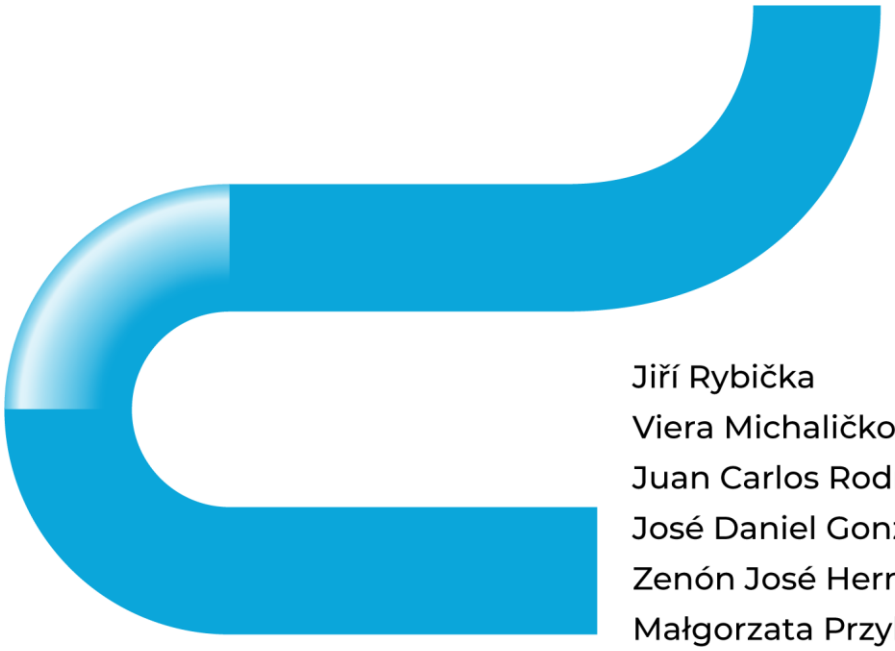
FITPED

C/C++ advanced



Jiří Rybička
Viera Michaličková
Juan Carlos Rodríguez-del-Pino
José Daniel González-Domínguez
Zenón José Hernández-Figueroa
Małgorzata Przybyła-Kasperek

www.fitped.eu

2021

Co-funded by the
Erasmus+ Programme
of the European Union

Work-Based Learning in Future
IT Professionals Education
(Grant. no. 2018-1-SK01-KA203-046382)

C/C++ Advanced

Published on

November 2021

Authors

Jiří Rybička | Mendel University in Brno, Czech Republic

Viera Michaličková | Constantine the Philosopher University in Nitra, Slovakia

Juan Carlos Rodríguez-del-Pino | University of Las Palmas de Gran Canaria, Spain

José Daniel González-Domínguez | University of Las Palmas de Gran Canaria, Spain

Zenón José Hernández-Figueroa | University of Las Palmas de Gran Canaria, Spain

Małgorzata Przybyła-Kasperek | University of Silesia in Katowice, Poland

Reviewers

Anna Stolińska | Pedagogical University of Cracow, Poland

Peter Švec | Teacher.sk, Slovakia

Eugenia Smyrnova-Trybulska | University of Silesia in Katowice, Poland

Piet Kommers | Helix5, Netherland

Graphics

Ľubomír Benko | Constantine the Philosopher University in Nitra, Slovakia

David Sabol | Constantine the Philosopher University in Nitra, Slovakia

Erasmus+ FITPED

Work-Based Learning in Future IT Professionals Education

Project 2018-1-SK01-KA203-046382

Co-funded by the
Erasmus+ Programme
of the European Union

ISBN 978-80-558-1780-4

Table of Contents

Files – Text Files

Chapter 1

1.1 I/O Stream library

📖 1.1.1

In C++, objects called streams are used to control files. A stream object contains a number of operations that can be performed on files. There are files of different types and are used in different modes, which is done by different stream variants.

Using the I/O Stream library, we can open the file by connecting it to the input, output or input/output stream. In order to do this, we should attach a library:

- fstream - Stream class to both read and write from/to files
- ifstream - Stream class to read from files
- ofstream - Stream class to write on files

All functions needed to access files are declared by these classes. To define the object of the required type of stream we use, for example:

```
ifstream file; // file stream not connected to any file
file.open ("name"); // connect the stream to a file "name"
```

Alternatively, we can use:

```
ifstream file ("name"); - a file stream connected to a file
"name"
```

If the file does not exist then it will be created. We can close the file explicitly by using

```
file.close();
```

or rely on the class destructor.

📝 1.1.2

In order to both read and write from/to file, you must attach the _____.

- fstream library
- ofstream library
- ifstream library

1.1.3

To declare an object named f1 of the output stream, we will use.

- ifstream f1;
- ofstream f1;
- ifstream f1("file_name");
- fstream f1;

1.1.4

An object declared as follows

```
fstream f2;
```

is an input-output stream object associated with a file named f2.

- False
- True

1.1.5

In order to declare an input stream connected to a file named *new_file* we use.

- ifstream str = new new_file;
- ifstream str("new_file");
- ofstream str("new_file");
- ifstream str(new_file);

1.2 Reading and writing from/to a file

1.2.1

In order to check whether the file was successfully opened, we can use a conditional statement *if(stream_name)*. Any stream can be assigned to a bool object. The object will be *true* if the stream is in the correct state; otherwise it will be *false*.

Alternatively, we can use the stream check function: *good()* or *is_open()*. Both functions return *true* if the file has been accessed. We use them as follows *stream_name.good()* or *stream_name.is_open()*

There is also *eof()* function that returns *true* if a file open for reading has reached the end.

🕮 1.2.2

There are different methods for reading data from a file.

First of all, we can read data from a file in analogy to stream cin>>

```
stream_name >> variable_to_which_the_data_will_be_saved
```

Data that has been read in this way is always treated as text. By using this method, we will not read any information about whitespace (enter, tab, space, etc.).

