block	Calculated actions or sequence of actions
2.		Logic block	Selection of the direction of execution of the algorithm depending on a certain condition
3.		Data I/O block	General designation of data input (output).
4.		Start (end)	The beginning or end of the algorithm, entry or exit

The basic structures of algorithms are a defined set of blocks and standard ways of connecting them to perform typical sequential events. The main structures include the following:

- linear;

- branched;
- cyclical.

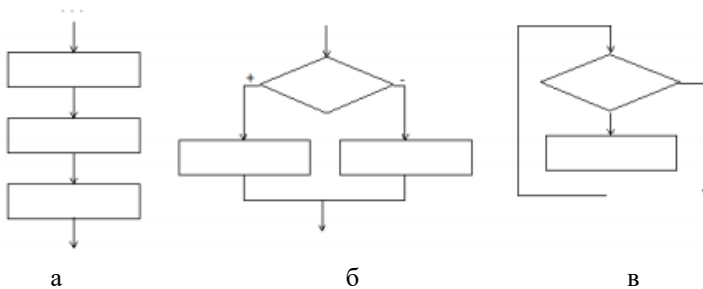


Fig.1.1. Linear structure (a), branching (b) and cyclic structure (c)

Algorithms in which actions take place one after the other are called linear algorithms.

An algorithm in which the action is performed along one of the possible branches of the problem solution, depending on the fulfillment of the conditions, is called branched. In branched algorithms, there is a condition, depending on the fulfillment or non-fulfillment of which a certain sequence of commands (actions) is executed. As a condition in a branched algorithm, any statement that can be understood by the executor can be used, which can be true or false. Such a statement can be expressed both in words and in a formula.

An algorithm in which some part of the operations (the body of the loop – a sequence of commands) is performed repeatedly is called a cyclical algorithm. The organization of cycles that never lead to a stop in the execution of the algorithm is a violation of the requirement of its effectiveness - obtaining a result in a finite number of steps.

Creating executable program code

Writing a program involves the execution of a certain number of actions, which can be divided into the following most important stages with more or less detail:

- problem statement;
- choosing a method of solving the problem;

- writing the source text of the program in C and C++;
- entering the source text of the program using a text editor; the text can be divided into several files (modules); at this stage, we receive source text files with the extension .c or .cpp;
- compilation of modules (each module separately or all modules together); at this stage, we receive an object file, that is, a file with the extension .obj;
- program syntax debugging;
- combining compiled modules into a program (this is often called program linking or linking); at this stage, the necessary standard libraries are added to the program and we get an executable file with the extension .obj);
- launching the program for execution;
- program debugging (program and algorithm testing);
- final design of the program.

When performing each of the specified stages of programming, there is a need to return to the previous stages, sometimes up to changing the formulation of the problem. Modern programming systems allow you to conveniently move from one stage of writing a program to another. This is done by the presence of the so-called integrated programming environment, which includes a text editor, compiler, linker, built-in debugger and, depending on the system or its version, gives the programmer additional conveniences for writing and debugging programs.

Some implementations of the C++ language, for example, Microsoft Visual C++, Borland C++, are implemented in the form of integrated development environments that allow you to perform all the above-described stages of creating an executable file in automatic mode.

Let's consider several free integrated development environments (Integrated development and Learning environment, IDLE) that can be used to develop programs in C/C++ languages:

– **Visual Studio.** An integrated development environment (IDLE) developed by Microsoft. You can download the latest version of the environment at <https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&channel=Release&version=VS2022&source=VSLand>

ingPage&passive=false&cid=2030

– **Dev-C++**. The distribution includes the MinGW compiler. The original version was developed by Bloodshed Software. At the moment, the continuation of the development is carried out by the Orwell company. You can download the latest version of the environment at <http://sourceforge.net/projects/orwelldevcpp/>;

– **Eclipse CDT**. An integrated environment based on the Eclipse platform. You can download the latest version of the environment at <https://eclipse.org/cdt/downloads.php>.

– **MinGW Codeblocks and GCC Compiler**.

Declaration of variables

Data in the program can be divided into variables and constants. Before use, variables and constants must be declared using the declaration statement.

A variable is a named area of memory into which values of a declared type are written during program execution. The variable is declared as follows:

type_of_variable name_of_variable;

```
| int a;  
| float g, sum;
```

A data type is understood as a set of permissible values of this data and a set of permitted operations on them. At the same time, the data type determines the size of memory occupied by variables and constants of this type.

Service words are used to describe the main types of the C++ language:

- int (integer);
- char (symbolic);
- bool (logical);
- float (valid);
- double (valid with double precision);
- void (empty, has no value).

The int, char, and bool types are called integers, and the float and double types are called floating-point values. The code that generates the compiler to handle integer values is different from the code for floating-point values.

C++ uses four type specifiers to specify the internal representation and value range of standard types:

- short;
- long;
- signed;
- unsigned.

Data types used in C++ are listed in Table 1.2.

Table 1.2 – Data types in C++

Type	Size memory, bytes	Range of values
[signed] char	1	-128...127
unsigned char	1	0...255
[signed] short [int]	2	-32768...32767
unsigned short [int]	2	0...65535
[signed] int	4	-2147483648...2147483647
unsigned int	4	0... 4294967295
[signed] long [int]	4	-2147483648...2147483647
unsigned long [int]	4	0...4294967295
float	4	3.4e-38...3.4e38
double	8	1.7e-308...1.7e308
long double	10	3.4e-4932... 3.4e4932

The order of calculation of the expression is determined by the arrangement of operation signs, round brackets and priorities of operation execution. Expressions with the highest priority are evaluated first.

The symbol "=" means a binary simple assignment operation, as a result of which the value of the right operand is assigned to the left operand:

name_of_variable = expression;

An example of declaring variables and constants:

```
int a=1, b;
const float g = 8.1;
```

Arithmetic operations

Arithmetic operations of the C++ language include:

- subtraction and unary minus;
- + addition;
- multiplication;
- / division;
- % % division by modulo;
- ++ increase by one (increment);
- decrease by one unit (decrement).

The operations of addition, subtraction, multiplication and division work in the same way as in most other algorithmic languages. They can be applied to all built-in data types. Operations are performed from left to right, that is, the value of the left operand is calculated first, then the value to the right of the operation sign. If the operands have the same type, then the result of the arithmetic operation has the same type. Therefore, when the division operation / is applied to integer variables or character variables, the remainder is discarded. So, $11/3$ will be equal to 3, and the expression $1/2$ will be equal to zero.

The operation of division modulo % gives the remainder from integer division. The % operation can only be applied to integer variables. The following example calculates the integer part and the remainder of the division of two integers.

```
#include <stdio.h>
main() {
    int x, y;
    printf("Введіть ділен і дільник:");
    scanf("%d%d", &x, &y);
    printf("\nЦіла частина %d\n", x/y);
    printf("Залишок від ділення %d\n", x%y);
}
```

The C language gives the user two more very useful operations specific to the C language. These are the unary operations ++ and --. The ++ operation adds one to the operand, the -- operation subtracts one from the operand. Both

operations can be before the operand or after the operand (prefix and postfix forms). The three operators written below give the same result, but differ when used in expressions:

```
x = x + 1 ; ++x ; x++.
```

A simple program will allow you to understand this difference.

```
#include <stdio.h>
main()
{
  int x=5;
  int y=60;
  x++;
  ++y;
  printf("x=%d y=%d\n", x, y);
  printf("x=%d y=%d\n", x++, ++y);
}
```

The result of this program will be the following:

```
x=6,y=61;
```

```
x=6, y=62.
```

Notice that the printed value of x has not changed on the second call to printf(), but the value of y has increased by one. In fact, the value of the variable x also increased by one, but after exiting the printf()/ function. The difference in the use of prefix ++x and postfix x++ forms is as follows:

x++ - the value of variable x is first used in the expression and only then the variable is increased by one;

++x - The variable x is first incremented by one, and then its value is used in the expression.

The priority of arithmetic operations is as follows:

1. ++, --
2. - (unary minus)
3. *, /, %
4. +, -

Operations of the same seniority are performed in order from left to right. Of course, in order to change the order of operations, parentheses can be used.

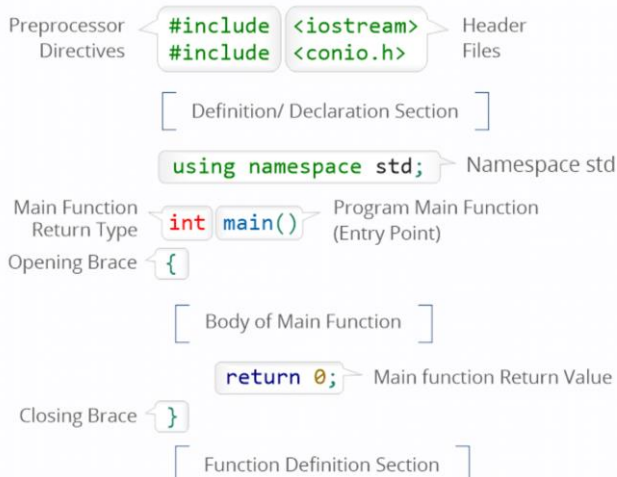
Mathematical functions of the C++ language

To use mathematical functions in C and C++ programs, you need to include the header file `<math.h>`. The main functions of this library:

- `cos(x)` – cosine;
- `acos(x)` – arccosine;
- `exp(x)` – exponent;
- `log(x)` – natural logarithm;
- `log10(x)` – decimal logarithm;
- `round(x)` – returns a value rounded to an integer (the returned value is a floating-point value);
- `floor(x)` – rounding to the nearest smaller integer;
- `ceil(x)` – rounding to the nearest larger whole number;
- `pow(x, y)` – elevation of x to the power of y ;
- `sin(x)` – sine;
- `asin(x)` – arcsine;
- `tan(x)` – tangent;
- `atan(x)` – arctangent;
- `sqrt(x)` – square root;
- `fabs(x)` – absolute value for floating-point numbers;
- `random(x)` – outputs a random number from 0 to the value of the argument.

Examples of programs in the language C++

Every C++ program must include the main function `main()`. This function is the starting point of entry into the program. The main structure of the C++ program is shown in the figure:



1.3. Work program

1.3.1. Familiarize yourself with the basic theoretical information on the topic of the work, using the given methodological instructions, lectures, as well as the recommended literature.

1.3.2. Install Visual Studio or Dev-C++

1.3.3. Following the installation points, install the environment on the PC.

1.3.4. Configure the compiler according to the tasks.

1.3.5. According to your number in group list (Variant number), write two programs (tasks 1 and 2) in the selected environment and run them for execution.

1.3.6. Prepare a report on the work done.

Report requirements

The report should include:

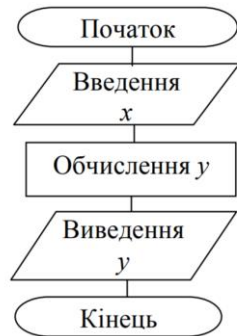
- Title page indicating the variant number
- Purpose of work
- Results of execution of individual tasks: block diagrams of algorithms, program texts, results of execution of the program copied from the monitor
- Answers to questions

An example of task 1. Develop an algorithm scheme and create a software project for calculation

$$y = \frac{0.2x^2 - x}{(\sqrt{3} + x)(1 + 2x)} + \frac{2(x-1)^3}{\sin^2 x + 1}$$

where x is an arbitrary variable to be entered.

```
# include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;
int main () {
    setlocale(0, ".1251");
    double y, x;
    cout<<"Введіть x= ";
    cin>>x;
    y=(0.2*x*x-x)/((sqrt(3)+x)*(1+2*x))+2*pow(x-1,3)/(pow(sin(x),2)+1);
    cout<<"y= "<< y;
    return 0;
}
```



1.4. Hardware and software

1.4.1. Personal computer.

1.4.2 Software: Visual Studio or Dev-C++

1.5. Questions

1.5.1. What is the basic structure of a C++ program?

1.5.2. What is the main function for?

1.5.3. How to create executable program code?

1.5.4. What is a variable?

1.5.5. How is a variable declared?

1.5.6. What is a constant?

1.5.7. What types of data do you know?

1.5.8. What memory size corresponds to the data types you know?

1.5.9. What type modifiers are there?

1.5.10. What is IDLE? What are the modern IDLEs?

1.5.11. How to start the program for execution?

1.5.12. How to write a comment in the program?

1.5.13. Do spaces and empty lines affect the execution of the program?

1.5.14. What function must the program contain?

1.5.15. What are the printf() and scanf() functions for?

Task 1

Variant number	Task	Variant number	Task
1	$z = \frac{e^{-x} - 12.34}{\lg x - \cos x}$	16	$y = \frac{e^{-x} + x - 1}{ \cos x + \sqrt{\cos x}}$
2	$y = \frac{\sqrt{x} - tgx}{\arccos x + \ln x}$	17	$y = \frac{e^{-2x}}{x + 1 + tgx}$
3	$g = \frac{\sin x^2 - \cos(x-1)}{\arctg x + \ln x}$	18	$y = \frac{x + \lg(x+1)}{\cos x}$
4	$p = \frac{e^x + 3x - 3}{\sin x + \sqrt{\cos x}}$	19	$u = \ln 1-x - \sin^2 x$
5	$y = \frac{e^{-x} - 4x}{\lg x}$	20	$y = e^{-x} + tg(x-1)$
6	$y = \arcsin\left(\frac{x}{1 + \cos x}\right)$	21	$c = \frac{2 + \ln x}{tgx}$
7	$y = x + \frac{\sin x + 4}{\ln x}$	22	$y = x^2 + x \cdot \lg x$
8	$y = tgx + \frac{e^x - \sqrt{x}}{tg^2 x}$	23	$y = \frac{x + \lg(x+1)}{\cos x}$
9	$u = \frac{\sin^2 x}{\sqrt{\ln(x-1)}}$	24	$y = \sin^3 x - \cos(x-1)^2$
10	$y = e^2 \cdot \lg x^4$	25	$c = 4 - \frac{\ln(x-1) + \ln x}{\sqrt{\cos(x-\pi)}}$
11	$y = \frac{e^{-x} - 4}{\cos x+1 }$	26	$y = \frac{\sin(x+1)}{\lg x} - 1$
12	$c = \sin^2 x - \frac{x}{\cos^3 x}$	27	$y = \sin^2(x-1)$
13	$y = \frac{x - \lg x}{1 + \cos 3x}$	28	$y = \frac{x \cdot \ln x - 2\sqrt{x}}{1 + \cos x}$
14	$c = \frac{\ln x + 5}{\cos x + 4x}$	29	$y = \frac{e^{x-1} - (x+1)^5}{\lg x}$

15	$y = \frac{\cos(x-1)}{\ln^2 x}$	30	$y = \frac{(x-1)^2}{\cos x + \sqrt{x}}$
-----------	---------------------------------	-----------	---

Task 2

Variant number	Task
1	$y = \frac{x^2 - z^2}{\lg x-7 }, x = \frac{\sin a}{\ln(a+b)-1}, z = \sqrt{\frac{a+b}{ab}}, a=3,5, b=1;$
2	$z = \ln \left \frac{x\sqrt{x} + \cos y}{1-y} \right , x = \frac{(a-b)^2}{b}, y = \cos a + ab, a=0,2, b=7;$
3	$y = -\sqrt{ x-z +1}, x = \frac{e^{-a} + \sin a}{2ba}, z = \frac{\arctg(b-a)+b}{1+a}, a=1,2, b=3;$
4	$z = c \cdot e^{-x+y} - c \cdot x, x = \frac{\lg(\alpha+2)}{\tg \alpha} + 0,1, y = \frac{\sin \alpha - \ctg \frac{c}{4}}{\ln \alpha + \ln c}, c=4,5, \alpha=2;$
5	$z = \frac{ x-1 + e^{-y}}{\lg x}, y = 2a(a+b), x = \frac{1+b}{\sqrt{a+b}}, a=1,75, b=0,4;$
6	$p = \frac{e^{-xy} + 4}{\sin xy}, x = a^2 + b^2, y = \frac{1}{b}, a=-2,4, b=0,87;$
7	$r = \frac{x+y}{(x-y)^2} + 1, x = \sin(ab), y = \ln a-b , a=1,8, b=-0,6;$
8	$\varphi = \frac{x^2}{1,3} + \ln y , x = k+6, y = \ln k + \lg m, k=14, m=2,2;$
9	$\alpha = \frac{x^3}{\tg(y-1)}, x = a + \frac{1}{b}, y = \frac{b}{2} + \sin b, a = \frac{1}{2}, b=1,4;$
10	$t = (m-y) + \cos my, m = \sqrt{ x+a }, y = a \cdot \sqrt{\sin x}, x=3, a=-1,7;$
11	$\varepsilon = \lg x - \sqrt{ y+1 }, x = (a-1)^2 + \cos b, y = a + \tg^2 b, a=1, b=-4;$
12	$\gamma = \tg \frac{x+1}{y-2} + k+x , x = \sqrt{m+n}, y = \frac{km-3}{6}, m=3, n=-2,2, k=0,8;$
13	$j = -m + \left \frac{\pi}{5} - y \right , x = \frac{2}{m} + mn, y = \sqrt{ m-3 } + \ln n, m=-2, n=3,87;$

14	$f = \frac{x-0,12}{y-1}$, $x=e^2+2$, $y=a-\sin a$, $a=6,45$;
15	$a=\sqrt{ \pi-y }+\sin x$, $y=(\beta-1)^2-0,3$, $x=\cos\alpha+\frac{\alpha}{2}$, $\alpha=4,4$, $\beta=1,87$;
16	$n=\sin x+tgy$, $x= \alpha+2 -\beta-3$, $y=\sin(\alpha-\beta)$, $\alpha=5$, $\beta=-0,1$;
17	$b=(\beta+z)^2+z+0,1$, $\beta=\lg k+x $, $z=2x-12,47$, $x=0,3$, $k=4$;
18	$y=ax^3+e^z$, $x=tg\frac{z}{a}+z$, $z=\sqrt[3]{a+1}$, $a=2$;
19	$t=\frac{x^2-y^3}{x+y}$, $x=\sqrt{1+z}$, $z=\sqrt{ 1+\sin y }$, $y=-1$;
20	$m= y-5,5 +\sin\frac{y}{4}$, $y=\ln x $, $x=\sqrt{e^2+3,41}$;
21	$g=e^z+\ln z$, $z=\sqrt{x+6}$, $x=21,4(\alpha-0,5)^2-\cos\alpha$, $\alpha=0$;
22	$t=\frac{m-y+\cos my}{m+y+17,14}$, $y=2m+\sqrt{e}$, $m=2,7$;
23	$\varphi=\cos\frac{x^2}{\pi}+y$, $x=e^2-1$, $y=\lg a+3a$, $a=2,3$;
24	$j=x^{-m}+\left \frac{\pi}{4}-y\right $, $x=\frac{n}{m}$, $y=\sqrt{ m-3 }+\ln n$, $m=-2$, $n=2,4$;
25	$i=\frac{\sqrt{m^3+2,5x}}{1-m}$, $x=\cos\frac{\pi}{y}-2$, $y=3m$, $m=1,4$;
26	$a=\gamma\cdot\sqrt{y+0,1}+\sin x$, $y=tg(x-1)$, $x=\lg \gamma+2 +1,4$, $\gamma=-3,6$;
27	$d=\sqrt{\sin(x-1)+\cos\gamma}$, $x=1+tg\gamma$, $\gamma=3,41$;
28	$y=\sin 2x+tgy$, $x=\ln \alpha-2 -\lg y+2 $, $y=e^{-\alpha}+\frac{\pi}{8}$, $\alpha=4,45$;
29	$h=\sin^2\left(\frac{x-y}{8\pi}\right)$, $x=y^2+0,15$, $y=1$;
30	$z=\lg x+1 -\ln 2^x-1 $, $x=e^k+\cos^2 ky$, $y=\sqrt{ k-e }$, $k=1,7$

№2. Development of programs of a branched structure

2.1. The purpose of the work

Learn to write programs with a branched structure. Learn the types of cyclic algorithms and cyclic operators of the C++ language.

2.2. Brief theoretical information

Relational operations and logical operations

Relational operations are used for comparison. The complete list of relational operations in the C language is as follows:

< less,

<= less than or equal to

> more

>= greater than or equal to

== is equal to

!= not equal.

There are also three logical operations in the C language:

&& and (AND),

|| or (OR),

! not (NOT).

Relational operations are used in conditional expressions, or, for short, conditions. Examples of conditional expressions:

a<0, 101>=105, 'a'=='A', 'a'!='A'

Each conditional expression is checked whether it is true or not. More precisely, it should be said that each conditional expression receives the value "true" ("true") or "incorrect" ("false"). There is no boolean type in the C language. Therefore, the result of a logical expression is an integer arithmetic value. In C, "true" is a non-zero value, "false" is zero. In most cases, the non-zero "true" value is one. So, from the above examples of conditional expressions, the 2nd and 3rd received the value "zero", and the 1st and 4th - non-zero values. Let's consider an example.

```

#include <stdio.h>
main()
{
    int tr, fal;
    tr = (101<=105); /*вираз "істинний" */
    fal = (101>105); /*вираз "невірний" */
    printf("true - %d false - %d\n", tr, fal);
    return 0;
}

```

Logical operations in the C language correspond to the classical logical operations AND(&&), OR (||) and NOT (!), and their result corresponds to the corresponding tables, which are commonly called truth tables:

X	Y	X AND Y	X OR Y	NOT X	X XOR Y
1	1	1	1	0	0
1	0	0	1	0	1
0	1	0	1	1	1
0	0	0	0	1	0

The XOR operation is called the exclusive-or operation. In the C language, there is no sign of the logical operation XOR, although it can be implemented using the operations AND, OR, and NOT. However, in the future we will consider bitwise operations, among which the "which excludes or" operation already exists.

Relational operations and logical operations have a lower priority than arithmetic operations. This means that the expression $12 > 10 + 1$ is treated as the expression $12 > (10 + 1)$.

The priority of logical operations and relational operations is as follows:

1. !
2. >, <, >=, <=
3. ==, !=
4. &&
5. ||

Boolean expressions, like all other expressions, can use parentheses with the highest priority. In addition, parentheses allow you to make logical expressions more clear and easy to read. Conditional and logical expressions are used in

control operators of the C language, such as `if` for and others. A feature of logical operations `&&` and `||` lies in the fact that if the value of the left operand (expression1) is zero when calculating the result of the operation (expression 1) `&&` (expression 2), then the value of the second operand will not have any effect on the result of the operation. In this case, the second operand is not evaluated. And therefore, the hopes that the calculation of the second operand may increase some variable due to the `++` operation are not justified. The same applies to the operation `||`. In this case, the second operand is not evaluated if the value of the left operand is not zero.

