

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
“IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE”

**Boiarinova Y.E., Kuchmii O.O.,
Tarasenko-Klyatchenko O.V.**

**FUNDAMENTALS OF PROGRAMMING
BASIC CONSTRUCTIONS**

Laboratory Work Tutorial

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів, які навчаються
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення мультимедійних та
інформаційно-пошукових систем»)

Kyiv
Igor Sikorsky Kyiv Polytechnic Institute
2023

Рецензенти:

Клятченко Ярослав Михайлович, к.т.н., доцент

Заболотня Т.М., к.т.н., доцент

Відповідальний редактор

Легеза Віктор Петрович, д-р техн. наук, проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського

за поданням Вченої ради факультету прикладної математики

Electronic online educational publication

Boiarinova Yulia, PhD, Associate Professor

Kuchmii Oksana, Assistant

Tarasenko-Klyatchenko Oksana PhD, Associate Professor

FUNDAMENTALS OF PROGRAMMING. BASIC CONSTRUCTIONS

LABORATORY WORK TUTORIAL

FUNDAMENTALS OF PROGRAMMING. BASIC CONSTRUCTIONS: Laboratory Work Tutorial [Електронний ресурс]: tutorial is aimed at students of the specialty 121 “Software Engineering” (educational program «Software Engineering of Multimedia and Information Retrieval Systems») / Igor Sikorsky Kyiv Polytechnic Institute; Yulia E.Boiarinova, Oksana O.Kuchmii, Oksana V. Tarasenko-Klyatchenko. – Electronic text data (1 file: 10 MB). – Kyiv: Igor Sikorsky Kyiv Polytechnic Institute, 2023. – 116 p.

This tutorial is developed for familiarizing students with basic of programming in C. The tutorial includes the introduction, 6 chapters, tasks for laboratory work, 2 appendixes and a list of recommended literature. For each laboratory task there are examples of implementation, description of the task, theoretical information, guidelines. The tutorial is aimed at students of the specialty 121 “Software Engineering”, educational program “Software Engineering of Multimedia and Information Retrieval Systems” of the Faculty of Applied Mathematics of Igor Sikorsky Kyiv Polytechnic Institute.

Y.E.Boiarinova, O.O. Kuchmii, O.V. Tarasenko-Klyatchenko, 2023

Igor Sikorsky Kyiv Polytechnic Institute, 2023

INTRODUCTION.....	4
1 INTRODUCTION TO PROGRAMMING	5
1.1 ALGORITHMS AND PROGRAMS	5
1.2 PROGRAMMING LANGUAGES	6
1.3 HISTORY OF LANGUAGE C	8
1.4 CHARACTERISTICS OF C-SYSTEMS.....	9
1.5 PROGRAMMING COMPILERS.....	10
2 FAMILIARITY WITH THE LANGUAGE C.....	11
2.1 ALPHABET OF LANGUAGE C	11
2.2 VARIABLES AND TYPES OF VARIABLES, DECLARATION OF VARIABLES IN C LANGUAGE	12
2.3 OUTPUT FUNCTION IN C LANGUAGE	13
2.4 INPUT FUNCTION IN C LANGUAGE	15
2.5 ASSIGNMENT OPERATIONS	16
2.6 EQUALITY AND RELATION OPERATIONS.....	17
3 PROGRAM MANAGEMENT	18
3.1 MANAGEMENT STRUCTURES.....	18
3.1.1 <i>Management structure if/else</i>	19
3.1.2 <i>Operations for increment and decrement.....</i>	21
3.1.3 <i>Logical operations</i>	22
3.1.4 <i>Logical multiplication</i>	22
3.1.5 <i>Logical addition</i>	23
3.1.6 <i>Logical operation of negation</i>	23
3.1.7 <i>Structure with multiple choice switch</i>	24
3.2 REPETITION STRUCTURES	27
3.2.1 Repetition structure for	27
3.2.2 The repetition structure while	28
3.2.3 The repetition structure <i>do/while</i>	29
3.3 COMMAND BREAK AND CONTINUE	30
4 ARRAY	32
4.1 ARRAY IN PROGRAMMING (LANGUAGE C)	34
4.2 ARRAY INVERSION	35
4.3 ACTIONS WITH ARRAY ELEMENTS	38
4.4 MULTIDIMENSIONAL ARRAYS	41
4.5 INPUT / OUTPUT OF MULTIDIMENSIONAL ARRAYS.....	43
FUNCTIONS	45

5.1	FUNCTION DEFINITION.....	46
5.2	PROTOTYPE OF FUNCTION	49
5.3	HEADER FILES	49
5.4	FUNCTION CALL.....	50
INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)		52
TASKS		68
TASK 1. THE SIMPLEST OPERATORS OF THE C PROGRAMMING LANGUAGE.....		68
TASK 2. CALCULATE THE FUNCTION		76
TASK 3. DEVELOP A PROGRAM USING THE OPERATOR SWITCH.....		84
TASK 4. CALCULATION OF THE SUM.....		90
TASK 5. CREATE PROGRAM WITH ONE-DIMENSION ARRAY		97
TASK 6. USING OF FUNCTIONS		104
APPENDIX A.....		113
APPENDIX B		114
SOURCES.....		116

INTRODUCTION

Recent decades have been marked by the rapid development of information technologies, which is connected, among other things, with the emergence of new methods of developing software products. New programming languages, new tools, new software development technologies appeared. Nowadays, the process of creating software applications has become much easier, but their value and quality depends, as before, on the professionalism of the programmer, on the depth of his knowledge. Learning to program in the C language provides an opportunity for a teacher at a professional level to teach a student how to develop software applications, to illustrate and explain how huge programs are built from the elementary constructions of the language.

The C programming language is a typed general-purpose programming language developed in 1969-1973 by Bell Labs employee Dennis MacAlistair Ritchie as a development of the B language. Initially, the compiler from this language was developed for implementation on the UNIX operating system, but was later ported to many other platforms. According to the design of the C language, its constructions are very close to typical machine instructions, thanks to which it found use in projects for which assembly language was peculiar. The C programming language is currently used in the creation of operating systems, drivers, console-type applications, for controlling microprocessor technology. Many modern programming languages are built on the basic constructions of the C language. These include C++, C#, Java, Python, etc.

1 INTRODUCTION TO PROGRAMMING

1.1 Algorithms and programs

If a specialist is faced with a task related to calculation, he, first of all, develops an algorithm.

The procedure for solving the problem in the form

- actions to be performed;
- the sequence in which these actions must be performed;

is called an algorithm. In other words, an algorithm is a defined sequence of actions, the execution of which ensures the achievement of the final goal.

The algorithm can be described in words, can be depicted graphically (block diagram). If there is a task to implement an algorithm on a computer, it must be translated into a programming language.

A program is a defined sequence of actions written in a programming language, the execution of which will lead to a final goal.

If expressed in the language of a programmer, the actions to be performed are statements, and determining the sequence of their execution is called program control.

Algorithms themselves always consist of three possible structures:

- 1) following (linear structure);
- 2) branching (structures with condition verification);
- 3) repetition (cyclic structures).

Any combination of basic structures, or only one of them, may be present in a given particular algorithm.

In the dictionary of any language there are words to describe the three possible structures of the algorithm, special graphic shapes are designed to graphically represent the algorithm, in any programming language there are tools for the implementation of a linear structure, a structure with condition verification, and a cyclic structure.

1.2 Programming languages

Programmers develop programs in various programming languages, some of which are directly understandable to the computer, while others wander through an intermediate stage - translation. All languages can be divided into three general types:

- 1) machine languages;
- 2) assembly language;
- 3) high-level languages.

Each computer can understand its own machine language. This language is directly related to its hardware part. Machine languages generally consist of a sequence of numbers (usually 0 and 1), which are commands to perform elementary operations. A specific machine language can only be used with a specific type of computer. Machine languages are not easy to use

As computers spread, it became quite certain that programming in machine languages inhibits the development of computer technology. Then there are assembly languages, where English abbreviations are used instead of a sequence of numbers to represent elementary operations.

Translator programs called assemblers have been developed to convert assembly language programs into machine language programs.

But even in assembly language, it was still necessary to write a large number of instructions to implement the solution of simple problems. To speed up the programming process, high-level languages were developed, in which it is enough to write one statement to perform several actions:

Special programs called compilers are used to convert high-level language programs into machine language programs.

A high-level programming language is largely close to human language and frees you from the knowledge of the specifics of a specific processor.

High-level programming languages appeared, became popular, and disappeared with the advent of new computer technologies.

The high-level language FORTRAN (FORmula TRANslator) was developed by IBM in 1954-1957 for use in scientific and engineering applications that require complex mathematical calculations. To this day, software continues to be developed in this language.

Among the "long-lived" is the high-level language COBOL (Common Business Oriented Language), which was created in 1959. This language is used to develop commercial application programs that require high accuracy when processing large amounts of data.

The PASCAL programming language (named after the mathematician, physicist, writer, philosopher Blaise Pascal) was developed by the Swiss scientist Nicholas Wirth in 1968 specifically to teach programming to students. Today, it is already used not only for training, but also as a tool for developing professional software.

1.3 History of language C

The C language originates from two languages, BCPL and B. This language was developed by Dennis Ritchie in 1972. The C language is hardware-independent, and programs written in this language can be ported to a large number of systems.

At the end of the 70s, Kernighan and Ritchie's book "The C Programming Language" appeared. This is probably the most popular book in the field of programming.

The use of the C language for different types of computers led to the appearance of different variants of the language, which were often incompatible.

In 1989, the American National Standards Committee developed a standard for the C language, called ANSI C. Thanks to this standard, programs written in the C language can run virtually unchanged on most computer systems.

At Bell Labs, Björn Strastrup developed an addition to the C language, namely C++. The C++ language provides object-oriented programming, which, in turn, significantly increases the productivity of software development by a team of programmers.

Almost all operating systems were developed in C or C++ languages.

1.4 Characteristics of C-systems

A set of software products that allow you to develop programs in the C language are called C-systems.

All C systems basically consist of three parts: an environment, a programming language, and a standard library. Library functions are not part of the C language; they perform operations such as input/output, mathematical calculations.

During the development of a C program, there are six stages: editing, preprocessing, compilation, layout, loading, execution.

The first stage is intended for the development of the program text. The programmer types the program and performs the necessary corrections with the help of the editor program (the result is the original text, which is stored in a file with the extension "c" or "cpp").

At the second stage, the preprocessor of the C language executes special commands - directives of the preprocessor, which indicate what should be attached to the program file, what should be replaced in the program text, etc. The preprocessor is automatically started by the compiler.

The third stage is performed by the compiler. The compiler translates (translates) the program from the C language into machine language code (the result is object code, which is stored in a file with the extension ".obj").

At the fourth stage, an executable image is created (saved in a file with the extension "exe"), which, in addition to the program itself, contains all functions that are referenced in the program itself.

The fifth stage is called loading, during which the program is placed in the computer's memory.

And, finally, the sixth stage is the stage of program execution by the computer processor.

1.5 Programming compilers

Special programs called compilers are used to convert high-level language programs into machine language programs. The input of the compiler is the program code in C, the result of the compiler is the so-called object code of the program, which contains processor commands in machine language.

There are many compilers and integrated development environments:

- Borland C++
- C++Builder
- Microsoft Visual C++
- Microsoft Visual Studio
- Dev-C++
- Code::Blocks
- Embarcadero RAD Studio and others.

2 FAMILIARITY WITH THE LANGUAGE C

2.1 Alphabet of language C

The alphabet of the C/C++ language consists of:

- capital and small letters of the Latin alphabet: "A", ..., "Z", "a", ..., "z";
- digits 0, 1, ..., 9;
- special characters: " ' () [] { } < > . , ; : ? ! ~ * + - = / \ | # % \$ & ~ @ and the underscore character _.

Programs consist of syntactic constructions called commands (other names are statements). Commands are built with lexemes - indivisible elements of language: words, numbers, symbols of operations.

Words are divided into identifiers and keywords.

An identifier is a name that the user gives to objects, for example, variables, constants, functions. All words can consist of lowercase or uppercase letters of the English alphabet, numbers, and also contain an underscore. An identifier always starts with a letter or an underscore. Lowercase and uppercase letters that have the same meaning in the C language are considered different symbols.

Reserved identifiers are called keywords. They are used to write commands. It is not possible to change the assignment of the keyword in the program. The main keywords of the C language: int, double, main, break, printf, scanf, while, for, switch, struct, return, etc. (see Appendix A).

2.2 Variables and types of variables, declaration of variables in c language

From a logical point of view, all data that is processed in the program is represented by variables. From a technical point of view, variables are cells in RAM.

Variables can be assigned specific values, which means writing these values to memory cells. Variable values can be used to evaluate expressions. This results in the operation of reading values from memory cells.

Variable values can be of different types (see Appendix B). The main types are:

int – contains whole numbers;

float, double – contains real numbers (with floating point);

char – contains one character.

Any variable has a name - a correct identifier of the C language.

Each variable must be declared. A variable declaration is a statement that causes a cell to be reserved for a variable. This cell is accessed by variable name. A variable declaration also contains an indication of the type of values for that variable. For example, the following statements will declare three variables a, b, c of the above types:

```
int a;
```

```
float b;
```

```
char c;
```

2.3 Output function in C language

The *printf* function is used to display the values of variables, expressions, and text literals on the screen

Function

```
printf ("Format String", object 1, object 2, ..., object n);
```

The Format String consists of the following elements:

- control characters;
- text submitted for direct output;
- formats designed to display the values of variables of different types.

Objects may be missing.

- Control characters are not displayed on the screen, but control the location of the displayed characters. A distinctive feature of the control character is the presence of a backslash '\' in front of it.
- The text is displayed in double quotes.
- Formats are required to indicate the form in which the information will be displayed. A distinctive feature of the format is the presence of the character percentage '%' in front of it.

Basic control characters

\ *n*' - new line;

\ *t*' - horizontal tab;

\ *v*' - vertical tab;

\ *b*' - return to the character;

\ *r*' - return to the beginning of the line;

\ *a*' Is an audible signal.

Formatting commands (format specifiers) for printf ()

	Format
%c	Character of type char
%s	Character string
%d	An integer of type int with a sign in decimal notation;
%o	An integer of type int with a sign in the octal number system
%x	An integer of type int with a sign in the hexadecimal number system
%u	An integer of type unsigned int;
%f	Decimal number is a single precision float precision format
%lf	Decimal number is a double-precision floating-point format
%ld	An integer of type long int with a sign in decimal notation
%lu	An integer of type unsigned long int
%lx	An integer of type long int with a sign in hexadecimal
%e	Decimal number in the form x.x e + xx
%E	Decimal number in the form x.x E + xx
%hd	An integer of type short with a sign in decimal notation
%hu	An integer of type unsigned short
%hx	An integer of type short with a sign in hexadecimal

2.4 Input function in C language

The *scanf()* data input function reads the data input from the keyboard, converts it to an internal format, and transmits the functions. The programmer sets the rules for interpreting the input data using the format string specifications.

General form of function *scanf()*

scanf ("Format string", variable address 1, variable address 2, ...);

The string of formats is similar to the function *printf()*.

The ampersand symbol '&' is used to generate the variable address

Address = & object

A string of formats and a list of arguments for the function are required.

Arithmetic of language C

Action in C	Arithmetic operation	Algebraic expression	Expression of C
Addition	+	$a+7$	$a+7$
Subtraction	-	$p-3$	$p-3$
Multiplication	*	ap	$a*p$
Division	/	a/p	a/p
Calculation of surplus	%	$x \bmod y$	$x\%y$

- All arithmetic operations use two operators. The result of dividing two integers will also be an integer ($7/4 = 1$, $17/18 = 0$).
- The operation of calculating the remainder of the fraction can be performed only with integers.
- In C, the calculation of an arithmetic expression is performed in an order that corresponds to the rules of seniority

2.5 Assignment operations

The C language provides several assignment operations.

`c = c + 3;` or `c += 3;`

Any type operator

`variable = variable operation expression;`

can be written as

`variable operation = expression;`

2.6 Equality and relation operations

- C operators either perform certain actions (such as calculating or inputting / outputting data) or make decisions. In the program, decision-making occurs based on whether a statement is true or false (the latter is called a condition).
- Conditions are set using equality and relation operations. Relationship operations have the same priority and are performed from left to right. Equality operations have lower priority and are also performed from left to right.

Standard operation	Operation in C	Example	Explanation
=	==	x==y	x is equal to y
≠	!=	x!=y	x is not equal to y
>	>	x>y	x is greater than y
<	<	x<y	x is less than y
≥	>=	x>=y	x is greater than or equal to y
≤	<=	x<=y	x is less than or equal to y

3 PROGRAM MANAGEMENT

3.1 Management structures

- The selection structure is used to select one of the alternative courses of action.
- The if structure is called a single-choice structure because it selects or ignores a single action. In the if structure, the specified action is performed only when the condition is true; otherwise the action is skipped.

General view of the structure *if*

if (condition) {block};

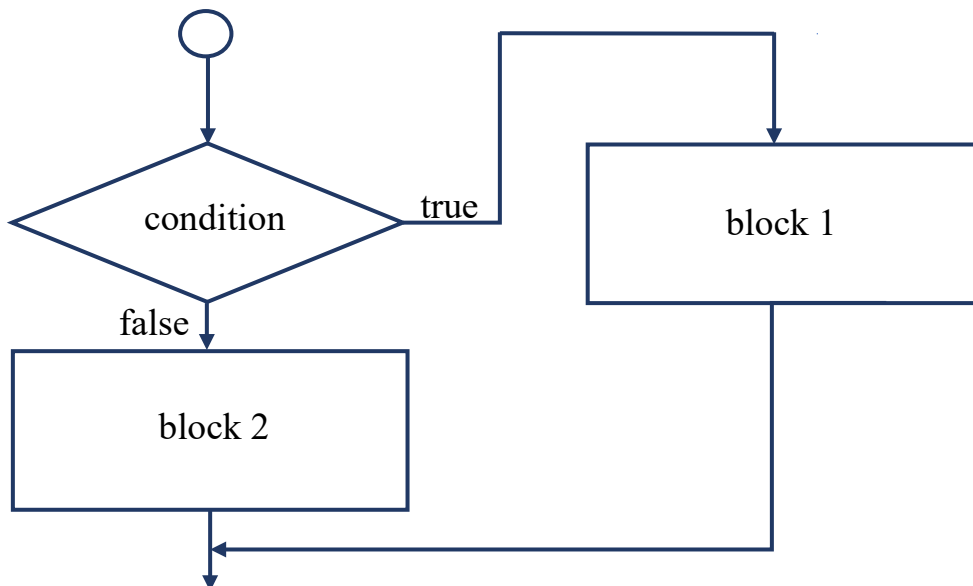
- The keyword if means condition - an expression, the result of which can have one of two values - either true (not equal to 0), or false (equal to 0);
- Block (as mentioned earlier) is one or more operators enclosed in curly braces

3.1.1 Management structure *if/else*

The if / else structure is called a double-choice structure because it chooses between two alternative actions. The if / else structure allows the programmer to specify that, depending on whether the condition is true or false, different actions must be performed.

General view of the structure *if/else*

```
if (condition) {block_1} else {block_2}
```



Example. Write a notification depending on the air temperature:

temperature less than -20 - very cold

less than -5 and more than -20 - cold

less than 0 more than -5 - cool

more than 0 - normal

Program

```
#include <stdio.h>

int main()
{
    int t;
    printf ("Input temperature\n");
    scanf ("%d",&t);
    if (t<=-20) {
        printf("Very Cold\n");
        printf("Very Cold\n");
    }
    else if (t<=-5) printf("Cold\n");
        else if (t<=0) printf("Cool\n");
            else printf("Normal\n");
    return 0;
}
```

3.1.2 Operations for increment and decrement

- The C language also provides a unary increment operation ++
- and unary decrement operation --.