We can also read data from a file using the function *get()*. Example

```
fstream file ("file_name");
char data;
file.get (data);
```

Another way to read data from a file is to use the function *getline()*.

An example of using the getline() function from the *<string>* library is given below

```
fstream file("file_name");
string data;
getline(file, data);
```

An example of using the function *getline()* from the fstream class is given below

```
fstream file("file_name");
char date[20];
file.getline(date, 20);
```

The second parameter indicates the maximum number of characters that can be written to the variable data.

In both cases, the data that is read is treated as text.

The last way that will be discussed in this course is to use the *read()* function. It is a safe method for binary data, which will be discussed in the next part of the course.

An example of using the function *read()* is given below

```
fstream file("file_name");
char date[20];
file.read(date, 20);
```

In order to check how many bytes of data have been loaded into the buffer, we use the function *gcount()*.

```
file.gcount();
```

1.2.3

Now the methods of writing data to a file will be discussed. It should be noted that we can add the data at the end of the file or overwrite it. It is not possible to add text between existing data.

First of all, we can write data to a file analogously to the stream cout<<

```
stream_name << variable_from_which_the_data_will_be_saved
```

Data that has been write in this way is always treated as text.

We can write data to a file by using the function *put()*. Example

```
fstream file("file_name");
char data = 'a';
file.put (data);
```

Another way to write data to a file is to use the function *write()*. An example of using the *write()* function is given below

```
fstream file("file_name");
string data = "Data to write";
file.write (& data[0], data.length());
```

The second parameter indicates the number of characters that will be written to the file.

1.2.4

What commands will allow you to check the state of the stream?

- stream_name.open();
- stream_name.good();
- if(stream_name)

- stream_name.is_good();

1.2.5

How to read data from a file that does not give any information about whitespace?

- stream_name>>variable
- stream_name<|<|variable
- stream_name.getline(variable,1)
- stream_name.read(variable,1)

1.2.6

The use of which function guarantees that data that has been read will not necessarily be treated as text?

- read
- getline
- get
- put

1.2.7

Which function allows you to specify the number of characters that will be written to the file?

- put
- read
- write
- getline

1.2.8

What is not possible during writing data to a file?

- adding new data to the end of a file
- overwriting existing data with new one
- adding new text between existing data in a file

1.2.9

Select the line with incorrect code

```
1. ifstream file("name");
2. file.put("Great");
3. char data[31];
4. file.read(data, 30);
```

- line no. 1
- line no. 2
- line no. 3
- line no. 4

1.3 Text files (programs)

1.3.1 Read char from file

Write a code that will read the content of the file "file1.txt" that contains two lines:

```
I want to learn C ++.
Programming in C ++ is fun and cool.
```

The user specifies which character from the file should be displayed on the screen. If the value provided by the user is too large, display None.

```
Input : 4
Output: a
```

```
Input : 10
Output:
```

1.3.2 File to screen

Write a code that will read the content of the given file that contains numbers and put them into the screen. In case of an empty file return None.

```
Input : file1.txt
Output: 3,5,1,7,3
```

```
Input : file2.txt
Output: 3,3,3,3,3
```

1.3.3 Students

Write a code that will read the content of a given file. The file contains in each line the name of a student and his/her absence in school in hours. The data is delimited by a colon. Put the data into two tables and sum all the absence hours and find the student with the worst attendance. Return also the name of the student and the average absence. In case of an empty file, return None.

```
Input : file1.txt
Output: Sum of absence: 80 h; Average: 8.00 h; The worst
attendance: Michal
```

```
Input : file2.txt
Output: None
```

Content of file1.txt:

```
Jano:10 Michal:20 Fero:0 Juraj:5 Jana:0 Karol:15 Martin:8
Klara:12 Oliver:10 Ulrich:0
```

Content of file2.txt:

1.3.4 Grants

Write a code that will read two given text files with the names of students and their average grades. The average is separated using a semicolon, the averages are written with a dot. Return the list of names of honoured students - those who have an average below or equal to 1.4 (we don't need the averages). In case of two empty files, return None.

```
Input : file1a.txt file1b.txt
Output: Michal Jana Karol Milos
```

Content of file1a.txt:

```
Jano;1.9 Michal;1.2 Fero;3.0 Juraj;2.25 Jana;1.0
```

Content of file1b.txt:

```
Karol;1.25 Martin;3.8 Klara;2.01 Oliver;1.9 Ulrich;2.4
Milos;1.2
```

1.3.5 Average

Write a code that will return the average of numbers saved in a given text file. Show the results with one decimal number. The numbers are saved in the file so that each number is in a separate row. In case of an empty file, return None.

```
Input : file1.txt
Output: 3.4
```

```
Input : file2.txt
Output: None
```

Content of file1.txt:

```
1
5
7
3
1
```

Content of file2.txt:

1.3.6 Students names I.

Write a code that will read the content of a given file that contains names in separate rows. Return the names starting with the letter B and the name of the student that is first and last in the file. In case of an empty file, return None.

```
Input : file1.txt
Output: Barbora Boris Jano Peter
```

```
Input : file2.txt
Output: None
```

Content of file1.txt:

```
Jano Jana Michal Adam Barbora Andrea Juraj Boris Simona Peter
```

Content of file2.txt:

⌨ 1.3.7 Students names II.