Assignment operation

The assignment operation in the C language is denoted simply by the sign `=`. Unlike other languages, C can use the assignment operator in expressions that also contain comparison operators or logical operators. In the fragment

```
if ((f=x-y)>0) printf ("Число x, больше чем y)
```

first, the size of `x-y` assigned to the variable `f` is calculated, then its value is equal to zero. Another feature of using the assignment operator in the C language is the ability to write the operator:

$$a=b=c=x*y$$

Such multiple assignment is commonplace in C and is done from right to left. First, the value of `x*y` is calculated, then this value is assigned to `c`, then `b`, and only then `a`. An expression to which a value can be assigned must be on the left side of the assignment operator. Such an expression in the C++ language, for example just a variable, is called an lvalue. The expression `2=2` is false because no value can be assigned to a constant: a constant is not the size of an lvalue.

C has additional assignment operations `+=`, `-=`, `/=`, `*=` and `%=`.

Instead of the expression `n = n + 5`, you can use the expression `n += 5`. Here `+=` is an additive assignment operation, as a result of which the dimension on the right is added to the value of the variable on the left.

Similarly

`m-=20` is the same as `m=m-20`;

`m*=20` is the same as `m=m*20`;

`m/=10` is the same as `m=m/10`;

`m%=10` is the same as `m=m%10`.

These operations have the same priority as the assignment operation =, that is, lower than the priority of arithmetic operations. There are a few additional assignment operations that we'll mention below. The operation $x+=5$ is performed faster than the operation $x=x+5$.

Operation condition ? :

The condition operation is the only C language operation that has three operands. This operation has the form:

$(\text{expression1})? (\text{expression2}): (\text{expression3})$

The expression (expression1) is calculated. If this expression has a non-zero value, then expression (expression2) is calculated. The result of the operation will be the value of the expression (expression2).

If the value of the expression (expression1) is zero, then the expression (expression3) is calculated and its value will be the result of the operation. In any case, only one of the expressions is calculated: (expression2) or (expression3). For example, it is convenient to use the condition operation to find the largest of two numbers x and y:

$\text{max} = (x > y) ? x : y ;$

or to find the absolute size of the number x:

$\text{abs} = (x > 0) ? x : -x ;$

If the second and third operands are sizes of the value type, that is, they can be in the left part of the assignment operation, then the result of the condition operation is also a size of the value type. With the help of this operation, you can solve the problem in one line: replace the largest of the numbers x or y with the value 1:

$(x > y) ? x : y = 1 ;$

Coma operation

The comma operation has the lowest priority of all C and C++ operations. The comma operation is performed from left to right, and its value is the value of the right operand. In the expression (expression1), (expression2) the value (expression1) will be calculated first, then the value (expression2). This value will be the result of the operation.

Sizeof operation

The sizeof operation has two forms: sizeof (type) or sizeof (expression).

The result of this operation is an integer value of type size or an expression in bytes. When using the second form, the value of the expression is not calculated, but only its type is determined. Usage examples: sizeof (short int) or sizeof (x). The void type cannot be used as a type in the sizeof operation.

Operations. and -> will be defined below.

Next, we will see that some operation signs have several meanings. So, the & operation sign has two meanings: a binary bitwise AND operation and a unary operation of taking an address.

Controlling operators

We already know the simplest form of two such operators: if and for. Let's consider these and other operators in more detail. Control operators can be divided into three categories:

1. Conditional statements if, if-else and switch.
2. Loop operators for, while and do-while.
3. Goto unconditional transition operator.

Conditional statement if

The full form of the operator is as follows:

```
if (condition) operator;  
else operator;
```

If the value of the condition is "true", then the operator (it can be a compound operator - block) that follows the condition is executed. If the condition takes the value "false", then the statement following the else keyword is executed. In the statement of the if statement, the second part (that is, the else statement) may be missing. Then, if the condition takes the value "false", the next statement of the program is executed immediately. In fact, an arbitrary expression can be used as a condition. The if statement only checks whether the value of this expression is non-zero (true) or null (false). With the help of the if

operator, you can, for example, calculate the value of the function $\text{sgn}(x)$ - the sign of x . The function $\text{sgn}(x)$ takes the value 1 if $x > 0$, the value -1 if $x < 0$, the value 0 if $x = 0$.

```
#include <stdio.h>
int main( ) {
    int sgn;
    float x; printf("Введіть число:"); scanf("%f",&x);
    if(x>0) { sgn=1;printf("число %f позитивне sgn");}
    if(x==0) { sgn=0;printf("число %f дорівнює нулю sgn");}
    if(x<0) { sgn= -1;printf("число %f негативне sgn");}
    return 0;}

```

It is often necessary to use the if-else-if construction:

```
if (condition) operator;
else if (condition) operator;
else if (condition) operator;
.....
else operator;
```

In this form, the conditions of the if statements are checked from top to bottom. As soon as one of the conditions takes the value "true", the statement that follows this condition will be executed, and the rest of the construction will be ignored. The if statements of the previous example can be written in another form:

```
#include <stdio.h>
int main( ) {
    int sgn;
    float x;
    printf(" Введіть число:"); scanf("%f", &x);
    if (x>0) { sgn=1;printf("Число %f позитивне \n", x); }
    else if(x<0) (sgn= -1;printf("число %f негативне \n", x); }
    else (sgn=0;printf("число %f дорівнює нулю \n", x); }
    return 0;}

```

As we have already said, some expression can be used as a condition of the if operator. So, in order to check whether the number x is equal to zero or not, you can write

```
if(x=0) printf("Число дорівнює нулю");  
else printf("Число не дорівнює нулю");
```

The same result can be obtained by the following operator:

```
if(!x) print("Число дорівнює нулю");  
else print("Число не дорівнює нулю");
```

The following construction is called a nested if statement:

```
if(x)  
if (y) operator1;  
else operator2;
```

In this form, it is not clear which of the if statements else refers to. In the C language, the else operator is associated with the nearest if in the corresponding block. In the last example, else refers to if(y). In order to attribute else to the if(x) operator, you need to arrange the operator brackets accordingly:

```
if(x)  
{  
if (y) operator 1;  
}  
else operator p2;
```

Now if(y) refers to another block.

Operator SWITCH

The C++ language has a built-in multiple choice operator called switch.

```
switch (expression)  
{  
case constant 1: sequence of operators
```



```

break;
case constant2: sequence of operators
break;
.....
case constantN: sequence of operators
break;
default sequence of operators
}

```

First, the expression in parentheses for the keyword is calculated, then the list of labels (case constant 1, etc.) is reviewed until a label corresponding to the value of the calculated expression is found. Next, the appropriate sequence of operators following the colon is executed. If the value of the expression does not correspond to any of the labels of the switch operator, then the sequence of operators following the keyword default is executed.

The construction of the switch operator is allowed when the word default and the corresponding sequence of operators may be absent.

Another previously unknown operator is break. When the keyword break occurs after a sequence of statements, the execution of the break statement leads to exiting the switch statement and moving to the next statement of the program. The presence of the break statement in the switch statement is optional. What will happen if there are no break statements? The answer to this question will be provided by the results of the following two versions of the program:

```

#include <stdio. h>
int main( ) {
char ch;
printf("Введіть заголовну літеру алфавіту.");
ch=getchar( );
if(ch>='A' && ch<='Я')
switch(ch)
{
case 'A': printf("Alex \n"); break;
case 'B': printf("Bob \n"); break;
case 'B': printf("Itan \n"); break;
case 'T': print("Max\n"); break;

```

```
default: printf("Other \n"); break;}  
else printf("Wrong letter \n");  
return 0;}
```

Suppose you run the first program and enter the letter B. The result of the program will be the following line:

Bob

We can see that all statements, starting with the label 'B', including the one following the word default, have been executed.

The break statement ends the sequence of statements related to each label. Then the next statement of the program is executed. If there is no break statement, then execution continues to the first break statement or to the end of the switch statement.

Loops

Loops are necessary when we need to repeat some action several times, usually while some condition is met. In the C++ language, three types of loop operators are known: for, while, and do-while.

The for loop

The basic form of the for loop has the following form:

for (initialization ; condition check ; change) operator;

In its simplest form, initialization is used to assign an initial value to a loop parameter. A condition check is usually a conditional expression, used to change the loop parameter each time the loop is repeated. These three loop header divisions must be separated by a semicolon. The loop is executed until the conditional expression is true. As soon as the condition becomes false, the next statement in the for loop starts executing. The simplest example of the for loop statement:

```
for(i=0; i<10; i++) printf("%d\n", i);
```

As a result of the execution of this statement, the numbers from 0 to 9 will be printed in the column. To print these numbers in reverse order, you can use the following statement:

```
for(i=9; i>0; i++) printf("%d\n", i);
```

The for loop is similar to similar loops in other programming languages, and at the same time, this statement in C is much more flexible, powerful, and applicable in many situations.

It is not necessary to use an integer counter as a loop parameter. Here is a fragment of the program that displays the letters of the alphabet on the screen:

```
unsigned char ch;  
for (ch='A'; ch<="Я"; ch++) printf("%c", ch);
```

The next fragment of the program

```
for(ch='0'; ch! ='N'); scanf("%c", &ch);
```

will be executed until the character 'N' is entered from the keyboard. Note that the place where the increment should be is empty. A loop from which there is no way out, a so-called endless loop, can be formed accidentally or intentionally.

Here are three examples of such cycles.

```
for (;)printf{ " An endless cycle \n";  
for(i=l;l;i++) printf("An endless cycle \n");  
for (i=10;i>6;i++) printf( "An endless cycle \n");
```

However, for such cycles, an exit can also be arranged. For this, the break operator, which we met above, is used. If the break operator is found in the component statement of the loop, then the execution of the loop is immediately terminated and the execution of the next statement of the program begins.

```
#include <stdio. h>  
main( ) {
```

```

    unsigned char ch;
    for(;;) {
        ch=getchar( ); /*Read symbol */
        if(ch=='Q') break; /*Check symbol */
        printf("%c", ch); /*Print symbol */
        ...
    }
    return 0;}

```

This program will print the characters entered until the character 'Q' is entered. Perhaps it is syntactically correct, the presence of an empty operator (absence of an operator) in the for loop.

For example *for (i=0;i<10000;i++)*;

While and do-while loops

The next loop statement in C++ is a while loop. The basic form has the following form: while (condition) operator; where the operator can be a simple, compound, or empty operator. A "condition", like all other statements, is just an expression. The loop is executed until the condition evaluates to true. When the condition becomes false, the program will transfer control to the next program statement. Just like in the for loop, in the while loop, the condition is first checked, and then the operator is executed. This is the so-called cycle with a premise.

Unlike previous loops in the do-while loop, the condition is checked at the end of the loop statement. The basic form of the do-while statement is as follows:

```

do{ sequence of operators }
while (condition);

```

Curly brackets are optional if there is one operator inside them. However, they are most often put for better readability of the program, as well as not to confuse (the programmer, not the compiler) with the while statement. The do-while statement is called a loop statement with a postcondition. No matter what the condition is at the end of the statement, the set of statements in curly brackets must be executed one (first) time. In for and while loops, the

statement may not be executed once.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
/* Гра " Вгадай число " Програма вибирає випадкове число від 1 до
100, ви повинні вгадати його */
main( ) { int s, x, n=0;
randomize( );
s=random(100)+1;
do
{ printf("Введіть число від 1 до 100: ");
scanf("%d", &x);
n++;
if(s<x) print("Загадане число менше\n");
if(s>x) printf("Загадане число більше\n");
} while (s-x);
printf("Ви вгадали число ! \n");
printf("Затратили на вгадування %d спроб\n", n);
return 0;}
```

Nested loops

When one loop is inside another, it is said to be a nested loop. Nested loops are often found, for example, when filling tables. As an example, consider a program for printing a multiplication table of integers..

```
#include <stdio.h>
int main( )
{ int i, j;
for(i=0;i<10;i++) {
for(j=1;j<5;j++) printf("%d * %d = %2d ", i, j, i*j);
printf("\n");
...
}
```

Operator break

The break operator has two uses. The first is the end of the case in the switch statement. The second is the immediate end of the cycle, not related to checking the usual condition of the end of the cycle. When the break statement occurs inside a loop statement, the loop is immediately exited and the statement following the loop statement is executed.

```
#include <stdio. h>
main( ) {
    int i;
    for(i=0;i<1000;i++)
    {
        printf(“%d-%d \n”, i, i*i*i);
        if (i*i*i>= 10000) break;
    }
}...
```

Operator continue

Another useful operator is the continue operator. If the continue statement is encountered in a loop statement, it transfers control to the beginning of the next iteration of the loop. In the while and do-while loops - to check the condition, in the for loop - to increase. This statement is necessary if you want to end the current iteration of the loop and not execute the remaining statements, but immediately go to the next iteration of the loop. For example, it can be used in a program that prints natural numbers that are multiples of seven.