<code>x=x+1;</code>	<code>x+=1;</code>	<code>++x;</code>	<code>x++;</code>
<code>y=y-1;</code>	<code>Y-=1;</code>	<code>--y;</code>	<code>y--;</code>

If increment or decrement operations are placed before a variable, they are called pre-increment and pre-decrement operations, respectively.

If increment or decrement operations are placed after a variable, they are called post-increment and post-decrement operations, respectively.

The operation of pre-increment (pre-decrement) on the variable causes an increase (decrease) of the variable by 1, and then the new value of the variable is used in the expression in which it appears.

The operation of the post-increment (post-decrement) over the variable causes the use of the variable in the expression in which it appears, and then increase (decrease) the variable by 1.

3.1.3 Logical operations

Language C provides logical operations that allow you to build complex conditions by combining simple ones. The following operations are logical operations:

&& - logical multiplication(logical “AND”);

|| logical addition (logical “OR”);

! – logical negation (logical “NOT”).

3.1.4 Logical multiplication

- If in some place of the program it is necessary to provide truth of two conditions at the same time for a choice of some branch of its performance, logical multiplication is applied.
- `if (condition1) && (condition2)) action;`
- This condition is true if both simple conditions are true.
- If at least one of these simple conditions is not true, or both simple conditions are not true, then the program ignores the output statement and goes to the statement that follows the `if`.

Expression A	Expression B	A&&B
T	T	T
T	F	F
F	T	F
F	F	F

3.1.5 Logical addition

If in some place of the program it is necessary to provide truthfulness of at least one of two conditions at the same time for a choice of some branch of its performance, logical addition is applied.

if (condition1) || (condition2)) action;

Expression A	Expression B	A B
T	T	T
T	F	T
F	T	T
F	F	F

3.1.6 Logical operation of negation

- The logical negation operation allows the programmer to "reverse" the condition.
- This operation, in contrast to the operations && i ||, is unary, ie only one condition is used as an operand.
- The logical negation operation is placed before the condition when it is necessary to select the branch of the program with a false condition

Expression A	!A
T	F
F	T

3.1.7 Structure with multiple choice switch

- Sometimes the algorithm involves a sequence of decisions, when there is an independent check of a variable or expression for equality of each of the constant values, and depending on this, different actions are performed.
- In the C language for processing of such situations the structure with multiple choice of switch is provided.
- The switch selection statement is a very convenient way to replace the multiple use of if statements.

switch (expression) operation;

where operation :

case const expression: {block}

or

default : {block }

You should also pay attention to the keyword default , it is not necessary, but at the same time it is necessary to handle unexpected situations. For example, when the value of a variable does not match one of the values of the case, then the code that is in the default is executed. This can be useful if we do not expect any of the case values to match the value of the variable in the switch conditions. In this case, the code in the default branch worked.

Example. Write a program that simulates the work of a calculator: enter two numbers and one of the arithmetic operations (+, -, *, /), the result display on the screen.

```

#include <conio.h>
#include <stdio.h>
int main()
{
int arg1, arg2;
    char oper;
    printf("First number:");
    scanf("%d",&arg1);
    printf(" Second number:");
    scanf("%d",&arg2);
    printf(" Operation:");
oper=getch();
    switch (oper) {
    case '+': printf("%d+%d=%d",arg1,arg2,arg1+arg2); break;
    case '-': printf("%d-%d=%d",arg1,arg2,arg1-arg2); break;
    case '*': printf("%d*%d=%d",arg1,arg2,arg1*arg2); break;
    case '/': printf("%d/%d=%f",arg1,arg2,(float)arg1/arg2); break;
    default:printf(" The operation symbol is entered incorrectly!"); break;
        }

    getch();
return 0;
}

```

This program variables *arg1*, *arg2* integers with which action will occur. The variable *oper* of character type (char - any one character) is intended to save the operation to be performed. The getch () function returns the character that was entered on the keyboard. This symbol is assigned to the variable *oper*.

The keyword switch is followed by the name of the variable oper in parentheses. This variable is a control expression in the structure. The value of this expression is compared to each constant expression after the case label.

If the values of the control and constant expressions match, the operators for this case are executed. The break statement causes control to be passed to the first statement after the switch structure.

If there is no match between the values of the control and constant expressions, the actions of the default block are performed. The default label may be missing. Then, in the absence of coincidence of the values of the control and constant expressions, any operator of the switch structure is not execute.

3.2 Repetition structures

Most programs include the repetitions or the loops.

The loop is a group of commands that are repeatedly executed by a computer as long as some condition of the continuation remains true.

The operators that are included in the repetition structure make up the body of this structure. The body of the repetition structure can be simple (single operator) or complex operator (block).

In language C there are 3 repetition structures:

- Repetition structures *for*
- Repetition structures *while*
- Repetition structures *do/while*

3.2.1 Repetition structure for

Such repetitions are sometimes called definite repetitions, because it is known in advance how many times the loop will be performed. The control variable is used to count the number of repetitions.

The control variable changes each time (usually increased by 1) when the loop body is executed.

When the value of the control variable indicates that the required number of repetitions is performed, the loop ends, the computer continues to execute the program from the operator that is following the structure of repetition.

General format of the structure *for*

for (expression1; expression2; expression3)

{ body of the loop }

where expression1 initiates the loop control variable,

expression2 is a condition for continuing the loop,

expression3 indicates how the loop control variable changes.

To implement the repetition structure controlled by the counter, the language C provides a structure **for**. This structure automatically controls all the details of such repetition

Example. Print number from 1 to 10

```
#include <stdio.h>

int main()
{   int counter ;
    for (counter=1; counter<=10; counter++)
        printf("%d ",counter);
    return 0;
}
```

3.2.2 The repetition structure **while**

- The repetition structure of **while** or repetition, controlled by the control value
- Such repetitions are called indeterminate repetitions, because it is not known in advance how many times you need to perform the loop. Control values are used to control the number of repetitions.
- To implement the repetition structure controlled by the control value, the language C uses the structures **while**, **do \ while**.

The format of the repetition structure **while**

while (condition) {block}

If the expression has a value that is not equal to zero (true), the operators of the body loop (block) are executed.

After that, the program returns to the calculation of the expression.

As soon as the value of the expression matches 0 (false), the loop stops, and the program starts working from the first statement after the while structure.

Example. Finding the first degree 2, which more than 1000

```
#include <stdio.h>

int main()
{   int product=2;
    while (product <= 1000)
        product = 2 * product;
    printf("Rezult=%d",product);
}
```

3.2.3 The repetition structure *do/while*

The **do/while** repetition structure is similar to the **while** structure. In the **while** structure, the loop continuation condition is checked at the beginning of the loop before executing the loop body operators.

The **do/while** structure checks the condition of the continuation of the loop after the execution of the loop body, and therefore the loop body will be executed at least once.

After completing the **do/while** loop, program execution continues from the statement that is written after the while sentence.

The format of the repetition structure *do/while*

do {block} while (expression)

where

block—sequence of actions(the body loop)

expression - condition for finish the loop

Example. Print number from 1 to 10

```
#include <stdio.h>

int main()
{
    int counter = 1;
    do {
        printf("%d ",counter);
        } while (++counter <= 10);
    return 0;
}
```

3.3 Command break and continue

- The **break** and **continue** statements are designed to change the progress of actions in the program.
- Execution of the break statement in the structures **switch, for, while, do / while** will lead to immediate exit from the structure.
- Execution of the program continues from the first statement, which is recorded after it.

Example

```
#include <stdio.h>

int main()
{
    int x;
    for (x = 1; x <= 10; x++) {
        if (x == 5)
            break;
        printf("%d ",x);
    }
    printf("Break on x == %d",x);
    return 0;
}
```


In the example, the **break** operator is used in the for structure. When the value of the variable x becomes equal to 5, the **break** statement is executed. This completes the loop, and the program continues to run from the printf statement. The loop is performed completely only four times.

Executing the **continue** statement in the **for**, **while**, **do / while** structures will skip the remaining statements in the body of these structures and execute the next iteration of the loop.

Example

```
#include <stdio.h>

int main()
{
    int x;
    for (x = 1; x <= 10; x++) {
        if (x == 5)
            continue;
        printf("%d ",x);
    }
    return 0;
}
```

In this example, the **continue** operator is used in the for structure. When the value of the variable x becomes equal to 5, the operator **continue**. This allows you to skip the printf statement and the program continues to perform the next iteration of the loop.

4 ARRAY

An array is a group of memory cells that have the same name and type.

To use a specific cell or array element, specify the name of the array and offset that cell relative to the first cell or the beginning of the array.

The offset is specified after the array name in square brackets and is called the array index.

For example, suppose that an array C has 12 elements of an integer type.

The first element in any array has an index of 0.

Thus, the first element is denoted by C [0].

The second element has an offset relative to the beginning of the array by one integer.

Thus, the second element is denoted by C [1].

In the general case, the i-th element of the array is denoted by C [i-1].

Arrays take up space in RAM.

To declare an array C or reserve 12 elements for an array of integers must be written

```
int C [12];
```

With one ad, you can reserve memory for multiple arrays. For example, the following example reserves memory for 100 elements of an array of integers **B** and 27 elements of an array of integers **X**:

```
int B[100], X[27];
```

Arrays can also be declared for other data types: int, char, float, double...

For example, a character string is an array of characters, that is, an array of type char.

Like variables of other types, array elements can be initiated by values at the time they are declared. To do this, after declaring the array put an equal sign "=" and in curved brackets write a list of values for all elements of the array. The values in the list are separated by commas:

```
int A123A [12] = {32, 27, 64, 18, 96, 12, 45, 23, 50, 9, 2, 0}.
```

```
A123A[0]=32; A123A[1]=27;...A123A[11]=0;
```

If the values in the list are less than the elements of the array, the remaining elements are automatically initiated by 0. However, specifying more values in the initialization list than the number of elements of the array will result in a syntax error.

If array elements are initiated at the time the array is declared, you can not specify the size of the array. Then the size of the array will be equal to the number of values in the initialization list. For example,

```
int n [] = {1, 2, 3, 4, 5};
```

an array of five elements will be created.

Character arrays have several unique features. A character array can be initiated by a string literal. For example,

```
char NAME1 [] = "first";
```

initiates the elements of the array **NAME1** by individual characters of the string literal "first".

The size of the array **NAME1** is equal to the length of the string "first". The line "first", in turn, consists of the corresponding letters and a special character at the end of the line - "\0". All character strings in C end in this character.

Thus, the compiler calculates the number of elements in the array **NAME1** as equal to 6.

The previous initiated is equivalent to the following:

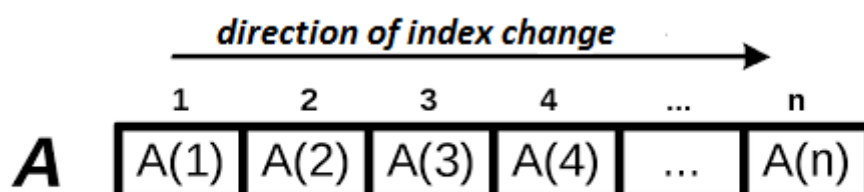
```
char NAME1 [] = {'f', 'i', 'r', 's', 't', '\0'};
```

A character array representing a string can be output using the printf function and the %s conversion specifier:

```
printf ("%s \n", NAME1);
```

Line characters are displayed until the end of line '\0' is detected.

GENERAL APPEARANCE OF THE ARRAY



4.1 Array in programming (Language C)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
------	------	------	------	------	------	------	------	------	------

One-dimensional array declaration

```
int Array[10];
```

```
float B[200];
```

```
char T[15];
```

Example. Input / output of one-dimensional array

```
#include <stdio.h>

int main ()
{
    int A[100];
    /* Array A consists of 100 integers numbers */
    int i,N;
    printf ("Input quantity elements of the array\n:");
    scanf ("%d",&N);
    printf (" Input array n:");
    for (i=0; i <= N-1; i++)
        scanf ("%d",&A[i]);
    printf (" The rezult array :");
    for (i=0; i <= N-1; i++)
        printf ("%3d",A[i]);
    printf ("\n");
    return 0;
}
```

4.2 Array inversion

Example 1. 1 way- inversion with additional array

```
#include <stdio.h>

int main ()
{
    int A[100],B[100];
    int i,N;
    printf ("Input quantity elements of the array \n:");
    scanf("%d",&N);
    printf (" Input array n:");
    for (i=0; i <= N-1; i++)
        scanf("%d",&A[i]);
    for (i=0; i <= N-1; i++)
        B[i]=A[N-1-i];
    for (i=0; i <= N-1; i++)
        A[i]=B[i];
    printf ("The rezult array\n");
    for (i=0; i <= N-1; i++)
        printf("%3d",A[i]);
    printf("\n");
    return 0;
}
```

Example 2. 2 way - inversion with additional variable

```
#include <stdio.h>

int main ()
{
    int A[100];
    int i,N,B;
    printf ("Input quantity elements of the array \n:");
    scanf("%d",&N);
    printf (" Input array n:");
    for (i=0; i <= N-1; i++)
        scanf("%d",&A[i]);
    for (i=0; i <= (N-1)/2; i++)
    {
        B=A[i];
        A[i]=A[N-1-i];
        A[N-1-i]=B;}
    printf ("The rezult array\n");
    for (i=0; i <= N-1; i++)
        printf("%3d",A[i]);
    printf("\n");
    return 0;
}
```

Example 3. 3 way- output inversion with indexes of array

```
#include <stdio.h>

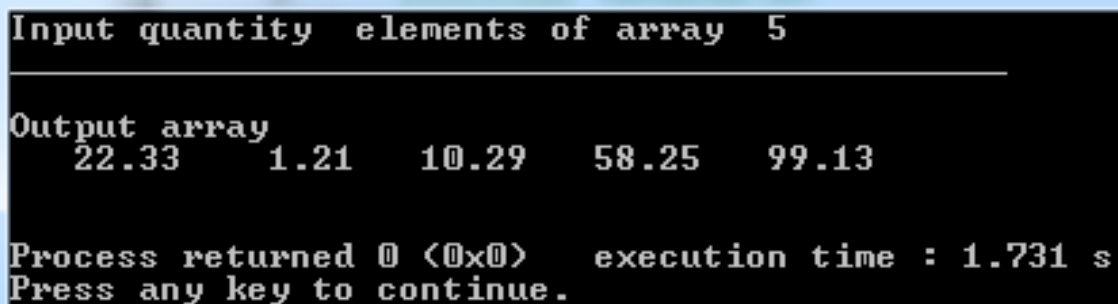
int main ()
{
    int A[100],B[100];
    int i,N;
    printf ("Input quantity elements of the array \n:");
    scanf("%d",&N);
    printf (" Input array n:");
    for (i=0; i <= N-1; i++)
        scanf("%d",&A[i]);
    printf ("The rezult array\n");
    for (i=N-1; i >= 0; i--)
        printf("%3d",A[i]);
    printf("\n");
    return 0;
}
```

4.3 Actions with array elements

Example 1. Filling the array with random numbers

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    int i,n;
    float x;
    float A[10];
    printf("Input quantity elements of array ");
    scanf("%d",&n);
    srand(time(NULL));
    for (i=0;i<n;i++)
    {
        x=(float)(rand()%9)/24; // fractional part of a number
        A[i]= rand()%100+x; // random integers from 0 to 99 + fractional part x
    }
    printf("_____ \n");
    for (i=0;i<n;i++)
    printf("  %5.2f",A[i]);
    return 0;
}
```



```
Input quantity elements of array 5
_____
Output array
  22.33   1.21  10.29  58.25  99.13

Process returned 0 (0x0)   execution time : 1.731 s
Press any key to continue.
_
```



```
Input quantity elements of array 4

Output array
69.08 11.13 57.17 38.33

Process returned 0 (0x0) execution time : 2.036 s
Press any key to continue.
-
```

Example 2. Search for the minimum element in the array

```
#include <stdio.h>

int main ()
{
    int Arr[100];
    int i,N,min;
    printf("Input quantity elements of array ");
    scanf("%d",&N);
    printf (" Input array by %d: elements\n",N);
    for (i=0; i<N;i++) scanf("%d",&Arr[i]);
    printf ("array :\n");
    for (i=0; i<N;i++) printf("%4d",Arr[i]);
    // min=-200;
    //min=1000;
    min=Arr[0];
    for (i=1; i <N; i++)
    if (min>Arr[i]) min=Arr[i];
    printf("\nMin=%d",min);
    return 0;
}
```

Example 3. Search for the maximum element in the array

```
#include <stdio.h>

int main ()
{
    int Arr[100];
    int i,N,max;
    printf("Input quantity  elements of array  ");
    scanf("%d",&N);
    printf (" Input array by %d elements\n",N);
    for (i=0; i<N;i++)  scanf("%d",&Arr[i]);
    printf ("array:\n");
    for (i=0; i<N;i++) printf("%4d",Arr[i]);
    // max=-200;
    //max=1000;
    max=Arr[0];
    for (i=1; i <N; i++)
    if (max<Arr[i]) max=Arr[i];
    printf("\nMax=%d",max);
    return 0;
}
```

4.4 Multidimensional arrays

Arrays in C can have several indexes.

Multidimensional arrays are used, for example, to represent tables consisting of values arranged in rows and columns. Arrays that require two indexes are two-dimensional arrays; three indices - three-dimensional, etc. The ANSI standard states that the C language must support at least 12 indexes.

Multidimensional array declaration

```
int A[3][4];
```

```
float B[3][7][2];
```

```
double QQQ[2][2];
```

```
char WW[4][3];
```

Example. Two-dimensional array `int A[3][4]`

	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

Multidimensional arrays, like one-dimensional ones, can be initiated by their declaration. For example, the declaration of a two-dimensional array *a* as follows

```
int a[3][4] = {{1, 2, 7, 6}, {3, 4, 8, 9}, {5, 6, 9, 10}};
```

equivalent to an declaration

```
int a[3][4];
```

and twelve assignment operations

```
a[0][0] = 1; a[0][1] = 2; a[0][2] = 7; a[0][3] = 6;
```

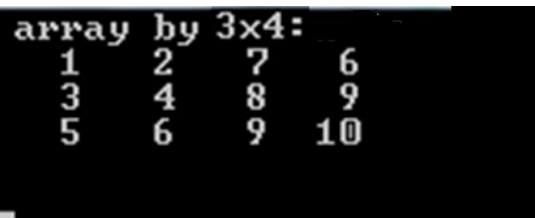
```
a[1][0] = 3; a[1][1] = 4; a[1][2] = 8; a[1][3] = 9;
```

```
a[2][0] = 5; a[2][1] = 6; a[2][2] = 9; a[2][3] = 10;
```

Example

```
#include <stdio.h>

int main ()
{
    int a[3][4] = {{1, 2, 7, 6}, {3, 4, 8, 9}, {5, 6, 9, 10}};
    int i, j;
    printf (" array by 3x4:\n");
    for (i=0; i <3; i++)
    {
        for (j=0; j<4; j++)
            printf ("%4d", a[i][j]);
        printf("\n");
    }
    printf("\n");
    return 0;
}
```



```
array by 3x4:
 1  2  7  6
 3  4  8  9
 5  6  9 10
```

4.5 Input / output of multidimensional arrays

Example program that allows the user to enter the dimension of the array (quantity of rows and columns), and then fill it and display

```
#include <stdio.h>

int main ()
{
    int Arr[10][10];
    int i,j,N,M;

    printf ("\nInput quantity of row of the array:");
    scanf("%d",&N);

    printf ("\nInput quantity of column of the array:");
    scanf("%d",&M);

    printf (" Input array by dimension %dx%d:\n",N,M);
    for (i=0; i <N; i++)
        for (j=0; j<M;j++)
            scanf("%d",&Arr[i][j]);

    printf ("array by  %dx%d:\n",N,M);
    for (i=0; i <N; i++)
    {
        for (j=0; j<M;j++)
            printf ("%4d",Arr[i][j]);
        printf("\n");
    }
    printf("\n");
    return 0;
}
```