Write a code that will read the content of a given file that contains the names each in separate rows (max. 100). Return the longest and shortest name and the name that occurs most often. In case of an empty file, return None.

```
Input : file1.txt
Output: Barbora Eva Jana
```

```
Input : file2.txt
Output: None
```

Content of file1.txt:

```
Jano
Jana
Michal
Adam
Eva
Barbora
Jana
Andrea
Juraj
Boris
Simona
Peter
```

Content of file2.txt:

⌨ 1.3.8 Telegram

The text file contains the text of a telegram. Write a code that will return the price of the telegram when each letter costs 0.10 Euro. Spaces and STOP do not count in the price. In case of an empty file, return None.

```
Input : file1.txt
Output: 3.1
```

```
Input : file2.txt
Output: None
```

Content of file1.txt:

```
I will not come today STOP I do not have time
```

Content of file2.txt:

1.3.9 Math

Write a code that will read the content of a given file. Each line contains a math equation with result, for example 11 + 23 = 44, the next line 15*2 = 31 and so on. There are adding, multiply and subtract. Read the file and return the "fix" of the equations: if the result is correct, write after the equation "OK", else write "Error:" and the correct result. After the last equation write a new line with the count of correct and wrong equations and percentage success. In case of an empty file, return None.

```
Input : file1.txt
Output:  1*9=10 Error: 9 5+2=7 OK 7-2=5 OK 3+3=3 Error: 3 1-
0=1 OK Correct: 3 Wrong: 2 Success: 60.0 percent
```

```
Input : file2.txt
Output: None
```

Content of file1.txt:

```
1*9=10
5+2=7
7-2=5
3+3=3
1-0=1
```

Content of file2.txt:

1.3.10 Numbers

Write a code that will search the given file so that will find all numbers. Let's assume the numbers are from the words separated by a space from each side. Return the maximum and minimum number. In case of an empty file, return None.

```
Input : file1.txt
Output: 3 123
```

```
Input : file2.txt
Output: None
```

Content of file1.txt:

```
I have bought 20 apples in the town but I brought home only 17
and 3 apples  have I eaten during the walk by the home with
number 123
```

Content of file2.txt:

Binary Files

Chapter 2

2.1 Binary files

📖 2.1.1

Let's go back to the open function, which was already used in the previous lesson.

The definition of this function is as follows:

```
void open( const char * filename, ios::openmode
file_open_mode);
```

The first parameter is a string representing the file's opening path and file name. The second parameter is optional and can take the following values:

- ios::app, set the file saving position at the end of the file. Data can be saved only at the end of the file;
- ios::ate, set the file pointer at the end of the file;
- ios::binary, data in the file is treated as binary data, not as text data;
- ios::in, file opened in reading mode;
- ios::out, file opened in write mode;
- ios::trunc, the contents of the file are deleted and replaced with new data.

📖 2.1.2

If we want to use several options at the same time, we use the symbol | (OR).

The ios::out mode is assigned to objects from the ofstream class by default. The ios::in mode is assigned to objects from the ifstream class by default. The ios::in | ios::out mode is assigned to objects from the fstream class by default.

📖 2.1.3

As was mentioned in the previous lesson, for binary files we do not use cin >> and cout << streams and get, getline and put functions. Only the use of reading and writing functions gives correct results for binary files.

The read and write functions have the following definitions:

```
read( char * buffer, streamsize);
write( const char * buffer, streamsize);
```

2.1.4

Examples of using these functions for binary files can be found below

```
#include <fstream>
int main () {
 ofstream file;
 file.open("example.bin", ios::out | ios::binary);
 if(file.is_open(){
 char temp[3];
 temp[0]='C';
 temp[1]='+';
 temp[2]='+';
 file.write(temp, 3);
 file.close();}
}
```

```
#include <fstream>
int main () {
 ifstream file;
 file.open("example.bin", ios::in | ios::binary);
 if(file.is_open(){
 char temp[3];
 file.read(temp, 3);
 file.close();}
}
```

2.1.5

The *ios::ate* flag in the open function means

- set the file pointer at the end of the file
- data can be written only at the end of the file
- contents of the file are deleted and replaced with new data
- file opened in write mode

2.1.6

The *ios::binary* flag in the open function means

- data in the file is treated as binary data
- set the file pointer at the end of the file
- contents of the file are deleted and replaced with new data

- file opened in write mode

2.1.7

In order to open a binary file in write mode and data will be saved at the end of the file we will use the following flags:

- ios::out
- ios::trunc
- ios::binary
- ios::app

2.1.8

In order to use several flags in the open function, we separate them with a sign

- |
- &
- ,
- ;

2.1.9

In order to write data in a binary file, we use

- write
- <|<|
- put
- read

2.1.10

In order to read data from a binary file, we use

- read
- >>
- get
- getline

2.1.11

The default mode assigned to objects of the ofstream class is

- ios::out
- ios::trunc
- ios::in
- ios::app

2.1.12

The default mode assigned to objects of the ifstream class is

- ios::in
- ios::out
- ios::trunc
- ios::app

2.1.13

Which line of the code is incorrect?

```
1. ifstream file;
2. file.open ("example.bin", ios::in || ios::binary );
3. char buffer [20];
4. if(file.is_open()){
5. file.read(bufer, 10);
6. file.close();}
```

- 2
- 1
- 3
- 4
- 5
- 6

2.1.14

The default mode assigned to objects of the fstream class is

- ios::out

- ios::trunc
- ios::in
- ios::app

2.2 Binary files (programs)

⌨ 2.2.1 Read char from file

Write a code that will read the content of the binary file "file1.bin".