```
#include <stdio. h>
main( )
{ int i;
    for(i=0;i<1000;i++)
    {
        if(i%7) continue;
        printf(“%8d”, i);
    }
}
```

2.3. Work program

2.3.1. Start the IDLE environment.

2.3.2. Compose the algorithm of the program for finding the value of the function, which is calculated depending on the value of the argument, from task 1 according to your version. Develop a block diagram of the algorithm.

2.3.3. Write a program to calculate the value of the function and task 1.

2.3.4. Compose the algorithm of the program for calculating the value of the function (task 2) on the specified interval with the specified step, according to your option. Draw block diagrams of algorithms.

2.3.5. Write programs for calculating the values of the functions from task 2.

2.3.6. Compile the algorithm of the program for calculating the value of the function (task 3) on a certain interval with a given step of changing the argument according to your version. Draw a block diagram of the algorithm.

2.3.7 Compile a program to calculate the value of the function from task 3.

Program requirements:

Task 1: data should be entered from the keyboard, final values and intermediate variables should be displayed on the screen in a convenient form.

Task 2:

- input data (initial and final value of the argument, step of changing the argument) to be entered by the input operator;
- display the calculation result in the form of a table (column 1 - point number, 2 - argument value, 3 - function value);
- write 2 versions of the program: 1 – using the goto unconditional transition operator, 2 – using the loop operator.

Task 3:

- boundaries of the function definition area, the step of changing the argument, the initial value of the argument must be entered by the input operator;
- display the calculation result in the form of a table (column 1 - point number, 2 - argument value, 3 - function value).

2.4. Hardware and software

2.4.1. Personal compute.

2.4.2. Software: Visual Studio or Dev-C++

2.5. Questions

- 2.5.1. What arithmetic operations do you know?
- 2.5.2. Name the relation operations.
- 2.5.3. What logical operations are used in the C++ language?
- 2.5.4. What does the assignment operation look like in C++?
- 2.5.5. The IF operator: record form, purpose.
- 2.5.6. The switch operator: record form, purpose.
- 2.5.7. What types of cycles can you name?
- 2.5.8. Name the cycle operators of the C++ language known to you.
- 2.5.9. Write down the general form of the for loop statement.
- 2.5.10. Write down the general syntax of the loop while statement.
- 2.5.11. Reproduce the general record of the do-while loop statement.

Task 1

Variant number	Task	Task	Task
1	$y = \begin{cases} -\cos^2(x-\pi), & -\pi < x < \frac{\pi}{4} \\ \sqrt{ x+1 }, & \frac{\pi}{4} \leq x \leq 1 \\ \frac{1}{x-1}, & 1 < x \end{cases}$	16	$y = \begin{cases} x\sqrt{x-5}, & 4, 0 \leq x < 2 \\ \arctg x^2, & 2 \leq x < 8 \\ \lg x-7 , & 8 \leq x \end{cases}$
2	$y = \begin{cases} -\frac{1,4+x}{\ln x}, & 1 < x < 3, 2 \\ x^2 - 0,75, & 0 < x \leq 1 \\ \cos^3 x^2 - \sin^3 x^2, & x \leq 0 \end{cases}$	17	$y = \begin{cases} -\arctg \frac{x+\pi}{x^2}, & 0 < x \leq 1 \\ \ln x^3 , & 1 < x < 10 \\ e^{-x}, & x \geq 10 \end{cases}$
3	$y = \begin{cases} e^{-2,5x^3} + 1, & x < 0, x \neq -1 \\ \sqrt{ \lg x - \ln x }, & 1 < x \leq 5, 5 \\ \frac{x-1}{x - \sin^2 x}, & x = -1, x > 5, 5 \\ 2x, & 0 \leq x \leq 1 \end{cases}$	18	$y = \begin{cases} \sin e^x - 2, & x \leq 4 \\ \frac{x^2 - 1, 2}{x + 4}, & 4 < x < 10 \\ x, & x \geq 10 \end{cases}$
4	$y = \begin{cases} 2x^3\sqrt{x^2+z^2}, & 1 < x < 20, 4 \\ \arctg(x-z), & 0 < x \leq 1 \\ e^{x+z}, & x \leq 0 \end{cases}$	19	$y = \begin{cases} -\sqrt[3]{cx}, & c > 9, x < -1 \\ \ctg \frac{c}{x}, & c < 0, -1 < x \leq 1 \\ \ln c^2 - x^2 , & c > 0, x < c \end{cases}$

5	$y = \begin{cases} \left \lg \frac{\pi - x}{16} \right , 0 < x < \frac{1}{4} \\ (x^2 - 2, 0, 4)^{-3,14}, \frac{1}{4} < x < 1 \\ \arccos \frac{x}{4}, x \geq 1 \end{cases}$	20	$y = \begin{cases} \frac{x+a}{e^{xa}}, xa < 1, x < 0 \\ -\ln^2 x, 2 < x, a \leq 0 \\ \lg \sqrt{a}, 0 < a, 0 \leq x \leq 2 \end{cases}$
6	$y = \begin{cases} e^{- x }, 1 \leq x \\ \lg \sqrt{1-x^2}, x < 1 \\ \arctg x, x \leq -1 \end{cases}$	21	$y = \begin{cases} x^e - e^{-x}, x < 2 \\ \lg x^2, x \leq -2 \\ \sin^2 x, x \geq 2 \end{cases}$
7	$y = \begin{cases} 2^{x-1} + 2, 7, 1, \pi \leq x < 8, 5 \\ \sqrt{\pi - x }, 8, 5 < x < \pi \\ x - 2, 7, 8, 5 \leq x \end{cases}$	22	$y = \begin{cases} 0, x \leq -10 \\ \operatorname{ctg} \frac{x-1}{e}, -10 < x \leq 0 \\ \ln x, 10 \leq x \\ \sqrt{x^3}, 0 < x < 10 \end{cases}$
8	$y = \begin{cases} \sqrt[3]{\lg x + \ln x^2}, x > 1 \\ e^{-x} + 1, x \leq 1 \end{cases}$	23	$y = \begin{cases} e^{-x} + x^2 - 1 , x > 1 \\ \lg \sqrt{ 1-x }, -\pi < x \leq 1 \end{cases}$
9	$y = \begin{cases} \arccos \frac{\pi - x}{2}, x < -1 \\ e^{-x^2}, -1 < x < 1 \\ \pi \ln^2 x, x > 1 \\ 10^{-3}, x = 1 \end{cases}$	24	$y = \begin{cases} \sin(x+1), x \leq -1 \\ \operatorname{ctg} \frac{x-1}{e}, -1 < x \leq 0 \\ \ln x, 1 \leq x \\ \sqrt{x^3}, 0 < x < 1 \end{cases}$
10	$y = \begin{cases} x^3 \sqrt{x^2 + 3}, 1 < x < 10 \\ \arctg(x-9), 0 < x \leq 1 \\ e^{x+1}, x \leq 0 \end{cases}$	25	$y = \begin{cases} (-1)^{3x-1} x^2, x \geq 5 \\ \ln x^{-1}, 0 < x < 1 \\ \cos x-1 , 1 < x < 5 \end{cases}$
11	$y = \begin{cases} \arcsin(-x^2 + 1), x < 0 \\ \lg^2(2x) + 4, 4, 0 \leq x \leq 3 \\ \frac{1}{-e^x}, x > 3 \end{cases}$	26	$g = \begin{cases} \frac{\pi}{8} \sin^2 \left(\frac{x-y}{3} \right), x = 1 \\ y^{-e}, 1 < x < 5 \\ e^{-x}, 5 \leq x < 11 \\ 0, 15x, x > 11 \end{cases}$

12	$y = \begin{cases} \sqrt{\sin^3(x-1)}, -6 \leq x \leq 2 \\ e^{-x}, x \geq 6 \\ 4,4 \cdot \lg^3 x , 6 > x > 2 \\ \ln x^2, x < -6 \end{cases}$	27	$y = \begin{cases} \operatorname{arctg}(\pi x), 0 < x < \pi \\ \ln(x-3,18), 2\pi \leq x \\ \frac{1}{\sqrt{x-\pi}}, \pi < x < 2\pi \\ \pi, x \leq 0 \end{cases}$
13	$y = \begin{cases} e^{-\pi} + x^{-e}, 1 < x \leq e \\ \ln^2(x-e), e < x < 10^3 \\ \frac{x^2}{e}, 0 < x \leq 1 \\ 2^x \operatorname{ctg}^3 x^2, x < 0 \end{cases}$	28	$y = \begin{cases} e^{-x}, 1 < x < 2 \\ x^e + 1, 2 \leq x \leq 5 \\ 1, x < 1 \end{cases}$
14	$y = \begin{cases} \sqrt{x}, x > 0 \\ 2 - x^2, x \leq 0 \end{cases}$	29	$y = \begin{cases} \sin^2 x, x \leq -1 \\ \sqrt{-x}, -1 < x < 0 \\ x - \lg x, x > 1 \end{cases}$
15	$y = \begin{cases} \frac{\sin x}{x}, x > 0 \\ 2x^2 + \ln x , x < 0 \\ 0, x = 0 \end{cases}$	30	$y = \begin{cases} 124 - e^x, x < 1 \\ \operatorname{tg}(x-1), 1 \leq x \leq 10 \\ 1, x > 10 \end{cases}$

Task 2

Variant number	Task
1	$y = \frac{\cos^2 x}{x^2 + 1}, 3,8 \leq x \leq 7,6, \Delta x = 0,6;$
2	$y = \frac{\operatorname{tg} 0,5x}{x^3 + 7,5}, 0,1 \leq x \leq 1,2, \Delta x = 0,1;$
3	$y = \frac{e^{2x} - 8}{x + 3}, -1 \leq x \leq 2,3, \Delta x = 0,7;$
4	$y = \frac{x + \cos 2x}{3x}, 2,3 \leq x \leq 5,4, \Delta x = 0,8;$
5	$y = \frac{x + \cos 2x}{x + 2}, 0,2 \leq x \leq 10, \Delta x = 0,8;$
6	$y = \frac{\cos^3 t^2}{1,5t + 2}, 2,3 \leq t \leq 7,2, \Delta t = 0,8;$
7	$z = \frac{x^3 + 2x}{3 \cos \sqrt{x} + 1}, 0 \leq x \leq 2, \Delta x = 0,4;$

8	$z = \frac{t + \sin 2t}{t^2 - 3}$, $2,4 \leq t \leq 6,9$, $\Delta t = 0,4$;
9	$y = \frac{x^3 - 2}{3 \ln x}$, $4,5 \leq x \leq 16,4$, $\Delta x = 2,2$;
10	$z = \frac{2,3t + 8}{2 \cos t + 1}$, $0 \leq t \leq 6,5$, $\Delta t = 1,1$;
11	$y = \frac{\arccos x}{2x + 1}$, $0,1 \leq x \leq 0,9$, $\Delta x = 0,1$;
12	$y = \frac{5tg(x+7)}{(x+3)^2}$, $1,2 \leq x \leq 6,3$, $\Delta x = 0,2$;
13	$y = \frac{1,5t - \ln 2t}{3t + 1}$, $2,5 \leq t \leq 9$, $\Delta t = 0,8$;
14	$y = \frac{2,5x^3}{e^{2x} + 2}$, $0 \leq x \leq 0,5$, $\Delta x = 0,1$;
15	$y = \frac{3x - 2}{2 \arctg x + 1}$, $3,2 \leq x \leq 5,2$, $\Delta x = 0,4$;
16	$y = \frac{5 \lg x}{x^2 - 1}$, $1,2 \leq x \leq 3,8$, $\Delta x = 0,4$;
17	$z = \frac{6x + 4}{\sin 3x - x}$, $2,3 \leq x \leq 7,8$, $\Delta x = 0,9$;
18	$z = \frac{2 \sin^2(x+2)}{x^2 + 1}$, $7,2 \leq x \leq 12$, $\Delta x = 0,5$;
19	$y = \frac{(3x+2)^2}{\sin x + 3}$, $4,8 \leq x \leq 7,9$, $\Delta x = 0,4$;
20	$y = \frac{2 \sin^3 x}{3x + 1}$, $-1 \leq x \leq 1$, $\Delta x = 0,25$;
21	$y = \frac{tg 2t - 3t}{t + 3}$, $0,2 \leq t \leq 0,8$, $\Delta t = 0,1$;
22	$y = \frac{3x + 1}{\arctg x}$, $0,1 \leq x \leq 1,5$, $\Delta x = 0,2$;
23	$y = \frac{2t + 8}{\cos 3t + 1}$, $2 \leq t \leq 6,5$, $\Delta t = 0,8$;
24	$y = \frac{\arccos x}{3x + 1}$, $0,1 \leq x \leq 0,9$, $\Delta x = 0,1$;
25	$y = \frac{(x+2)^2}{\sqrt{x^2 + 1}}$, $2,3 \leq x \leq 8,3$, $\Delta x = 0,6$;

26	$y = \frac{t - \ln 2t}{3t + 1}, 2,1 \leq t \leq 8,5, \Delta t = 0,7;$
27	$y = \frac{x^2 + 2x}{\cos 5x + 2}, -2 \leq x \leq 4,5, \Delta x = 0,5;$
28	$y = \frac{\ln x+1 + 5}{2x + 3}, 0,2 \leq x \leq 0,9, \Delta x = 0,15;$
29	$y = \frac{x + \cos 2x}{3x}, 2,7 \leq x \leq 8, \Delta x = 0,7;$
30	$z = \frac{\arcsin 2x + x }{x^2 + 1}, 0 \leq x \leq 0,4, \Delta x = 0,2.$

Task 3

Variant number	Task
1	$w = \begin{cases} y + \sin y, & -6,5 < y < 0,5 \\ \ln(y + \sqrt[3]{y}), & 0,5 \leq y \leq 8; \Delta y = 0,5; \end{cases}$
2	$x = \begin{cases} 1,26^v + v, & 0 \leq v < 0,1 \\ \operatorname{arctg}(v + 0,4), & 0,1 \leq v < 4; \Delta v = 0,1 \end{cases}$
3	$f = \begin{cases} y + 0,1 \cos y, & -2 < y \leq 0,5 \\ \lg(y + \sqrt{y + 0,6}), & 0,5 < y \leq 3; \Delta y = 0,5 \end{cases}$
4	$y = \begin{cases} \sin x + e^x, & -2 \leq x \leq 0 \\ \operatorname{arctg}(x - 0,3), & 0 < x \leq 3; \Delta x = 0,5 \end{cases}$
5	$w = \begin{cases} z - \sin z, & -2 \leq z \leq 0,5 \\ \operatorname{arctg}(x - 0,3), & 0,5 < z \leq 3; \Delta z = 0,5 \end{cases}$
6	$v = \begin{cases} t + \cos t, & 0 \leq t \leq 0,5 \\ \operatorname{arctg}(t + \ln t), & 0,5 < t \leq 2; \Delta t = 0,3 \end{cases}$
7	$y = \begin{cases} \operatorname{arctg} x + e^x, & 0 \leq x \leq 0,5 \\ \ln(x + \sin x), & 0,5 < x \leq 8; \Delta x = 0,5 \end{cases}$
8	$w = \begin{cases} 0,3^v - v^2 + \cos v, & -3 < v \\ \operatorname{ctg}(0,34v - 0,2), & 1 < v \leq 7; \Delta v = 1 \end{cases}$
9	$z = \begin{cases} x^3 + \sin x, & 0 \leq x \leq 0,3 \\ \operatorname{arctg}(x + \ln x), & 0,3 < x \leq 2; \Delta x = 0,3 \end{cases}$

10	$w = \begin{cases} 0,6v - 0,3^v, & -2 < v \leq 0,3 \\ \ln(v + \sqrt{v + \cos v}), & 0,3 < v \leq 5; \Delta v = 0,5 \end{cases}$
11	$u = \begin{cases} x - 0,8 \sin x, & 0 \leq x \leq 2,2 \\ \operatorname{arctg}(\ln x + 0,3), & 2,2 \leq x \leq 3; \Delta x = 0,4 \end{cases}$
12	$v = \begin{cases} \cos z - z, & 0 \leq z < 0,5 \\ \ln(z + \sqrt{z}), & 0,5 < z \leq 7; \Delta z = 4; z \neq 4 \end{cases}$
13	$u = \begin{cases} 1,3t - \sin t, & -4 \leq t \leq -2 \\ \lg(t + \sqrt{t}), & -2 < t \leq 4; \Delta t = 0,5 \end{cases}$
14	$u = \begin{cases} 0,2t + \operatorname{arctg} t, & -2 \leq t \leq 0 \\ \arcsin(0,25t), & 0 < t \leq 5; \Delta t = 0,8 \end{cases}$
15	$y = \begin{cases} \operatorname{arctg} z + z, & -2 \leq z < 0 \\ \lg z + \sqrt{z}, & 0 < z \leq 5; \Delta z = 0,5 \end{cases}$
16	$r = \begin{cases} z + \cos z, & -1 < z \leq 0 \\ \operatorname{arctg}(z + \ln z), & 0 < z \leq 1; \Delta z = 0,4 \end{cases}$
17	$w = \begin{cases} v^2 + \sqrt[3]{v}, & 0 \leq v \leq 0,5 \\ \ln(v + \sin v), & 0,5 < v \leq 8; \Delta v = 0,5 \end{cases}$
18	$y = \begin{cases} x - e^x, & -2 \leq x < 2 \\ \operatorname{arctg}(x + \sqrt{x} - 1,4), & 2 \leq x \leq 5; \Delta = 0,5 \end{cases}$
19	$t = \begin{cases} 1,3y + \sin y, & 0 \leq y \leq 0,3 \\ \operatorname{arctg}(y + \sqrt{y}), & 0,3 \leq y \leq 2; \Delta y = 0,3 \end{cases}$
20	$x = \begin{cases} w + \cos w, & 0 \leq w < 0,5 \\ \operatorname{arctg} w - \lg(w + \sqrt{w}), & 0,5 \leq w \leq 2; \Delta w = 0,2 \end{cases}$
21	$t = \begin{cases} x^2 - e^x, & 0 \leq x < 0,4 \\ \ln(\operatorname{arctg} x + x), & 0,4 < x \leq 2; \Delta x = 0,2 \end{cases}$
22	$f = \begin{cases} t + e^x, & -0,5 \leq t \leq 0,5 \\ \sqrt{t} - \ln(t + \operatorname{arctg} t), & 0,5 < t \leq 4,5; \Delta t = 0,5 \end{cases}$
23	$u = \begin{cases} \operatorname{arctg} v - e^v, & 0 \leq v \leq 1 \\ \lg(v + \cos v), & 1 \leq v \leq 3; \Delta v = 0,5 \end{cases}$
24	$x = \begin{cases} \sin z - z^2, & 0 \leq z \leq 0,2 \\ \operatorname{arctg}(\ln z + \sqrt{z}), & 0,2 \leq z \leq 3; \Delta z = 0,2 \end{cases}$

25	$f = \begin{cases} v^2 - \sqrt[3]{v}, & -2 < v < 0 \\ \operatorname{arcctg}(v + \ln v), & 0 \leq v < 3; \Delta v = 0,5 \end{cases}$
26	$z = \begin{cases} x - \ln x, & 1 \leq x < 2 \\ x^2 - 2, & 2 \leq x \leq 5; \Delta x = 0,2 \end{cases}$
27	$y = \begin{cases} t^2 - \sin t, & -\pi \leq t < 0 \\ \sin t - t^2, & 0 \leq t \leq \pi; \Delta t = 0,1 \end{cases}$
28	$f = \begin{cases} x x^2 - 3 , & 0 \leq x < \sqrt{3} \\ \sqrt{x^4}, & \sqrt{3} \leq x \leq 10; \Delta x = 0,5 \end{cases}$
29	$y = \begin{cases} x^3 \cos x, & -\pi \leq x \leq 0 \\ x^2 \sin x^2, & 0 < x \leq \pi; \Delta x = 0,1 \end{cases}$
30	$z = \begin{cases} t - 5t^2, & 5 \leq t < 7 \\ \sqrt{t} + t^2, & 7 \leq t \leq 10; \Delta t = 1 \end{cases}$

№3. Development of programs with one-dimensional arrays.

3.1. The purpose of the work

To study the organization of data of the same type in the form of arrays in the C++ language, their declaration, methods of accessing elements and programming of processing algorithms.

3.2. Brief theoretical information

Pointers

Pointers are variables that store the addresses of another variable. Most often, these addresses indicate the memory location of other variables. For example, if the variable x contains the address of the variable y, then the variable x is said to "point" to y.

Pointer variables (or pointer type variables) must be declared accordingly. The format of a pointer variable declaration is as follows:

```
type *variable_name;
```

In this entry, the type element means the base type of the pointer, and it must be a valid C++ type. The variable_name element is the name of a pointer variable.

To understand what has been said, consider the following example. To declare the variable p as a pointer to an int value, use the following instruction: *int *p;*

To declare a pointer to a float value, use the following instruction:

```
float *p;
```

In general, using an asterisk (*) before a variable name in a declaration statement turns that variable into a pointer.

The type of data a pointer will refer to is determined by its base type. Let's consider another example:

```
int *ip; // A pointer to an integer value
```

```
double *dp; // Pointer to a value of type double
```

As noted in the comments for this program, the variable `ip` is a pointer to an `int` value because its base type is `int`, and the variable `dp` is a pointer to a `double` value because its base type is `double`. So, in the previous examples, the variable `ip` can be used to point to an `int` value, and the variable `dp` to a `double` value. However, remember: there is no real way to prevent a pointer from referencing "treasury-what". This is why pointers are potentially dangerous.

Operators of work with pointers

Two operators are used with pointers: "*" and "&". The "&" operator is unary. It returns the memory address at which its operand1 is located. For example, in the process of executing such a piece of program code

```
ptr = &balance;
```

the address of the `balance` variable is placed in the `ptr` variable. This address corresponds to the area in the computer's internal memory that belongs to the `balance` variable. Executing this instruction has no effect on the value of the `balance` variable. The purpose of the "&" operator can be "translated" in Ukrainian as the "address of the variable" before which it is located. Therefore, the above assignment instruction can be expressed as follows: "ptr variable gets address of balance variable". To better understand the essence of this assignment, let's assume that the variable `balance` is located in the memory area with address 100. Therefore, after the execution of this instruction, the variable `ptr` will acquire the value 100.

The second operator for working with pointers (*) serves as a complement to the first (&). It is also a unary operator, but it accesses the value of the variable located at the address given by its operand. In other words, it refers to the value of the variable addressed by the given pointer. If (continuing work with the previous assignment instruction) the `ptr` variable contains the address of the `balance` variable, then during the execution of the instruction

```
value = *ptr;
```


the value variable will be assigned the value of the balance variable pointed to by the ptr variable. For example, if the variable balance contains a value of 3200, then after the last instruction is executed, the variable value will contain the value 3200, because this is exactly the value that is stored at address 100. The purpose of the operator "*" can be expressed by the phrase "at address". In this case, the previous instruction can be read as follows: "variable value acquires the value (located) at the address ptr".

```
include <iostream>
#include <conio>
using namespace std;

int main() {
int balance;
int *ptr;
int value;
balance = 3200;
ptr = &balance;
value = *ptr;
cout << "Баланс дорівнює: " << value << endl;
getch(); return 0;
}
```

As a result of execution, this program displays the following results on the screen: The balance is: 3200

The multiplication sign (*) and the at-address operator are denoted by the same asterisk symbol. These operations are in no way related to each other.

The "*" and "&" operators have higher precedence than any of the arithmetic operators, except for the unary minus, which has the same precedence as the pointer operators.

When assigning a value to an area of memory addressed by a pointer, it (the pointer) can be used on the left side of the assignment statement. For example, in the process of executing such an instruction (if p is a pointer to an integer type)

```
*p = 101;
```

the number 101 is assigned to the memory area, in p, which is addressed

by the pointer. Thus, this instruction can be read as follows: "put the value 101 at the address p." To increment or decrement a value located in an area of memory addressed by a pointer, you can use an instruction similar to the following:

```
(*p)++;
```

The parentheses are required here because the "*" operator has lower precedence than the "++" operator.

Arrays

An array is a finite set of elements of the same type.

The k-dimensional array is declared according to the following format:

```
<type> <array name> [i1][i2]*...*[ik]
```

where the name of the array is the correct identifier, i1, i2, ..., in - the maximum number of array elements for each dimension.

Each index of the array must be written in separate square brackets []. The number of array indexes is unlimited. By default, the total size of the array should not exceed one segment (64K). To declare an array larger than 64K, the huge modifier is specified between the type and the array name.

The array index for each dimension varies from 0 to m-1, where m=1,...,k.

The array type can be specified as:

- 1) one of the main types (int, char, float, double or synonyms);
- 2) another array type;
- 3) a pointer, including a pointer to a function;
- 4) structure (structure);
- 5) union.

For example:

```
int mass[10]; // array of 10 integers
```

```
char line[80]; //array of 80 characters
```

```
float a[5][4]; //two-dimensional array of 5x4 elements of type float
```

```
double b[7][7][7]; // an array of 7x7x7 elements of type double
```

```
int *x[10] //array of 10 pointers to type int
```

When declaring an array, its initialization is allowed. The initial values of the

elements must be separated by commas in curly braces { }. The value of the elements for each individual dimension can be enclosed in curly brackets, between which a comma is placed, for example:

```
int year[12]={31,28,31,30,31,30,31,31,30,31,30,31};  
float mas[2][3]={{0.0,0.1,0.2},{1.0,1.1,1.2}};  
The last initialization is equivalent to the following:  
float mas[2][3]={0.0,0.1,0.2,1.0,1.1,1.2}.
```

The number of elements cannot be greater than the specified size. If the number of elements is less than the specified dimension, then the assignment takes place only for the initial elements, and other elements take null values if the array is declared as external or static, and take undefined values in other cases:

```
static int vec[10]={1,8,3}; /* The first three elements will take the specified  
values, and the next seven - zero values */  
auto float matr[3][4]={{0.0}, {1.0,1.1},{2.0,2.1,2.2}};  
/* elements with indices [0][0],[1][0],[1][1],[2][0],[2][1],[2][2] will take the  
value and all others – undefined values */.
```

With explicit initialization of the array, its dimensions may not be specified. Then the dimension of the array is determined by the number of specified elements, for example:

```
int i_array []={4,-2,7,10,-3,5,3}  
/* declared and initialized array of seven integers */
```

When explicitly initializing two-dimensional arrays, the first index can be omitted, and the second must be specified:

```
float f_mas[][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

In this example, an array consisting of 3 rows and 4 columns is declared. This announcement is equivalent to the following one:

```
float f_mas[][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

According to the ANSI standard, an array can be initialized anywhere, including inside a function for old compilers. To initialize an array in the middle

of a function, it must be declared static:

```
static float vec[]={5.2,6.3,8.4};
```

In memory, array elements are placed in a contiguous region, so that the right index changes first.

An example of placing an array in memory int matrix[2][3]

[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]
0	2	4	6	8	12 байт

The array element is referenced by specifying the name of the array and its indexes. Indices must be constants, variables, or integer expressions. Each index is enclosed in separate square brackets, e.g:

```
i_array[3]; /* access to the fourth element of the i_array array */
```

Another way to access array elements is to use pointers.

The name of one array is a pointer to its null element, for example:

```
int vec{5};
```

```
vec → vec[0] vec[1] vec[2] vec[3] vec[4]
```

```
vec=&vec[0];
```

```
vec+i=&vec[i];
```

```
*vec=vec[0];
```

```
*(vec+i)=vec[i];
```

According to the declaration of the two-dimensional array int matrix[2][3]; notation matrix[i], where i=0,1, determines the memory location of the i-th line, and the name of the matrix array determines the pointer to the beginning of the int matrix array[2][3]. The array name matrix is compatible with the pointer type to an array of three integers: int (*vec)[3];

pointers to rows	Elements		
matrix[0]	[0][0]	[0][1]	[0][2]
matrix[1]	[1][0]	[1][1]	[1][2]

The values of the pointers matrix and *matrix are the same, but not compatible in type. The compatibility and equivalence rules are given below:

```
matrix=&matrix[0]=matrix[][m];
```

```
matrix+i=&matrix[i];
```

```
*matrix=matrix[0]=&matrix[0][0];
```

```
*(matrix+i)=matrix[i]=&matrix[i][0];
```

```
*(matrix+i)+j=matrix[i]+j=&matrix[i][j];
```

```

**matrix=*matrix[0]=matrix[0][0];
**(*matrix+i)=*matrix[i]=matrix[i][0];
*(*(matrix+i)+j)=*(matrix[i]+j)=matrix[i][j];

```

Similarly, to declare a three-dimensional array `int fx [5][4][8]`; the notation `fx[i]` specifies the address of the *i*-th two-dimensional array with dimensions of 4x8 elements in memory, and the notation `fx[i][j]` specifies the address of the 8-element *j*-th row of the *i*-th two-dimensional sub-array.

Given that the first *p* indices of the array ($p < k$) determine the address of the corresponding subarray, a form similar to the addressing of memory cells through pointers is allowed for addressing array elements or their pointers:

For example:

```
(ip)[name_array [i1][i2]/ *... */[i(p-1)]].
```

ip - variable or integer expression, but not a constant.

For example:

```
(i)[vec] /* equivalent to addressing vec[i] */
```

```
(j)[matrix[i]] /* еквівалентно звертанню matrix[i][j] */.
```

When working with arrays, you need to remember that the name of the array is a constant, so modification operations of the value determined by this constant are not allowed, for example:

```
int a[5],y,n,
```

```
int *z; /* the following operations are not allowed */
```

```
a=y; a++; a+=n; z=&a;
```

In the C++ language, one cannot assign an entire array to another of the same type and size. Arrays are copied element by element using loop operators, arrays of pointers.

In the C language, it is allowed to use arrays of pointers to all data types, for example:

```
int *r[3];
```

`r[3]` - an array containing the addresses of three elements of type `int`.

To access the value located at the address `r[i]`, it is necessary to use the operation of dereferencing the indexed pointer `*r[i]`. Working with an array of pointers is demonstrated by the following example:

```
int i;
```

```
float x1=1, x2=2, x3=3, sum=0, *r[3];
r[0]=&x1;
r[1]=&x2;
r[2]=&x3;
for(i=0;i<3;i++) sum = sum+*r[i];
printf("Сума = %f\n",sum);
```

The scanf library function is used to enter array elements, which has the following format:

```
int scanf("format string", list of pointers to array elements);
```

The printf library function is used to output array elements, which has the following format:

```
int printf("format string", list of array elements);
```

The I/O format string must contain specifications for the data list formats.

The format specification must begin with a percent symbol (%)

Код	Format
%c	Symbol
%d	Signed decimal
%i	Signed decimal
%e	Exponential representation (string letter e)
%E	Exponential representation (capital letter E)
%f	A floating point value
%g	Uses the shorter of the two formats: %e or %f (if %e, uses the lowercase letter e)
%G	Uses the shorter of the two formats: %E or %F (if %e uses a capital letter e)
%o	Unsigned octal integer
%s	A string of characters
%u	Unsigned decimal integer
%x	Unsigned hexadecimal integer (lowercase letters)
%P	Ppointer. The corresponding argument must be a pointer to an integer. This specifier stores in this whole the number of characters output to the output stream up to the current moment (before the specifier %n is detected)
%%	Prints the % character

To enter a long integer (long) or real with double precision (double), the character l is specified before the format symbol. For long double type input, the L character is specified before the format character.

The format string can include other characters from the ASCII table.

If such characters are included in the format string of the scanf function, then they must occur in the input stream. If additional characters are included in the format string of the printf function, they will be placed in the output stream. In this way, you can display auxiliary messages. An example of input-output of elements of a one-dimensional array