```
Input quantity of row of the array:4
Input quantity of column of the array:2
Input array by 4x2:
1
2
3
4
5
6
7
8
Output array by 4x2:
1 2
3 4
5 6
7 8

Process returned 0 (0x0)   execution time : 16.211 s
Press any key to continue.
_
```

FUNCTIONS

The best way to develop a program and support large programs is to design the program in the form of small separate parts - modules.

The modules of the C language are called functions. C programs are developed, as a rule, by combining new functions that are developed by the programmer with the functions that are included in the system library of the C language.

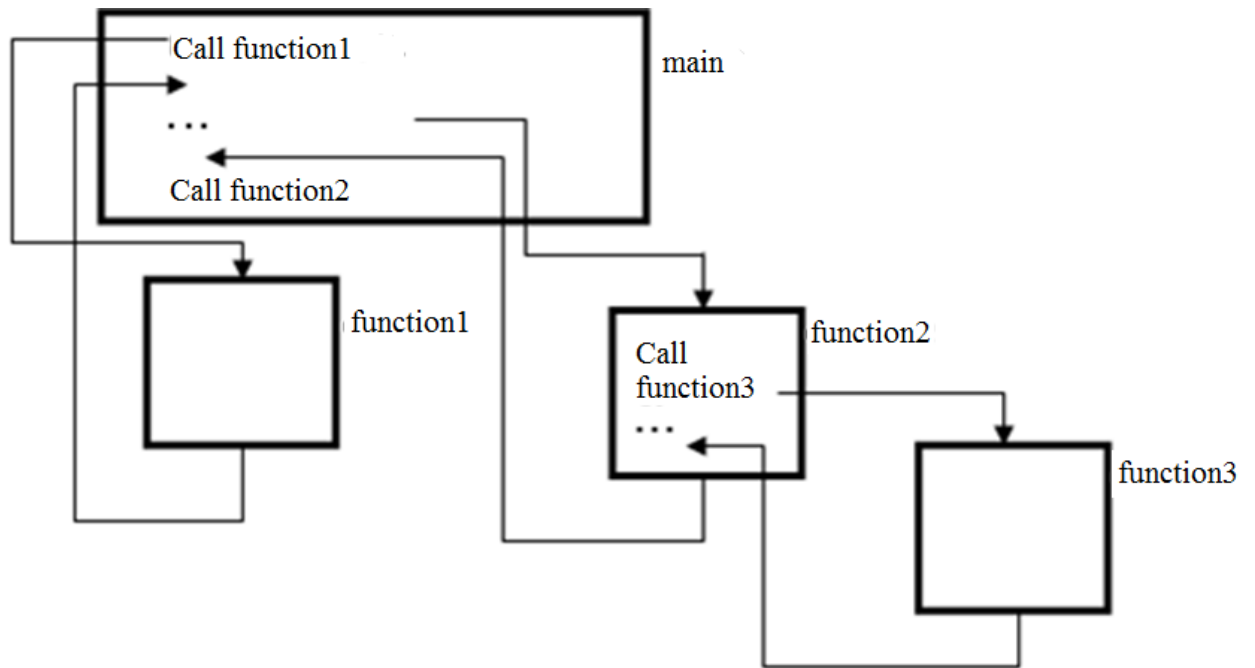
The standard C language library provides a large set of functions for performing general mathematical calculations, character string processing, I / O, and more. Standard functions simplify the work of the programmer.

The programmer can write the functions, that is to allocate in separate modules some tasks to which performance it is possible to address from many points of the program. These functions are often called programmer-defined functions.

Calling a function is called a function call. The function call specifies its name and passes the information (as arguments) needed to perform the function.

After executing the function, the program returns to the place where the function was called. A function call can be written to either the main function or any other function.

Example of scheme of interaction of functions in the program



Function allows to separate the program into modules.

All variables that are declared in the body of a function are local variables - they are known only to the function in which they are defined.

Most functions have a list of parameters. Parameters allow functions to exchange information. Function parameters are also local variables.

5.1 Function definition

```

function_type_return  function_name (parameter_list)
{
  Local definition
  Operators
}
  
```

The **function_name** can be any valid identifier.

The **function_type_return** is the type of returned value.

If the void keyword is specified as the type, it means that the function returns nothing.

If the type of returned value is not specified, the compiler assumes that the type has value int.

Parameter_list is a list of parameter declarations (comma-separated) that a function receives when it is called.

If the function does not get values, the `parameter_list` is denoted by the `void` keyword.

The type of each parameter must be described, except for the `int` type. If no type is specified, the parameter is considered to be of type `int`.

Local declaration and **operators** in the middle of curly braces make up the body of the function.

Before the first call, the function must be defined as above or using a prototype.

The compiler uses a function prototype to verify that the function call has the correct return type, the correct number of arguments, the correct argument type and the correct argument sequence.

Examples of function declarations

```
int Fun1 (int a, int b, int c);  
char Fun2 (int x, int y);  
void Fun3 (char ch, int num, int line);  
float Fun4 (float q, float t, float r, int k) ;  
int Fun5 (void);  
void Fun6 (void);  
float Fun7 (float q, double t, float r, int k) ;
```

There are ways to return control to the point in the program where the function was called.

If the function does not return a result, control returns as soon as the right curly brace occurs, terminating the body of the function,
or when performing the operator `return`.

If the function returns a result, the statement
`return expression; // return (expression);`
returns the value of the expression.

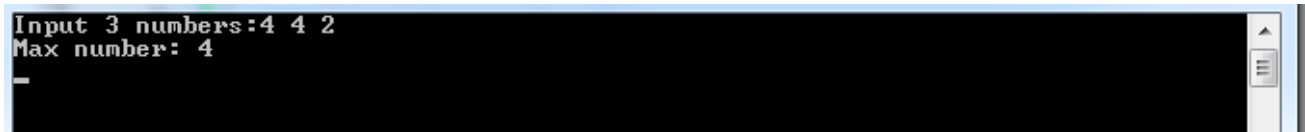
Example. Find the maximum of the three numbers and return this value.

```
#include <stdio.h>

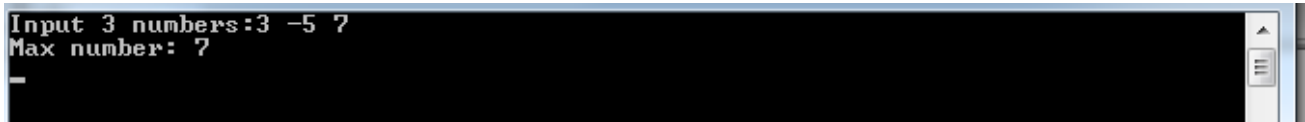
int maximum(int x, int y, int z);

main ()
{
    int a, b, c;
    printf ("Input 3 numbers:");
    scanf ("%d %d %d", &a, &b, &c);
    printf ("Max number: %d\n", maximum(a, b, c));
    return 0;
}

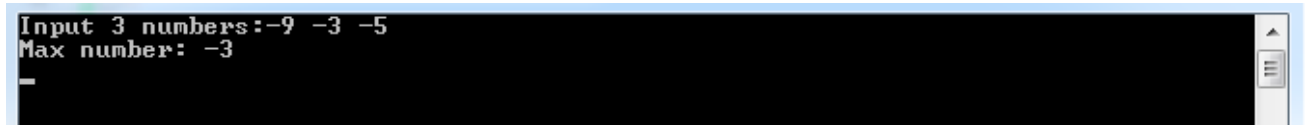
/*Definition of function maximum*/
int maximum(int x, int y, int z)
{
    int max=x;
    if (y > max)
        max=y;
    if (z > max)
        max=z;
    return max;
}
```



```
Input 3 numbers:4 4 2
Max number: 4
-
```



```
Input 3 numbers:3 -5 7
Max number: 7
-
```



```
Input 3 numbers:-9 -3 -5
Max number: -3
-
```

5.2 Prototype Of Function

Earlier it was stated that each function in the C language should have its own prototype. The function prototype informs the compiler about the type of data that returns the function; the number of parameters that the function receives; Type and order of parameters. The compiler uses the prototype to check the correctness of the function.

The prototype of the maximum function that is given above is the line

```
int maximum (int x, int y, int z);
```

This prototype reveals that maximum receives three whole -type parameters (int) and returns the result of the type int. The call that does not correspond to its prototype will cause a syntactic error. The error also occurs if the function prototype does not correspond to the function itself.

Another important consequence of the use of prototypes of functions is the automatic translation of arguments, that is, the forced transformation of function arguments to the appropriate type. In other words, the values of arguments that do not meet exactly the types of parameters in the prototype of the function are converted into the appropriate type before call. Automatic type of type translation is also used in expressions that are formed from the values of two or more types. The main content of the rule of such transformation - the lower type will be transferred to the higher.

5.3 Header files

As noted earlier, each function must have its own prototype. Prototypes of a certain number of functions can be combined into a separate file called the title file. The name of such a file is a permissible ID, and the extension is the letter "h".

Each standard library has its own header file, which contains prototypes for all functions of this library, as well as identifying data and constants required for these functions. Standard Language Libraries Files are recorded in corner brackets <,> and are included in the program #include directive:

```
#include <stdio.h - contains prototypes of standard input/output library;
```

`#include <conio.h>` - contains prototypes of standard library of console input/withdrawal;

`#include <math.h>` - contains prototypes of functions of the mathematical library;

`#include <stdlib.h>` - contains prototypes of functions to convert numbers into text and text into numbers, prototypes of memory placement functions, generation of random numbers, etc.;

`#include <string.h>` - contains prototypes of text processing;

`#include <time.h>` - contains prototypes of standard date and time management library.

The programmer can create a header file and place prototypes of developed functions. Such a file is also included in the `#include` directive, but writes in characters. " For example, the `Maximum.H` header file can be included in the program by directive:

`#include "maximum.h"`

at the beginning of the program.

The difference in the headline of the system library header and the headline file designated by the headline programmer is where the pre -processor will look for these files. If the title file name is recorded in double paws, the pre -processor believes that such a file is placed in the same directory as the file with the source text of the program. If the file name is recorded in corner brackets, the search files will be kept in special directory, which are determined depending on the specific implementation of the compiler.

5.4 Function call

In the C language, as in many other programming languages, there are two ways of calling a function - a call by value and a link to the link.

When the argument is used in the call by value, a copy of the argument value is transmitted to the function. Changes that occur with a copy do not affect the value of the variable in the function that transmits this copy. The transmission of the argument by value should be used when the calling function does not need to be changed. In the above example in the call of the function `maximum` (`a`, `b`, `c`) arguments are transmitted

by value, that is, three numbers are transmitted to the input of the function, which are assigned to variable x, y, z, respectively.

If the body of the maximum function would somehow change the initial values of x, y, z, this would not affect the variable A, B, C.

When the argument of the function is transmitted by the link, the address of this argument is transmitted to the function being called. This, of course, leads to the fact that any changes in the argument in the caused function occur and with a corresponding variable in the function that causes. This allows you to return the function more than one result (**return** operator), but as much as you like. For example, the **scanf** system function is transmitted to a reference (address) to a variable that will be assigned to the keyboard. The assignment of the value of variables is the result of function.

INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

The integrated development environment (IDE) Code :: Blocks can be used to perform home tasks, laboratory work. Code :: Blocks it is a free software (download from <http://www.codeblocks.org/>)

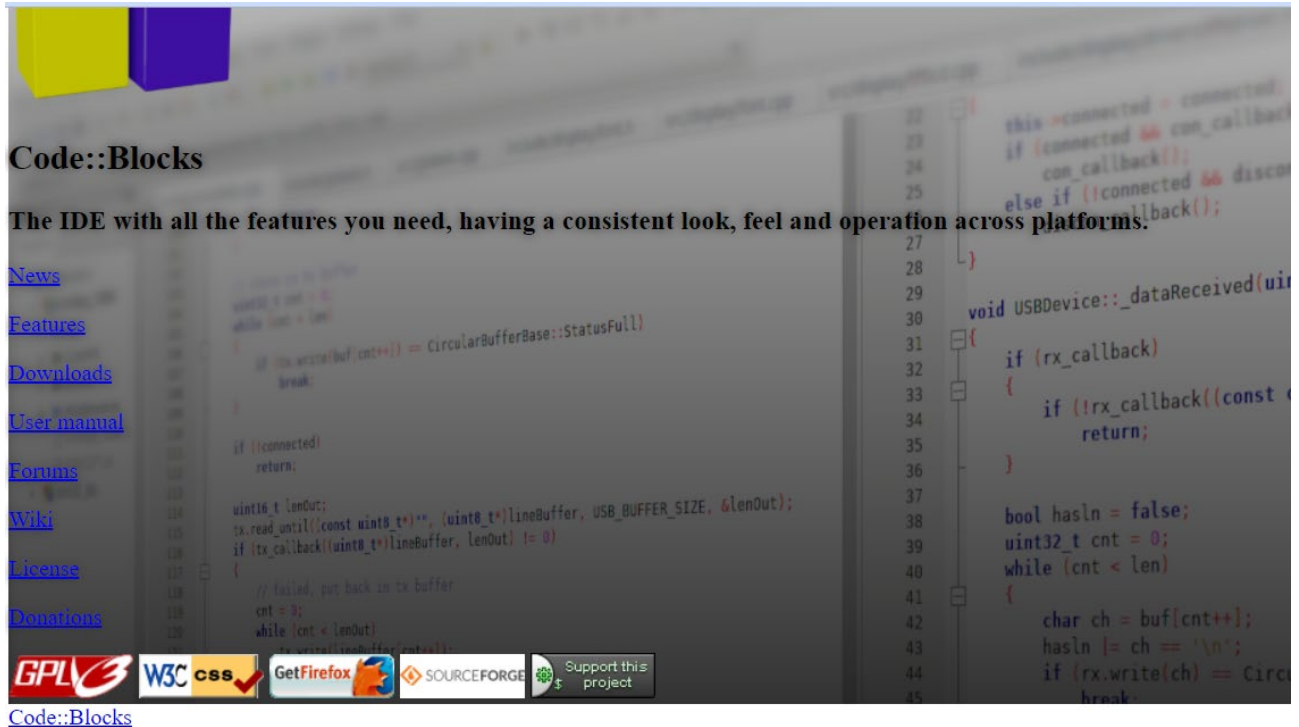


Figure 1. <http://www.codeblocks.org/>

You must to select downloads -> binary release (fig.2)-> operation system

Binary releases

Please select a setup package depending on your platform:

- [Windows XP / Vista / 7 / 8.x / 10](#)
- [Linux 32 and 64-bit](#)
- [Mac OS X](#)

NOTE: For older OS'es use older releases. There are releases for many OS version and platforms on the Sourceforge.net page.

NOTE: There are also more recent nightly builds available in the [forums](#) or (for Ubuntu users) in the [Ubuntu PPA repository](#). Please note that we consider nightly builds to be stable, usually.

NOTE: We have a [Changelog for 20.03](#), that gives you an overview over the enhancements and fixes we have put in the new release.

NOTE: The default builds are 64 bit (starting with release 20.03). We also provide 32bit builds for convenience.

Figure 2. Select operation system

Select program with “*mingw*”(fig.3)

 **Microsoft Windows**

File	Download from
codeblocks-20.03-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-nosetup.zip	FossHUB or Sourceforge.net

NOTE: The codeblocks-20.03-setup.exe file includes Code::Blocks with all plugins. The codeblocks-20.03-setup-nonadmin.exe file is provided for convenience to users that do not have administrator rights on their machine(s).

NOTE: The codeblocks-20.03mingw-setup.exe file includes additionally the GCC/G++/GFortran compiler and GDB debugger from [MinGW-W64 project](#) (version 8.1.0, 32/64 bit, SEH).

NOTE: The codeblocks-20.03(mingw)-nosetup.zip files are provided for convenience to users that are allergic against installers. However, it will not allow to select plugins / features to install (it includes everything) and not create any menu shortcuts. For the “installation” you are on your own.

If unsure, please use codeblocks-20.03mingw-setup.exe!

Figure 3. Select type of Code::Blocks

The first launch of Code::Blocks IDE

After running the Code::Blocks IDE, a splash screen will appear showing its version as it loads.

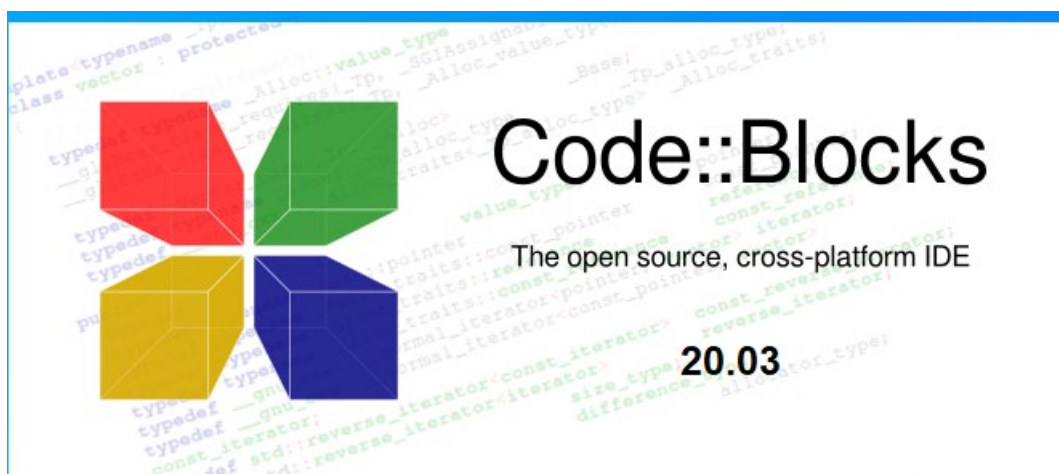


Figure 4. Code::Blocks screensaver

After that, a dialog box will appear with a list of found compilers; from it you need to choose the one that will be used by default. To perform practical work, you need to select the default GNU GCC Compiler and click OK.

In fig. 5 shows the dialog box for install folder.

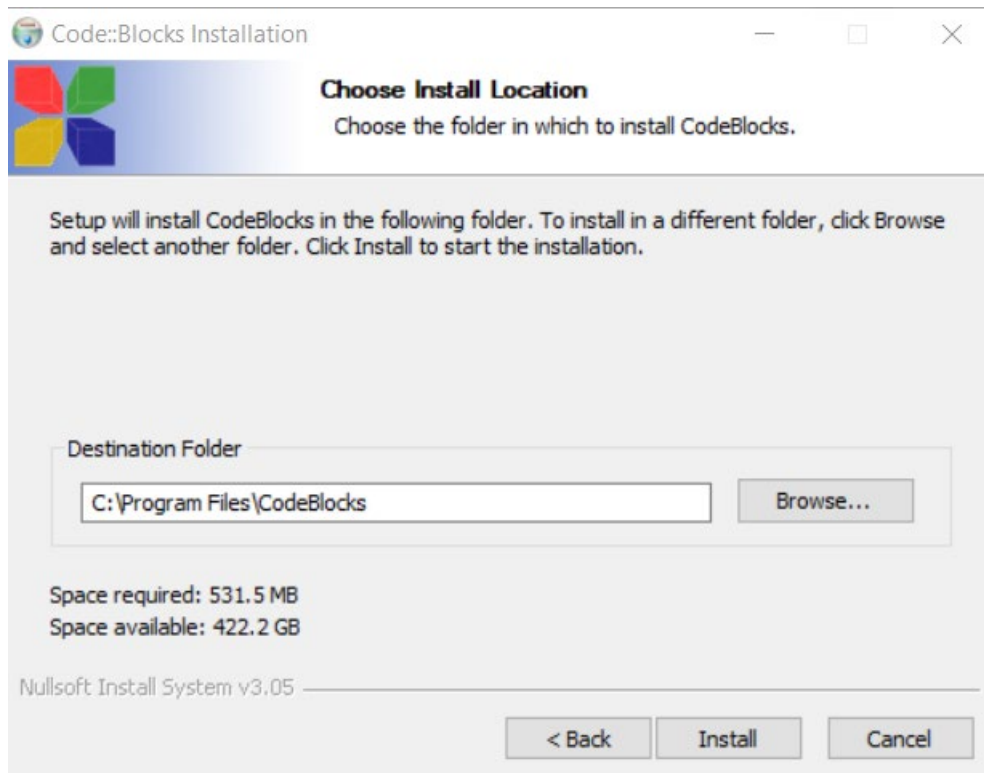


Figure. 5. Dialog box

After installing, you will be transferred to the main window of the program (see Fig. 6).

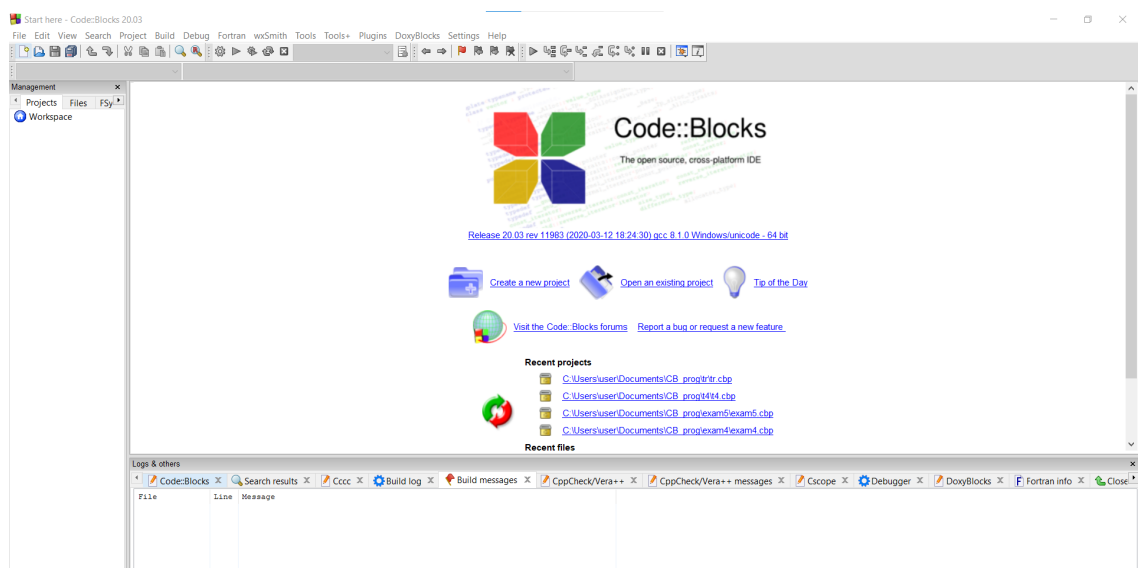


Figure 6. The main Code::Blocks IDE window

Creating a project in Code::Blocks

Create a new project starts the project creation wizard.

Open an existing project opens the Open file dialog box, in which you can open a file of an existing project. The Open file dialog box can be called with the Ctrl+O keys.

File->New->Project... menu item – creation of a new project (or the Create a new project item in the "Quick Start" window) (fig.7).

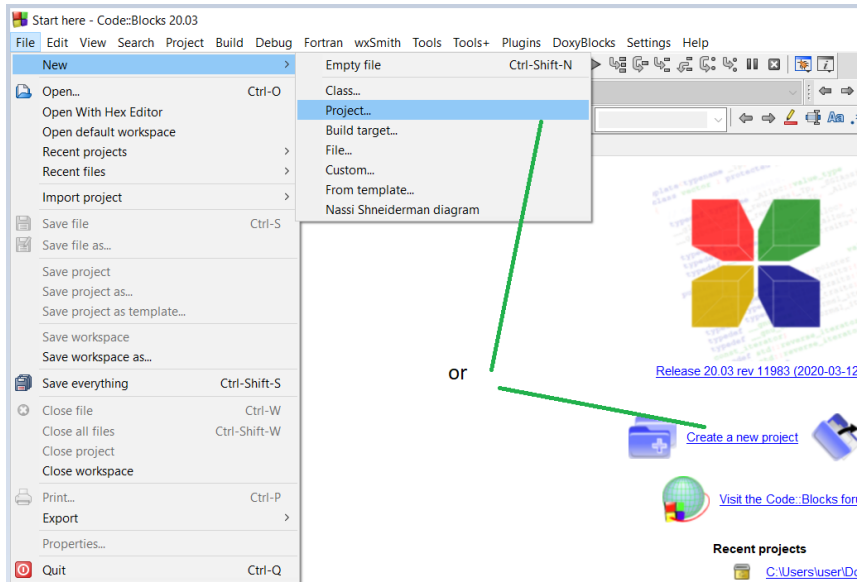


Figure 7. Create new project

In the New from template dialog box that appears, you need to specify the project template. To perform laboratory work, you can select Console application (see Fig. 8), press GO, after which the wizard for creating console applications will appear.

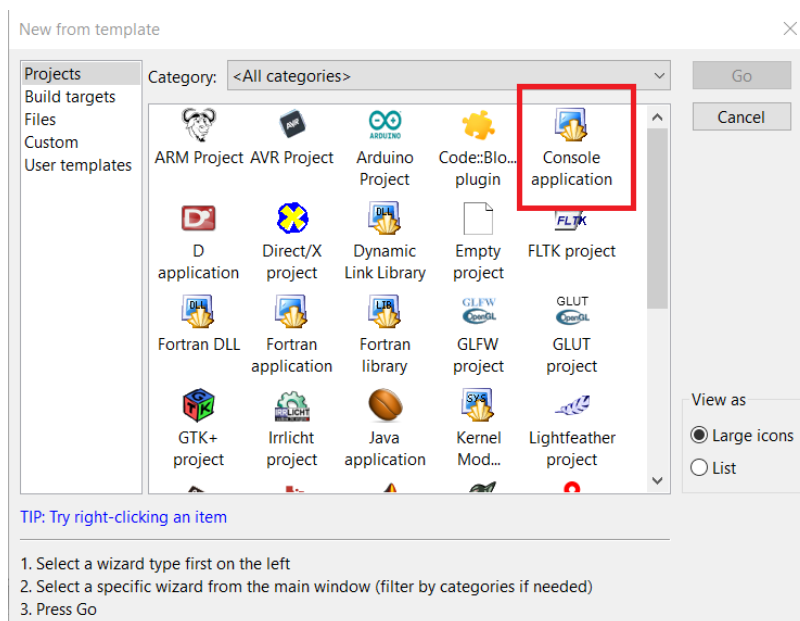


Figure 8. New from template project template selection dialog box

Click on “Concole application” and press “Go”

If you check the Skip this page next time check box in the dialog box of the wizard (see Fig. 9), this page of the wizard will be skipped when creating a new project.

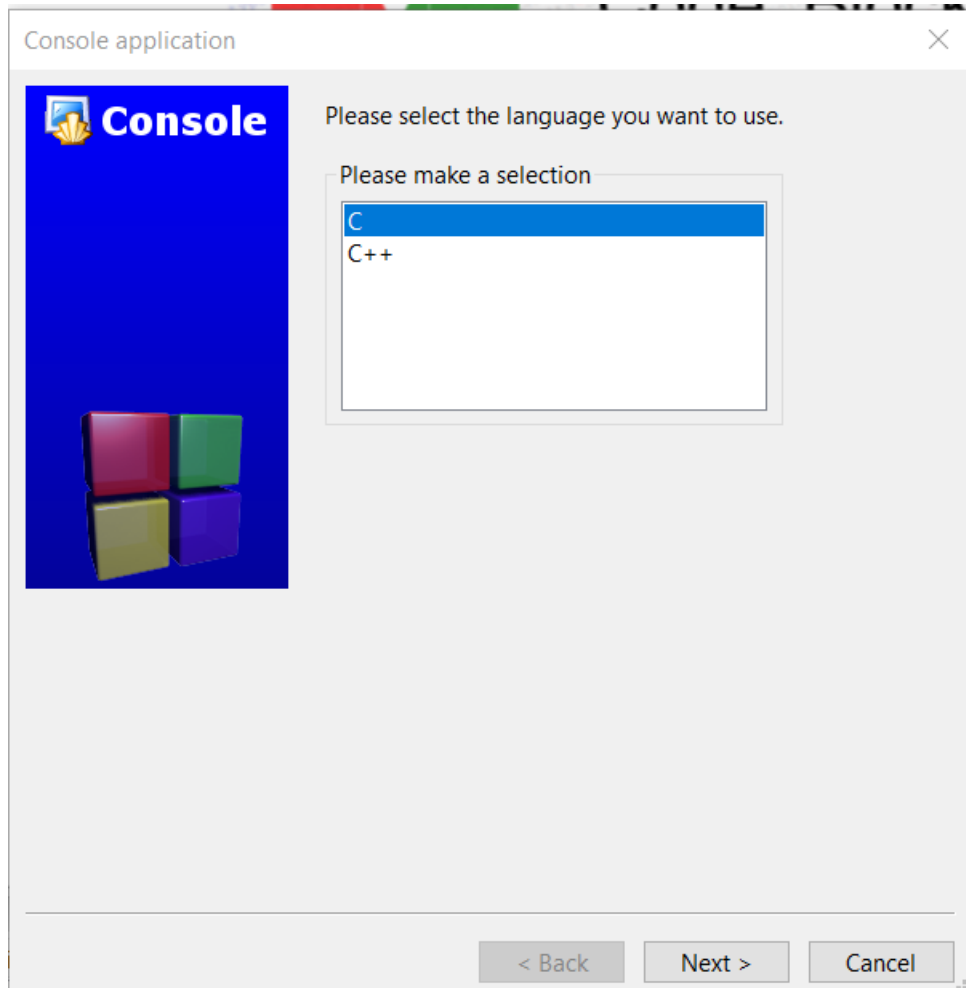


Figure 9. Dialog window of the wizard

Choose the C language and press “Next”

In the Project title field (Fig. 10) you need to specify the name of the project, in the Folder to create project in field - the folder where it will be saved.

Optional fields Project filename - specify the name of the project file, Resulting filename - specify the final name of the directory and project.

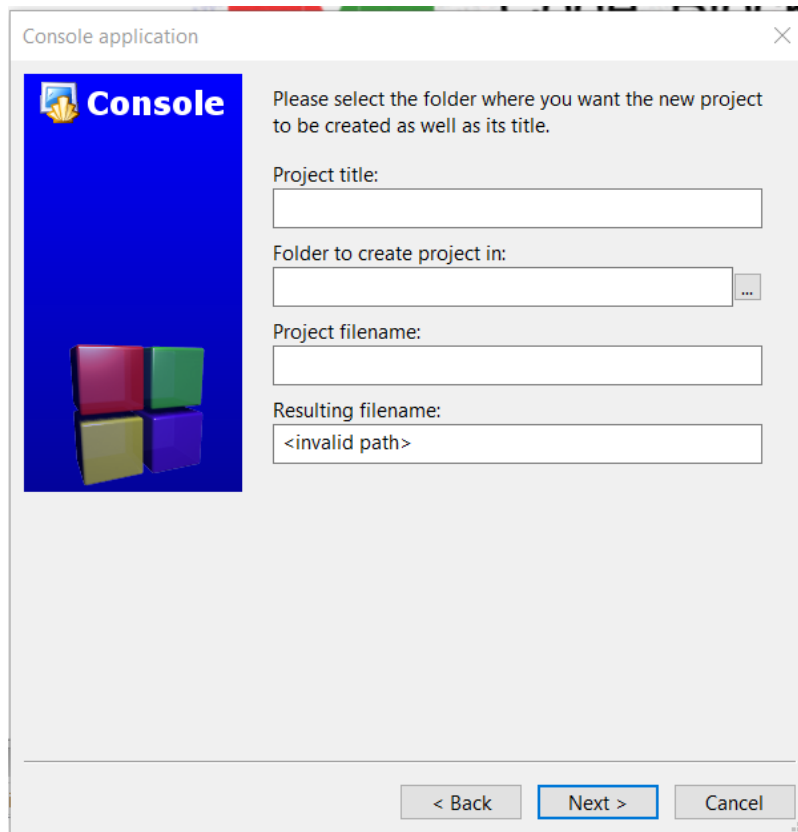


Fig. 10. Wizard page for specifying the name of the project and the folder where it will be saved

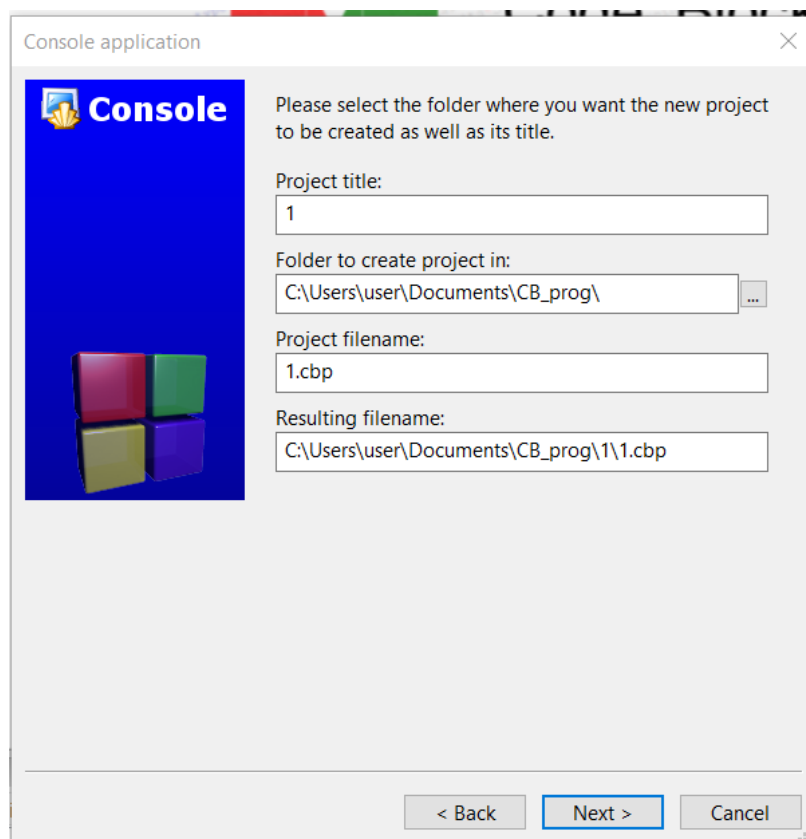


Figure 10a. Example the page with the name of the project and the folder where it will be saved

On the page (see Fig. 11), you need to choose the compiler that will be used when compiling the application.

Build scripts help you get multiple versions of the same application:

- Debug – compilation script, used when debugging the application;
- Release – script for compiling the finished application.

If there is no need to create a certain script, then you need to remove the corresponding check mark.

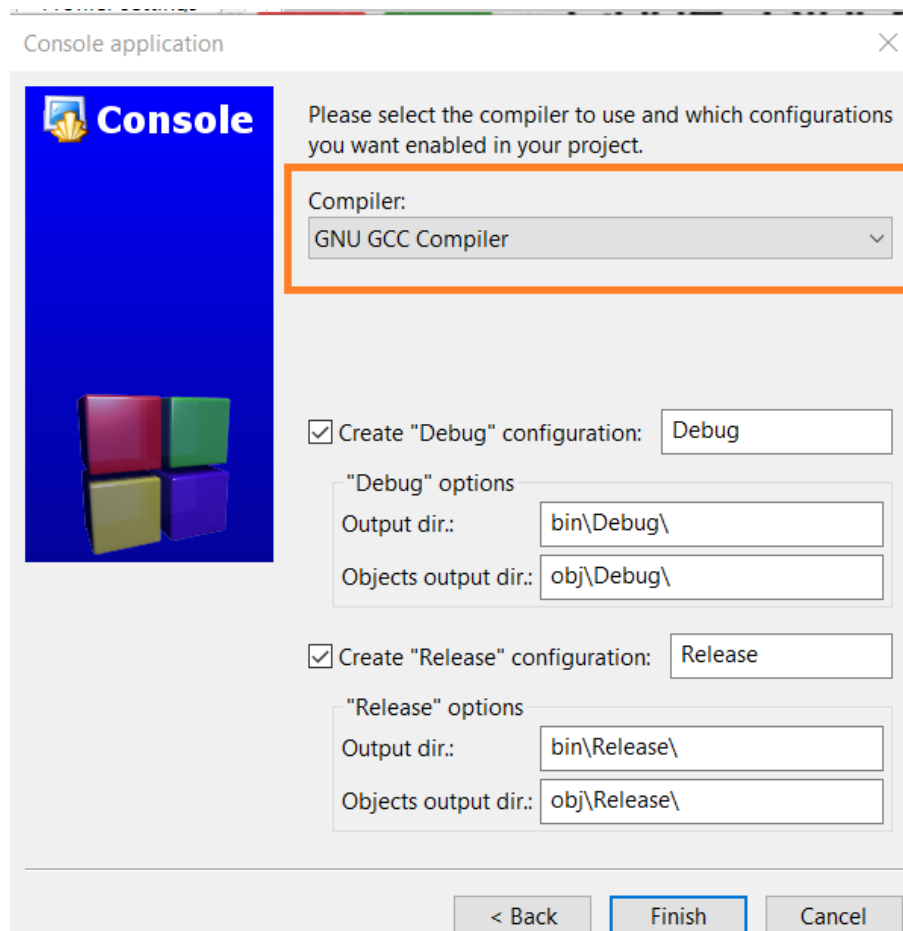


Figure 11. Wizard page for selecting a compiler

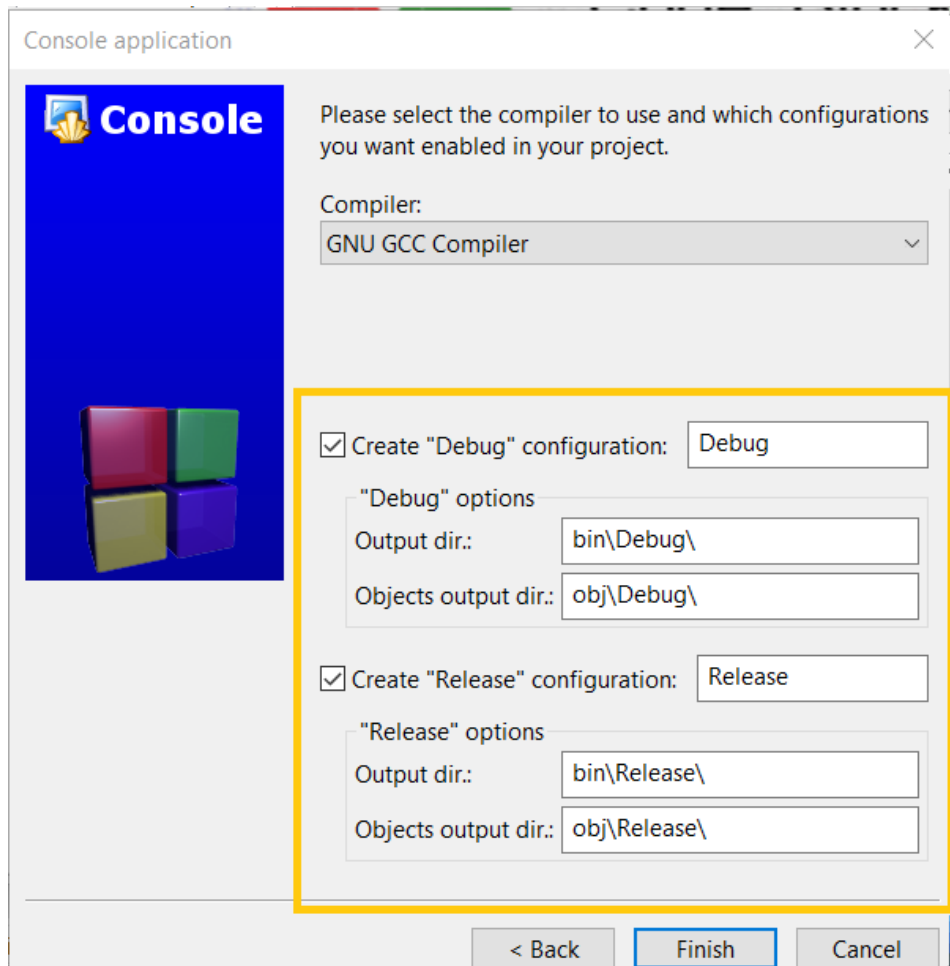


Figure 11a. Wizard page for selecting build script

In each scenario, it is possible to specify the directories where the files of the compiled application will be placed:

- Output dir – for all files;
- Objects output dir - for object files.

After clicking Finish, the project will be created and opened.

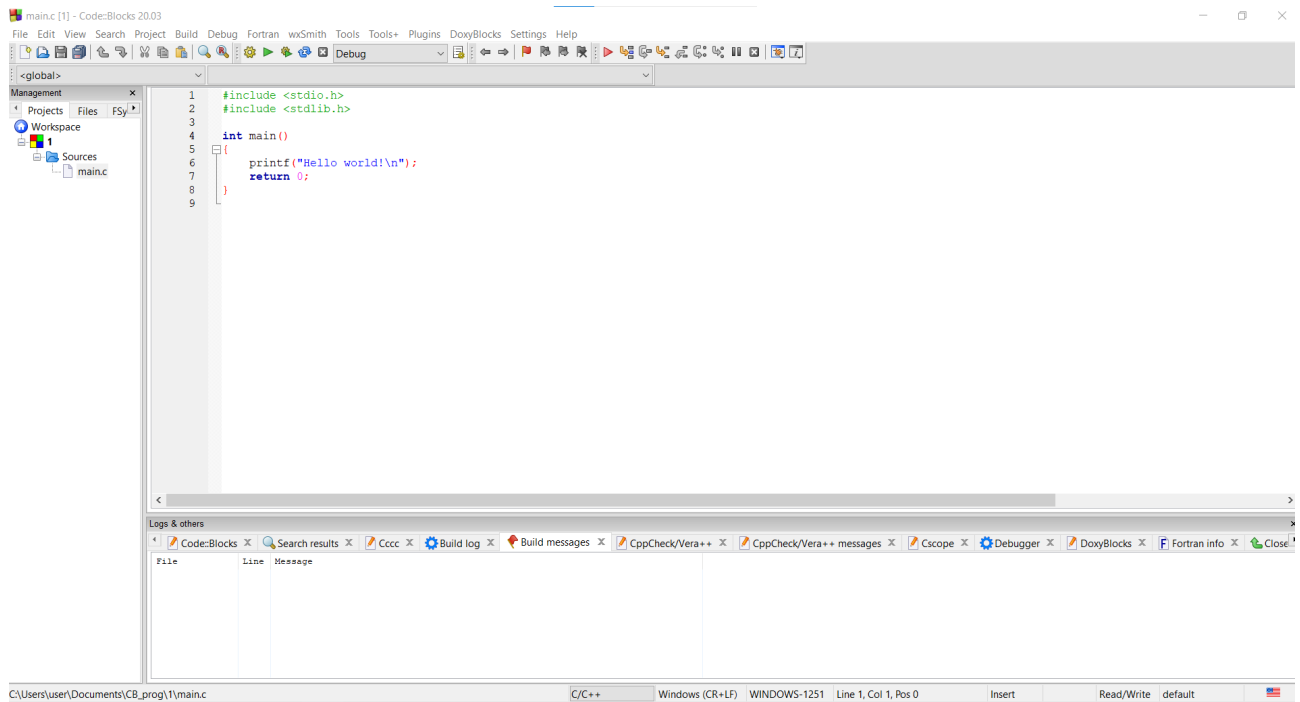


Figure 12. The main Code::Blocks IDE window

In the code editor window (Fig. 12) there is a Management panel on the left, where the hierarchical structure of the project is displayed in the form of a tree, consisting of one main.cpp file, which is located inside the virtual directory Sources. The Sources directory is inside the created project. The project belongs to the Workspace workspace and is named the name given to it when it was created. Double-clicking on the main.cpp file will cause it to open in the main window.

To close the file, you can click the corresponding icon next to the file name, or use the Ctrl+W keys.

To switch between open files, you can use the Ctrl+Tab keys, or click on their titles with the mouse.

If the file has been changed, an asterisk appears on its tab to the left of the file name. To save the file, press Ctrl+S or select the Save button in the toolbar.

Through the View menu (Fig. 13), you can control the appearance of the Code::Blocks IC. To display or hide toolbars, you need to go to the menu View->Toolbars and check the corresponding panels

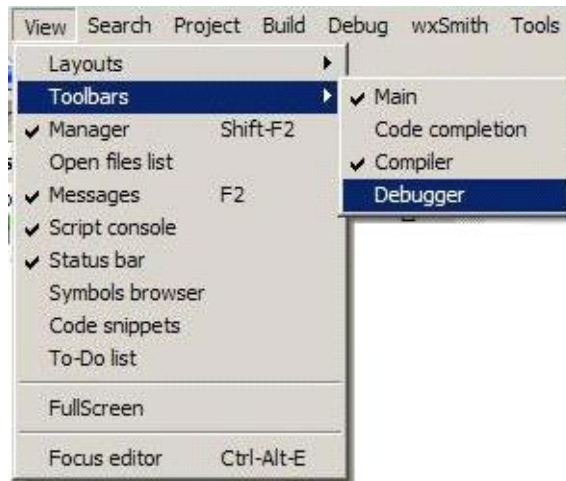


Figure 13. View menu

- Main – the main toolbar, on which the main actions for working with projects are displayed (Fig. 14);



Figure 14: Main panel

- Code completion – a panel for viewing code objects (Fig. 15);



Figure 15: Code completion panel

- Compiler – a panel with buttons for controlling application compilation (Fig. 16);

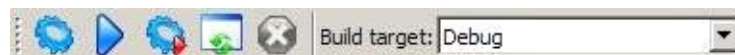


Figure 16: Compiler panel

- Debugger – a panel with buttons for controlling application debugging (Fig. 17).



Figure 17: Debugger panel

The View->Messages menu item or the F2 key will show or hide the Messages compiler message window at the bottom of the screen.

Encoding setting

UTF-8 encoding must be set for normal operation (see status bar). If a different encoding is installed, then it must be changed and after the change, reopen all open files.

To configure the Code::Blocks encoding, you need to select the Settings->Editor... menu item, after which the Configure editor window of the code text editor settings will open (Fig. 18).

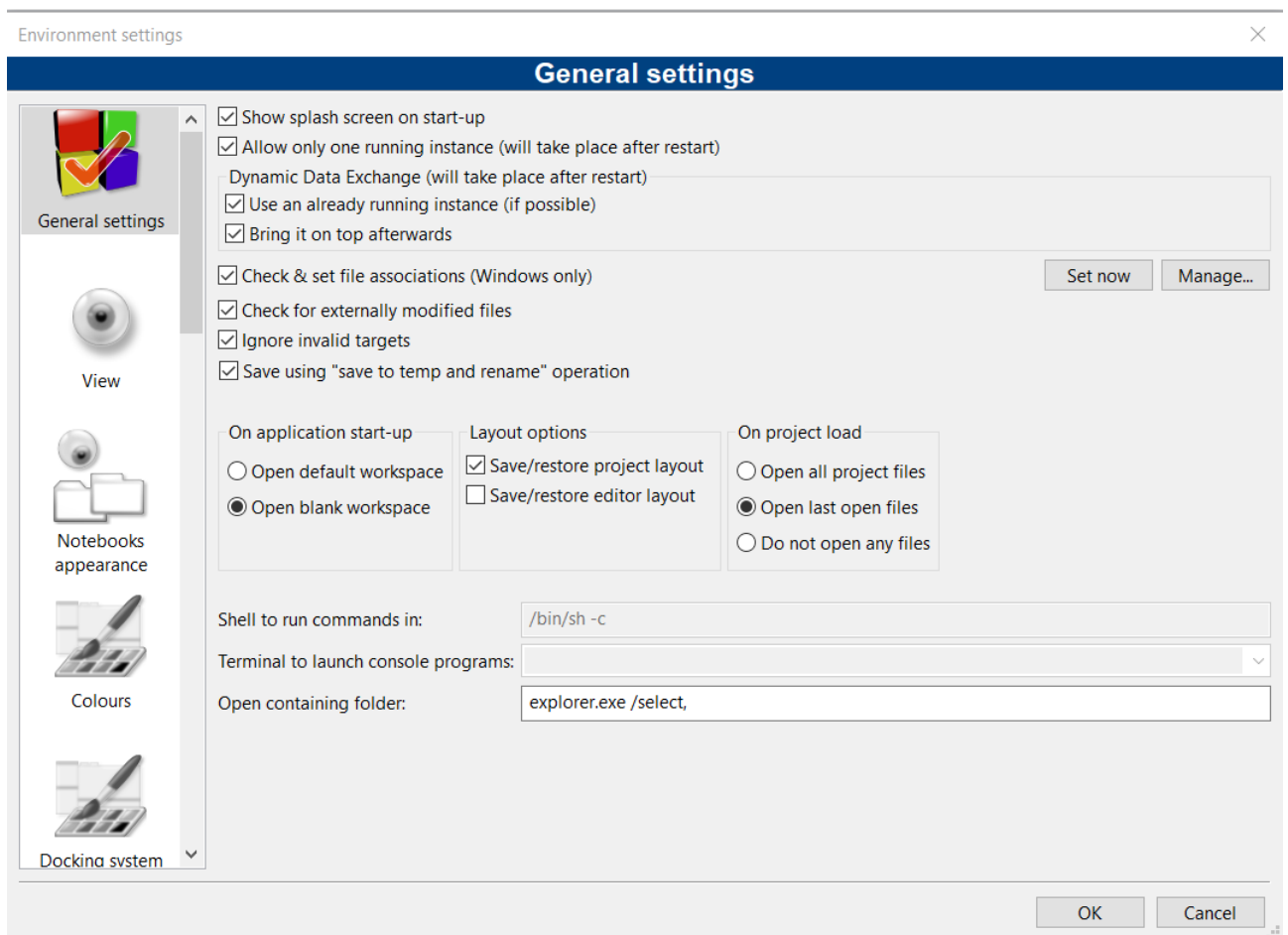


Figure 18. Configure editor code text editor settings window

In the Default encoding when opening files drop-down list, you need to select UTF-8 encoding, click OK and reopen all open files in ISR.

Work with several open projects

With several projects open at the same time, the active project is highlighted in bold, and regardless of which file is currently open in the main window, the active project will be compiled.

To switch projects, hover the mouse over the project you want to make active and press the right mouse button. In the drop-down menu (Fig. 19), select Activate project. To close this project, select the Close project menu item.

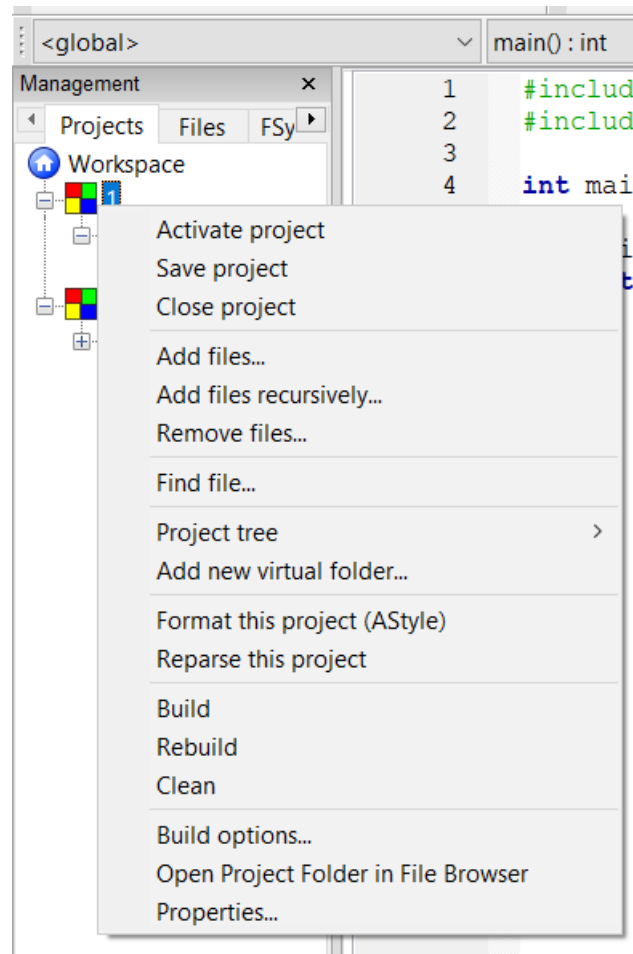


Figure 19. Project menu

Compilation and assembly of the project

To assemble the project, compile and run the application, press F9 or the Build->Build and run menu item.

- Build – assembly of the entire application
- Complete current file – compilation of the currently open file
- Run – launch of an already compiled application

- Build and run – collect and run
- Rebuild - reassemble
- Clean – clean the project from compiled and temporary files.

If you only need to compile the project without starting it, you need to select the Build menu item or press Ctrl+F9.

When creating the project, it was necessary to define the application assembly scenarios. Two build scenarios, Debug or Release, also allow you to get two independent versions of the program with different build options, and accordingly, with different options for optimizing the application. In order to use them, you need to select the appropriate build script. This can be done in the Compiler toolbar (Fig. 16), where you can specify one of two scenarios in the Build target drop-down list. You can change the compilation options for each of the scenarios in the Project build options window (Fig. 20) by selecting the appropriate Debug or Release scenario.

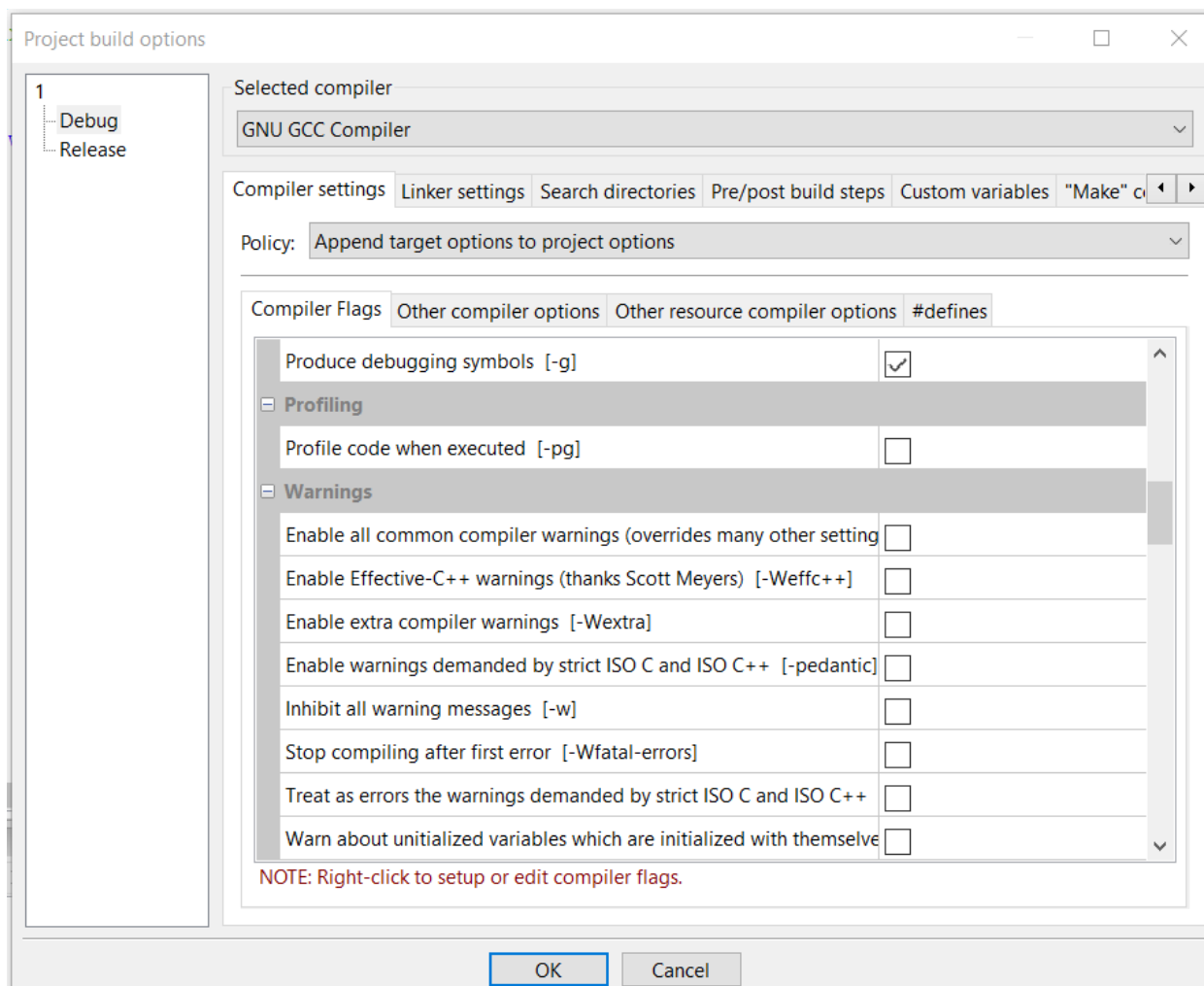


Figure 20. "Project build options" compilation settings window

GCC compiler options

In some cases, additional setting of compiler options is required. These settings are made for a specific project in the Project build options window (Fig. 20), which can be called from the project menu by selecting the Build options... item (Fig. 19). Globally, these settings are set for the entire program in the Compiler and debugger settings window (Fig. 21).

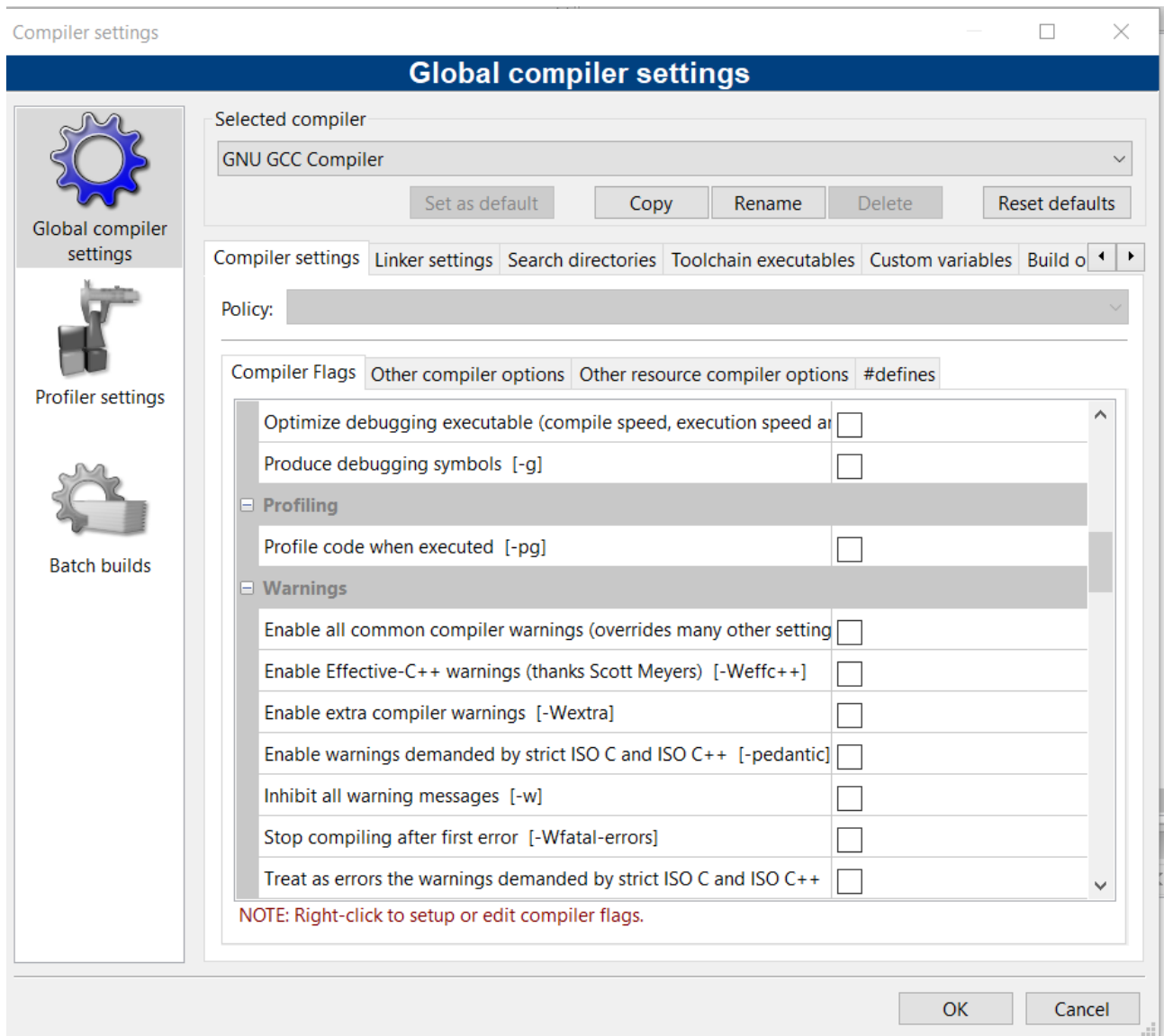


Figure 21. Compiler and debugger settings window

If the program is not compiled and the message "cc1plus.exe: error: unrecognized command line option „-Wfatal-errors“" appears in the Messages panel in the Build log tab, you need to go to the menu item Settings -> Compiler and debugger... and in the window that opened, provided that the GNU GCC Compiler is selected, in the

corresponding drop-down list of Selected compiler, on the Compiler Flags tab, you need to uncheck the item "Stop compiling after first error [-Wfatal-errors]".

If the Messages panel is not visible at the bottom of the screen, press F2 to open it.

Debugging the project

The program uses the Gdb debugger, which you must install, to find runtime errors. To work with this debugger, use the Debug menu. To start the debugger, you need to select the Debug->Start menu item, after which the debugger will start, and if you do not specify a breakpoint, it will run the entire application step by step. To view the contents of variables, breakpoints must be specified. A breakpoint is set either by clicking the left mouse button on the gray separator bar next to the line number, or by placing the cursor on the line where you want to stop.

To view the contents of variables and arrays, the Watches panel (window) is used (Fig. 22).



Figure 22. Watch panel

In order to add variables to this window, you need to click on the Watches window with the right mouse button, after which you need to select the Add watch item in the drop-down menu.

The Edit watch dialog box will open, where you need to enter the name of the variable in the Keyword field and click the OK button. To view the array, you need to mark the Watch as array marker.

CREATION AND ADJUSTMENT NEW PROGRAMS

1. Open a new editor window to enter a new program.
2. Type the text of the new program.
3. It is imperative to write the program text to disk before starting the program for execution for the first time, because the program text may be lost due to errors in the program or a computer failure.
4. Start the program for execution.
5. If syntax errors were made in the program, a corresponding message will appear on the screen, and the cursor will show the location of the error.
6. View the program execution results.
7. If incorrect results are obtained, then you need to correct algorithmic errors and run the program again for execution.
8. Repeat points 4-7 until correct results are obtained.
9. Save the debugged program on disk.

TASKS

TASK 1. The simplest operators of the C programming language

Formulation of the problem

Develop a program that calculates and outputs the values of t_1 and t_2 according to formulas that correspond to the variant of the individual task (see below). Determine the areas of valid values of formula parameters and specify arbitrary values from these areas.

Parameters named a and b are integers (type `int`), and other parameters are floating points (float type).

The values of the parameters with the names x and y must be entered from the keyboard, and the values of the rest must be set as the initial values of the corresponding variables.

Content of the report

1. Statement of the task, a specific version of the task.
2. Text of the program.
3. Tests for debugging the program and the results obtained for them.
4. Calculation results for arbitrary input data.

Variants of tasks

Variant	Task
1.	$t1 = \frac{1}{b^3} \left(\log \left(\frac{x}{y} \right) + \frac{b^3}{x} \cdot tg \left(\frac{2}{ax} \right) \right)$ $t2 = \frac{2}{a^2} \cdot tg(ax) - \left(\sqrt{\frac{x^2}{a} - \frac{2}{a^2}} \right) \cdot \cos(bx)$
2.	$t1 = \frac{xb}{y} + \frac{b}{y^2} \cdot \log(ax + y)$ $t2 = \frac{1}{2ab} \cdot \left(\log \sqrt{2ab - x} \right) - \frac{1}{2dy}$

3.	$t1 = \frac{1}{b} (\log (\frac{ax}{2}) - \frac{1}{\sin(bx)})$ $t2 = \frac{1}{a} \cdot \frac{1}{(n-1)x^2} - \frac{b}{(n-2)x^{n-2}}$
4.	$t1 = \frac{1}{c} (\frac{1}{ax+b} + \frac{y}{d} \cdot \log (\frac{yx+b}{ax+d}))$ $t2 = \frac{\sin(by)}{2a} + \frac{x\sqrt{c^2+b^2}}{2y\sqrt{c^2-b^2}}$
5.	$t1 = \frac{b}{(\sqrt{a}-b) \cdot (b+x)} - \frac{x}{(a-b)^2} \cdot \log(a+x)$ $t2 = \frac{1}{a} \cdot (\log (\frac{\cos(ax)}{\sin(ax)}) - \frac{1}{2x})$
6.	$t1 = \frac{1}{(a-b)^2} \cdot (\frac{1}{a+x} - \frac{1}{(1-x)^2}) + \frac{2}{(a-y)^3} \cdot x$ $t2 = \frac{b}{2a} \cdot \frac{\sin(ax)}{\cos^2(ax)} + \frac{2a}{b} \cdot \cos(ax)$
7.	$t1 = \frac{1}{2a^2} \cdot (\frac{y\sqrt{c^2+b^2}}{x\sqrt{c^2-b^2}}) + \sqrt{y-x}$ $t2 = \frac{2x}{b^2} (\sin(ax) - 0.5) - \frac{1}{2a} \cdot \cos(\frac{ax}{2})$
8.	$t1 = \frac{1}{a^2} (\log(x) - \frac{b^2}{2x} + \frac{2b}{x^2})$ $t2 = \frac{\cos(ax)}{2a \cdot \sin^2(ax)} + \frac{1}{2a} \cdot \log(\operatorname{tg} \frac{ax}{2})$
9.	$t1 = \frac{1}{a^4} (\frac{y^3}{3} - 3bx + 3d \cdot \log(x) + \frac{b^3}{x})$ $t2 = \frac{1}{1-\cos^2(ax)} + \frac{1}{a} \cdot \log(\frac{ax}{2})$
10.	$t1 = \frac{1}{b^2+1} (\log(\frac{y}{x}) + \frac{ax}{2y})$ $t2 = -\frac{x}{2a} \cdot \frac{\cos(ax)}{\sin(ax)} + \frac{2}{a^2} \log(\sin(\frac{ax}{2}))$

11.	$t1 = \frac{ax}{y} + \frac{b}{y^2} \cdot \log (yx + d))$ $t2 = \frac{-1}{a(n-2)} \cdot \frac{\sin(ax)}{\cos(n-1) \cdot bx}$
12.	$t1 = \frac{1}{2a^2} \cdot (\log \left(\frac{y\sqrt{c^2 + b^2}}{x\sqrt{c^2 - b^2}} \right) + 1)$ $t2 = \frac{1}{a} \cdot \operatorname{tg} \left(\left \frac{ax}{2} \right \right) + \frac{1}{a} \log \left(\frac{ax}{2} \right) \cdot \sqrt{b^2 - 1}$
13.	$t1 = \frac{1}{c} \left(\frac{1}{ax + b} + \frac{y}{c} \cdot \log \left(\frac{yx + d}{ax + b} \right) \right)$ $t2 = \frac{1}{a} \cdot \operatorname{tg} \left(\left \frac{ax}{2} \right \right) + \frac{1}{2} \log \left(\frac{ax - 1}{2} \right)$
14.	$t1 = \frac{b}{(a-b) \cdot (b-x)} - \frac{2}{(a-b)^3} \cdot \log \left(\frac{a+x}{b+x} \right)$ $t2 = \frac{1}{2a} \cdot \left(\frac{\cos(ax)}{\sin^2(ax)} - \log \left(\frac{ax+1}{2} \right) \right)$
15.	$t1 = \frac{-1}{(a-b)^2} \cdot \left(\frac{1}{a+x} + \frac{1}{1+x} \right) + \frac{2}{(a-b)^3} \cdot \log \left(\frac{y}{a} \right)$ $t2 = \frac{1}{1 - \sin^2(ax)} + \left(\frac{1}{a} \cdot \operatorname{tg} \left(\left \frac{ax}{2} \right \right) \right)$
16.	$t1 = \frac{1}{b^2} \left(\log \left(\frac{y}{3x} \right) + \frac{ax}{y} \right)$ $t2 = \frac{\cos(ax)}{2a \cdot \sin^2(ax)} + \frac{1}{2a} \cdot \log \left(\frac{bx}{2} \right)$
17.	$t1 = \frac{1}{c} \left(\frac{b}{a} \cdot \log(ax + b) + \frac{d}{y} \cdot \log(yx + d) \right)$ $t2 = \frac{1}{2ab} \cdot \left(\log \left(\frac{x\sqrt{c^2 - b^2}}{2y\sqrt{c^2 + b^2}} \right) \right)$

18.	$t1 = \frac{1}{b^3} (\log \frac{y}{x} - \frac{a^3 x^2}{2y^2})$ $t2 = \frac{\sin(ax)}{2a \cdot \cos^2(x)} + \frac{1}{2a} \cdot \log \left(\frac{ax}{2} \right)$
19.	$t1 = \frac{1}{a} \left(\frac{-1}{(n-2)x^2} + \frac{b}{(n-1)x^{n-2}} \right)$ $t2 = \frac{1}{a} \left(\log \left(\frac{ax}{2} \right) - \frac{1}{\sin(ax)} \right)$
20.	$t1 = \frac{1}{a^4} \left(\frac{x^3}{3} - 3bx + 3b^2 \cdot \log(x) + \frac{b^3}{x} \right)$ $t2 = \frac{2x}{a^2} \cdot \sin(ax) - \left(\frac{x^2}{a} - \frac{2}{a^2} \right) \cdot \cos(ax)$
21.	$t1 = \frac{aby}{3} + \frac{d}{x} \cdot \log(ax + y)$ $t2 = \frac{1}{2} \cdot \left(\log \left(\frac{y}{x} \right) - \frac{ab}{2y} \right)$
22.	$t1 = \frac{1}{3a} \left(\log \left(\frac{yx}{2} \right) + \frac{1}{\cos(bx)} \right)$ $t2 = \frac{b}{a} \cdot \left(\frac{x^2}{(d-1)} - \frac{y^2}{(b-2a)} \right)$
23.	$t1 = \frac{bx}{(\sqrt{a}-b) \cdot (\sqrt{b}+a)} + \frac{y}{(x-b)^2} \cdot (a+x)$ $t2 = \frac{1}{ay} \cdot \left(\operatorname{tg} \left(\frac{bx}{2} \right) - \log \left(\frac{5}{x} \right) \right)$
24.	$t1 = \frac{1}{3d} \left(\frac{1}{ax+b} + \frac{d}{x} \cdot \log \left(\frac{ax+b}{ax-d} \right) \right)$ $t2 = \frac{\sqrt{y}b}{2a} + \frac{x\sqrt{c^2+b^2}}{2y\sqrt{c^2-b^2}}$

25.	$t1 = \frac{1}{ab} \cdot \left(\frac{\sqrt{c^2 - b^2}}{\sqrt{c^2 + b^2}} \right) + d\sqrt{y+x}$ $t2 = \frac{2}{x} \cdot \sin(ax) - \frac{y}{2a} \cdot \cos\left(\left \frac{ax}{2}\right \right)$
26.	$t1 = \frac{xb}{4y} + \frac{b}{2y^2} + \frac{x}{y^4} \cdot \sqrt{dy}$ $t2 = \frac{x}{2ab} \cdot (\log \sqrt{yab-1}) - \frac{1}{2ab} \cdot \sin(by)$
27.	$t1 = \frac{1}{c} \left(\frac{1}{ax+b} + \frac{1}{(ax+b)^2} \cdot \cos(yb+d) \right)$ $t2 = \frac{a}{b} \cdot tg(ax) + \frac{1}{2} \log(by+1)$
28.	$t1 = \frac{x}{a^2} \left(\sqrt{dy} - \frac{b^2}{2x} + \frac{2b}{x^2} \right)$ $t2 = \frac{\cos^2(ax)}{\sin^2(ax)} + \frac{1}{2a} \cdot \cos^2(ax) + \sin^2(ax)$
29.	$t1 = \left(\frac{b^2}{y} + \frac{2b}{y} + \frac{1}{y} \right) \cdot \left(\frac{y}{(b-1)^2} \right)$ $t2 = \frac{1}{2ax} \cdot (\sqrt{2ab+b^2}) - \frac{y}{2dx}$
30.	$t1 = \frac{1}{a^2} \left(\frac{x^2}{3} + 3bx + 3b^2 - \frac{b^3}{x} \right)$ $t2 = \frac{2y}{a^2} \cdot \sin(bx) - \left(\frac{x^2}{a} \right) \cdot \cos(bx)$

Instructions for completing the Task 1

Library <math.h>

- $\text{sqrt}(x)$ - finding the root of the number x
- $\text{tan}(x)$ is the tangent of x
- $\text{sin}(x)$ is the sine of the number x
- $\text{cos}(x)$ is the cosine of the number x

- $\log(x)$ is the logarithm of the number x
- $\text{abs}(x)$ is the modulus of the number x
- $\text{pow}(x, y)$ - raising the number x to the power of y

Example. Calculate by formula

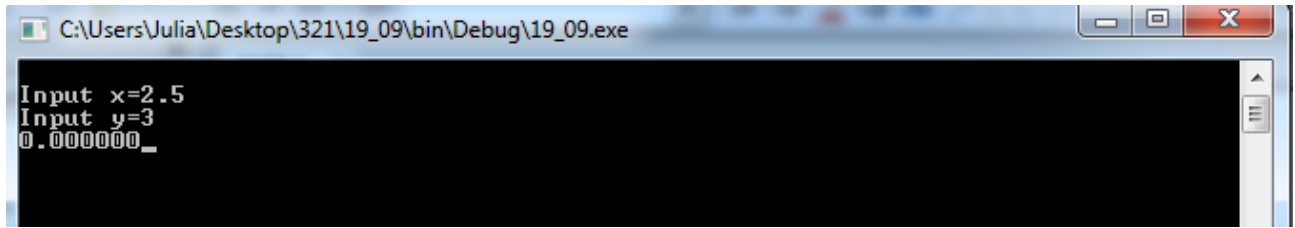
$$t1 = \frac{1}{a}(x^2 + 2)(5y - \ln x)$$

Mathematics expression	Expression with language C
$t1 = 1/a * (x^2 + 2)(5y - \log(x))$	$t1 = 1/a * (x * x + 2) * (5 * y - \log(x))$ or $t1 = 1/a * (\text{pow}(x, 2) + 2) * (5 * y - \log(x))$

Text of program1