The user specifies which character from the file should be displayed on the screen. If the value provided by the user is too large, display None.

```
Input : 4
Output: a
```

```
Input : 10
Output: None
```

⌨ 2.2.2 File to screen

Write a code that will read the content of the given binary file that contains numbers and put them into the screen. In case of an empty file return *None*.

```
Input : file1.bin
Output: 3,5,1,7,3
```

```
Input : file2.bin
Output: 3,3,3,3,3
```

⌨ 2.2.3 Students

Write a code that will read the content of a given binary file. The file contains in each line the length of string and then the string with the name of a student, colon and his/her absence in school in hours. Put the data into two tables and sum all the absence hours and find the student with the worst attendance. Return also the name of the student and the average absence. In case of an empty file, return *None*.

```
Input : file1.bin
Output: Sum of absence: 80 h; Average: 8.00 h; The worst
attendance: Michal
```

```
Input : file2.bin
Output: None
```

Content of file1.bin:

```
Jano:10 Michal:20 Fero:0 Juraj:5 Jana:0 Karol:15 Martin:8
Klara:12 Oliver:10 Ulrich:0
```

Content of file2.bin:

2.2.4 Grants

Write a code that will read two given binary files with the names of students and their average grades. The average is separated using a semicolon, the averages are written with a dot. In each line, the length of the string is saved. Return the list of names of honoured students - those who have an average below or equal to 1.4 (we don't need the averages). In case of two empty files, return *None*.

```
Input : file1a.bin file1b.bin
Output: Michal Jana Karol Milos
```

Content of file1a.bin:

```
Jano;1.9 Michal;1.2 Fero;3.0 Juraj;2.25 Jana;1.0
Content of file1b.bin:Karol;1.25 Martin;3.8 Klara;2.01
Oliver;1.9 Ulrich;2.4
Milos;1.2
```

2.2.5 Average

Write a code that will return the average of numbers saved in a given binary file. Show the results with one decimal number. In case of an empty file, return *None*.

```
Input : file1.bin
Output: 3.4
```

```
Input : file2.bin
Output: None
```

Content of file1.bin:

```
1 5 7 3 1
```

Content of file2.bin:

Standard I/O Streams

Chapter 3

3.1 Standard I/O streams

🕮 3.1.1

In the previous lessons, we learned the rules of using text files and binary files. On this occasion, the following issues were discussed:

- methods for declaring streams of various types,
- function open with its parameters,
- ways of writing and reading data to/from a file,
- ways to check if the stream has been correctly linked to a file,
- function close.

🕮 3.1.2

We will continue to discuss the input and output streams. Input-output operations are defined in the iostream library. Three objects were defined in the iostrem library:

- cin - an istream class object representing the standard input;
- cout - an ostream class object representing the standard output;
- cerr - an ostream class object representing the standard diagnostic output for error messages.

At the iostream input, the types are distinguished, moreover, it is safe in terms of types and expandable. So cin>>x reads to x according to the type of x. Similarly cout<<x writes x according to the type of x.

By default, the operator >> omits whitespace.

```
for (int i; cin >> i && 0 <i;)
  cout << i << '\n';
```

This loop will take a sequence of positive integers separated by whitespaces and will print them each in a separate line.

As for the files, also in the case of standard input and output, we can use the functions get, getline, put, read, write.

An example of using the get and put functions:

```
char sign;
while (cin.get (sign))
cout.put (sign);
```

Everything that has been read will be displayed on the screen, including spaces.

🕮 3.1.3

In C++, a file can be seen as an array of uninterpreted bytes that are indexed from 0 to length-1. Two independent positions are assigned to each stream opened to read and write:

- the current reading position is called "get pointer"
- the current writing position is called "put pointer"

Only get pointer is assigned to ifstream streams, and only put pointer is assigned to ofstream streams.

🕮 3.1.4

These positions can be operated by using the following functions:

- tellg - returns a streampos object, points to the place where the data will be read from the file

```
streampos tellg();
```

- tellp - returns a streampos object, points to the place where the data will be written to the file

```
streampos tellp();
```

- seekg - sets get pointer. The definition of this function is given below

```
seekg( streamoff offset, ios::seekdir direction);
```

The first parameter of the seekg function specifies the number of bytes to move the get pointer position. The second parameter is optional and specifies the place, relative to which the position will be moved. The default value is ios::beg i.e. the beginning of the file. The value ios::cur means the current position, and ios::end means the end of the file.

- seekp - sets put pointer. The definition of this function is given below

```
seekp( streamoff offset, ios::seekdir direction);
```

This function is analogous to the seekg function.

After making the jump to the new position, it is a good practice to check that the pointer is set correctly. For this purpose, you can use the fail function, which

belongs to the fstream class. This function returns the value true if the operation on the file was unsuccessful.

Another function that can be useful for working with files is the eof function. This function will return the value true if there is no more data to read in the file.

🕮 3.1.5

An example that presents the use of the functions described above, is presented below. In this example, the get pointer will be set to the end of the file, while the put pointer will be in the middle of the file.

```
#include <fstream>
int main() {
  fstream file;
  file.open( "file1");
  if(file){
    streampos begin,end;
    begin = file.tellg();
    file.seekg (0, ios::end);
    end = file.tellg();
    file.seekp=((end-begin)/2,ios::begin);
    file.close();
  }
  return 0;
}
```

🕮 3.1.6

Now we will discuss how to control the write buffer.

The fstream class has an internal buffer, which is designed to speed up working with files. When you save data to a file, they are initially saved in the buffer until it is overflow. This is because access to a given disk area requires a lot more time than access to the cache memory. There is a function in the fstream class that allows you to control the internal buffer. The flush function forces the entire buffer contents to be written to disk, regardless of whether the buffer is full or not.

🕮 3.1.7

An example of using the *flush* function is given below.

```
#include <fstream>
int main(){
  fstream file( "file1" );
  if( file.is_open() ){
      file << "Programming in C ++ is pleasant ";
      file.flush();
  file<< "and useful";

    file.close();
  }
  return 0;
}
```

📖 3.1.8

In the <sstream> header, there are input and output of string streams.

The string stream operations are as follows:

```
stringstream ss (m); - ss is an empty chain stream in m mode
stringstream ss (); - default constructor: stringstream ss
(ios_base :: out | ios_base :: in)
stringstream ss (s, m); - ss is a string stream whose buffer
is initialized using string s in m mode
stringstream ss (s); - stringstream ss (s, ios_base :: out |
ios_base :: in);
stringstream ss (ss2); - constructor - ss2 is moved to ss; ss2
becomes empty
ss = move (ss2); - assignment - ss2 is moved to ss; ss2
becomes empty
s = ss.str(); - s is a string copy of characters found in ss
ss.str(s); - the ss stream buffer is initialized using string
s; if the ss stream is in ios :: ate mode, values are added at
the end of s
ss.swap(ss2); - replaces ss and ss2.
```

📖 3.1.9

The standard output can be modified using some functions - manipulators. To use them, we should attach the <iomanip> or <ios> heading. Manipulators have been mentioned yet in the input/output format section. They will be described here in more detail.