```
#include <stdio.h>
int main( ){
    float vec[10];
    // Bscið
    for(int i=0;i<10;i++)
        scanf("%f",&vec[i]);
    // Bucið
    for(int i=0;i<10;i++)
        printf("%g ", vec[i]);
    printf("\n");
    return 0;
}
```

3.3. Work program

3.3.1. Start the IDLE environment.

3.3.2. Make a program algorithm for determining some parameters of one-dimensional arrays (task 1) according to your option. Draw a block diagram of the algorithm.

3.3.3. Compile a program for determining some parameters of one-dimensional arrays (task 2) according to your option.

Report requirements:

Fill the array of task 2 with the input operator.

3.4. Hardware and software

3.4.1. Personal computer.

3.4.2 Software: Visual Studio or Dev-C++

3.5 Questions

3.5.1. How is an array declared?

3.5.2. What are the possible types of arrays?

3.5.3. What will the array look like if the number of elements smaller than the size of the array is specified during its initialization?

3.5.4. What are the ways to access an array element?

3.5.5. How to copy one array to another?

3.5.6. How to describe an array of pointers?

3.5.7. What is de-referencing for?

3.5.8. How is input-output of arrays carried out?

3.5.9. Name the format specifiers you know.

Task 1

1) Find and print the sum of positive array elements

$B(6)=(5.0;-2.3;-6.9;-1.1;2.0;6.6)$.

2) Count and print the number of positive elements that are in even places

$C(8)=(-6.3;-1.0;10.3;-8.8;6.3;-1.1;0.0;0.1)$.

3) Print the arithmetic mean of the negative elements of the array:

$A(6)=(6.3;-2.1;4.2;5.3;-7.2;-4.5)$.

4) Find the minimum array element $B(7)=(6.3;-1.6;1.1;0.1;-2.0;2.3;6.3)$.

5) Print the sum of negative elements that are in even positions in the array

$X(17)=(-2.3;4.0;-8.9;6.3;4.9;-7.8;-6.5;5.1;3.8;-4.3;-5.1;7.2)$.

6) Print the arithmetic mean of the non-negative elements of the array that are in odd position

$B(10)=(6.3;0.0;-8.3;7.2;6.1;-4.2;5.7;6.4;5.6;-4.8)$.

7) Find and print the number of positive array elements

$C(9)=(1.6;2.1;-3.1;0.0;1.1;-2.2;3.7;8.9;9.2)$.

8) Calculate the positive product of array elements $D(5)=(1.1;-6.2;0.0;2.3;5.1)$.

9) Find the sum of array elements y $X(6)=(3.5;-6.3;2.1;0.1;5.1;-2.1)$, the value of which is less than 0.21.

- 10) Calculate the product of the modules of the values of the array elements
 $Y(7)=(-2.2;0.2;3.1;2.1;-3.1;6.1;0.5)$.
- 11) Determine the number of array elements $B(5)=(2.2;3.1;-3.6;0.1;2.1)$, the value of which is less than 0.99.
- 12) Determine the number of negative array elements
 $D(5)=(1.2;25.3;-2.3;-3.1;0.0)$.
- 13) Calculate the product of array elements $(B)=(2,3;4,3;-15,2;1,1;-1,2;-3,3)$, the value of which is greater than 2.0
- 14) Print the serial numbers of the negative elements of the array
 $W(8)=(-7,9;1,0;1,1;-2,2;5,0;-1,1;2,0)$.
- 15) Count the number of array elements $Y(6)=(2,1;3,6;-6,3;4,1;2,2;-2,3)$, whose values are greater than 2.3.
- 16) Calculate the product of array elements $A(5)=(3,1;-7,8;6,2;-3,3;1,1)$, which are greater than -5.4.
- 17) Calculate the sum of the values of the negative elements of the array
 $X(8)=(-1,2;6,3;0,2;-0,7;1,1;2,3;-3,6;2,2)$.
- 18) Determine the positive numbers of array elements
 $C(7)=(1,1;2,3;-6,4;0,0;2,1;2,3;1,2)$.
- 19) Calculate the product of array elements $A(5)=(1,3;6,3;2,4;-3,6;-2,5)$.
- 20) Determine the minimum array element $X(6)=(2,1;-3,6;-2,0;0,0;-6,3;1,0)$ and the number of this item
- 21) Print the number of the first negative element of the array
 $A(8)=(3,2;-6,3;2,0;-3,3;-6,6;-2,2;0,2,1)$.
- 22) Determine the number of the maximum element of the array
 $C(6)=(2,3;7,9;12,3;-6,8;-22,3;0,0)$.
- 23) Print the numbers of negative elements of the array
 $D(7)=(2,2;-3,3;2,1;-3,0;-7,1;-5,1;0,0)$.
- 24) Find the minimum array element $B(6)=(21,3;30,5;-6,8;0,3;-1,2;5,3)$.
- 25) Determine the maximum module element of the array
 $C(8)=(-3,6-5,3;2,1;0,1;-0,7;5,3;6,6;-2,2)$.

26) Determine the number of non-negative array elements

$$D(7)=(-2,3;2,3;0,0;3,2;6,0;-6,0;3,2) .$$

27) Count the number of elements in the array

$$D(8)=(6,3;26;-3,6;2,1;0,0;6,6;-7,2;1,1) .$$

which do not exceed the number 5 in modulus..

28) Calculate the arithmetic mean of array elements

$$A(5)=(3,2;6,3;-3,3;2,3;5,5) .$$

29) Find the number of positive array elements

$$B(6)=(6,2;-3,2;0,0;3,3;2,2;-3,6) .$$

30) Determine which number has the smallest element of the array

$$C(7)=(3,3;0,0;-3,3;-6,17;6,6;2,1) .$$

Task 2

Variant number	Size of array	Data type	Task
1	13	int	Calculate the number and sum of even elements of the array
2	15	float	Calculate the arithmetic mean of positive elements
3	19	int	Calculate the factorial of the value of the last element
4	10	int	Calculate the product of elements whose value is less than 6
5	14	float	Calculate the arithmetic mean of odd elements
6	20	float	Place array elements in reverse order
7	18	float	Calculate the sum of array elements that are multiples of 3
8	15	int	Calculate the sum of elements whose absolute value does not exceed 10
9	10	int	Calculate the arithmetic mean of the minimum and maximum elements of the array
10	12	int	Output the number of elements whose values
11	20	int	greater than the value of the first element of the array
12	8	float	Determine the indices of the minimum and maximum elements of the array

13	15	float	Calculate the product of odd array elements
14	13	int	Calculate the arithmetic mean of elements whose value is greater than the value of the last element of the array
15	15	int	Calculate the number of positive elements whose value is less than 20
16	19	int	Calculate the arithmetic mean of elements that are multiples of 5
17	10	float	Determine the minimum of positive elements
18	18	float	Calculate the number of positive, negative and zero elements
19	15	int	Calculate the product of single-digit array elements
20	10	float	Calculate the sum of only two-digit elements
21	12	int	Calculate the percentage content of positive, negative and zero elements
22	20	float	Check if the array is in ascending order
23	18	int	Check if the array is sorted in descending order
24	15	float	Replace all negative elements with minimal ones
25	10	float	Replace all negative elements with minimal ones
26	12	int	Remove the smallest array element from the array
27	20	int	Calculate the number of positive and negative elements
28	14	float	Calculate the sum of only three-digit elements
29	11	int	Calculate the arithmetic mean of elements whose value is greater than the value of the last element of the array
30	16	float	Calculate the sum of array elements whose values belong to the interval [3, 6]

№4. Developing programs with multidimensional arrays

4.1. The purpose of the work

To study the organization of data of the same type in the form of multidimensional arrays in the C++ language, their declaration, methods of accessing elements, and programming processing algorithms..

4.2. Brief theoretical information

Organizing multidimensional arrays

The dimensionality of an array is determined by the number of indices. Elements of a one-dimensional array (vector) have one index, elements of a two-dimensional array (matrix, table) have two indices: the first of them is the row number, the second is the column number. The number of indices in arrays is unlimited. When placing elements of an array in computer memory, the rightmost index is changed first, then the rest are changed from right to left.

A multidimensional array is declared in the program as follows:

```
<type> <name> [ <size1> ] [ < size 2> ] ... [ < size N> ];
```

For example

```
int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };
```

The first dimension represents the number of rows [2], while the second dimension represents the number of columns [3]. The values are arranged in row order and can be visualized as follows:

	col 0	col 1	col 2
row 0	1	4	2
row 1	3	6	8

Accessing elements of a two-dimensional array

To access an element of a two-dimensional array, you need to specify the row and column index numbers. This instruction accesses the value of the element in the first row (0) and third column (2) of the matrix array.

```
int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };
```

```
printf("%d", matrix[0][2]); // output 2
```

To change the value of an element, refer to the element's index number in each dimension:

```
matrix[0][0] = 9;
```

Example of I/O of elements of a two-dimensional array:

```
#include <stdio.h>
int main(){
    int i,j,mas[3][4];
    // Ввід
    for(i=0;i<3;i++)
        for(j=0;j<4;j++)
            scanf("%d",&mas[i][j]);
    // Вивід по рядках
    for(i=0;i<3;i++){
        for(j=0;j<4;j++)
            printf(" %d ",mas[i][j]);
        printf("\n");
    }
    return 0;
```

Example of working with a matrix in which the maximum and minimum elements are swapped:

```
#include <iostream>
using namespace std;

int main(){
    setlocale(0, ".1251");
    double a[4][6],min,max;
    int i,j,imin,jmin,imax,jmax;

    cout<<"Введіть матрицю з 4-х рядків і 6-ти стовпців:"<<endl;
    for(i=0; i<4; i++)
        for(j=0; j<6; j++) cin>>a[i][j];
```

```

min=max=a[0][0];
imin=jmin=imax=jmax=0;
for(i=0; i<4; i++)
    for(j=0; j<6; j++)
        {
            if(a[i][j]<min){ min=a[i][j]; imin=i; jmin=j; }
            if(a[i][j]>max){ max=a[i][j]; imax=i; jmax=j; }
        }
a[imin][jmin]=max;
a[imax][jmax]=min;

cout<<"Матриця, в якій поміняні місцями максимальний і
мінімальний елементи."<<endl;

for(i=0; i<4; i++)
{
    for(j=0; j<6; j++) cout<<a[i][j]<<"\t";
    cout << endl;
}
system("pause">>void");
return 0;
}

```

Functions in C++

Functions are the building blocks of the C++ language, self-contained units of a program designed to solve specific tasks, usually repeated several times.

The basic form of a function definition is:

```

type <function name>(list of variables)
{
    commands
}

```

The type of a function determines the type of value that the function returns using the return statement. If the type is not specified, the function defaults to returning an integer value (of type int). The parameter list consists of a list of

parameter types and names, separated by commas. A function may have no parameters, but parentheses are required in any case.

The parameter list must specify the type for each parameter. An example of a valid parameter list is:

```
f(int x, int y, float z)
```

Example of an incorrect parameter list:

```
f(int x, y, float z)
```

Let's give an example of a function that implements the reduction of a number a to a natural power b:

```
float step(float a, int b)
{
    float i;
    if(a<0) return (-1); /* основа негативна */ a=1;
    for(i=b;i-->0) a*=a;
    return a;
}
```

This function returns -1 if the base is negative, and n if the base is non-negative.

The return operator has two uses.

First, this operator causes an immediate exit from the current function and a return to the calling program.

Second, this operator can be used to return the value of the function.

It should be noted right away that there may be several return operators in the function body, or there may be none. In this case, the return to the calling program occurs after the last operator in the function body is executed.

Another example is a function for finding the largest of two numbers:

```
max(int a, int b)
{
    int m;
    if(a>b) m=a;
    else m=b;
}
```

```
    return m;  
}
```

It is also possible to write this function without using an additional variable:

```
max(int a, int b)  
{  
    if(a>b) return a;  
    else return b;  
}  
// Можно еще короче:  
max(int a, int b)  
{  
    if(a>b) return a;  
    return b;  
}  
// А можно и так:  
max(int a, int b)  
{  
    return (a>b)? a: b;  
}
```

If a function is supposed to return a value but does not, the compiler issues a warning. All functions that return values can be used in C++ expressions, but they cannot be used on the left side of an assignment operator, except when returning a pointer value.

There are some special features to using functions that return pointers. Pointers are neither of type `int` nor `unsigned int`. Their values are memory addresses of data of a certain type. The function must be described accordingly. Consider an example of a function that returns a pointer to type `char`. This function finds the first space in a string and returns its address.

```
char* find(char* string)  
{  
    int i=0;  
    while (string[i] != ' ' && (string[i] != '\0')) i++;
```



```
    if(s[i]) return &s[i]; /* повертає адреса першого пробілу */  
    else return NULL; /* повернення нульового покажчика */  
}
```

When a function does not return any value, it must be declared as a void function.

You are not required to declare a function as void, in which case it will default to int and return no value. This will cause a compiler warning, but will not prevent compilation. However, it is a good practice to declare the return type of the function..

Function prototypes

A feature of the ANSI C++ language standard is that in order to generate correct machine code for a function, it must be informed of the return type of the result, as well as the number and types of arguments, before the first call. For this purpose, C++ uses the concept of a function prototype. The function prototype is specified as follows: type <function name>(parameter list);

Using a function prototype is a function declaration (declaration). Most often, the function prototype completely coincides with the title in the function description, although this is not always the case. When declaring a function, it is important for the compiler to know the function name, the number and type of parameters, and the type of the return value. In this case, the names of the formal parameters of the function do not play any role and are ignored by the compiler. Therefore, the function prototype can look like this:

```
int func(int a, float b, char* c);
```

or:

```
int func(int, float, char*);
```

These two announcements are absolutely equal.

Let's consider an example.

```
#include <stdio. h>  
float sqr(float a); /* this is a function prototype, a function declaration  
*/
```

```

int main()
{
    float b;
    b=5.2;
    printf("Квадрат числа %f дорівнює %f", b, sqr(b));
    return 0;
}

float sqr(float a) /* This is a description of the function */
{
    return a*a;
}

```

The following two examples of using functions will cause a compile-time error. In the first example, the compile-time error is that the return value does not match the declared type of the function. C and C++ will automatically convert the data to another type, but only when possible. An integer type cannot be automatically converted to a pointer to an integer..

```

#include <stdio.h> /* The example is incorrect. */
int *sqr(int *i) /* Function prototype */
main( )
{
    int i;
    sqr(&x);
}
int *sqr(int *i) /* Function declaration */
{
    return *i=(*)*(*)i;
}

```

```

#include <stdio.h>
/* The example is incorrect. */
int sqr(int *i) /* Function prototype */
main()

```

```

{
  int x=10:
  sqr(&x, 10); /* Argument number mismatch */
}
int sqr(int *i)
{
  *i=(*i)*(*i);
}

```

Note that if we fix these programs, the function will return the square of the number *i* not through the function value, but through the function parameter.

If the function has no arguments, then when declaring the prototype of such a function, the keyword `void` should be written instead of arguments. This should also apply to the `main()` function. Its declaration should be of the form `void main(void)` or `main(void)`.

```

#include <stdio.h>
void line_(void);
main (void)
{
  line_();
}
void line_(void)
{
  int i;
  for(i=0;i<80;i++) printf("-");
}

```

C++ header files contain prototypes of standard functions that are related to that header file. Examples of such header files are `stdio.h`, `string.h`, `conio.h`, etc.

Scope and area of visibility

The scope rules of a variable are the rules that determine what data is available from a given place in the program.

In the C++ language, each function is a separate program block. You cannot

get into the body of a function except by calling this function. In particular, you cannot use the local goto operator to jump to the body of another function.

From the point of view of the scope of variables, there are three types of variables: global, local, and formal parameters. Scope rules determine where each of them can be used.

Local variables are variables declared inside a block, in particular inside a function. The C++ language supports a simple rule: a variable can be declared inside any program block. A local variable is available inside the block in which it is declared. Recall that a block is opened with a curly brace and closed with a curly brace. The scope of a local variable is the block.

A local variable exists while the block in which this variable is declared is executed. When you exit the block, this variable (and its value) is lost..

```
#include <stdio.h>
void f(void);
main(void)
{
    int i;
    i=1;
    f();
    printf("В функції main значення i дорівнює %d\n", i);
}
void f(void)
{
    int i;
    i=10;
    printf("В функції f() значення i дорівнює %d\n", i);
}
```

The example shows that when the function is called, the value of the variable `i` declared in `main()` has not changed.

Formal parameters are variables declared in the function description as its arguments. Functions can have a number of parameters that are used when calling functions to pass values to the function body. Formal parameters can be used in the function body in the same way as local variables, which they essentially are. The scope of formal parameters is the block that is the body of

the function.

Global variables are variables declared outside of functions. Unlike local variables, global variables can be used anywhere in the program, but before their first use they must be declared. The scope of a global variable is the entire program.

Using global variables has its drawbacks:

- they occupy memory for the entire duration of the program;
- using global variables makes functions less general and makes them more difficult to use in other programs;
- using external variables causes errors due to side effects. These errors are usually difficult to find.

4.3. Work program

4.3.1. Run the IDLE environment.

4.3.2. Write the program algorithm (task 1) according to your version. Draw a flowchart of the algorithm.

4.3.3. Write the program (task 2) according to your version.

Program requirements:

Fill arrays with the input operator.

4.4. Hardware and software

1.4.1. Personal computer.

1.4.2 Software: Visual Studio or Dev-C++

4.5 Questions

4.5.1. Which of the following declarations of two-dimensional arrays are incorrect and why?

a) `int C[1..5, 1..5]`; c) `double C[1..5][1..5]`;

b) `double C[5][5]`; d) `int C: [5][5]`;

4.5.2. How are the elements of two-dimensional arrays placed in the operating memory?

4.5.3. Write the operator for declaring a matrix of integers S of size 7 by 3?

4.5.4. Is it possible to process a two-dimensional array by organizing the outer loop by columns and the inner loop by rows?

Task 1

Variant number:

1. In a matrix of real numbers with 5 rows and 4 columns, calculate the number of positive, negative, and zero elements.
2. In a 4×5 matrix of integers, determine the largest element and its indices.
3. Determine the minimum element of the main diagonal of a square matrix of size 5×5 and the row number in which it is contained.
4. Interchange the elements of the first row of a 4×4 matrix of real numbers with the elements of its non-main diagonal.
5. In a 3×5 matrix of integers, replace the negative elements with zeros.
6. Determine the maximum and minimum elements of a 6×6 matrix of real numbers.
7. Calculate the vector of arithmetic mean values of the elements of the rows of a 5×4 matrix of real numbers.
8. Calculate the sum of the elements of the non-main diagonal of a 5×5 matrix of integers.
9. For a 5×5 matrix of integers, calculate the transpose matrix.
10. Determine the column number of the 3×6 real number matrix with the smallest element.
11. Determine the minimum element of the non-principal diagonal of the 5×5 integer matrix and the column number in which it is located.
12. Calculate the vector of sums of row elements of the 7×3 integer matrix.
13. Replace negative elements in odd rows of the 7×4 real number matrix with zeros, and positive elements with ones.
14. Calculate the sums of the elements of the main and non-principal diagonals of the 5×5 real number matrix and the difference between these sums.
15. In the 7×5 real number matrix, calculate the sum of all negative elements of the first four rows.
16. In the 4×5 integer matrix, replace all negative elements with zeros.
17. Calculate the product of the minimum element of a 4×5 integer matrix by the arithmetic mean of the matrix.
18. In a 6×4 integer matrix, calculate the arithmetic mean of positive elements.
19. In a 5×4 integer matrix, replace the positive elements in odd rows with 1, and the negative elements in even rows with 0.

20. In a 6×3 real number matrix, calculate the product of all negative elements in even rows.

21. In a 3×5 integer matrix, calculate the number of elements that are less than the arithmetic mean.

22. In a 5×3 real number matrix, replace all elements that are greater than 2.5 with 1.

23. Calculate the vector of sums of the absolute values of the elements of the rows of a 4×5 real number matrix.

24. In a matrix of integers of size 7×4 , determine the smallest element from the number of positive and the largest from the number of negative and swap them.

25. Calculate the vector of elements of the main diagonal of a matrix of real numbers of size 5×5 .

26. Calculate the vector of sums of squares of elements of columns of a matrix of real numbers of size 3×5 .

27. In a matrix of integers of size 5×5 , swap elements of the main and non-main diagonals.

28. In a matrix of integers of size 5×5 , replace all even elements with zeros.

29. Calculate the difference of the sums of elements of the first row and last column of a matrix of real numbers of size 4×6 .

30. Calculate the vector of sums of elements of the main and non-main diagonals of a matrix of real numbers of size 6×6 .

Task 2

Variant number:

1. From the matrix $A(n,n)$ ($n \leq 6$) obtain a new matrix $B(n,n)$ by dividing all elements of the matrix A by its maximum element in modulus.

2. In the matrix $A(6,8)$ it is necessary to change the row containing the minimum element to the row containing the maximum element. Assume that these elements are unique.

3. From the matrix $A(m,n)$ ($m \leq 5$, $n < 6$) obtain the numbers a_1, \dots, a_m , where a_i is the value of the first positive element of the i -th row.

4. Transpose the matrix $A(m,n)$ ($m \leq 4$, $n < 6$) and print the resulting matrix.

5. The coordinates of m vectors are given by the matrix $A(m,n)$ ($m \leq 6$, $n < 7$). It is necessary to calculate the lengths of these vectors, print the values, and among these find and indicate the number of the vector of the minimum length.

6. Perform such a transformation of the matrix $A(m,n)$ ($n \leq 4$) in which all positive elements are replaced by the sum of the corresponding indices, and negative ones by the product of the indices.

7. Find the minimum element of the matrix $A(m,n)$ ($m \leq 5$) and display it on the screen. Replace the elements of the matrix that lie below the main diagonal with the minimum element.

8. In the matrix $A(6,6)$, delete the 4th row and print the resulting matrix.

9. Perform such a transformation of the matrix $A(m,n)$ ($m \leq 5$, $n < 7$), in which the last column will take the place of the first, and all the others will be shifted one column to the right.

10. The coordinates of n vectors are given by the matrix $A(m,n)$ ($m \leq 5$, $n \leq 6$). Calculate the lengths of these vectors, print and plot their values in a one-dimensional array. Among the elements of the array, find the maximum element and its number.

11. In the matrix $A(3,7)$, delete the 5th column and print the resulting matrix.

12. The matrix $A(m,n)$ ($m \leq 4$, $n \leq 3$) contains positive and negative elements. Formulate two arrays from the elements of this matrix: B - containing positive elements, and C - negative. Count the number of elements in these arrays.

13. The integer matrix $A(n,n)$ ($n \leq 5$). Find the smallest of the values of the elements of the column that has the maximum sum of the moduli of the elements. Specify the column number.

14. Perform such a transformation of the matrix $A(m,n)$ ($m \leq 7$, $n \leq 3$), in which the last row is swapped with the first, the penultimate with the second, etc. Print the transformed matrix.

15. In the matrix $B(m,n)$ ($m \leq 4$, $n \leq 6$), all elements of which are different. In each row, the element with the smallest value is selected, then the largest among these numbers is selected. Display the found element on the screen.

16. Square all odd elements of the matrix $A(m,n)$ ($m \leq 5$, $n \leq 4$) and formulate a one-dimensional array from these squares.

17. In the matrix $A(m,n)$ ($m \leq 5$, $n \leq 5$), replace the element of the main diagonal with the elements of the side diagonal.

18. In the matrix $A(m,n)$ ($m \leq 5$, $n \leq 4$), find the sum of the elements that frame this matrix and swap the minimum element of the left side with the maximum element of the right side.

19. Make the following transformation of the matrix $A(m,n)$ ($m \leq 5$, $n \leq 5$).

Replace the elements above the side diagonal with those that are systematic with respect to the side diagonal;

20. In the matrix $A(m,n)$ ($m \leq 5$, $n \leq 5$), replace the elements above the side diagonal with those symmetrical with respect to this diagonal.

21. In the matrix $A(m,n)$ ($m \leq 5$, $n \leq 5$), replace the elements of the diagonal adjacent to the main diagonal from above with the element of the diagonal adjacent to the main diagonal from below.

22. In the matrix $A(m,n)$ ($m \leq 5$, $n \leq 4$), arrange the last two rows in descending order.

23. Arrange the elements of the matrix $A(m,n)$ ($m \leq 5$, $n \leq 4$) above the main diagonal in ascending order and write them below the main diagonal in rows.

24. Arrange the elements of the matrix $A(m,n)$ ($m \leq 5$, $n \leq 4$) in descending order and place them in rows.

25. Perform the following transformation of the matrix $A(m,n)$ ($n \leq 5$) in which all positive elements are replaced by the sum of the corresponding indices, and negative ones by the difference of the indices.

26. Perform the following transformation of the matrix $A(m,n)$ ($m \leq 5$, $n \leq 5$): replace the elements that are above the side diagonal with those that are symmetric to the main diagonal.

27. In the matrix $B(m,n)$ ($m \leq 5$, $n \leq 5$), all elements of which are different. In each row, the element with the largest value is selected, then the smallest among these numbers is selected. Print the found element.

28. Perform such a transformation of the matrix $A(m,n)$ ($m=5$, $n=6$), in which the last column takes the place of the first, and all the others are shifted one column to the right

29. In the matrix $A(6,6)$, delete the 3rd row and print the resulting matrix.

30. Perform such a transformation of the matrix $A(m,n)$ ($m=8$, $n=3$), in which the last row is swapped with the first, the penultimate with the second, etc. Print the transformed matrix.

№5. Developing programs with string variables

5.1. The purpose of the work

Learn to work with text data, gain the knowledge and skills necessary for programming based on the creation and use of user-defined functions, and learn to use them in practice in the process of developing programs in the C++ programming language.

5.2. Brief theoretical information

Characters

Character type variables are declared using the `char` keyword and occupy 1 byte in memory. The `char` type is an integer type and can be specified with or without a sign. The method of interpreting `char` type variables can be specified implicitly or explicitly. The implicit form of the `char` type is determined by a compiler option. In an integrated environment, this option is set using the Options/Compile/Code Generation menu. The explicit form is determined using the signed and unsigned type modifiers.

Examples of declarations:

```
char c;  
unsigned char t;  
signed char v;
```

The value of a `char` type variable determines the code of one of the 256 characters of the code table.

If the `char` type is considered signed; then the most significant bit of the code determines the sign. In this case, the range of `char` type values is from -128 to 127. For the unsigned `char` type, the range of code values is from 0 to 255. Initialization of `char` type variables can be done implicitly or explicitly.

Implicitly static and global `char` variables are initialized with the value `'\0'`. Local variables that are not static take on an undefined value.

Explicit initialization of `char` variables can be done when they are declared or when using an assignment operation or input functions. A `char` variable can be assigned a numeric or symbolic value. A symbolic constant is specified in quotes

either explicitly or by its octal or hexadecimal code, which must be preceded by the `\` character, for example:

```
char c1='A';  
char c1='x41';  
char c3,c4=0x41;
```

In all cases, the variables will take on the value 0x41 (or decimal 65), which, depending on the context of use, can be interpreted as a number or a symbol with the corresponding code.

Arrays of characters

A character array is a sequence of char elements located in a contiguous area of memory. An example of declaring a character array, for example:

```
char buffer[10];
```

A character array can contain Latin letters, Cyrillic characters, punctuation marks, and control characters. Control characters are specified by their mnemonic designation or ASCII code value, preceded by the `\` character.

A character array can be initialized in one of the following ways:

1. By default, global and static arrays are initialized with the character '0';
2. Explicit character-by-character initialization during declaration;
3. Character-by-character initialization using an assignment operation or character-by-character input.

If the character array is initialized when it is declared, the number of elements can be specified explicitly, for example:

- `char buffer[10]={'T','u','r','b','o',' ','C'};`

or implicitly:

- `char buffer[]={ 'T','u','r','b','o',' ','C'};`

In the first example, an array of 10 elements is specified. The first 7 of them will be assigned values, and the following C will take the value '0' if the array is global or static, and undefined values - in other cases. The number of elements of the initialization list must not exceed the specified value.

In the second example, the number of elements of the array is determined

implicitly by their list, as a result, an array of 7 characters will be created, which will take values from the specified initialization list.

The array elements can be accessed using an index or a pointer. An array index is an integer expression written in square brackets after the array name. The index of the first element of an array is 0. For example, to access the character 'r' of a previously defined array, you must write `buffer[2]`.

When accessing characters using an index, it is necessary to remember that the compiler does not control its permissible value. Incorrect use of the index can lead to an access outside the array without issuing a warning or error message.

Considering that the array name is a constant pointer to the first byte of this array (element `buffer[0]`), to access the *-th* element of the character array, you can use the `*` operation, for example; `*(buffer+i)`

Character-wise initialization using the assignment operation is performed by accessing individual elements of the array in one of the following ways:

```
buffer[1]='U';  
*(buffer+2)='R';
```

Element-by-element input of a character array is organized using a loop operator, for example:

```
for(i=0;i<10;i++)  
getchar(buffer[i]);
```

To input characters, you can use specially designed library functions `getchar`, `getch`, `getche`; or the `scanf` formatted input function.

Character arrays are also called character buffers. Special functions are designed to work with character buffers, the prototypes of which are given in the `mem.h` file.

Strings

A string in C is an array of characters that ends with the character `'\0'` (NULL). Strings are declared in the same way as character arrays, for example::

```
char line[20]; /* a string of 20 characters is declared */
```

In fact, 19 characters can be written to such an array, and the 20th character is the end-of-line character '\0', for which memory must be reserved. Each character of the string occupies 1 byte in memory. The maximum length of the string depends on the selected memory model.

Initial initialization of character strings can be done in two ways. In the first method, characters are specified as array elements, for example::

```
charstr[6]= {'w','h','i','l','e','\0'};
```

With this method of declaring a string, the character '\0' must be specified explicitly at the end of the array of elements. If the number of characters listed in curly braces is less than the specified size, the string is padded from the right to the specified length with '\0' characters.

If the '\0' character is not specified at the end of the initialization list, then not a string, but an array of characters will be formed in memory.

With the second initialization method, the string is not divided into individual characters, but is specified in quotes, for example:

```
char str[] = "while";
```

With this initialization method, the compiler will add the '\0' character to the end of the string.

The string of characters can be empty. Such a string consists of only one '\0' character and is initialized as follows:

```
char str[]="";
```

An array character is referenced by its serial number, for example:

```
char c, str[]="while";
```

```
c=str[2]; //c='i'
```

The name of a character string is a pointer to a zero element (str ==&str[0]), that is, it takes a constant value assigned to it at the compilation stage. Therefore, modification of this value is not allowed, for example, the operation str++ is not allowed.

To declare a character string, you can use a pointer, for example:

```
char *str1 = "while";
```

```
or char *str1;
```

```
str1="while";
```

The variable `str1` contains the address of the string "while". Unlike the previous description of a string as an array of characters, its modification is allowed, for example, the operation `str1++` is allowed and the new value of the variable `str1` points to the next element.

The expression `*(str1+i)` provides access to the *i*th element of a character string, for example:

```
c = *(str1+2) /*c='i' */
```

Character strings can be combined into arrays of arbitrary size. In practice, the most common work is with two-dimensional character arrays (one-dimensional arrays of character strings).

Depending on the description, character strings are stored in a continuous memory area as two-dimensional character arrays or in different memory areas as separate strings. The first way of description:

```
charmasstr[4][10]={"green", "red", "white", "blue"};
```

defines a rectangular character array. All four strings will have the same length. Short strings are padded on the right with '\0' characters to the specified length (up to 10 characters).

Omitting the second index, we will have the addresses of each string, for example, the entry `masstr[2]` defines a pointer to the string "white", and the entry `*(masstr[2]+j)` defines the *j*th character of this string (*j* = 0,...,9).

To access characters, you can also use the method typical of numeric arrays, for example, the entry `masstr[0][1]` defines the character 'r' of the string "green".

The second way of description:

```
char *ptrstr[4]={"green", "red", "white", "blue"};
```

defines an array of 4 pointers to strings of different lengths. The string values may not be stored consecutively, but in different memory areas.

To access the *j*th character of the *i*th string, you can use the following expression; `*(ptrstr[i]+j)`.

Organization of standard output and input of character data. Formatted data output

It is implemented using the `printf` function, which has a variable number of arguments. The prototype of the function is given in the table above.

Function reference:

```
printf("format string", arg1,arg2,...);
```

The format (control) string is used to specify the number and types of arguments and can include:

1. regular characters that are displayed on the display screen;
2. data conversion specifications, each of which displays the value of the next argument from the list on the screen,
3. control character constants:
 - \a - causes a sound signal;
 - \n - transition to a new line;
 - \b - return to the left position;
 - \r - transition to the beginning of the current line;
 - \f - transition to a new page;
 - \t - horizontal tab;
 - \v - vertical tab;
 - \ddd - octal character code;
 - \' - apostrophe;
 - \xdd - hexadecimal character code;
 - \" - double quotes;
 - \0 - null character (empty);
 - \\ - backslash;

The conversion specification has the following format:

```
%[alignment][width][precision]conversion character
```

Square brackets are not specification characters, but only indicate that this field can be omitted. The conversion specification begins with a % sign and ends with a conversion character (format), between which there can be:

1. The "-" sign (minus), which indicates that the converted parameter should be left-aligned in its field (by default it is right-aligned);
2. A string of numbers - the minimum width of the field. If the value of the variable exceeds the width of the field, as many characters as necessary are printed;
3. A colon;
4. A string of numbers - the maximum number of characters that must be printed for the char type. The arguments of the printf function can be variables, constants, expressions, function calls. The values of the arguments must

correspond to given specification.

Character output

Conversion specification: %[-][width]s

The string characters are output until the '\0' character is encountered, or to the specified precision. If the string length is greater than the specified precision, the rest of the string is discarded.

This is done using the `scanf` function, which can have a variable number of arguments. The `scanf` function is mainly intended for reading a set of data of different types. Function format:

```
scanf("format string", arg1, arg2, ...);
```

A characteristic feature of this function is that its arguments must be pointers to values. For each argument in the format string, its own conversion specification is specified.

The width determines the number of characters that should be read from the input stream and assigned to the character array. If width is omitted, a single character is entered. Blank characters can be entered with this specification.

Example:

```
char a[5], b; /* Input stream: 1234567890 */
scanf("%5c", a);
a[4]='\0'; /* Result a = 1234\0 */
scanf("%c", &b); /* Result b = 5 */
```

Character string input

Conversion specification: %[*][width]s

Width specifies the maximum length of the input string. Strings in the input stream must be separated by blank characters. Leading blank characters are ignored. Reading occurs up to the first blank character (space, tab, newline), or until the specified width is reached. A '\0' character is appended to the end of the string in memory to fill the declared length.

Example:

```
char a[5], b[6]; /* Input stream; 1234567890 */
```



```
scanf("%3s", a); /* Result a = 123\0\0 */
scanf("%5s", b); /* Result b = 45678\0 */
```

Unformatted output functions

Character output

The putchar function is used in a program to display a character on the video terminal screen.

Function reference: putchar(character);

After the character is output, the cursor remains in the output line. If the output line is full, then when the next character is output, the transition to the beginning of a new line occurs. Examples of character output:

```
char ch='b';
putchar('a'); /* a */
putchar('\n'); /* transition to a new line */
putchar('\007'); /* sound signal */
putchar(ch); /*b */
putchar(getchar()); /* output of the entered character */
```

Character string output

The puts function is used to output a character string. Function reference: putchar(string pointer);

The puts function stops outputting characters if it encounters the '\0' character. This character is not output. The string of characters output by the puts function always starts with a new line on the screen. Examples of using the puts function:

```
char str1 [] = "abcdefgh";
char *str2 = "1234567890";
puts("Message Output");
puts(str1); /*abcdefgh*/
puts(str2); /* 1234567890 */
puts(&str1[4]); /*efgh*/
puts(str2+6); /* 7890 */
```

Unformatted Input Functions

Character Input

The `getchar`, `getch`, and `getche` functions are used to input characters. The prototype of the `getchar` function is in the `stdio.h` file, and the prototypes of the `getch` and `getche` functions are in the `conio.h` file.

The `getchar` function is designed to read a character from the keyboard, display it on the screen, and transfer it to the program. The function implements buffered input - the character is transferred to the program after it is typed on the keyboard and the <Enter> key is pressed.

The variable to which the function value is assigned must have the `char` type. For example:

```
char ch;  
ch=getchar();  
while((ch=getchar())!='*') {/* loop body */}
```

The `getch` function implements unbuffered input - the character is transferred to the program immediately after it is typed on the keyboard without pressing the <Enter> key. A characteristic feature of this function is also that the entered character is not displayed on the screen.

Calling the `getch` function:

```
variable = getch();
```

The `getche` function also implements unbuffered input of a character, but with its display on the screen. Calling the `getche` function:

```
variable = getche();
```

If the `getchar()`, `getch()` or `getche()` functions encounter an end-of-file code in the input stream, they return the EOF character. The EOF variable declaration is given in the standard file `stdio.h`. The end-of-file character is simulated by simultaneously pressing the two "Ctrl+Z" keys (when entering a character from the keyboard). This can be used as a condition for exiting the loop if the entered character is used in the loop body, for example:

```
char ch;
while((ch=getchar())!=EOF)
{ }
```

Entering character strings

The gets function is used to enter a character string. This function reads characters from the input stream until it encounters the newline character '\n', which is formed when the <Enter> key is pressed. The '\n' character is not included at the end of the line, and instead the '\0' character is automatically formed, for which an additional byte of memory must be reserved.

Before using the gets function, you must allocate memory for the string of characters. Calling the gets function:

```
gets(pointer to string);
```

For example:

```
char name[81];
gets(name);
```

The following example demonstrates incorrect use of the gets function:

```
static char *name;
gets(name);
```

Although the function argument is specified correctly as a pointer to a character type, no memory is reserved for the string of characters. Placing a string in memory at this address can overwrite other information and, as a result, lead to unpredictable consequences.

Unlike the scanf function, the gets function allows you to enter empty characters, for example, spaces. The gets function returns a pointer to the entered string of characters. If the data is read incorrectly or if the end of the file is encountered in the input stream, the gets function returns a NULL pointer, for example:

```
while(gets(name)!=NULL) { }
```

Processing character strings

When processing character strings, it is most often necessary to perform operations such as determining the length of a string, copying, concatenating, and comparing strings. Functions are provided for working with strings, the prototypes of which are given in the `string.h` file.

The length of a string is determined using the `strlen()` function, which has the following prototype:

```
unsigned strlen(char *str);
```

The function returns the number of characters in the string up to the null character terminating the string `'\0'`. Example:

```
char str[20]="Character string";  
int k;  
k=strlen(str);
```

The variable `k` will take a value equal to the number of characters in the string `str`, i.e. `k=14`.

When performing a copy operation, it is necessary to correctly define the string that receives character data. This string can be defined as a character buffer in static or dynamic memory.

In static memory, a string is declared as an array of characters, for example; `char buf[20]`. Dynamic memory for a string is allocated using the `malloc()` or `calloc()` functions:

```
char *buf;  
buf=(char *)malloc(20);
```

In both cases, it is necessary to provide for memory allocation for the end-of-string character `'\0'`. Copying strings is done using the library function `strcpy()`, the prototype of which looks like:

```
char *strcpy(char *str1, char *str2);
```

This function copies the string `str2` into the string `str1`. Both strings are specified by their own pointers. The function returns a pointer to the receiving string `str1`. The length of the string `str1` must be sufficient to store the string `str2`,

including the null character at the end of the string. For example;

```
char str1 [15];  
char str2[]='GCC MinGW';  
strcpy (str1, str2);
```

The `strcat()` function concatenates two strings:

```
char *strcat(char *str1, char *str2);
```

The `str2` string will be appended to the end of `str1` string. The length of `str1` string must be sufficient to accommodate the result of the concatenation, including the string termination character `'\0'`. Example;

```
char str1 [20]="Turbo ";  
char *str2="Ci";  
strcat(str1,str2);
```

The result of the `strcat()` function will be the string `str1`, which will take the value of the character constant "Turbo Ci"

The `strcmp()` function is used to compare strings, which has the following prototype:

```
int strcmp(char *str1, char *str2);
```

This function compares the strings `str1` and `str2` and returns an integer less than 0 if `str1<str2`; equal to 0 if `str1=str2`; greater than 0 if `str1>str2`. Strings are compared character by character from left to right until the first mismatch of character codes. When comparing, it is necessary to remember that in the ASCII table, lowercase letter codes are greater than uppercase letter codes. Example:

```
char str1[]="Turbo C" char str2[]="TURBO C++"  
if(strcmp(str1,str2)) put("The first string is greater than the second");
```

Other functions for processing character strings are given in Table 5.1-5.3.

Table 5.1. Functions for working with strings `#include <string.h>`

Function	Prototype	Action
atof	double atof(char *str);	Converts the string str to a double-precision real number. The conversion is performed up to the first illegal character or the '\0' character. If it cannot be converted, it returns 0
atoi	int atoi(char*str);	Converts the string str to a decimal integer. If the number exceeds the int range, it returns the 2 lower bytes. If it cannot be converted, it returns 0
atol	long atol(char*str);	Converts the string str to a long decimal integer
ecvt	char *ecvt(double v, int dig, int *dec, int *sign);	Converts the real v to a string: dig - the number of digits of the number to be converted to a string, dec - the position of the decimal point from the beginning of the string (if dec<=0, then the position of the decimal point is to the left of the number), sign is {0,1} - the sign of the number. The '\0' character is appended. Returns a pointer to a string
fcvt	char *fcvt(double v, int dig, int*dec, int*sign);	Same as ecvt, only dig - the number of digits after the point
gcv	char *gcv(double v, int dig, char *buf);	Converts a real v to a string. Unlike ecvt() and fcvt(), it places the string in a previously declared buffer buf. dig is the number of characters in the string. The resulting string contains the fixed-point or floating-point representation of the number, depending on whether the number can fit in dig positions
itoa	char *itoa(int v, char *str, int baz);	Converts an integer v to a string str in the baz number system (2<=baz<=36). Returns a pointer to a string
ltoa	char *ltoa(long v, char *str, int baz);	Converts a long integer v to a string of characters str
strcat	char *strcat(char *sp, char *si);	Assigns the string si to the string sp

strchr	char *strchr(char *str, char c);	Finds the first occurrence of the character c in the string str.
strcmp	int strcmp(char *str1, char *str2);	Compares the strings str1 and str2. Result: <0 if str1<str2; =0 if str1=str2; >0 if str1>str2
strncmpi	int strncmpi(char *str1, char *str2);	Compares the strings str1 and str2 in a case-insensitive manner for literal characters. Returns the same value as strcmp..
strcpy	char *strcpy(char *sp, char *si);	Copies string si to string sp
strcspn	int strcspn(char *str1, char *str2);	Determines the length of the first segment of string str1 that contains characters that are not included in the set of characters of string str2
strlen	unsigned strlen(char *str);	Calculates the length of string str
strlwr	char *strlwr(char *str);	Converts uppercase letters in string to lowercase letters
strncat	char *strncat(char *sp, char *si, int kol);	Assigns kol characters of string si to string sp
strncmp	int strncmp(char *str1, char *str2, int kol);	Compares kol of the first characters of strings str1 and str2. The result is similar to the strcmp function
strncmpi	int strncmpi(char *str1, char *str2, int kol);	Compares the first kol characters of the strings str1 and str2 without taking into account the case of literal characters. The result is similar to the strcmp function
strncpy	char *strncpy(char *sp, char *si, int kol);	Copies kol characters of the string si to the string sp
strpbrk	char *strpbrk(char *str1, char *str2);	Finds the first occurrence of an arbitrary character from the set of characters of the string str2 in the string str
strrchr	char *strrchr(char *str, char c);	Finds the last occurrence of the character c in the string str
strset	char*strset(char*str, intch);	Writes the character ch to all positions of the string str. Returns a pointer to str
Strnset	char *strset(char *str, int ch, zise t n);	Writes the character ch to the first n positions of the string str. Returns a pointer to str. The '\0' character is not erased if n > strien(str)

Strspn	int strspn(char *str1, char *str2);	Finds the length of the first segment of the string str1 that contains characters from the set of characters included in the string str2
Strstr	char *strstr(char *str1, char *str2);	Returns a pointer to the element of the string str1 that is the beginning of the substring str2, and NULL if str2 is not included in str1
Strupr	char *strupr(char *str);	Converts lowercase letters of the string str to uppercase
Ultoa	char *ultoa(unsigned long v, char *ctr, int baz); n+-	Converts an unsigned long integer v to a string of characters

Table 5.2. Character validation and conversion functions #include <ctype.h>

Function	Prototype	Action
Isalnum	int isalnum(int c);	Returns a value other than 0 if c is a letter (A-Z,a-z) or a digit (0-9), and 0 otherwise
Isalpha	int isalpha(int c);	Returns a value other than 0 if c is a letter (A-Z,a-z), and 0 otherwise
Isascii	int isascii(int c);	Returns a value other than 0 if the character code of c is from 0 to 127, and 0 otherwise
Isctrl	int isctrl(int c);	Returns a value other than 0 if c is a control character (0x7F or 0x00-0x1 F), and 0 otherwise
Isdigit	int isdigit(int c);	Returns a value other than 0 if c is a digit (0-9), and 0 otherwise
Isgraph	int isgraph(int c);	Returns a value other than 0 if c is a character with a graphic designation (0x21-0x7E), and 0 otherwise
Slower	int islower(int c);	Returns a value other than 0 if c is a lowercase character, and 0 otherwise
ispnnt	int isprint(int c);	Returns a value other than 0 if c is a printed character (0x20-0x7E). and 0 otherwise
ispunct	int ispunct(int c);	Returns a value other than 0 if c is a control character
Toupper	int toupper(int c);	Converts the letter c to uppercase

Table 5.3. Function for working with buffers (character arrays)
`#include<mem.h>` or `#include<string.h>`

Function	Prototype	Action
memcpy	void *memcpy (void *dest, void *src, size_t n);	Copies a block of n bytes from src to dest. Buffers must not overlap. Returns a pointer to dest.
memccpy	void *memccpy (void *dest, void *src, int c, size_t n);	Copies a block of n bytes from src to dest. Buffers must not overlap. Copying continues until: 1. the character c is encountered, which is also copied to dest. Returns a pointer to the next byte after the character c; 1. 2. until n bytes are copied. Returns a NULL pointer
memmove	void *memmove (void *dest, void *src, size_t n);	Copies a block of n bytes from src to dest. Buffers may overlap. Returns a pointer to dest
movmem	void *moymem (void *src, void *dest, unsigned n);	Copies a block of n bytes from src to dest. Buffers may overlap. Returns a pointer to dest
movedata	void movedata (unsigned srcseg, unsigned srcoff, unsigned destseg, unsigned destoff, size_t n);	Copies n bytes from srcseg: srcoff to destseh: destoff
memcmp	int memcmp (void *s1, void *s2, size_t n);	Compares the first n bytes of two buffers s1 and s2 in lexicographical order. Returns: <0 if s1<s2; ==0 if s1==s2; >0 if s1>s2
memicmp	int memicmp (void *s1, void *s2, size_t n);	Same as memcmp, but case insensitive.
memchr	void *memchr (void *s, int c, size_t n);	Searches for the character c in the first n bytes of buffer s. Returns a pointer to the character c. If the character is not found, returns NULL

5.3. Work program

5.3.1. Run the IDLE environment

5.3.2. Write the algorithm for the program for task 1 according to your version.

5.3.3. Write the program for task 1 according to your version.

5.3.4. Write the algorithm for the program for processing a text array (Appendix 2) according to your version.

5.3.5. Write the program for processing a text array according to your version.

Requirements for programs

- input data to be entered by the input operator (tasks 1, 2);
- print the results of the programs according to the task.

5.4. Hardware and Software

5.4.1. Personal computer.

5.4.2. Software: IDLE.

5.5. Questions

5.5.1. How to declare a character type variable?

5.5.2. What format specifier is used for input and output of characters?

5.5.3. What are character arrays? How to declare them?

5.5.4. How to refer to a specific element in a character array?

5.5.5. What is a character string? How to declare it?

5.5.6. What is an array of character strings? How to declare it?

5.5.7. What are the functions of unformatted input-output of character strings?

Task 1.

The results of the examination session of 1st year students are presented in the following table

	Прізвище	Інформат іка	Вища матем.	Фізика	Програму вання
1.	Іванчук С.О.	4	3	3	4
2.	Панченко І.А.	5	4	4	5

3.	Заєць О.М.	3	4	4	4
4.	Вельбицький П.О.	4	3	3	3
5.	Сидоренко В.Р.	2	3	3	2
6.	Кравченко З.І.	3	5	4	5
7.	Якубів Р.Н.	5	4	4	3
8.	Зоренко П.М.	4	2	3	3
9.	Берестяк Г.С.	4	5	5	5
10.	Дячик Н.С.	5	5	4	4

Variants:

1. Print a table containing the numbers, surnames and the number of “5”, “4”, “3”, “2” for each student in the group, and also count the total number of “5”, “4”, “3”, “2” in the group.

2. Print a table containing the numbers, surnames, grades and average score of those students in the group whose average score is more than 4, and also count the number of such students in the group.

3. Print a table containing the numbers, surnames and grades of students who have at least one “3”, and also count the number of such students in the group.

4. Print a table containing the numbers, surnames and grades of students who do not have any “5”. Count the number of such students.

5. Print a table containing the numbers, surnames and grades of each student, and at the end indicate the average score of the group in each discipline.

6. Print a table containing the numbers, surnames, grades and average score of each student in the group.

7. Print a table containing the numbers, surnames and grades of students in higher mathematics, and also calculate the average score of the group in this subject.

8. Print a table containing the names and grades of those students who have the highest and lowest average score in the group.

9. Print a table containing the numbers, surnames, grades and average score of students in the group whose average score is less than 4.

10. Print a table containing the numbers, surnames, grades of students who have only good and excellent grades.

11. Print a table containing the numbers, surnames, grades and the number of “3” in the grades of each student.

12. Print a table containing the numbers, surnames and grades of those students who received good and excellent grades in computer science, and also count the number of such students.

13. Print a table containing the numbers, surnames and grades of those students who received a satisfactory or unsatisfactory grade in higher mathematics, and also count the number of such students.

14. Print a table containing the numbers, surnames and examination grades of students. At the end, indicate the discipline with the highest average score.

15. Print a table containing the numbers, surnames and grades of students who received at least one unsatisfactory grade.

16. Print the number of “2”, “3”, “4”, “5” from each discipline.

17. Print a table containing the numbers, surnames and the number of “2”, “3”, “4”, “5” in the grades of each student.

18. Print a table containing the numbers, surnames and grades of students in the subjects “Higher Mathematics” and “Informatics”.

19. Print a table containing the average exam scores of a student in each subject.

20. Print a table containing the numbers, surnames, grades and the number of positive grades of each student.

21. Print a table containing the numbers, surnames and grades of students who have positive grades in computer science. Print the number of such students.

22. Print a table containing the numbers, surnames and grades of students who have positive grades in higher mathematics. Print the number of such students.

23. Print a table containing the numbers, surnames and grades of students who have grades “good” and “excellent” in computer science. Print the number of such students.

24. Print a table containing the numbers, surnames and grades of students who have positive grades in physics. Print the number of such students.

25. Print a table containing the numbers, surnames and grades of students who have grades “good” and “excellent” in programming. Print the number of such students.

26. Print a table containing the names and grades of those students who have the highest and lowest average score in the group.

27. Print a table containing the numbers, surnames, grades and average score

of students in the group whose average score is less than 3.

28. Print a table containing the numbers, surnames, grades of students who have only excellent grades.

29. Print a table containing the numbers, surnames, grades, and the number of “2”s in the grades of each student.

30. Print a table containing the numbers, surnames, and grades of those students who received good and excellent grades in programming, and also count the number of such students.

Task 2

Variants:

1. Given a text array A(10). Find and print the elements of the smallest length. Print this element, its serial number and length (number of characters).

2. In the text array B(12), find the element with the largest length, print it together with the number and length.

3. In the text array C(15), find the sum of the lengths of the elements with the smallest and largest length.

4. From the elements of the text array B(20), form arrays whose elements have the same length.

5. In the text array A(15), swap the elements with the smallest and largest lengths.

6. In the text array A(13), swap the following: the 1st element with the 13th, the 2nd with the 12th, etc. Print the original and transformed arrays.

7. Given an array A(10), print the elements in ascending order of their length.

8. Array B(10) contains the students' last names. Sort it alphabetically.

9. Given a text array: paper, water, tower, canal, height, volume. Merge the 2nd and 4th elements of the array and put the resulting text variable in second place. Delete the 4th element of the array.

10. Given a text array B(12). Sort it in descending order of the lengths of its elements.

11. Given a numeric array of grades: 3, 4, 4, 5, 2, 3, 3, 4. Form a text array of grades by replacing: 3 with satisfactory, 4 with good, etc. Print the resulting array.

12. Given an array of text variables B(10). Create an array C(10) containing the elements of the array B(12), increased by 5 each

13. The condition is the same as in 12 to calculate the average score of each student.

14. Given a text array A(10). Print its elements in ascending order of their lengths

15. Given a text array A(10). Print its elements in descending order of their lengths.

16. In the text array A(8) of data containing 8 words, calculate the sum of the lengths of the elements that are in even places.

17. In the text array F(10), calculate the sum of the lengths of the first 7 elements.

18. In a text array of 9 elements, find the sum of the lengths of elements from the 2nd to the 6th.

19. Given a text array B(12). Sort it in descending order of the lengths of its elements and write the resulting array to the array A\$(12).

20. Given an array of text variables B(10). Create an array c(10) containing the elements of the array B(12) written in reverse order.

21. Given a text array A(10). Find and print the elements of the longest length. Print this element, its serial number and length (number of characters).

22. In a text array C(15), find the difference in the lengths of the elements with the smallest and largest length. Print these elements.

23. Given a text array B(12). Sort it in ascending order of the lengths of its elements.

24. In a text array of 10 elements, find the sum of the lengths of elements from the 3rd to the 9th.

25. Given an array of text variables B(10). Create an array C(10) containing the elements of the array B(12), increased by 3 each.

26. In the text array A(15), swap the elements with the smallest and largest lengths.

27. In the text array A(15), swap the following: the 1st element with the 15th, the 2nd with the 14th, etc. Print the original and transformed arrays.

28. Given a text array A(20). Find and print the elements with the largest length. Print this element, its ordinal number and length (number of characters).

29. In the text array B(14), find the element with the largest length, print it along with the number and length.

30. In the text array F(12), calculate the sum of the lengths of the first 6 elements.

№6. Developing programs with file variables. Working with files

6.1. The purpose of the work

Learn functions and algorithms for organizing work with files.

6.2. Brief theoretical information

Structure

In the C++ programming language, a structure is a collection of variables united by a common name, which provides a convenient means of storing related data in one place. A structure is a collection of different data types, since they consist of several different, but logically interconnected Programming in C++. For these same reasons, structures are sometimes called composite or conglomerate data types. A structure is a combination of one or more objects (variables, arrays, pointers, other structures). Like an array, it is a collection of data, but differs from it in that its elements must be addressed by name, and its different elements do not necessarily have to belong to the same type.

Structures are convenient to use where various data related to the same object must be combined. For example, a high school student is characterized by the following data: last name, first name, date of birth, class, age.

A structure is declared using the struct keyword, followed by its type, a list of elements enclosed in curly braces. It can be represented in the following general form:

```
struct type {  
    element type 1 element name 1;  
    element type n element name n;  
};
```

An element name can be any identifier. Several identifiers of the same type can be written in one line, separated by commas.

```
struct date {  
    int day;  
    int month;
```

```
int year;
};
```

Following the curly brace that ends the list of elements, variables of this type can be written, for example:

```
struct date {...} a, b, c;
```

In this case, the corresponding memory is allocated.

The resulting type name can be used to declare an entry, for example: struct date day;. Now the variable day has the type date.

Structures can be nested one above the other. For better understanding of the structure, we use Cyrillic (Ukrainian letters) in identifiers; this cannot be done in C++.

For example:

```
struct STUDENT {char LastName [15];
firstName [15];
struct DATA{ int DAY, MONTH, YEAR;} BIRTHDATE;
int class, age;};
```

The DATA type above includes three elements: Day, Month, Year, which contain integer values (int).

The STUDENT record includes the elements: LASTNAME [15]; FIRSTNAME [15]; BIRTHDATE, CLASS, AGE. LASTNAME [15] and NAME [15] are character arrays with 15 components each. The BIRTHDATE variable is represented by the component element (nested structure) DATE. Any date of birth corresponds to the day of the month, month, and year. The CLASS and AGE elements contain integer values (int). After introducing the DATE and STUDENT types, you can declare variables whose values belong to these types.

For example:

```
struct STUDENT STUDENTS [50];
```

The STUDENTS array consists of 50 elements of type STUDENT.

In the C++ language, it is allowed to use arrays of structures; records can consist of arrays and other records.

To refer to a separate component of the structure, you must specify its name, put a period and immediately write the name of the desired element after it.

For example:

```
STUDENTS [1]. CLASS = 3;  
STUDENTS [1]. BIRTHDAY. DAY = 5;  
STUDENTS [1]. BIRTHDAY. MONTH = 4;  
STUDENTS [1]. BIRTHDAY. YEAR = 1979;
```

The first line indicates that the 1st student is in the third grade, and the following lines indicate his date of birth: 5.04.79.

Each type of structure element is defined by the corresponding declaration line in curly braces. For example, the STUDENTS array has the type STUDENT, and the year is an integer. Since each record element belongs to a specific type, its compound name can appear wherever values of that type are allowed.

Input/output libraries and file handling in C++

Input/output operations in C are organized using library functions. It should be noted that the C++ system follows the ANSI standard, also called buffered or formatted input/output.

At the same time, the C++ system supports another input/output method, the so-called UNIX-like, or unformatted (unbuffered) input/output.

We will pay attention to the first method - the ANSI standard.

The C++ language also supports its own object-oriented input/output.

It is important to understand what a file and a stream are and what the difference between these concepts is. The C input/output system supports an interface that does not depend on what physical input/output device is actually used, that is, there is an abstract level between the programmer and the physical device. This abstraction is called a stream. The method of storing information on a physical device is called a file.

Although devices are very different (terminal, disk drive, magnetic tape, etc.), the ANSI C standard associates each device with a logical device called a stream. Since streams are independent of physical devices, the same function can write information to disk, to magnetic tape, or to the screen.

There are two types of streams in C: text and binary.

A text stream is a sequence of characters. However, there may not be a one-to-one correspondence between the characters fed into the stream and those

output to the screen.

A binary stream is a sequence of bytes that uniquely correspond to what is on an external device.

A file in C is a concept that can be applied to everything from a file on disk to a terminal. A stream can be associated with a file using the file open statement. Once a file is open, information can be transferred between it and your program.

Not all files are the same. For example, from a file on disk, you can select the 5th record or replace the 10th record. At the same time, information can only be transferred to a file associated with a printer device sequentially in the same order. This illustrates the main difference between streams and files: all streams are the same, which cannot be said about files.

The file opening operation associates a stream with a specific file. The file closing operation breaks this association. If a stream was opened for output, then when the file closing operation is performed, the corresponding buffer is written to an external device. If the program terminates normally, all files are automatically closed.

Each stream associated with a file has a control structure called FILE. It is described in the header file `stdio.h`.

File pointer

The link between a file and a stream in the ANSI C I/O system is the file pointer. A file pointer is a pointer to information that specifies various aspects of a file: name, status, and current position. A file pointer specifies the name of a file on disk and its use in the stream associated with it. A file pointer is a pointer to a structure of type FILE, which is defined in the `STDIO.H` file. The following functions are also defined in the `STDIO.H` file:

Function	Function action
<code>fopen()</code>	Open a file
<code>fclose()</code>	Close a file
<code>putc()</code>	Write a character to a stream
<code>getc()</code>	Read a character from a stream
<code>fseek()</code>	Change the file position pointer to the specified location
<code>fprintf()</code>	Format write to a file

fscanf() Format read from a file
feof() Returns true if the end of the
file is reached
ferror Returns false if an error is detected
fread() Reads a block of data from a stream
fwrite() Writes a block of data to a stream
rewind() Sets the file position pointer to the beginning
remove() Destroys a file

To declare a file pointer, the operator

```
FILE *fput;
```

Let's consider the functions listed above in more detail.

The fopen() function performs two actions: first, it opens a stream and associates a file on disk with that stream; second, it returns a pointer associated with that file. The function prototype is

```
FILE *fopen(char filename, char mode);
```

where mode is a string containing the mode in which the file is opened. The possible file opening modes are listed below:

Mode	Action
------	--------

"r"	Open for reading
-----	------------------

"w"	Create for writing
-----	--------------------

"a"	Open for appending to an existing file
-----	--

"rb"	Open binary file for reading
------	------------------------------

"wb"	Open binary file for writing
------	------------------------------

"ab"	Open binary file for appending
------	--------------------------------

"r"	Open file for reading and writing
-----	-----------------------------------

"w+"	Create file for reading and writing
------	-------------------------------------

"a+"	Open file for appending or creating for reading and writing
------	---

"r+b"	Open text file for reading and writing
-------	--

"w+b"	Create binary file for reading and writing
-------	--

"a+b"	Open binary file for appending or creating for reading and writing
-------	--

"rt"	Open text file for reading
------	----------------------------

"wt"	Create text file for writing
------	------------------------------

"at"	Open text file for appending
------	------------------------------

"r+t"	Open text file for reading and writing
-------	--

"w+t"	Create text file for reading and writing
-------	--

"a+t" Open text file for appending or creating for reading and recording

If you are going to open a file named test for writing, then just write

```
FILE* fp;  
fp=fopen("test","w");
```

However, it is recommended to use the following method to open the file:

```
FILE *fp;  
If((fp=fopen("test","w"))==NULL)  
{  
    puts("Не можу відкрити файл \n");  
    exit(1);  
}
```

This method detects an error when opening a file. The NULL constant is defined in stdio.h. The exit() function we used has a prototype in the file stdLIB.h

```
void exit(libint val);
```

and terminates the program, and returns the size of val to the operating system (the calling program). Before terminating, the program closes all open files, frees buffers, and displays all necessary messages on the screen. In addition, there is an abort() function with a prototype of

```
void abort(int val);
```

This function immediately terminates the program without closing files or freeing buffers. It sends the message "abnormal program termination" to the stderr stream.

If the file is open for writing, the existing file is destroyed and a new file is created. When opening a file for reading, it is required that it exists. In the case of opening for reading and writing, the existing file is not destroyed, but is created if it does not exist.

Writing a function to a stream is done by the putc() function with the prototype:

```
int putc(int ch, FILE *fptr);
```

If the operation was successful, the written character is returned. In case of

an error, EOF is returned.

The `getc()` function reads a character from a stream opened for reading by the `fopen()` function. The prototype of the `getc()` function is

```
int getc(FILE *fptr);
```

Historically, `getc()` returns an `int` value. The same can be said about the `ch` argument in the description of the `putc()` function. In both cases, only the low-order byte is used. The function returns the EOF character if the end of the file is reached or an error occurs while reading the file. To read a text file, you can use the construction

```
ch=getc(fptr);  
while(ch!=EOF) { ch=getc(fptr) };
```

When reading a binary file, it will not be possible to determine the end of the file, just like when reading a text file. The `feof()` function with the prototype `int feof(FILE *fptr)` is used to determine the end of a text file.

The function returns "true" if the end of the file has been reached, and "zero" if not. The following construction reads a binary file to the end of the file:

```
while(!feof(fptr)) { ch=getc(fptr); }
```

This construction can also be used for text files. The `fclose()` function, declared as

```
int fclose(FILE *fptr);
```

returns "zero" if the file closing operation was successful. Any other value indicates an error. If the file closing operation is successful, the corresponding data from the buffer is read into the file, the file control unit associated with the stream is freed, and the file becomes available for further use.

If an error occurs when reading or writing a text file, the corresponding function returns EOF. To determine what actually happened, the `ferror()` function with the prototype

```
int ferror(FILE *fptr);
```

which returns "true" if the last file operation was performed and "false" otherwise. The `ferror()` function must be executed immediately after each file operation, otherwise its error message may be lost.

The `rewind()` function sets the file position indicator to the beginning of the file specified as the function argument, the prototype of this function is

```
void rewind(FILE *fptr);
```

Borland C++ defines two more buffered input/output functions: `putw()` and `getw()`. These functions are not part of the ANSI C language standard. They are used to read and write integers. These functions work exactly like `putc()` and `getc()`.

The ANSI C language standard also includes the `fread()` and `fwrite()` functions used to read and write blocks of data:

```
unsigned fread(void *buf,int bytes, int c, FILE *fptr);
```

```
unsigned fwrite(void *buf,int bytes, int c, FILE *fptr);
```

where `buf` is a pointer to the memory area from which information will be exchanged; `c` is the number of record units, each with a length of `bytes` to be read (written); `bytes` is the length of each record unit in bytes; `fptr` is a pointer to the corresponding file.

Reading and writing to a file does not have to be done sequentially; it can be done directly by accessing the desired file element using the `fseek()` function, which sets the file position pointer to the desired location. The prototype of this function is

```
int fseek(FILE *fptr,long numbytes, int origin);
```

where `fptr` is a pointer to the corresponding file; `numbytes` - the number of bytes from the starting point to set the current position of the file pointer, `origin` - one of the macros defined in `stdio.h`:

Starting point	Macro	Value
Start of file	SEEK SET	0
Current position	SEEK CUR	1
End of file	SEEK END	2

When a program starts executing, five specified streams are automatically opened. The first three are standard (`stdin`), standard output (`stdout`), and standard error (`stderr`). They are normally associated with the console, but they can be redirected to another stream. You can use `stdin`, `stdout`, and `stderr` as file pointers in all functions that use the `FILE` type.

In addition, C++ opens `stdprn` and `stdaux` streams, associated with the printer and the computer's serial port, respectively. These streams are opened and closed automatically.

The ANSI standard also includes the functions `fprintf()` and `fscanf()`, which work similarly to the functions `printf()` and `scanf()`, except that they are

associated with files on disk. The prototypes of these functions are, respectively

```
int fprintf(FILE *fptr, const char*string,...);
```

```
int fscanf(FILE *fptr, const char*string,...);
```

where `fptr` is a pointer to a file returned by the `open()` function.

The `remove()` function removes the specified file. The prototype of this function is:

```
int remove(char *filename);
```

The function returns 0 if the operation is successful and non-zero otherwise.

Since the C language is related to the UNIX operating system, a second input/output system has been created in the C++ system. This system complies with the UNIX standard. The function prototypes are in the `io.h` file. These functions are:

`read()` - reads a data buffer,

`write()` - writes to a data buffer,

`open()` - opens a file,

`close()` - closes a file,

`fseek()` - searches for a specified byte in a file,

`unlink()` - destroys a file.

When writing programs for working with files, it is necessary to remember that:

- in the program that performs operations of reading from a file or writing to a file, a pointer to the `FILE` type must be declared;
- in order for the file to be accessible, it must be opened, specifying for what action the file is opened: reading, writing or updating data, as well as the file type (binary or text);
- when working with files, errors are possible, therefore it is recommended to check the result of file operations using the `ferror` function (`fopen()`);
- reading data from a text file can be performed using the `fscanf()` function, writing – `fprintf()`;
- after finishing work with the file, it must be closed (`fclose()` function);

An example of writing to a file and reading from a file an array of structures that describe information about students in a group and store the student's name, age, and group name.