```
#include <stdio.h>
#include <math.h>
int main()
{int a=2;
float x,y,t1;
printf("\nInput x=");
scanf("%f",&x);
printf("Input y=");
scanf("%f",&y);
t1=1/a*(x*x+2)*(5*y-log(x));
printf("%f",t1);
return 0;
}
```

The result of this program1



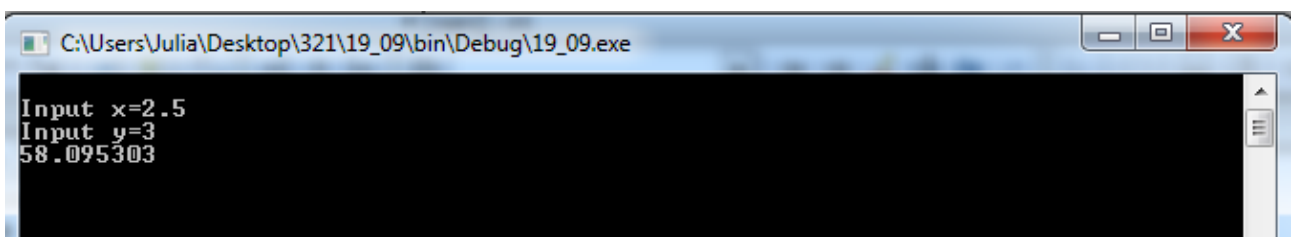
It is not correct result.

We must to change line $t1=1/a*(x*x+2)*(5*y-\log(x))$; and write correct program.

Text of new program2