For example, the code

```
cout << setprecision (4) << 344566.4444;
```

will display only four digits of the number, that is 3.446e + 005

The manipulators are listed below.

Input and output manipulators from the <ios> header

- cout << boolalpha Uses symbolic representation of true and false (input and output)
- cout << noboolalpha
- cout << showbase Add prefixes to numbers on the Output: 0 to octal and 0x to hexadecimal numbers
- cout << noshowbase
- cout << showpoint Always shows a decimal point
- cout << noshowpo
- cout << showpos Shows + before positive numbers
- cout << noshowpos
- cout << uppercase Applies capital letters to the numbers on the output, e.g. 1.2E10 and 0X1A2
- cout << nouppercase
- cout << internal Applies padding according to the designation in the formatting pattern
- cout << left Completes after value
- cout << right Completes before value
- cout << dec Integer value with base 10
- cout << hex Integer value with base 16
- cout << oct. Integer value based on 8
- cout << fixed Floating point format dddd.dd
- cout << scientific Scientific format d.ddddEdd
- cout << hexfloat Uses a number with base 16 in the mantissa and exponent;
- cout << defaultfloat Uses the default floating point format
- cin >> skipws Skip white characters
- cin >> noskipws

Input and output manipulators from the <iomanip> header

- cout << setbase(b) Integers with base b
- cout << setfill(int c) Set c as the fill character
- cout << setprecision(n) Sets the precision to n digits
- cout << setw(n) The width of the next field is n characters
- cin >> get_money(m, intl) Read from cin using the money_get; m is long double or basic_string; if intl == true, use standard three-letter currency names

- cout << put_money(m, intl) Writes m to cout using the money_put; money_put specifies types acceptable for m; if intl == true, use standard three-letter currency names
- cin >> get_time(tmp, fmt) Loads to * tmp in the fmt format and using the time_get facet of the cin object
- cout << put_time (tmp, fmt) Writes * tmp to the cout according to the fmt format, uses the time_put facet of the cout object

3.1.10

What does the *get pointer*, which is assigned to the read and write stream, indicate?

- current reading position
- current writing position
- pointer to the end of a file
- pointer to the beginning of a file

3.1.11

What does the *put pointer*, which is assigned to the read and write stream, indicate?

- current writing position
- current reading position
- pointer to the end of a file
- pointer to the beginning of a file

3.1.12

Which function returns a *streampos* object pointing to the place where the data from the file will be read?

- tellg
- seekp
- seekg
- tellp

3.1.13

Which function allows you to set the *put pointer* that is assigned to the stream?

- seekp
- seekg
- tellg
- tellp

3.1.14

What is the default value of the second parameter of the *seekg* function?

- ios::beg
- ios::cur
- ios::end
- ios::act

3.1.15

A function that returns the value *true* if there is no more data to read in the file is

- eof
- fail
- is_open
- good

3.1.16

When we save the data to a file, they are initially saved _____.

- in the buffer
- on the disk

3.1.17

The _____ function of the fstream class ensures that the entire contents of the buffer are written to the disk.

- seek
- splash
- flush

3.1.18

What type of object is returned by the *tellg* function?

- streampos
- streamoff
- seekdir
- int

3.1.19

The _____ is by default assigned to the *ifstream* streams.

- get pointer
- put pointer

3.2 I/O streams (programs)

3.2.1 Number of chars

Write a code that will calculate the number of characters contained in the given file (each character should be counted, including newline and space). Put half of the file's content into the screen. In case of an empty file return N*umber of characters in the file: 0*.

```
Input : file1.txt
Output: Number of characters in the file: 9
3
5
1
```

```
Input : file2.txt
Output: Number of characters in the file: 9
3
3
3
```

3.2.2 Natural numbers calculator

Write a code that will calculate operations on natural numbers +, -, *, /. If an error occurs, print the message using the standard diagnostic output.

```
Input : 3 2 /
Output: Error. The result of the operation is not a natural
number.
```

```
Input : 3 4 +
Output: 3+4= 7
```

⌨ 3.2.3 Incorrect type

Write a code that will calculate the sum of the value of items from some store. Elements are always given according to the scheme: ID number, item name, item value. Using the property that the standard input recognizes types, check if the items are correctly entered. The first number is the number of elements.

```
Input : 4  1000 table 12.5 1002 chair 4.5 1004 bookstand 21.5
1006 box 7.1
Output: The sum of the elements is equal: 45.6
```

```
Input : 4  table 1000 12.5 1002 chair 4.5 1004 bookstand 21.5
1006 box 7.1
Output: Error. Incompatibility of types.
```

⌨ 3.2.4 Words – statistics

Write a code that will read a given text. Return the number of all words and the longest and the shortest word and the average length of the word in the text. Do not count the signs ",.?!:;

```
Input :
Earth treading stars that make dark heaven light:
Such comfort as do lusty young men feel
When well apparell had April on the heel
Of limping winter treads, even such delight
Among fresh female buds shall you this night
Inherit at my house; hear all, all see,
And like her most whose merit most shall be:
Which on more view, of many mine being one
May stand in number, though in reckoning none,
Come, go with me.
Output:
Number of words: 77
The longest word: "reckoning" has 9 chars
The shortest word: "as" has 2 chars
Average length of words :4.36364 chars
```

```
Input :
The Gardens are bounded on one side by a never ending line of
omnibuses, over which your nurse has such authority that if
she holds up her finger to any one of them it stops
immediately. She then crosses with you in safety to the other
side. There are more gates to the Gardens than one gate, but
that is the one you go in at, and before you go in you speak
to the lady with the balloons, who sits just outside.
Output:
Number of words: 83 The longest word: "immediately." has 12
chars The shortest word: "a" has 1 chars Average length of
words :3.96386 chars
```