```

#include <iostream>
#include <stdio.h>
using namespace std;
const int size = 10; // константа, розмір масивів
struct Anketa{ // оголошення структури, що описує анкету
студента
    char name[20];
    int age;
    char group[10];
} studArray[size], studArray2[size]; // оголошення двох масивів
структур

int main(int argc, char** argv) {
    FILE *fp;
        //заповнюємо масив структур
    for (int i=0; i<size; i++){
        cout<<"Name = ";cin>>studArray[i].name;
        cout<<"Age = "; cin>>studArray[i].age;
        cout<<"Group = ";cin>>studArray[i].group;
    }
    // Відкриваємо файл для запису.
    if ((fp=fopen("balance", "w"))==NULL){
        printf("can't open file");    return 1;    }
    // Одним викликом зберігаємо весь масив studArray.
    fwrite(studArray, sizeof studArray, 1, fp);
    fclose(fp);
    // Відкриваємо файл для зчитування.
    if ((fp=fopen("balance", "r"))==NULL){
        printf("can't open file");
        return 1;
    }
        // Одним "махом" зчитуємо весь масив studArray2.
    fread(studArray2, sizeof studArray2, 1, fp);
        // Відображаємо вміст масиву.
    for (int i=0; i<size; i++){
        cout<<"Student "<<i<<" "<<studArray2[i].name;

```



```

        cout<<" "<<studArray2[i].age<<"
"<<studArray2[i].group<<endl;
    }
    fclose(fp); return 0;
}

```

6.3. Work program

6.3.1. Start the IDLE environment.

6.3.2. Create an algorithm for a program for creating and reading text files according to your version.

6.3.1. Write a program for task 1 according to your version. Use an array of structures to describe the information in the file.

Program requirements

Enter input data using the input operator; display the result of program execution according to the task.

6.4. Hardware and software

6.4.1. Personal computer.

6.4.2. Software: IDLE

6.5. Questions

6.5.1 What input/output methods does C++ support?

6.5.2. What is a file and a stream?

6.5.3. What types of streams do you know?

6.5.4. What input and output functions do you know?

6.5.5. In which header files are the prototypes of input/output functions located?

6.5.6. What functions perform input/output of a string of characters?

6.5.7. What is a file pointer, its definition?

6.5.8. List the main functions of working with files.

6.5.9. What file opening modes do you know, how are they set?

6.5.10. What is the purpose of the exit() function?

6.5.11. What function is intended for closing files?

6.5.12. What functions are intended for reading and writing data blocks.

6.5.13. What function establishes access to a specific file element?

6.5.14. What is a structure?

Task 1

Variant number:

1. Create a file “BOOK” that contains information about the books in your library. The information should include the author’s last name, book title, publisher, and year of publication. Using the generated file, print out information about books published by the “Prosvita” publishing house.

2. Create a file that contains information about household refrigerators: name of the refrigerator, cost, volume of the refrigerator compartment, manufacturer. Using the generated file, print out information about refrigerators that cost more than 55,000 hryvnias.

3. During a football game, a file is created that includes the player’s last name and the number of points scored during the game. Using the generated file, print out the last names of the 3 most productive players on the team.

4. Create a file that contains information about trains that go to Kyiv (train number, full name, travel time). Using the generated file, print out information about trains whose travel time does not exceed 6 hours.

5. Generate a file containing information about trains that depart from Zdolbuniv station (including transit trains): train number, destination station, departure time, travel time. Using the generated file, print out information about trains that go to Lviv.

6. Generate a file containing information about students who were born in the summer (June, July, August).

7. Create a file “stud”, which has the following structure: student’s last name, year of birth, gender. Print, using the file, a list of male students and indicate their age. At the end of the list, print the average age of the students.

8. Create a file “EXAM” based on the results of the examination session (three exams). Information about students is entered in symbolic form in the following order: last name **N1**N2**N3, where N is the score. Using the file, print the results of the session in the form of a table. Provide for printing the table header with the names of the disciplines.

9. Record the ski race protocol in the file “SCI”. For each participant, enter: last name, time of participation in the race (hour, minute, second). Using the generated file, print the names of the participants who met the standards (time less than 30 minutes).

10. Create a file “CAR”, which contains information about car owners: last name, brand, color. Using the created file, print out information about owners who have gray “Audi” cars.

11. Create a file containing data about books in your personal library: author’s last name, book title, publisher, year of publication, number of pages. Using the created file, print out information about books published by the “Mir” publishing house, and also calculate the total number of such books.

12. Create a file containing information about tape recorders: brand, its cost, class. Using the created file, print out information about first-class tape recorders.

13. Create a file - a telephone directory. The information should contain the subscriber’s last name, first name and patronymic, phone number. Print out the entire directory.

14. Create a file “FRIEND” from the last names and dates of birth of your friends. Using the generated file, print the names of those born in winter.

15. chess players participate in a chess tournament. Create a file that includes the names and game results (win - 1, draw -1, loss - 0). Using the generated file, process the results of the championship and print the names of the teams that took the prize places and the number of victories of each team.

16. teams participate in the football championship. Create a file of teams and match results (win - 2 points, draw -1, loss - 0). Using the generated file, process the results of the championship and print the names of the teams that took the prize places and the number of victories of each team.

17. In the attendance log, students’ missed classes are noted every day for each subject. Create a file that includes the last name and date of attendance for one subject (1 - present, 0 - absent) by each student in the group. Using the created file, create a list of those students who have more than 5 absences.

18. Create a file that includes the last names and grades of students during the semester in the discipline “Computer Engineering”. Using the created file, print the last names of those students whose average grade in the discipline is 4.

19. 20 sports journalists were asked to name the 3 best football players of the season. Create a file that includes the last names of the football players, the number of points scored by each journalist (3 - first place, 2 - second place, 1 - third place). Using the created file, identify the 3 best players.

20. Create a file that includes the names and positions of teachers who teach the discipline “Computer Engineering” at all faculties of the institute.

Using the created file, print the names of those teachers that begin with the letter “B”.

21. Create a file that contains the names and first names of students in your group. Using the created file, print the names of those students that begin with the letter “K”.

22. Create a file that contains the names and initials of teachers who teach in your group and the corresponding subjects. Print the names of teachers from the Department of Electrical Engineering and Automation.

23. Create a file that contains the results of your certification. Using the created file, print the subjects in which you received a “5”. If you did not receive any five, then print the subjects in which you received a grade of “4” or “3”.

24. Create a file “Journal”, which contains information about the library’s journals. The information should include the author’s last name, article title, journal title, publisher and year of publication. Using the generated file, print out information about journals published by the publishing house “NUVGP file”.

25. Create a file containing information about laptops: manufacturer’s name, cost, screen diagonal. Using the generated file, print out information about laptops costing over 25,000 hryvnias.

26. Create a file “GROUP” from the last names and dates of birth of students in the group. Using the generated file, print out the last names of those born in the summer.

27. In the attendance log, students’ missed classes are noted daily for each subject. Create a file that includes the last name and date of attendance for one subject (1 - present, 0 - absent) by each student in the group. Using the generated file, create a list of students who have more than 10 absences.

28. Create a file that includes the names and grades of students during the semester in the discipline “Higher Mathematics”. Using the generated file, print the names of those students whose average score in the discipline is 3.

29. Protocol of the swimming competition in the file “Competition”. For each participant, enter: last name, time of participation in the competition (minutes, seconds). Using the generated file, print the names of participants who met the standards (time less than 2 minutes).

30. Create a file “CAR”, which contains information about car owners: last name, make, color. Using the generated file, print information about owners who have “Ford” cars.

№7. Developing programs with custom classes. Working with classes and objects

7.1. The purpose of the work

Formation of skills in working with classes, algorithms for their declaration and processing. Acquire object-oriented programming skills.

7.2. Brief theoretical information

Class

The main difference between C++ and C is the ability to process a new data type - class. A class is a user-defined data type (variables of any type, other classes or pointers), which is created to describe an abstract set of objects - members of a class. The idea of a class is to combine data and algorithms for processing them. Data are called class fields, algorithms - methods, and the actual combination is encapsulation. Methods process fields and external data, in fact, they implement the idea of a class. Classes have the inheritance property, which ensures that the descendant class (hereinafter - derived class) uses the fields and methods of the base class (ancestor, parent class). Each class can have an arbitrary number of descendants, which makes it possible to create hierarchical inheritance trees. A descendant can overlap some methods of the ancestor, and then a method with the same name for different classes will be executed differently. This is called method polymorphism.

Encapsulation

Creating a new class is similar to creating a new structure:

```
class <class name>
{
  <access specifier>:
  <field type 1> <field names 1>;

  <field type N> <field names N>;
  <class method declarations or descriptions>;
};
```

Class methods are functions that are defined for fields or external variables.

In general, the concept of a class resembles the concept of a structure in C++, with the exception of a number of points: - it contains a number of specifications for accessing class objects; - as a rule, it includes elements - functions that specify the rules for processing class objects; - special functions - a constructor and a destructor - are used to create and destroy class objects. Unlike the fields of a structure, which are always accessible, members and methods in a class can have different levels of access.

Access specifiers are described as follows:

Access specifier	Description
private	Accessibility only for class methods
protected	Available only for class methods and derived class methods
public	Accessibility for any external function

The access specifier may be absent in the class description. In this case, the private specifier is active by default, unless explicitly specified otherwise. Note that a structure is completely similar to a class, except that its fields and methods are public by default. You cannot inherit descendant classes from structures.

For example, let's create a TPoint class that will contain the coordinates of a point and the following methods: lighting, extinguishing, and moving the point. The description of the TPoint class looks like this:

```
class TPoint
{ protected:
int x,y; // Координати
public:
TPoint(int a, int b); // Ініціалізує поля координат числами a і b
void On() // Рисує точку поточним кольором
{Draw(getcolor());}
void Off() // Вимикає точку - малює її кольором фону
{Draw(getbkcolor());}
virtual void Draw(int color) // Рисує точку кольором color
{putpixel(x, y, color);}
```

```
// Переміщає точку на екрані на dx вправо і на dy вниз  
void Move (int dx, int dy);  
};
```

The TPoint function creates an instance of a class and fills its fields with specific values. Such a class method is called a constructor. A constructor always has a class name. The meaning of the virtual keyword in the description of the Draw() method will be explained below.

Several constructors can be defined in a class, which differ in the list of parameters. When creating class objects, a constructor with the corresponding number and type of input parameters will be called (constructor overloading).

A constructor may or may not have input parameters. A constructor that does not have input parameters is called a default constructor.

A class may or may not have an explicit constructor. Then the system automatically allocates memory and initializes the object in a standard way. The standard procedure does not take into account the features of the class, therefore it may be incorrect and lead to errors in the program.

A destructor is a function-method of a class that is responsible for correctly releasing memory when an object is destroyed. The destructor always has a class name, preceded by the "~" symbol. If the destructor is not defined in the program, it is generated automatically at the program compilation stage.

Local objects are deleted by the destructor when they go out of scope, and global objects - after the main() function is completed. The destructor, if necessary, can also perform any other actions - outputting the final values of data elements or text messages when debugging the program.

Constructors and destructors cannot return values, and therefore there is no result type in their description.

Note that it is advisable to change the values of class fields using methods. For example, you can change the location of a point using the Move method.

Outside of the class description, the method header looks like this:

```
<class name>::<method name>( <list of formal parameters>);
```

Let's describe the methods of the created class:

```
TPoint::TPoint(int a, int b)
```

```

{
  x = a; y = b;
} void TPoint::Move(int dx, int dy)
{
  Off();
  x+=dx; y+=dy;
  On();
}

```

A variable (object) of a class is declared similarly to other classes:

```
TPoint n(12,24), m(100,200);
```

A class method is called like this:

```
<object name>.<method name>(<list of actual parameters>);
```

You can declare and use a Point instance of the TPoint class, for example, like this:

```

TPoint Point(50,50);
Point.On();
Point.Move(35, 70);
Point.Of();

```

or using dynamic variables:

```

TPoint* PointPtr = new Point(100,100);
PointPtr->On();
PointPtr->Move(35, 70);
PointPtr->Of();

```

Operation overloading

A special operator method is defined for classes, namely:


```
<type>operator<symbol>(<formal parameters>)  
{<method body>}
```

In this case, all arithmetic operations, assignment commands, assignment commands combined with arithmetic operations, and various pairs of parentheses can be used as symbols, for example: **operator+**, **operator/=**, **operator=**, **operator()**, etc. The rules for describing your own operator methods are similar to the rules for creating regular functions or methods.

Using the TPoint class and operator(), draw 1000 points randomly placed on the screen.

```
#include <graphics.h>  
#include <conio.h>  
#include <stdlib.h>  
class TPoint  
{  
protected: int x, y;  
public:  
TPoint(int a = 0, int b = 0)  
{  
x = a; y = b;  
}  
void On()  
{Draw(getcolor());}  
void Off()  
{Draw(getbkcolor());}  
virtual void Draw(int color)  
{putpixel(x, y, color);}  
TPoint& operator()(int i, int j)  
{  
x = i, y = j;  
return *this;  
}  
};  
void main()  
{
```

```

int gdriver = DETECT, gmode, errorcode;
initgraph(&gdriver, &gmode, "");
TPoint P;
randomize();
for(int i = 0; i < 1000; i++) P(random(i), random(i)).On();
getch();
closegraph();
}

```

In this program, we use operator() to set the coordinates of a point. To be able to use, for example, the commands P(30,80).On() or P(100,200).Off(), operator() must be of type Tpoint. Therefore, in the description of the operator-method operator(), we use the reference TPoint&. The return command must return an instance (variable value) of type TPoint. Since at the time of the class description, the name of the variable of this class is unknown, in C++ there is a special keyword **this** - a pointer to this variable.

Example of operator overloading

A program for working with complex numbers. The members of the class are the real and imaginary parts of a complex number. The methods of the class are: displaying a complex number on the screen; a constructor for initializing complex numbers; a destructor; calculating and outputting the modulus of a complex number; calculating and outputting the argument of a complex number.

To initialize a complex number, an overloaded constructor has been created:

complex() initialization of a complex number of the form $a + i \cdot a$ (if the parameter in parentheses is not specified, the complex number $1 + i$ is initialized by default);

complex(double, double) initialization of a complex number of the form $a + i \cdot b$.

The destructor **~complex()** has also been created, although its use is not mandatory for our class of problems.

To work with complex numbers, overloaded operations have been described:

bool operator ==(complex) comparison of two complex numbers;

complex operator +(complex) addition of two complex numbers;

complex operator -(complex) subtraction of two complex numbers;
complex operator *(complex) multiplication of two complex numbers;
complex operator /(complex) division of two complex numbers;
complex operator =(complex) assignment operation for complex numbers;
void operator ++() prefix increase by 1 of the real part;
void operator ++(int) postfix increment by 1 of the imaginary part;
void operator --() prefix decrement by 1 of the real part;
void operator --(int) postfix decrement by 1 of the imaginary part.

```
#include <iostream>
#include <math.h>
#include <windows.h>
#define pi 3.1415926
using namespace std;

class complex {
private:
    double x;
    double y;

public:
    complex();
    complex(double, double);
    ~complex();

    double modul();
    double argument();
    void show_complex();

    void operator++();
    void operator++(int);
    void operator--();
    void operator--(int);
    complex operator+(complex);
    complex operator-(complex);
```

```

    complex operator*(complex);
    complex operator/(complex);
    complex operator=(complex);
    bool operator==(complex);
};

bool complex::operator==(complex chislo)
// Порівняння двох комплексних чисел
{
    if (x == chislo.x && y == chislo.y) {
        return true;
    }
    return false;
};

complex complex::operator+(complex a)
//Додавання
{
    complex temp;
    temp.x = x + a.x;
    temp.y = y + a.y;
    return temp;
};

complex complex::operator-(complex a)
//Віднімання
{
    x = x - a.x;
    y = y - a.y;
    return *this;
};

complex complex::operator*(complex a)
//Множення
{
    x = x * a.x - y * a.y;

```

```
    y = y * a.x + x * a.y;  
    return *this;  
};
```

```
complex complex::operator/(complex a)  
//Ділення  
{  
    x = x * a.x + y * a.y;  
    y = y * a.x - x * a.y;  
    return *this;  
};
```

```
complex complex::operator=(complex a)  
//Операція присвоєння  
{  
    x = a.x;  
    y = a.y;  
    return *this;  
};
```

```
//Префіксне збільшення на 1 дійсної частини  
void complex::operator++()  
//Префіксне збільшення на 1 дійсної частини  
{  
    x++;  
};
```

```
void complex::operator--()  
//Префіксне зменшення на 1 дійсної частини  
{  
    x--;  
};
```

```
void complex::operator++(int)  
//Постфіксне збільшення на 1 уявної частини  
{
```

```

    y++;
};

void complex::operator--(int)
//Постфіксне зменшення на 1 уявної частини
{
    y--;
};

double complex::modul()
{
    return pow(x * x + y * y, 1 / 2.0);
};

double complex::argument()
{
    return atan2(y, x) * 180 / pi;
};
void complex::show_complex()
{
    if (y >= 0)
        cout << x << "+" << y << "i" << endl;
};

complex::complex()
//Конструктор 1
{
    x = 1;
    y = 1;
};

complex::complex(double x0, double y0)
//Конструктор 2
{
    x = x0;
    y = y0;
};

```

```

};

complex::~~complex()
    //Деструктор
    {};

int main()
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    /*Ввід комплексного числа c1 за допомогою конструктора 1 */
    complex c1;
    cout << "Комплексне число c1=\t";
    c1.show_complex();

    /*Ініціалізація числа c2 за допомогою конструктора 2 */
    complex c2(-1.0, 2.0);
    cout << "Комплексне число c2=\t";
    c2.show_complex();
    cout << endl;
    c2++;
    cout << "Комплексне число c2 після префіксної операції ++
=\t";
    c2.show_complex();

    ++c2;
    cout << "Комплексне число c2 після постфіксної операції ++
=\t";
    c2.show_complex();

    c2 = c2 - c1;
    cout << "Комплексне число c2 після віднімання віднього c1=\t";
    c2.show_complex();

    cout << endl;
    complex c3;

```

```

c3 = ((c2 + c1) / (c2 + c2)) * c1;
cout << "Обчислення виразу c3=((c2+c1)/(c2+c2))*c1 : c3=\t";
c3.show_complex();

cout << "Порівняння чисел c2 та c3 -\t";
if (c2 == c3)
    cout << "Числа рівні \n";
else
    cout << "Числа не рівні між собою\n";

cout << "Модуль комплексного числа c3=\t" << c3.modul() <<
endl;
cout << "Аргумент комплексного числа c3=\t" <<
c3.argument();