```
#include <stdio.h>
#include <math.h>
int main()
{
    int a=2;
    float x,y,t1;
    printf("\nInput x=");
    scanf("%f",&x);
    printf("Input y=");
    scanf("%f",&y);
    t1=1.0/a*(x*x+2)*(5*y-log(x));
    printf("%f",t1);
    return 0;
}
```

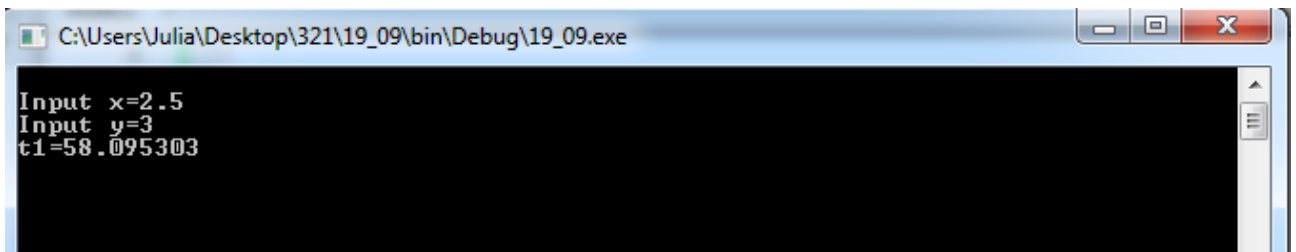
Result of new program2



Text of new program3

```
#include <stdio.h>
#include <math.h>
int main()
{int a=2;
float x,y,t1;
printf("\nInput x=");
scanf("%f",&x);
printf("Input y=");
scanf("%f",&y);
t1=(float)1/a*(pow(x,2)+2)*(5*y-log(x));
printf("%f",t1);
return 0;
}
```

Result of new program3