⌨ 3.2.5 Space and special characters

Write a code that will read a given text. Return the number of spaces, tabulators, newlines, commas, colons, semicolons and dots in the text.

```
Input :
Earth treading stars that make dark heaven light:
Such comfort as do lusty young men feel
When well apparell had April on the heel
Of limping winter treads, even such delight
Among fresh female buds shall you this night
Inherit at my house; hear all, all see,
And like her most whose merit most shall be:
Which on more view, of many mine being one
May stand in number, though in reckoning none,
Come: go with me.
Output:
Number of space and special characters:
Space: 70
Tab: 0
New line: 10
Dot: 1
Coma: 6
Colon: 3
Semicolon: 1
```

```
Input :
The Gardens are bounded on one side by a never ending line of
omnibuses, over which your nurse has such authority that if
she holds up her finger to any one of them it stops
```

```
immediately. She then crosses with you in safety to the other
side. There are more gates to the Gardens than one gate, but
that is the one you go in at, and before you go in you speak
to the lady with the balloons, who sits just outside.
Output:
Number of space and special characters:
Space: 83
Tab: 0
New line: 1
Dot: 3
Coma: 4
Colon: 0
Semicolon: 0
```

⌨ 3.2.6 Operator overloading - Date class

Write a code for the Date class with an object of three elements: unsigned short day, string month and unsigned short year. We assume that the values are given in this order when creating the object. Overload the input and output operators for this class so that the object output will be in the form year-day-month.

```
Input : 4 May 2010
14 April 1997
6 January 2021
Output:
2010-4-May
1997-14-April
2021-6-January
```

```
Input :  30 March 2019
18 September 2004
4 May 2010
14 April 1997
6 January 2021
Output: 2019-30-March
2004-18-September
2010-4-May
1997-14-April
2021-6-January
```

⌨ 3.2.7 Rows – statistics

Write a code that will read a given text. Return the number of all rows and the longest and the shortest row in the text.

```
Input :
Earth treading stars that make dark heaven light:
Such comfort as do lusty young men feel
When well apparell had April on the heel
Of limping winter treads, even such delight
Among fresh female buds shall you this night
Inherit at my house; hear all, all see,
And like her most whose merit most shall be:
Which on more view, of many mine being one
May stand in number, though in reckoning none,
Come, go with me.
Output:
Number of rows: 10
The longest row has 50 chars
The shortest word has 18 chars
```

```
Input :
The Gardens are bounded on one side by a never
ending line of omnibuses, over which
your nurse has such authority that if she holds
up her finger to any one of them it stops immediately.
She then crosses with you in safety to the other side.
There are more gates to the Gardens than one gate,
but that is the one you go in at, and before you go in
you speak to the lady with the balloons, who sits just
outside.
Output: Number of rows: 8
The longest row has 65 chars
The shortest word has 37 chars
```

⌨ 3.2.8 Different systems

Write a code that will read three numbers. The first one defines the system (a system with a base 8, 10 or 16), the second determines how many characters should go to the standard output (the missing digits should be filled in *) the third is the number in a decimal system that we want to write in the given format.

```
Input : 8 12 12584
Output: *******30450
```

```
Input : 16 8 87592
Output: ***15628
```

3.2.9 Number of decimal places

Write a code that will read a number and will display it with the given number of decimal places.

```
Input : 2112.15121566 5
Output: 2112.15122
```

```
Input : 1212.151216 10
Output: 1212.1512160000
```

Preprocessor – Directives and Macros

Chapter 4

4.1 Directives and macros

📖 4.1.1

Directives and macros are used to change the text of a program before it is recognized by the proper compiler. At the beginning of the line containing the directive, the sign # should be inserted. At the end of the line containing the directive, we do not put the sign ; . Directives may be placed on any line of code but they will be applied to part of the code located below it.

📖 4.1.2

The two basic commands to define a macro are #define and #undef.

The #define directive is used to exchange one string in the code with a different string. So if there is a line in the code

```
#define NAME new value
```

then every occurrence of NAME in the rest of the code will be replaced by the "new value".

The #undef directive is used to delete a macro definition. So if there is a line in the code

```
#undef NAME
```

the macro NAME will be deleted from the rest of the code.

📖 4.1.3

It is a good practice to write macro names in capital letters. Be careful when using macros. Macro names can not be overloaded, nor to apply recursion to them.

📖 4.1.4

An example of the use of #define and #undef directives is given below

```
#include <iostream>
#define SIZE 20
#define BEGIN {
```

```
#define END }
int main()
BEGIN
  int tab[SIZE];
  for(int i=0;i<20; i++)
  tab[i]=i;
  #undef SIZE
  #define SIZE 10
  int tab1[SIZE];
  for(int i=9; i>=0; i--)
  tab1[9-i]=i;
  return 0;
END
```

4.1.5

At the beginning of the line containing the directive, the sign _____ should be inserted.

- #
- @
- \\
- $

4.1.6

Directives may be placed on any line of code.

- True
- False

4.1.7

What effect will the macro have #define X y?

- Any occurrence of the X token in the code below the macro will be replaced by y.
- Any occurrence of the X token in the code will be replaced by y.
- Defining a new variable X, which takes the value of variable y.
- Defining a new variable y, which takes the value of variable X.

4.1.8

Each occurrence of the SIZE token below the following macros

```
#define SIZE 200
#undef SIZE
#define SIZE 50
```

will be replaced by the value of

- 50
- 200
- 150
- 250

4.1.9

At the end of the line containing the directive, we put the sign ;.