return 0;
}

```

7.4. Work program

7.4.1. Start the programming environment.

7.4.2. Write a program to solve the problem according to your.

Program requirements

Create a class according to the version. Supplement the class with overloaded constructors (one - without parameters, initializes the class fields with default values, e.g. 0, the second - parameterized, initializes the class fields with data entered by the user from the keyboard) and a destructor (displays a message about the destruction of the object).

7.4. Hardware and software

7.4.1. Personal computer.

7.4.2. Software: IDLE.

7.5. Questions

7.5.1. What are class fields?

7.5.2. What are class methods?

- 7.5.3. What properties do classes have?
- 7.5.4. What is class inheritance?
- 7.5.5. What is method polymorphism?
- 7.5.6. How is operator overloading implemented in C++?
- 7.5.7. What is the purpose of a class constructor?

Task 1

Variant number:

1. Create a class of fractions. The members of the class are the numerator and denominator. The methods of the class are: inputting a fraction from the keyboard; outputting a fraction to the screen; calculating and outputting the value of the fraction. Supplement the class with overloaded operations "+", "-", "*", "/". Write a program that demonstrates working with the class.

2. Create a class of vectors. The members of the class are the Cartesian coordinates of the beginning and end of the vector in space. The methods of the class are: inputting a vector from the keyboard; outputting a vector to the screen; calculating and outputting the length of the vector. Supplement the class with overloaded operations "+", "-". Write a program that demonstrates working with the class.

3. Create a class of matrices of size 3×3 . The members of the class are the elements of the matrix. The methods of the class are: inputting a matrix from the keyboard; outputting a matrix to the screen; calculating and outputting the determinant of the matrix. Add the class with overloaded operations "+", "-". Write a program that demonstrates working with the class.

4. Create a class of segments on a plane. The members of the class are the coordinates of the ends of the segment. The methods of the class are: entering a segment from the keyboard; displaying the segment on the screen; calculating and displaying the length of the segment on the screen. Add the class with overloaded operations "+" (adding segments according to the rules of adding vectors), "-" (subtracting segments according to the rules of subtracting vectors). Write a program that demonstrates working with the class.

5. Create a class for working with time within a day. The members of the class are hours, minutes and seconds. The methods of the class are: entering time from the keyboard; displaying time on the screen; calculating and displaying the time remaining until the end of the day. Add the class with overloaded

operations "+", "-". For all operations, provide for checking whether the obtained value is outside the permissible limits. Write a program that demonstrates working with the class.

6. Create a class for working with a circle. The members of the class are the radius of the circle and the coordinates of its center. The methods of the class are: entering a circle from the keyboard; displaying a circle on the screen; calculating the area of the circle and the circumference of the circle and displaying the result on the screen. Supplement the class with overloaded operations "*" (multiplying the radius by a number); "/" (dividing the radius by a number). Write a program that demonstrates working with the class.

7. Create a class of rectangles. The members of the class are the lengths of the sides of the rectangle. The methods of the class are: entering a rectangle from the keyboard; displaying a rectangle on the screen; calculating the perimeter and its area and displaying the result on the screen. Supplement the class with overloaded operations prefix ++ increases the lengths of the sides of the rectangle by 1; prefix -- decreases the lengths of the sides of the rectangle by 1. Write a program that demonstrates working with the class.

8. Create a class of triangles. The members of the class are the lengths of the sides of the triangle. The class methods are: inputting a triangle from the keyboard; outputting a triangle to the screen; calculating the perimeter and area and outputting the result to the screen. Supplement the class with overloaded operations prefix ++ increases the lengths of the sides of the triangle by 1; prefix -- decreases the lengths of the sides of the triangle by 1. Write a program that demonstrates working with the class.

9. Create a class for working with dates. The members of the class are the year, month, and day of the month. The class methods are: inputting a date from the keyboard; outputting a date to the screen; calculating and outputting the time of year corresponding to a given date. Supplement the class with overloaded operations "+" (adding years, months, and days, respectively); "-" (subtracting years, months, and days, respectively). For all operations, provide for checking whether the resulting value is outside the permissible limits. Write a program that demonstrates working with the class.

10. Create a class of matrices of size 4×4 . The members of the class are its elements. The methods of the class are: inputting a matrix; outputting a matrix to the screen; calculating and outputting the maximum element of the matrix to the screen. Supplement the class with overloaded operations "+", "-". Write a

program that demonstrates working with the class.

11. Create a class of points. The members of the class are the coordinates of a point on a plane. The methods of the class are: entering a point from the keyboard; displaying the coordinates of a point on the screen; calculating and displaying the number of the quadrant of the coordinate system in which the point is located. Supplement the class with overloaded operations "+" (adding the corresponding coordinates of two points); "-" (subtracting the corresponding coordinates of two points). Write a program that demonstrates working with the class.

12. Create a class of polynomials of dimension 4. The members of the class are the coefficients of a polynomial. The methods of the class are: entering the coefficients of a polynomial; displaying the polynomial on the screen; calculating and displaying the value of a polynomial for a given value x . Supplement the class with overloaded operations "+" (term-wise addition of polynomials); "-" (term-wise subtraction of polynomials). Write a program that demonstrates working with the class.

13. Create a class of triads of numbers. The members of the class are three real numbers. The methods of the class are: entering three numbers; displaying numbers on the screen; calculation and output of the largest and smallest number. Supplement the class with overloaded operations "+" (term-wise addition of triads of numbers); "-" (term-wise subtraction of triads of numbers). Write a program that demonstrates working with the class.

14. Create an ellipses class. The members of the class are the lengths of the semiaxes of the ellipse. The methods of the class are: inputting an ellipse from the keyboard; outputting the ellipse to the screen; calculating the area and perimeter of the ellipse and outputting the result to the screen. Supplement the class with overloaded operations "+" (adding the corresponding semiaxes of the ellipse); "*" (multiplying the semiaxes of the ellipse by a number). Write a program that demonstrates working with the class.

15. Create a class of points in space. The members of the class are the Cartesian coordinates of a point. The methods of the class are: inputting a point from the keyboard; outputting the coordinates of a point to the screen; calculating and outputting the polar coordinates of a point. Add the class with overloaded operations "+" (point coordinates are added); "-" (point coordinates are subtracted). Write a program that demonstrates working with the class.

16. Create a class of fractions. The members of the class are the numerator

and denominator. The methods of the class are: inputting a fraction from the keyboard; outputting a fraction to the screen; calculating and outputting the value of the fraction. Supplement the class with overloaded operations "+", "-", "*", "/". Write a program that demonstrates working with the class.

17. Create a class of vectors. The members of the class are the Cartesian coordinates of the beginning and end of the vector in space. The methods of the class are: inputting a vector from the keyboard; outputting a vector to the screen; calculating and outputting the length of the vector. Supplement the class with overloaded operations "+", "-". Write a program that demonstrates working with the class.

18. Create a class of matrices of size 3×3 . The members of the class are the elements of the matrix. The methods of the class are: inputting a matrix from the keyboard; outputting a matrix to the screen; calculating and outputting the determinant of the matrix. Add the class with overloaded operations "+", "-". Write a program that demonstrates working with the class.

19. Create a class of segments on a plane. The members of the class are the coordinates of the ends of the segment. The methods of the class are: entering a segment from the keyboard; displaying the segment on the screen; calculating and displaying the length of the segment on the screen. Add the class with overloaded operations "+" (adding segments according to the rules of adding vectors), "-" (subtracting segments according to the rules of subtracting vectors). Write a program that demonstrates working with the class.

20. Create a class for working with time within a day. The members of the class are hours, minutes and seconds. The methods of the class are: entering time from the keyboard; displaying time on the screen; calculating and displaying the time remaining until the end of the day. Add the class with overloaded operations "+", "-". For all operations, provide for checking whether the obtained value is outside the permissible limits. Write a program that demonstrates working with the class.

21. Create a class for working with a circle. The members of the class are the radius of the circle and the coordinates of its center. The methods of the class are: entering a circle from the keyboard; displaying a circle on the screen; calculating the area of the circle and the circumference of the circle and displaying the result on the screen. Supplement the class with overloaded operations "*" (multiplying the radius by a number); "/" (dividing the radius by a number). Write a program that demonstrates working with the class.

22. Create a class of rectangles. The members of the class are the lengths of the sides of the rectangle. The methods of the class are: entering a rectangle from the keyboard; displaying a rectangle on the screen; calculating the perimeter and its area and displaying the result on the screen. Supplement the class with overloaded operations prefix ++ increases the lengths of the sides of the rectangle by 1; prefix -- decreases the lengths of the sides of the rectangle by 1. Write a program that demonstrates working with the class.

23. Create a class of triangles. The members of the class are the lengths of the sides of the triangle. The class methods are: inputting a triangle from the keyboard; outputting a triangle to the screen; calculating the perimeter and area and outputting the result to the screen. Supplement the class with overloaded operations prefix ++ increases the lengths of the sides of the triangle by 1; prefix -- decreases the lengths of the sides of the triangle by 1. Write a program that demonstrates working with the class.

24. Create a class for working with dates. The members of the class are the year, month, and day of the month. The class methods are: inputting a date from the keyboard; outputting a date to the screen; calculating and outputting the time of year corresponding to a given date. Supplement the class with overloaded operations "+" (adding years, months, and days, respectively); "-" (subtracting years, months, and days, respectively). For all operations, provide for checking whether the resulting value is outside the permissible limits. Write a program that demonstrates working with the class.

25. Create a class of matrices of size 4×4 . The members of the class are its elements. The methods of the class are: inputting a matrix; outputting a matrix to the screen; calculating and outputting the maximum element of the matrix to the screen. Supplement the class with overloaded operations "+", "-". Write a program that demonstrates working with the class.

26. Create a class of points. The members of the class are the coordinates of a point on a plane. The methods of the class are: entering a point from the keyboard; displaying the coordinates of a point on the screen; calculating and displaying the number of the quadrant of the coordinate system in which the point is located. Supplement the class with overloaded operations "+" (adding the corresponding coordinates of two points); "-" (subtracting the corresponding coordinates of two points). Write a program that demonstrates working with the class.

27. Create a class of polynomials of dimension 4. The members of the class

are the coefficients of a polynomial. The methods of the class are: entering the coefficients of a polynomial; displaying the polynomial on the screen; calculating and displaying the value of a polynomial for a given value x . Supplement the class with overloaded operations "+" (term-wise addition of polynomials); "-" (term-wise subtraction of polynomials). Write a program that demonstrates working with the class.

28. Create a class of triads of numbers. The members of the class are three real numbers. The methods of the class are: entering three numbers; displaying numbers on the screen; calculation and output of the largest and smallest number. Supplement the class with overloaded operations "+" (term-wise addition of triads of numbers); "-" (term-wise subtraction of triads of numbers). Write a program that demonstrates working with the class.

29. Create an ellipses class. The members of the class are the lengths of the semi-axes of the ellipse. The methods of the class are: inputting an ellipse from the keyboard; outputting the ellipse to the screen; calculating the area and perimeter of the ellipse and outputting the result to the screen. Supplement the class with overloaded operations "+" (adding the corresponding semi-axes of the ellipse); "*" (multiplying the semi-axes of the ellipse by a number). Write a program that demonstrates working with the class.

30. Create a class of points in space. The members of the class are the Cartesian coordinates of a point. The methods of the class are: inputting a point from the keyboard; outputting the coordinates of a point to the screen; calculating and outputting the polar coordinates of a point. Add the class with overloaded operations "+" (point coordinates are added); "-" (point coordinates are subtracted). Write a program that demonstrates working with the class.

№8. Developing programs with custom classes. Inheritance

8.1. The purpose of the work

Learn to work with classes and objects. Acquire object-oriented programming skills.

8.2. Brief theoretical information

Inheritance

The general form used to declare inheritance is:

```
class <descendant name> : <access> <ancestor name>  
{  
<added class fields>;  
<declarations or descriptions of added and overridden methods>;  
}
```

The class whose properties are inherited is called the base class, and the class that inherits the properties of the base class is called the derived class. Here, access is one of three keywords: public, private, or protected.

Accessing elements of the base class

The access specifier determines how elements of the base class are inherited by the derived class. If the access specifier of the inherited base class is public, then all public members of the base class become public in the derived class. If the access specifier of the inherited base class is private, then all public members of the base class become private in the derived class. In both cases, all private members of the base class remain private and inaccessible to the derived class. If the access specifier is private, then the public members of the base class become private in the derived class, but these members remain accessible to the member functions of the derived class.

Sometimes there is a situation when the members of the base class, while remaining private, were accessible to the derived class. To implement this idea, C++ introduced the access specifier protected (protected). The access specifier protected is equivalent to private with the only difference: protected members of

the base class are accessible to members of all derived classes of this base class. Outside the base class or derived classes, protected members are inaccessible. The access specifier `protected` can be anywhere in the class declaration, although, as a rule, it is placed after the declaration of private members and before the declaration of public members. The full general form of a class declaration is as follows:

```
class class_name
{
... // private members
protected:
... // protected members
public:
... // public members
};
```

Constructors, Destructors, and Inheritance

A base class, a derived class, or both can have constructors and/or destructors. If both the base and derived classes have constructors and destructors, then the constructors are executed in the order of inheritance, and the destructors are executed in the reverse order. Thus, the base class constructor is executed before the derived class constructor. For destructors, the reverse order is correct: the derived class destructor is executed before the base class destructor.

When considering the concept of inheritance, it is necessary to pay attention to the peculiarities of passing arguments for the derived and base class constructors. When initialization is performed only in the derived class, the arguments are passed in the usual way. However, some difficulties arise when passing an argument to the base class constructor. First of all, all the necessary arguments of the base and derived classes are passed to the derived class constructor. Then, thanks to the extended form of the declaration of the derived class constructor, the corresponding arguments are passed further to the base class.

The syntax for passing arguments from a derived class to a base class is as follows:


```

derived_class_name ( arg_list1 ) :
base_class_name ( arg_list2 )
{
... //derived class constructor body
}

```

It is permissible to use the same arguments for both base and derived classes. It is also permissible for a derived class to ignore all arguments and pass them directly to the base class.

In most cases, the base and derived class constructors do not use the same arguments. Then, if you need to pass one or more parameters to each class constructor, you should pass all the arguments required by the constructors of the derived class to the derived class constructor. Then, the derived class constructor simply passes the arguments it needs to the base constructor.

Multiple Inheritance

There are two ways in which a derived class can inherit more than one base class. First, a derived class can be used as a base class for another derived class, creating a multi-level class hierarchy. In this case, the original base class is an indirect base class for another derived class. Second, a derived class can directly inherit from more than one base class. In such a situation, the combination of two or more base classes helps create a derived class.

When a class is used as a base for a derived class, which in turn is a base for another derived class, the constructors of these three classes are called in the order of inheritance. Destructors are called in the reverse order.

If a derived class directly inherits multiple base classes, the following extended declaration is used:

```

class derived_class_name :
access_spec1 base_class_name1,
access_spec2 base_class_name2,
...
access_specN base_class_nameN
{
... // class body
}

```

When multiple base classes are inherited, the constructors are used from left to right in the order specified in the derived class declaration. Destructors are executed in the reverse order. When a class inherits multiple base classes whose constructors require arguments, the derived class passes those arguments using the extended form of the derived class constructor declaration:

```

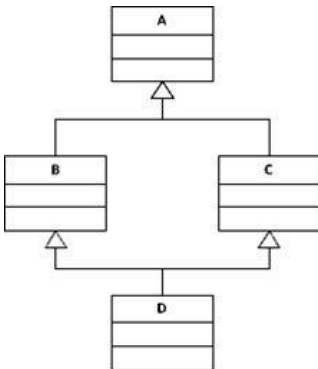
derived_class_name ( arg_list ):
    base_class_name1( arg_list1 ),
    base_class_name2( arg_list2 ),
    . . .
    base_class_nameN( arg_listN )
    {
    ... // derived class constructor body
    }

```

When a derived class inherits a class hierarchy, each derived class must pass the required arguments to the previous base class in the chain.

Virtual base classes

In the case of multiple direct inheritance by a derived class from the same base class, a problem may arise. Consider this option:



Here, the base class D is inherited by the derived classes B and C. The derived class A directly inherits these classes. This means that class D is actually inherited by class A twice – through classes B and C. However, if a member of class Base is used in class A, this will cause ambiguity, as it will have two copies of class D. To prevent this situation, C++ includes the virtual base class mechanism. In this case, the base class access specifier must be preceded by the

keyword `virtual`, for example:

```

class derived2: virtual public base

```

8.3. Work program

8.3.1. Launch the programming environment

8.3.2. Write a program to solve the problem according to your variant number.

Program requirements

Create a class in accordance with the specified subject area. Provide the ability to work with an arbitrary number of records, and also implement:

- constructors without parameters and with parameters;
- a function for displaying information about the object on the screen;
- a function for searching for the necessary information by a specific attribute;
- a function for editing records.

When developing a program, data protection should be implemented (description with the private modifier) to isolate the data elements of the class from subroutines in which this class is used.

8.4. Hardware and Software

8.4.1. Personal computer.

8.4.2. Software: IDLE.

8.5 Questions

8.5.1. Explain the OOP principle of inheritance. Give examples.

8.5.2. What access specifiers do you know?

8.5.3. What happens to the public and private members of a base class if the base class is inherited as public by a derived class?

8.5.4. What happens to the private and public members of a base class if the base class is inherited as private by a derived class?

8.5.5. Explain why the protected protection category is needed?

8.5.6. What is multiple inheritance?

Task 1

Variant number:

1. Create a base class **FIGURE**, in which the coordinates of one of the vertices of a geometric figure are defined. Create derived classes: **RECTANGLE** (additionally the lengths of two sides are defined), **CIRCLE** (additionally the

radius is defined), RIGHT TRIANGLE (additionally the lengths of the legs are defined).

2. Create a class FUNCTION for calculating the dependence of y on x . Create derived classes: LINEAR FUNCTION ($y(x)=ax+b$, two real values are added - coefficients a and b), QUADRATIC FUNCTION ($y(x)=cx^2+ax+b$, three real values a , b , c are added), CUBIC FUNCTION ($y(x)=dx^3+cx^2+ax+b$, four real values a , b , c , d are added).

3. Create a base class PROGRESSION with one valid data field - the first term of the progression. Create derived classes: ARITHMETIC PROGRESSION and GEOMETRIC PROGRESSION. Each class has an additional field of type double - a constant difference for arithmetic and a constant ratio for geometric progression. Define functions for calculating the sum and the n th element.

4. Create a base class - a geometric QUADRANT, in which the coordinates of the lower left vertex of the geometric figure are defined, and derived classes - SQUARE (has an additional field of type double - the length of a side), RECTANGLE (has two additional fields of type double - the lengths of two sides), TRAPEZONE (the lengths of four sides are specified).

5. Create a base class FIGURE, in which the coordinates of one of the vertices of the geometric figure are defined. Create derived classes: CUBE (has an additional field of type double - edge length), TETRAHEDRON (regular, has an additional field of type double - edge length), SPHERE (has an additional field of type double - radius).

6. Create an abstract class CURVES for calculating the dependence of y on x . Create derived classes: LINE ($y(x)=ax+b$, two real values are added - coefficients a and b), PARABOLA ($y(x)=ax^2+b$, real values a , b are added), HYPERBOLA FUNCTION ($y(x)=ax^3+ b$, real values a , b are added).

7. Create a base class FIGURE, in which the coordinates of one of the vertices of a geometric figure are defined. Create derived classes: CONE (has additional fields of type double - base radius and height), CYLINDER (has additional fields of type double - base radius and height), SPHERE (has additional field of type double - radius).

8. Create a base class EQUATION for calculating the value of x . Create derived classes: class LINEAR EQUATIONS ($ax+b=0$, two real values are added - coefficients a and b) and class QUADRATIC EQUATIONS ($cx^2+ax+b=0$, three real values a , b , c are added).

9. Create an abstract class TRIANGLE. A triangle is given by two sides and the angle between them. Define derived classes RIGHT-ANGLED, ICESCOPE triangle. Add methods for calculating the area.

10. Create a class geometric SOLID, in which the coordinates of one of the vertices of a geometric figure are defined. Create derived classes PYRAMID (has an additional field of type double – height) and SPHERE (has an additional field of type double – radius). The pyramid class has derived classes TRIANGULAR (has an additional field of type double – base area) AND QUADRIGULAR (has a rectangle at the base, has additional fields of type double – side lengths).

11. Create a base class FIGURE, in which the coordinates of one of the vertices of a geometric figure are defined. Create derived classes: RECTANGLE (additionally the lengths of two sides are defined), CIRCLE (additionally the radius is defined), RIGHT TRIANGLE (additionally the lengths of the legs are defined).

12. Create a class FUNCTION for calculating the dependence of y on x. Create derived classes: LINEAR FUNCTION ($y(x)=ax+b$, two real values are added - coefficients a and b), QUADRATIC FUNCTION ($y(x)=cx^2+ax+b$, three real values a, b, c are added), CUBIC FUNCTION ($y(x)=dx^3+cx^2+ax+b$, four real values a, b, c, d are added).

13. Create a base class PROGRESSION with one valid data field - the first term of the progression. Create derived classes: ARITHMETIC PROGRESSION and GEOMETRIC PROGRESSION. Each class has an additional field of type double - a constant difference for arithmetic and a constant ratio for geometric progression. Define functions for calculating the sum and the nth element.

14. Create a base class - a geometric QUADRANT, in which the coordinates of the lower left vertex of the geometric figure are defined, and derived classes - SQUARE (has an additional field of type double - the length of a side), RECTANGLE (has two additional fields of type double - the lengths of two sides), TRAPEZONE (the lengths of four sides are specified).

15. Create a base class FIGURE, in which the coordinates of one of the vertices of the geometric figure are defined. Create derived classes: CUBE (has an additional field of type double – edge length), TETRAHEDRON (regular, has an additional field of type double – edge length), SPHERE (has an additional field of type double – radius).

16. Create an abstract class CURVES for calculating the dependence of y on x. Create derived classes: LINE ($y(x)=ax+b$, two real values are added - coefficients a and b), PARABOLA ($y(x)=ax^2+b$, real values a, b are added), HYPERBOLA FUNCTION ($y(x)=ax^3+ b$, real values a, b are added).

17. Create a base class FIGURE, in which the coordinates of one of the vertices of a geometric figure are defined. Create derived classes: CONE (has additional fields of type double – base radius and height), CYLINDER (has additional fields of type double – base radius and height), SPHERE (has additional field of type double – radius).

18. Create a base class EQUATION for calculating the value of x. Create derived classes: class LINEAR EQUATIONS ($ax+b=0$, two real values are added - coefficients a and b) and class QUADRATIC EQUATIONS ($cx^2+ax+b=0$, three real values a, b, c are added).

19. Create an abstract class TRIANGLE. A triangle is given by two sides and the angle between them. Define derived classes RIGHT-ANGLED, ICESCOPE triangle. Add methods for calculating the area.

20. Create a class geometric SOLID, in which the coordinates of one of the vertices of a geometric figure are defined. Create derived classes PYRAMID (has an additional field of type double – height) and SPHERE (has an additional field of type double – radius). The pyramid class has derived classes TRIANGULAR (has an additional field of type double – base area) AND QUADRIGULAR (has a rectangle at the base, has additional fields of type double – side lengths).

21. Create a base class FIGURE, in which the coordinates of one of the vertices of a geometric figure are defined. Create derived classes: RECTANGLE (additionally the lengths of two sides are defined), CIRCLE (additionally the radius is defined), RIGHT TRIANGLE (additionally the lengths of the legs are defined).

22. Create a class FUNCTION for calculating the dependence of y on x. Create derived classes: LINEAR FUNCTION ($y(x)=ax+b$, two real values are added - coefficients a and b), QUADRATIC FUNCTION ($y(x)=cx^2+ax+b$, three real values a, b, c are added), CUBIC FUNCTION ($y(x)=dx^3+cx^2+ax+b$, four real values a, b, c, d are added).

23. Create a base class PROGRESSION with one valid data field - the first term of the progression. Create derived classes: ARITHMETIC PROGRESSION and GEOMETRIC PROGRESSION. Each class has an

additional field of type double - a constant difference for arithmetic and a constant ratio for geometric progression. Define functions for calculating the sum and the nth element.

24. Create a base class - a geometric QUADRANT, in which the coordinates of the lower left vertex of the geometric figure are defined, and derived classes - SQUARE (has an additional field of type double - the length of a side), RECTANGLE (has two additional fields of type double - the lengths of two sides), TRAPEZONE (the lengths of four sides are specified).

25. Create a base class FIGURE, in which the coordinates of one of the vertices of the geometric figure are defined. Create derived classes: CUBE (has an additional field of type double – edge length), TETRAHEDRON (regular, has an additional field of type double – edge length), SPHERE (has an additional field of type double – radius).

26. Create an abstract class CURVES for calculating the dependence of y on x. Create derived classes: LINE ($y(x)=ax+b$, two real values are added - coefficients a and b), PARABOLA ($y(x)=ax^2+b$, real values a, b are added), HYPERBOLA FUNCTION ($y(x)=ax^3+b$, real values a, b are added).

27. Create a base class FIGURE, in which the coordinates of one of the vertices of a geometric figure are defined. Create derived classes: CONE (has additional fields of type double – base radius and height), CYLINDER (has additional fields of type double – base radius and height), SPHERE (has additional field of type double – radius).

28. Create a base class EQUATION for calculating the value of x. Create derived classes: class LINEAR EQUATIONS ($ax+b=0$, two real values are added - coefficients a and b) and class QUADRATIC EQUATIONS ($cx^2+ax+b=0$, three real values a, b, c are added).

29. Create an abstract class TRIANGLE. A triangle is given by two sides and the angle between them. Define derived classes RIGHT-ANGLED, ICESCOPE triangle. Add methods for calculating the area.

30. Create a class geometric SOLID, in which the coordinates of one of the vertices of a geometric figure are defined. Create derived classes PYRAMID (has an additional field of type double – height) and SPHERE (has an additional field of type double – radius). The pyramid class has derived classes TRIANGULAR (has an additional field of type double – base area) AND QUADRIGULAR (has a rectangle at the base, has additional fields of type double – side lengths).

List of recommended literature

1. Programming: Principles and Practice Using C++ is a Book (2nd Edition) / Bjarne Stroustrup. 2014. 1312 p.
2. C++. Основи програмування. Теорія та практика: підручник / О. Г. Трофименко, Ю. В. Прокоп, І. Г. Швайко, Л. М. Буката та ін. ; за ред. О. Г. Трофименко. Одеса : Фенікс, 2010. 544 с.
3. Програмування C++ в прикладах і задачах / Васильєв О. 2024. 382 p.
4. Вступ до програмування мовою C++. Організація обчислень : навч. посіб. / Ю. А. Белов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставровський. К. : Видавничо-поліграфічний центр "Київський університет", 2012. 175 с.