```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe
Input x=2.5
Input y=3
t1=58.095303
```

TASK 2. Calculate the function

Formulation of the problem

1. Write a program using the if / else structure without using logical operations
2. Write a program using logical operations
3. Compare the results

Variant	Task
1.	$y = \begin{cases} x^2 - 2, & \text{if } x \in (-\infty; -1) \\ \frac{2}{5}x^3 + 12x, & \text{if } x \in [0; 2) \end{cases}$
2.	$y = \begin{cases} x + 9, & \text{if } x \in [20; \infty) \\ x^2 - 5x, & \text{if } x \in [5; 15) \end{cases}$
3.	$y = \begin{cases} \frac{2}{7}x^3 - 7, & \text{if } x \in (-\infty; -7) \\ \frac{1}{7}x^2 + 2x - 1, & \text{if } x \in [-7; 0) \end{cases}$
4.	$y = \begin{cases} 2x^3 - x^2 - 2, & \text{if } x \in (-5; -2) \\ x + 10, & \text{if } x \in (12; \infty] \end{cases}$
5.	$y = \begin{cases} x - 5, & \text{if } x \in (-\infty; -5) \\ 9x^4 + 1, & \text{if } x \in [0; 20) \end{cases}$
6.	$y = \begin{cases} x^2 + 8, & \text{if } x \in (-4; -1) \\ x^2 + 10x - 256, & \text{if } x \in (10; \infty] \end{cases}$
7.	$y = \begin{cases} x^2 - x + 17, & \text{if } x \in [0; 17) \\ \frac{1}{5}x + 1, & \text{if } x \in [20; 25) \end{cases}$
8.	$y = \begin{cases} x^2 - 2, & \text{if } x \in (-\infty; -7) \\ \frac{2}{3}x^3 + 12x, & \text{if } x \in (6; 12] \end{cases}$
9.	$y = \begin{cases} x^4 - x^2 + 25, & \text{if } x \in (-5; 0) \\ -56 + 25x, & \text{if } x \in [-\infty; -12) \end{cases}$

10.	$y = \begin{cases} 4x^2 - 15, & \text{if } x \in [-3; -1) \\ \frac{1}{9}x^3 - x, & \text{if } x \in [0; 3) \end{cases}$
11.	$y = \begin{cases} x^3 - 8, & \text{if } x \in [0; 15) \\ x^2 + x - 13, & \text{if } x \in (-10; -2) \end{cases}$
12.	$y = \begin{cases} 19x + 7, & \text{if } x \in (0; 7) \\ x^3 - \frac{1}{3}x, & \text{if } x \in [-12; -5] \end{cases}$
13.	$y = \begin{cases} x^2 - 2, & \text{if } x \in (-\infty; -1) \\ x^4 - x^2 + 2, & \text{if } x \in (1; \infty) \end{cases}$
14.	$y = \begin{cases} x^2 - 7, & \text{if } x \in [0; 1] \\ x + 10, & \text{if } x \in (1; \infty) \end{cases}$
15.	$y = \begin{cases} 4x^2 - 15, & \text{if } x \in [-10; -8) \\ \frac{1}{5}x^3 - x, & \text{if } x \in [-5; 5) \end{cases}$
16.	$y = \begin{cases} x^5 - 1, & \text{if } x \in (-\infty; -1) \\ 3x^3 + 1, & \text{if } x \in [0; 9) \end{cases}$
17.	$y = \begin{cases} x^2 - 27, & \text{if } x \in [-1; 3] \\ \frac{3}{5}x^2 + 5, & \text{if } x \in [10; 15) \end{cases}$
18.	$y = \begin{cases} 5x^3 + 2x^2 + 2, & \text{if } x \in (-\infty; -3) \\ x + 10, & \text{if } x \in [-1; 5] \end{cases}$
19.	$y = \begin{cases} 2x^3 - x^2, & \text{if } x \in (-25; -20) \\ 5x + 10, & \text{if } x \in [-20; 20] \end{cases}$
20.	$y = \begin{cases} 10x - 6, & \text{if } x \in (-15; 0] \\ x^3 - \frac{1}{3}x, & \text{if } x \in (0; 3) \end{cases}$
21.	$y = \begin{cases} x^3 - 2, & \text{if } x \in (5; +\infty) \\ x^2 + 5, & \text{if } x \in (-5; 0) \end{cases}$
22.	$y = \begin{cases} 2x^2 - 16, & \text{if } x \in [-4; 4) \\ \frac{1}{9}x^2 - x - 1, & \text{if } x \in [10; 20) \end{cases}$

23.	$y = \begin{cases} \frac{1}{5}x^2 + 5, & \text{if } x \in [5; 10) \\ x^3 - 9, & \text{if } x \in [0; 2) \end{cases}$
24.	$y = \begin{cases} x^4 - x^2 - 1, & \text{if } x \in (-2; 0) \\ -5 + 25x, & \text{if } x \in (-\infty; -12) \end{cases}$
25.	$y = \begin{cases} x + 2.5, & \text{if } x \in (-10; -2) \\ 2x^4 + x + 1, & \text{if } x \in [0; 3) \end{cases}$
26.	$y = \begin{cases} 2x^2 - 5, & \text{if } x \in (-\infty; -5) \\ x^2 + 10, & \text{if } x \in (5; \infty) \end{cases}$
27.	$y = \begin{cases} 2x^2 - 0.5, & \text{if } x \in [-1; 1) \\ \frac{1}{5}x^2 - x, & \text{if } x \in [10; 20) \end{cases}$
28.	$y = \begin{cases} 2x - x^2, & \text{if } x \in (-5; -2) \\ 5x - 10, & \text{if } x \in [-1; 1] \end{cases}$
29.	$y = \begin{cases} 3x - 2/3, & \text{if } x \in (-3; 3) \\ 2x - 1, & \text{if } x \in [5; 10] \end{cases}$
30.	$y = \begin{cases} 2x^2 - 6.5, & \text{if } x \in [-3; 4) \\ \frac{1}{8}x^2 - 1, & \text{if } x \in [8; 16) \end{cases}$

Instructions for completing the Task 2

Example 1. Calculate function (without logical operations)

$$y = \begin{cases} x^2 - 2, & \text{if } x \in (-5; 0] \\ x + 7, & \text{if } x \in (-\infty; -1) \cup [1; 5) \end{cases}$$

```
#include <stdio.h>

int main()
{
    float x,y;
    printf("\nInput x=");
    scanf("%f",&x);
    if (x<-10) printf("y=%f",x+7);
    else if(x<=-5) printf("function does not exist");
        else if(x<=0) printf("y=%f",x*x-2);
            else if(x<1) printf("function does not exist");
                else if(x<5) {y=x+7;
                    printf("y=%f",y);
                }
                else printf("function does not exist");

    getch();
    return 0;
}
```

```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe
Input x=-15
y=-8.000000_
```

```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe
Input x=-10
function does not exist
```

```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe
Input x=-7
function does not exist
```

```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe
Input x=-5
function does not exist
```

```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe
Input x=-2
y=2.000000
```

```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe
Input x=0
y=-2.000000
```

```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe
Input x=0.5
function does not exist
```

```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe
Input x=1
y=8.000000
```

```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe
Input x=5
function does not exist_
```

```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe

Input x=10
function does not exist_
```

```
C:\Users\Julia\Desktop\321\19_09\bin\Debug\19_09.exe

Input x=3
y=10.000000
```

Example 2. Calculate function (without logical operations)

$$y = \begin{cases} x^2 - 5, & \text{if } x \in (-5; 0] \cup (5; 10) \\ \frac{2}{3}x - x^3, & \text{if } x \in (0; 5] \cup [10; 15) \end{cases}$$

```
#include <stdio.h>

int main()
{
    float x;
    printf("Enter x\n");
    scanf("%f",&x);
    if (x<=-5)    printf("error\n");
    else
        if (x>15)    printf("error\n");
    if (x>-5)
        if (x<=0) printf("y=%.5f\n",x*x-5);
        if (x>5)
            if (x<10) printf("y=%.5f\n",x*x-5);
            if (x>0)
                if (x<=5) printf("y=%.5f\n",2.0/3.0*x-x*x*x);
                else if (x>=10)
                    if (x<15) printf("y=%.5f\n",2.0/3.0*x-x*x*x);
    return 0;
}
```

```
C:\Users\Julia\Desktop\321\333\bin\Debug\333.exe
Enter x
-45
error
```

```
C:\Users\Julia\Desktop\321\333\bin\Debug\333.exe
Enter x
-5
error
```

```
C:\Users\Julia\Desktop\321\333\bin\Debug\333.exe
Enter x
2
y=-6.666667
```

Example 3. Calculate function (with logical operations)

$$y = \begin{cases} x^2 - 5, & \text{if } x \in (-5; 0] \cup (5; 10) \\ \frac{2}{3}x - x^3, & \text{if } x \in (0; 5] \cup [10; 15) \end{cases}$$

```
#include <stdio.h>

int main()
{
    float x;
    printf("Enter x\n");
    scanf("%f",&x);
    if ((x<=-5)|| (x>=15))    printf("error!!!\n");
    if (((x>-5)&&(x<=0))||((x>5)&&(x<10)))
        printf("y==%.5f\n",x*x-5);
    if (((x>0)&&(x<=5))||((x>=10)&&(x<15)))
        printf("y==%.5f\n",2.0/3.0*x-x*x*x);
    return 0;
}
```

```
C:\Users\Julia\Desktop\321\333\bin\Debug\333.exe
Enter x
55
error!!!
```

```
C:\Users\Julia\Desktop\321\333\bin\Debug\333.exe
Enter x
-1
y== -4.000000
```

```
C:\Users\Julia\Desktop\321\333\bin\Debug\333.exe
Enter x
4
y== -61.333333
-
```

```
C:\Users\Julia\Desktop\321\333\bin\Debug\333.exe
Enter x
11
y== -1323.666667
-
```

Conclusions: results are equal.

TASK 3. Develop a program using the operator switch

1. Develop a program that allows you to enter a letter of the English alphabet (the first 10 letters) and display which letter is a vowel or consonant.
2. Develop a program that allows you to enter the month number, and the entered number to determine what time of year it is (winter, spring, summer, autumn).
3. Develop a program that allows you to enter the number of the day of the week, and determine the name of the day of the week by the entered number.
4. Develop a program that allows you to enter the number of the month, and the number entered to determine the number of days in this month (for non-leap year).
5. In some educational institution it was decided to evaluate the quality of students' work on the following scale: "2" - unsatisfactory; "3" - satisfactory; "4" - good; "5" - excellent, less than 2 - "very, very bad". Develop a program that allows you to enter a score on the scale and display the appropriate message.
6. Develop a program that allows you to enter an integer from 1 to 10. Depending on the number entered, you need to output a certain number of rows with stars. The number of stars is equal to the line number (in the first line - 1, in the second - 2, etc.).
7. Develop a program that allows you to enter a symbol and display whether the symbol is a number. Use the character code table.
8. Develop a program that displays the following menu: 1 - red, 2 - blue, 3 - white, 4 - yellow, 5 - green, 6 - orange, 7 - white, 8 - black. After selecting the menu number, the program should display the color name.
9. Develop a program that allows you to enter a letter of the English alphabet (the last 10 letters) and display which letter is a vowel or consonant.
10. The user enters an integer in the range of 1 to 7, which determines the number of learning tasks on a topic. Print a line with the specified number of tasks and the same number of exclamation marks.

11. The user enters some integer in the range of 10-20. Display the string - the name of this number on the screen
12. Print the name of the rainbow color depending on the selected number (1- red, 2 orange, 3-yellow, 4-green, 5- light blue, 6-blue, 7-purple)
13. Ticket price for 1 segments of travel – 10,50 UAH. Create a program that shows the total fare to station A (3 segments), B (5 segments), C (7 segments), B (8 segments).
14. Depending on the number of the chosen specialty display its name: 121 - Software Engineering, 122 - Computer Science and Information Technology, 123 - Computer Engineering, 124 - Systems Analysis, 125 - Cybersecurity, 126 - Information Systems and Technologies
15. Ticket price for 1 person – 119,2 UAH 20 . The user enters the number of passengers and receives the cost of tickets. (Number of people from 1 to 10)
16. Units of length are numbered as follows: 1 - decimeter, 2 - kilometer, 3 - meter, 4 - millimeter, 5 - centimeter. The user specifies the number of the unit of length (integer in the range [1... 5] and the length of the segment in these units (real number). Find the length of the segment in meters.
17. Units of mass are numbered as follows: 1 - kilogram, 2 - milligram, 3 - gram, 4 - ton, 5 - quintal. The user specifies the unit number of the mass [integer in the range [1... 5] and the body weight in these units (real number) .Find the body weight in kilograms.
18. The user enters the radius value. The program allows you to count the elements: 1 - diameter, 2 - the length of the circle $L = 2 \cdot \pi \cdot R$, 3 - the area of the circle $S = \pi \cdot R^2$. Use 3.14 as the value of π .
19. The user enters an integer in the range [1... 12] and receives the corresponding number in English
20. The year is divided into 4 quarters. Depending on the quarter number, display the names of the months included in it.
21. The user enters 2 values of the legs of a right triangle (a, b). The hypotenuse is calculated by the Pythagorean theorem. Calculate the values of 1-sine ($\sin = a /$

- c), 2-cosine ($\cos = b / c$), 3-tangent ($\text{tg} = a / b$), 4 - cotangent ($\text{ctg} = b / a$)) of this triangle.
22. Develop a program that allows you to enter a letter of the English alphabet (the first 10 letters) and display which letter is a vowel or consonant.
23. Develop a program that allows you to enter the month number, and the entered number to determine what time of year it is (winter, spring, summer, autumn).
24. Develop a program that allows you to enter the number of the day of the week, and determine the name of the day of the week by the entered number.
25. Develop a program that allows you to enter the number of the month, and the number entered to determine the number of days in this month (for non-leap year).
26. The user enters an integer in the range [101... 110] and receives the corresponding number in English
27. The year is divided into 4 quarters. Depending on the quarter number, display the names of the months included in it.
28. The user enters 2 values of the legs of a right triangle (a, b). The hypotenuse is calculated by the Pythagorean theorem. Calculate the values of 1-sine ($\sin = a / c$), 2-cosine ($\cos = b / c$), 3-tangent ($\text{tg} = a / b$), 4 - cotangent ($\text{ctg} = b / a$)) of this triangle.
29. Develop a program that allows you to enter an integer from 2 to 12 with step 2(2,4,6,8,10,12). Depending on the number entered, you need to output a certain number of rows with stars. The number of stars is equal to the line number (in the first line - 2, in the second - 4, etc.).

Instructions for completing the Task 3

Example. Enter integer number from keyboard from 1 to 4 and output this number as word

```
#include <stdio.h>

int main()
{
    int input;
    printf("Enter number from 1 to 4\n");
    scanf( "%d", &input );
    switch ( input ) {
        case 1:
            printf("First\n");
            break;
        case 2:
            printf("Second\n");
            break;
        case 3:
            printf("Third\n");
            break;
        case 4:
            printf( "Fourth\n" );
            break;
        default:
            printf( "ERROR\n" );
    }
    return 0;
}
```

```

#include <stdio.h>
int main()
{
    int input;
    printf("Enter number from 1 to 4\n");
    scanf( "%d", &input );
    switch ( input ) {
        case 1:
            printf("First\n");
            break;
        case 2:
            printf("Second\n");
            break;
        case 3:
            printf("Third\n");
            break;
        case 4:
            printf( "Fourth\n" );
            break;
        default:
            printf( "ERROR\n" );
    }

    return 0;
}

```

Results

```

Enter number from 1 to 4
2
Second

```

```

Enter number from 1 to 4
1
First

```

```
Enter number from 1 to 4
7
ERROR
```

```
Enter number from 1 to 4
3
Third
```

```
Enter number from 1 to 4
4
Fourth
```

Task 4. Calculation of the sum

According to the individual variant to make the program for calculation of the sum of an infinite series, summing up members of a series which values on the module exceed the set accuracy $s = 10^{-4} = 0.0001$. Determine the number of terms. Perform the calculation for x ($-2 < x < 2$)

№	Функція	№	Функція
1	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!}$	16	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{2k+1}}{2^k (2k-1)!}$
2	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^{2k}}{(k)! 2^{k-1}}$	17	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^k}{(k+4)!}$
3	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1} x^k}{(k)!}$	18	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{3k-1}}{k^2 (k+1)!}$
4	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{2k+1}}{k(2k+1)!}$	19	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^{2k}}{2^k (k)!}$
5	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{2k-1}}{k(k+3)!}$	20	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{3k-1}}{2k(k+3)!}$
6	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{2k+1}}{2k(k+1)!}$	21	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{k+2}}{k(2k+1)!}$
7	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^{2k}}{(2k-1)!}$	22	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{2k-1} x^{2k+1}}{k(2k)!}$

8	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1} x^{3k-1}}{(2k)!}$	23	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{3k} x^{2k-1}}{2^{2(k-1)} (k+1)!}$
9	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^{k+3}}{(k+2)! k^2}$	24	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^{k-1}}{(k+1)(k+3)!}$
10	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^{2k}}{(2k+1)!}$	25	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{2k-1} x^{2k+1}}{k(2k-1)!}$
11	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1} x^{3k-1}}{(k)! (k+2)}$	26	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{k+3}}{(k+1)! (2k)}$
12	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{3k-1}}{(3k)! (k+3)}$	27	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{3k-1} x^{3k-1}}{2^k (k+3)!}$
13	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1} x^{3k+1}}{3k(k+1)!}$	28	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1} x^{3k-1}}{(2k)!}$
14	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{2(k+1)}}{(k)! (k+2)}$	29	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^{2(k+1)}}{(2k+3)!}$
15	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^{3(k-2)}}{(k+3)(3k)!}$	30	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1} x^{2k-1}}{2^{k+1} (k+1)!}$

Instructions for completing the Task 4

Example 1. Calculate by formula

$$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^k}{(k+1)!(k+2)}$$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
//f(x)=\sum_{k=1}^{\infty} \frac{(-1)^k x^k}{(k+1)!(k+2)!}
int main()
{
    int t,N, f=1,k;
    float y,x,q;
    y=0;
    printf("\n\tinput x ");
    scanf("%f",&x);
    k=1;
    do
    {
        f=1;
        for(t=1;t<=k+1;t++)
            {f=f*t;}
        q=pow((-1),k)*pow(x,k)/((k+2)*f) ;
        printf("\n q [%d]=%f ",k,q);
        y=y+q;
        k++;
    }while(fabs(q)>0.0001);
    printf("y=%f",y);
    return 0;
}
```

```
input x -1.2

q[1]=0.200000
q[2]=0.060000
q[3]=0.014400
q[4]=0.002880
q[5]=0.000494
q[6]=0.000074
y=0.277774

Process returned 0 (0x0)   execution time : 4.119 s
Press any key to continue.
```

Example 2.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
//f(x)=\sum_{k=1}^T \frac{((-1)^k x^k)}{(k+2)(k+1)!}
int main()
{
    int t,N, f=1,T,k;
    float y,x,q;
    y=0;
    printf("\n\input x ");
    scanf("%f",&x);
    q=(-1)*x/((1+2)*(1*2)) ;
    k=2;
    y=y+q;
```

```

printf("\n q[1]=%f",q);
while (fabs(q)>0.0001)
{
    f=1;t=1;
    while (t<=k+1)
        {f=f*t;
        t++;}
    q=pow((-1),k)*pow(x,k)/((k+2)*f) ;
    printf("\n q[%d]=%f",k,q);
y=y+q;
k++;
}

    printf("\n y=%f",y);
    printf("\n\n\n\n");
return 0;
}

```

```

        input x  -1.2

q[1]=0.200000
q[2]=0.060000
q[3]=0.014400
q[4]=0.002880
q[5]=0.000494
q[6]=0.000074
y=0.277774

Process returned 0 (0x0)   execution time : 8.817 s
Press any key to continue.

```


Example 3.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
//f(x)=\sum_{k=1}^{\infty} \frac{((-1)^k x^k)}{(k+2)(k+1)!}
int main()
{
    int t,N, f=1,k;
    float y,x,q;
    y=0;
    printf("\n\tinput x ");
    scanf("%f",&x);
    for (k=1; ;k++)
    {
        f=1;t=1;
        while (t<=k+1)
        {f=f*t;
            t++;}
        q=pow((-1),k)*pow(x,k)/((k+2)*f) ;
        printf("\n  q[%d]=%.12f",k,q);
        y=y+q;
    }
    printf("\n  y=%.0f",y);
    printf("\n\n\n\n");
    return 0;
}
```

```
input x -1.2
```

```
q[1]=0.200000002980
```

```
q[2]=0.060000006109
```

```
q[3]=0.014400001615
```

```
q[4]=0.002880000509
```

```
q[5]=0.000493714411
```

```
q[6]=0.000074057163
```

```
y=0.277774
```

TASK 5. Create program with one-dimension array

1. Given an array of numbers. Find the value of the maximum element. If there are more than one element, then determine how many of them.
2. Given an array of numbers. Find how many pairs of identical adjacent elements.
3. Fill an array of 10 elements with random integers. Arrange the array in ascending order. Find the sum of odd elements of the array, display the results.
4. Fill an array of integers with random 14 elements. Sort the array in ascending order. Display maximum and minimum elements, find their difference.
5. Fill an array of integers with random 10 elements. Arrange the array in ascending order. Calculate the number and product of the elements of the array, greater than 20 and less than 50 and display the result.
6. Fill an array of integers with random 13 elements. Sort the array in descending order. Calculate the number and sum of elements of the array greater than 10. Display the result.
7. Fill an array of integers with random 10 elements. Arrange the array in ascending order. Calculate the number and sum of elements of the array, which are divisible by 5 without remainder.
8. Fill an array of integers with random 15 elements. Sort the array in descending order. Calculate the sum of the elements of the array, which are multiples of 3 and display the result.
9. Fill an array of integers with random 12 elements. Sort the array in ascending order. Find the number and sum of even elements of the array and display the result.
10. Given an array of numbers. Find the value of the minimum element. If there are more than one element, then determine how many of them.