- False
- True

4.2 Directives and macros (programs)

4.2.1 Compiled_in_CPP

Define the necessary macros to make the program fill the array so that it contains cubes of natural numbers (starting from 0 to the value defined by the user).

```
Input : 1
Output: 0 1
```

```
Input : 5
Output: 0 1 8 27 64 125
```

Compiled_in_CPP.cpp

```
#include <|iostream>
using namespace std;
//Add the code here

int main()
BEGIN
```

```
int value;
cin>>value;
//Add the code here

char tab[SIZE];
FOR(int i=0;i<|=SIZE; i++)
BEGIN
tab[i]=i*i*i;
cout<|<|tab[i]<|<|" ";
END
return 0;
END
```

4.2.2 Different size of the table

Write a code that using the macro SIZE defines three different tables with a different length defined by the user. The first array should contain subsequent natural numbers starting from 0. In the second array, the products of two consecutive natural numbers should be saved starting from 0. In the third array, the products of three consecutive natural numbers should be saved starting from 0. Display the contents of arrays on the screen.

```
Input : 2 6 8
Output: 0 1
0 2 6 12 20 30
0 6 24 60 120 210 336 504
```

```
Input : 5 1 9
Output: 0 1 2 3 4
0
0 6 24 60 120 210 336 504 720
```

Different_size.cpp

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

int main()
{
int value1,value2,value3;
cin>>value1;
cin>>value2;
cin>>value3;
//Add the code here
```

```
int tab1[SIZE];
//Add the code here

int tab2[SIZE];
//Add the code here

int tab3[SIZE];
//Add the code here

return 0;
}
```

4.2.3 Area and perimeter of the circle

Write code that will calculate the area and perimeter of a circle with a precision of 8 decimal numbers. Define a macro PI that will have PI values with an accuracy of 8 decimal numbers.

```
Input : 3
Output: Area of the circle: 28.27433385 Perimeter of the
circle: 18.84955590
```

```
Input : 4
Output: Area of the circle: 50.26548240 Perimeter of the
circle: 25.13274120
```

4.2.4 Guess the number

Write a code that allows the user to guess a natural number less than or equal to 100 defined as a macro (equal to 76). If the user specifies the too big or too small number, please write it. If the user guesses the number, please write "Congratulations, you guessed the number".

```
Input : 5
Output: Too small
```

```
Input : 16
Output: Too small
```

4.2.5 Calculate an operation

Write a code that will allow counting the sum, difference, product and division of two real numbers. Operation is distinguished by providing values from 1 to 4: 1 for

sum, 2 for difference, 3 for a product, 4 for division. This should be defined using macros. The user first gives the number corresponding to the type of operation and then the two real numbers.

```
Input : 1 2.6 5.8
Output: 8.4
```

```
Input : 2 1.51 2.812
Output: -1.302
```

4.3 Macro with parameters

🕮 4.3.1

Using the defined directive, it is also possible to define a macro with parameters. This macro works like a function, of course, it does not recognize types.

The following macro

```
#define GETMIN(a,b) ((a)>(b)?(b):(a))
```

will return a minimum of two numbers. So for example

```
int x=4, y=8;
cout<<GETMIN(x,y);
```

will display 4 on the screen.

🕮 4.3.2

It should be remembered that such a macro can be problematic because it works on the basis of character substitution. For example, the following macro

```
#define MULTIPLICATION(a) a*2
```

with the application

```
int x=4;
cout<<MULTIPLICATION(x+3);
```

will display on screen 10, not 14. This will happen because after substitution we get x + 3 * 2.

📖 4.3.3

Therefore, the first multiplication will be made. To avoid such errors, it is good to include macro arguments in parentheses. So the macro

```
#define MULTIPLICATION(a) ((a)*2)
```

with the application

```
int x=4;
cout<<MULTIPLICATION(x+3);
```

will display on screen 14.

📖 4.3.4

When creating a macro, operators # and ## are often used.

If we use the # operator before the parameter name, the string containing the name of the parameter will be returned, not its value.

For example the macro

```
#define name(a) cout<<"The parameter name is "<<#a
```

and the line of code

```
name(Piter);
```

will display on the screen The parameter name is Piter.

📖 4.3.5

If we use the ## operator between two parameters, it will glue two strings.

For example the macro

```
#define name(a,b) cout<<"The parameter name is "<<a##b
```

and the line of code

```
name(de,Morgan);
```

will display on the screen The parameter name is deMorgan.

4.3.6

The following macro

```
#define TEST1(a) 4*a+1
```

with the application

```
int x=2;
cout<< TEST1(x+5);
```

will display on the screen

- 14
- 4*2+5+1
- 29
- 32

4.3.7

The following macro

```
#define TEST2(a) 4*(a)+1
```

with the application

```
int x=2;
cout<< TEST2(x+5);
```

will display on the screen

- 29
- 12
- 14
- 32

4.3.8

If in the macro definition we use the # operator before the parameter name, we get _____.

- a string containing the parameter value
- the parameter name

4.3.9

If we use the ## operator between two parameters in the macro definition, it will result in _____.

- glueing two strings
- calculating the symmetrical difference
- creating one string by inserting a space between the two strings

4.3.10

Parameters in the macro definition are enclosed by parentheses _____ and separated by a sign _____.

- []
- {}
- #
- &
- ()
- ;
- ,

4.4 Macro with parameters (programs)

4.4.1 Distance

Write a macro that will calculate the distance that the animal will pass, which runs with the predefined speed given in m/s for a given number of minutes. Display the result in kilometres.

```
Input : 1.4 2
Output: 0.168
```

```
Input : 6 10
Output: 3.6
```

4.4.2 Trapezium field

Define the necessary macro with parameters in order to calculate a trapezium field. Then use this macro for the parameters' values defined by the user. Display results on the screen. If one of the defined values is less than or equal to zero, display the information "Wrong number".

```
Input : 5 2 3
Output: 10.5
```

```
Input : 5 2 2
Output: 7
```

4.4.3 Triangle field

Define the necessary macro with parameters in order to calculate a triangle field. Then use this macro for the parameters' values defined by the user. Display results on the screen. If one of the defined values is less than or equal to zero, display the information "Wrong number".