11. Given an array of numbers. Sort it in descending order. Find the average value of the elements of the array that is greater than 30.
12. Fill an array of integers with random 10 elements. Sort the array in descending order. Find the sum of squares of odd elements of the array and display the results.
13. Fill an array of integers with random N elements. Make a mirror image of the elements relative to the middle of the array. It's not allowed to use additional arrays.
14. Fill an array of integers with random N elements. Find the largest and smallest element of the array and swap them. If there are several maximum or minimum elements, it is not necessary to swap the maximum and minimum elements of the array.
15. Fill an array of integers with random 13 elements. Sort the array in descending order. Calculate the number and sum of elements of an array greater than 10. Display the result.
16. Fill an array of integers with random 10 elements. Sort the array in ascending order. Calculate the number and sum of elements of the array, which are divisible by 5 without remainder.
17. Fill an array of integers with random N elements. Sort the array in descending order. Calculate the sum of the squares of the elements of the array and display the result.
18. Fill an array of integers with random 12 elements. Sort the array in ascending order. Find the number and sum of even elements of the array, display the result.
19. Given an array of numbers. Find the value of the minimum element. If there are more than one element, then determine how many of them.
20. Fill an array of float with random 10 elements. Arrange the array in ascending order. Calculate the number and product of the elements of the array, greater than 5 and less than 10 and display the result

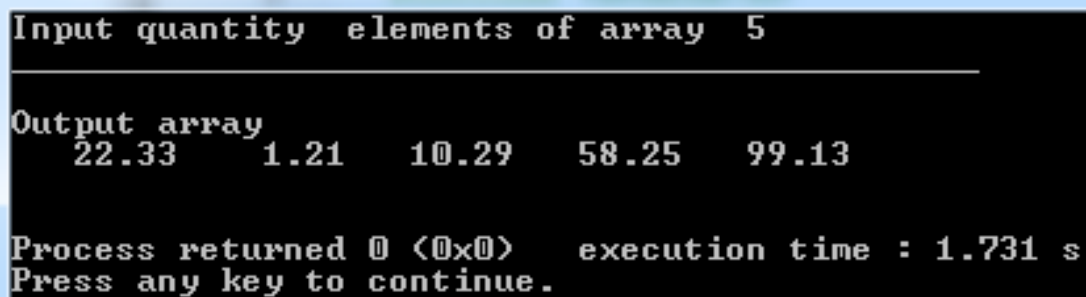
21. Given an array of numbers. Sort it in descending order of absolute values of array elements.
22. Fill an array of integers with random 10 elements. Sort the array in descending order. Find the sum of squares of odd elements of the array and display the results.
23. An array of size N is given. Display the elements of the array located between its minimum and maximum elements.
24. Given an array of numbers. Sort it in ascending order. Find the arithmetic mean of the elements of the array.
25. Enter an array of integers with 10 elements from the keyboard. Sort the array in ascending order. Find the sum of squares of even elements of the array and display the results.
26. Given an array of numbers. Find the value of the minimum element. If there are more than one element, then determine how many of them.
27. Fill an array of integers with random 15 elements. Sort the array in descending order. Calculate the number and sum of elements of the array greater than 12. Display the result.
28. Fill an array of integers with random 18 elements. Calculate the number and product of elements of the array smallest than 10. Display the result.
29. Given an array of numbers. Sort it in descending order of absolute values of array elements.
30. Fill an array of integers with random 9 elements. Arrange the array in ascending order. Calculate the number and sum of elements of the array, which are divisible by 3 without remainder.

Instructions for completing the Task 4

Example 1. Filling the array with random numbers and output result

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    int i,n;
    float x;
    float A[10];
    printf("Input quantity elements of array ");
    scanf("%d",&n);
    srand(time(NULL));
    for (i=0;i<n;i++)
    {
        x=(float)(rand()%9)/24; // fractional part of a number
        A[i]= rand()%100+x; // random integers from 0 to 99 + fractional part x
    }
    printf("_____ \n");
    for (i=0;i<n;i++)
    printf("  %5.2f",A[i]);
    return 0;
}
```



```
Input quantity elements of array 5
_____
Output array
  22.33   1.21   10.29   58.25   99.13
Process returned 0 (0x0)   execution time : 1.731 s
Press any key to continue.
_
```

```
Input quantity  elements of array  4

Output array
  69.08   11.13   57.17   38.33

Process returned 0 (0x0)   execution time : 2.036 s
Press any key to continue.
```

Example 2. Search for the minimum element in the one-dimension array

```
#include <stdio.h>

int main ()
{
    int Arr[100];
    int i,N,min;
    printf("Input quantity  elements of array  ");
    scanf("%d",&N);
    printf (" Input array by %d: elements\n",N);
    for (i=0; i<N;i++)  scanf("%d",&Arr[i]);
    printf ("array :\n");
    for (i=0; i<N;i++) printf("%4d",Arr[i]);
    min=Arr[0];
    for (i=1; i <N; i++)
        if (min>Arr[i]) min=Arr[i];
    printf("\nMin=%d",min);
    return 0;
}
```

```

Input quantity elements of array 5
      Input array by 5 elements
2 4 -1 0 3
array :
  2   4  -1   0   3
Min=-1
Process returned 0 (0x0)   execution time : 23.867 s
Press any key to continue.

```

Example 3. Search for the **last** maximum element in the one-dimension array

```

#include <stdio.h>

int main ()
{
    int Arr[100];
    int i,N,max,maxi;
    printf("Input quantity elements of array ");
    scanf("%d",&N);
    printf (" Input array by %d elements\n",N);
    for (i=0; i<N;i++)
        scanf("%d",&Arr[i]);
    printf ("array :\n");
    for (i=0; i<N;i++) printf("%4d",Arr[i]);
    max=Arr[0];maxi=0;
    for (i=1; i <N; i++)
        if (max<=Arr[i]) {max=Arr[i];maxi=i;}
    printf("\nMax=A[%d]=%d",maxi,max);
    return 0;
}

```



```
Input quantity elements of array 5
      Input array by 5 elements
9
2
8
9
2
      9      2      8      9      2
Max=A[3]=9
Process returned 0 (0x0)   execution time : 12.715 s
Press any key to continue.
```

TASK 6. Using of functions

Write a variant solution program with the obligatory use of functions to describe the actions that must be repeated, namely:

- input of array elements;
- output of array elements;
- execution on the array of actions specified in the variant.

1. Given three arrays of integers A [10], B [12], C [8] and an integer M. When sequentially viewing the elements from the beginning of the arrays to find in each of them the number of elements exceeding a given number M.
2. Given three arrays of real numbers A [8], B [15], C [10]. Divide all the elements of the array by its minimum element.
3. Given three matrices of integers A [7], B [7], C [7]. Assign the variable p a value of 2 if all matrices are equal to each other, a value of 1 if any two matrices are equal, 0 - otherwise.
4. Given two matrices of integers A [10], B [12]. Calculate each of these items that are more than 15 and have an odd index.
5. Given two matrices of integers A [10], B [12]. Calculate each of these items that are less than 15 and have an even index.
6. Given three matrices of integers A [5], B [7], C [10]. In each of them find all the maximum elements and replace them with 0.
7. Given three arrays of integers A [12], B [10], C [7]. In each of them, find the sum of the elements placed after the first negative element
8. Given three arrays of integers A [9], B [11], C [13]. In each of them find the sum of the elements placed after the first 0
9. Given three arrays of integers A [4], B [13], C [8]. In each of them find the product of the elements placed before the first 0
10. Given three arrays of integers A [10], B [12], C [14]. In each of them find all the minimum elements and replace them 1.

11. Given three arrays of real numbers X [9], Y [16], Z [12]. In each of these arrays, replace the elements with indexes divided to 3 by 5
12. Given three arrays of real numbers X [12], Y [14], Z [10]. In each of these array elements even replace 0
13. Given three arrays of real numbers X [9], Y [11], Z [15]. In each of these arrays, replace the odd elements with 1
14. Given three arrays of integers A [17], B [8], C [12] and integer K. Calculate in each array the sum of the elements that belong to the element with number K.
15. Given three arrays of integers A [11], B [10], C [14] and an integer K. Calculate in each array the product of the elements following the element with number K
16. Given three arrays of integers A [11], B [14], C [12] .Each array in descending order
17. Given three arrays of integers A [10], B [7], C [13]. Perform in each of them a cyclic shift of the elements to the right by 1 position.
18. Given three arrays of integers A [12], B [5], C [9]. Perform in each of them a cyclic shift of the elements to the left by 1 position.
19. Given three arrays of integers A [11], B [10], C [14]. If the first element of the array is negative, then calculate the sum of the elements, otherwise the product of all elements.
20. Given three arrays of integers A [11], B [10], C [14]. If the first element of the array is 0, then calculate the sum of the elements, otherwise - return 0
21. Given three matrices of integers A [7][4], B [6][5], C [6][8]. Calculate the sum of the elements separately in each row of the given matrices.
22. Given three matrices of integers A [5][4], B [5][5], C [6][7]. Calculate the sum of the elements separately in each column of the given matrices.
23. Given three matrices of integers A [5][5], B [4][6], C [7][3]. In each of them find all the maximum elements and replace them with zeros.
24. Given three matrices of integers A [3][4], B [7][2], C [4][6]. In each of them find all the minimum elements and replace them with 1.

25. Given two matrices of integers A [4][7], B [7][4]. Swap the first column with the last and output result
26. Given two matrices of integers A [3][7], B [2][5]. Find the sum of elements in each row and output result
27. Given two matrices of integers A [3][7], B [2][5]. Find the sum of elements in each column and output result
28. Given two matrices of integers A [5][6], B [3][4]. Swap the first row with the last and output result
29. Given three arrays of integers A [10], B [12], C [8] and an integer T. When sequentially viewing the elements from the end of the array to find an element that exceeds the given T.
30. Given three arrays of integers A [16], B [15], C [17]. Each array in descending order

Instructions for completing the Task 6

Example. There are three arrays of real (float) numbers A[8], B[13], C[10].

Divide all elements of the array by its maximum element

```
#include <stdio.h>

void InputArray (const N,float X[N]);
void OutArray (const N,float X[N]);
void modifyArray(const N,float X[N],float max_array);

int main ()
{
    float A[8];
    float B[13];
    float C[10];
    float max_array;
    printf ("Input Array A \n");
    InputArray(8,A);
    OutArray(8,A);
```

```

modifyArray(8,A,max_array);
printf ("\n\nInput Array B \n");
    InputArray(13,B);
    OutArray(13,B);
    modifyArray(13,B,max_array);
    printf ("\n\nInput Array C \n");
    InputArray(10,C);
    OutArray(10,C);
    modifyArray(10,C,max_array);
return 0;
}

void InputArray (const N,float X[N])
{
    int i;
    for (i=0; i <N; i++)
    {
        printf("\nEnter element of array %d  ",i+1);
        scanf("%f",&X[i]);
    }
}

void OutArray (const N,float X[N])
{
    int i;
    printf("\nArray\n" );
    for (i=0; i <N; i++)
        printf("%.3f ",X[i]);
}

void modifyArray(const int N, float X[N],float max_array)
{
    int i;

```

```

max_array=X[0];
for(i = 0; i <N; i++)
{
    if (max_array<X[i]) max_array=X[i];
}
printf("\nMax element %f ",max_array);
printf("\nModify array\n");
for(i = 0; i <N; i++)
{
    if (max_array==0) {printf("\nError\n"); break;}
    X[i]=X[i]/max_array;
    printf("%.3f ",X[i]);
}
}

```

```

Input Array A
Enter element of array 1  1
Enter element of array 2  2
Enter element of array 3  3
Enter element of array 4  4
Enter element of array 5  5
Enter element of array 6  6
Enter element of array 7  7
Enter element of array 8  8
Array
1.000  2.000  3.000  4.000  5.000  6.000  7.000  8.000
Max element 8.000000
Modify array
0.125  0.250  0.375  0.500  0.625  0.750  0.875  1.000  _

```

```

Input Array B
Enter element of array 1  1
Enter element of array 2  2
Enter element of array 3  3
Enter element of array 4  4
Enter element of array 5  5
Enter element of array 6  6
Enter element of array 7  7
Enter element of array 8  8
Enter element of array 9  9
Enter element of array 10 10
Enter element of array 11 11
Enter element of array 12 12
Enter element of array 13 13

Array
1.000  2.000  3.000  4.000  5.000  6.000  7.000  8.000  9.000 10.000 11.000 1
2.000 13.000
Max element 13.000000
Modify array
0.077 0.154 0.231 0.308 0.385 0.462 0.538 0.615 0.692 0.769 0.846 0.923 1.000

```

```

Input Array C
Enter element of array 1  9.1
Enter element of array 2  2.3
Enter element of array 3  5.7
Enter element of array 4  9.3
Enter element of array 5  4.4
Enter element of array 6  6.7
Enter element of array 7  2.2
Enter element of array 8  12.3
Enter element of array 9  5.89
Enter element of array 10 1.12

Array
9.100  2.300  5.700  9.300  4.400  6.700  2.200 12.300  5.890  1.120
Max element 12.300000
Modify array
0.740 0.187 0.463 0.756 0.358 0.545 0.179 1.000 0.479 0.091 _

```

Example 2.

Enter a two-dimensional array from the keyboard.

Find the first negative and last positive on the main diagonal of the matrix and display their positions on the screen

```
#include <stdio.h>
#include <stdlib.h>
#define K 3
#define P 4
void input (int AA[K][P],int k,int p){
    int i,j;
    for (i = 0; i < k; i++) {
        for (j = 0; j < p; j++) {
            printf("A[%d][%d]",i,j);
            scanf("%d",&AA[i][j]);
        }
    }
}
void output (int AA[K][P],int k, int p){
    int i,j;
    for (i = 0; i < k; i++) {
        for (j = 0; j < p; j++) {
            printf("%d\t ",AA[i][j]);
        } printf("\n");
    }
}
void SearchArr(int AA[K][P],int k, int p){
    int i,j,bp=-1,bn=-1;
```



```

        int pos,neg;

        for (i = 0; i < k; i++){
            if (AA[i][i]>0){ pos=AA[i][i]; bp=i;}}
        for (i = 0; i < k; i++){
            if (AA[i][i]<0) {neg=AA[i][i]; bn=i;} break;}
        /*for (i = n-1; i >=0; i--){
            if (AA[i][i]<0) {neg=AA[i][i]; bn=i;} }*/
        if(bn!=-1) printf("\nFirst    negative element=  %d, position i=%d,
j=%d",neg,bn,bn);

        else printf("\nElement is error!");

        if(bp!=-1)    printf("\nLast    positive element=  %d, position i=%d,
j=%d",pos,bp,bp);

        else printf("\nElement is error!");
    }

    int main () {
        int A[10][10];
        srand(time(NULL));
        input (A,K,P);
        printf ("The value of the original array\n");
        output (A,K,P);
        printf("\n");
        SearchArr (A,K,P);
        return 0;
    }

```

The value of the original array

-3	-2	-3	-2	-1
0	-2	0	0	-3
-3	-2	1	1	-3
-1	-3	-3	1	1
-2	-3	-1	-1	-3

First negative element= -3, position i=0, j=0
Last positive element= 1, position i=3, j=3_

APPENDIX A

RESERVED WORDS

Keywords of C			
auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

APPENDIX B

```
*main.c X
1  #include <stdio.h>
2  int main()
3  {int t;
4   printf ("Input temperature\n");
5   scanf ("%d",&t);
6   if (t<=-20) {
7       printf("Very Cold\n");
8       printf("\tVery Cold\n");
9   }
10  }
11  else if (t<=-5) printf("Cold\n");
12          else if (t<=0) printf("Cool\n");
13          else printf("Normal\n");
14  return 0;
15  }
16
```

```
D:\Bib\345\bin\Debug\345.exe
Input temperature
-30
Very Cold
    Very Cold

Process returned 0 (0x0)   execution time : 5.539 s
Press any key to continue.
-
```

```
D:\Bib\345\bin\Debug\345.exe
Input temperature
-7
Cold

Process returned 0 (0x0)   execution time : 9.312 s
Press any key to continue.
-
```

```
D:\Bib\345\bin\Debug\345.exe
Input temperature
0
Cool
Process returned 0 (0x0)   execution time : 17.082 s
Press any key to continue.
_
```

```
D:\Bib\345\bin\Debug\345.exe
Input temperature
-1
Cool
Process returned 0 (0x0)   execution time : 10.580 s
Press any key to continue.
_
```

SOURCES

1. Stewart, Bill (7 січня 2000). History of the C Programming Language.
2. Patricia K. Lawlis, c.j. kemp systems, inc. (1997). Guidelines for Choosing a Computer Language: Support for the Visionary Organization. Ada Information Clearinghouse.
3. Programming Language Popularity. 2009.
4. TIOBE Programming Community Index. 2009.
5. Bjarne Stroustrup. FAQ (англ.).
6. Ritchie, Dennis. The Development of the C Language.
7. Brian W. Kernighan and Dennis M. Ritchie: The C Programming Language, 2nd ed.,
8. King, K.N. (2008). C Programming: A Modern Approach (2 ed.). W. W. Norton.
9. Gustedt, Jens (2019). Modern C (2 ed.). Manning.
10. C Programming Absolute Beginner's Guide 3rd Edition by Greg Perry, Dean Miller, 2013
11. <https://www.programiz.com/c-programming>
12. <https://www.tutorialspoint.com/cprogramming/index.htm>
13. <https://www.cprogramming.com/>