```
Input : 2 1
Output: 1
```

```
Input : 3 7
Output: 7.5
```

4.4.4 Interval

Write a macro to see if the given number is from a given interval. Ensure that the upper and lower bounds of the interval are correctly entered. The first number given is always the searched one and then are the interval boundaries followed.

```
Input : 10 2 13
Output: true
```

```
Input : 5 15 7
Output: false
```

⌨ 4.4.5 Time of a day

Write a code that will for a given hour (1-12) and period (a.m., p.m.) decide whether it is a day or a night (let's assume the sun is coming out and sets down at 6:00).

```
Input : 5 a.m.
Output: night
```

```
Input : 3 p.m.
Output: day
```

4.5 Predefined macros

🕮 4.5.1

We will now discuss a group of macros that are predefined by the compiler and always available for use. These macros are listed below

- *__cplusplus* - Defined in C ++ build (but not in C). It can take different values depending on the C ++ standard. It adopts the value 199711 for ISO C ++ 1998/2003, the value 201103 for ISO C ++ 2011 and the value 201402 for ISO C ++ 2014.
- *__DATE__* - Date in format Mmm dd yyyy.
- *__TIME__* - Time in hh:mm:ss format.
- *__FILE__* - The name of the current file that is compiled.
- *__LINE__* - The current line number of the code in the file.
- *__STDC_HOSTED__* - The value 1, if the implementation is hosted, i.e. with all available standard headers; 0 otherwise.

🕮 4.5.2

There is also a group of macros that can be defined depending on whether the function is available. These macros are listed below

- *__STDC__* - In compilers compatible with ANSI or later this macro has the value 1.
- *__STDC_MB_MIGHT_NEQ_WC__* - It has the value 1 if the wchar_t encoding can give the character a different value than the value of this element as a regular literal.
- *__STDCPP_STRICT_POINTER_SAFETY__* - It is equal to 1 if the implementation has strict indicator security.
- *__STDCPP_THREADS__* - It has a value of 1 if the program can have several threads.

The predefined identifier *__func__* is also available, but it is not a macro. This identifier returns the name of the function currently being executed.

Compilers before C99 may not have the *__func__* identifier available or may have another one available but working in a similar way, e.g. *__FUNCTION__* for C89.

There are also other predefined macros specific for some compilers, such as Microsoft compilers, but this topic is beyond the scope of this course.

📖 4.5.3

Below, is an example of the use of predefined macros.

```
#include <iostream>
using namespace std;
int main(){
                cout<<__DATE__<<endl;
                cout<<__TIME__<<endl;
                cout<<__FILE__<<endl;
                cout<<__LINE__<<endl;
                cout<<__STDC_HOSTED__;
 return 0;
}
```

Sample result on the screen:

Apr 1 2019

15:36:10

C:\Users\Project\C\Macra.cpp

7

1

4.5.4

The predefined macro *__DATE__* gives the date in the format

- Mmm dd yyyy
- dd:mm:yy
- dd:mm:yyyy

- mm dd yy

4.5.5

Choose descriptions

__DATE__ - _____

__TIME__ - _____

__FILE__ - _____

- The name of the current file that is compiled.
- The name of the current file that is compiled.
- Date in format Mmm dd yyyy.
- The current line number of the code in the file.
- Time in hh:mm:ss format.
- Date in format Mmm dd yyyy.
- Time in hh:mm:ss format.
- Time in hh:mm:ss format.

4.5.6

The macro *__STDC_HOSTED__*

- It has the value of 1 if the implementation is hosted.
- It has the value of 1 if the program can have several threads.
- It has the value of 1 if the implementation has strict indicator security.
- It can take different values depending on the C ++ standard.

4.5.7

The macro *__cplusplus*

- It can take different values depending on the C ++ standard.
- It always has the value of 201103.
- It has the value of 1 if the program is written in C++.
- It has the value of 0 if the program is written in C.

4.5.8

The macro *__STDCPP_STRICT_POINTER_SAFETY__*

- It has the value of 1 if the implementation has strict indicator security.
- It has the value pf 1 if the implementation is hosted.
- It has the value of 1 if the program can have several threads.
- It can take different values depending on the C++ standard.

4.6 Predefined macros (programs)

4.6.1 Basic information

Write a program that, depending on the value of the natural number given by the user, writes the following information on the screen:

1: the name of the file

2: the available C ++ standard

3: the line number

4: information if the implementation is hosted

for other values, the program should write "You gave the wrong value".

```
Input : 1
Output: Predefined_macros.cpp
```

```
Input : 2
Output: 199711
```

4.6.2 Debug information

Write a function named division that displays on the screen the result of dividing two numbers. If an error occurs, it displays information about the file name and the name of the function in which the error occurred.

```
Input : 4.2 2.1
Output: 2
```

```
Input : 5 0
Output: Error in Debug_information.cpp division
```

4.6.3 Is compiler compatible with ANSI or newer standard?

Write a program with an external function that takes a certain double value as a parameter. At first, the function checks if the compiler is compatible with ANSI or newer standard and then displays the parameter on the screen with the accuracy of 4 decimal numbers.

```
Input : 2.12345684
Output: This compiler is compatible with ANSI or newer
standard 2.1235
```

```
Input : 52216.125478
Output: This compiler is compatible with ANSI or newer
standard 52216.1255
```

4.6.4 Error message

Write a program that calculates the multiples of the natural number given by the user, which is smaller than the square of this number. Multiples are saved into a table with size SIZE. If the table is full, an error message is printed. The message contains the file name and the reason for the error and is defined by the macro ERROR(reason).

```
Input : 5
Output: 5 10 15 20
```

```
Input : 15
Output: 15 30 45 60 75 90 105 120 135 150 ERROR in file:
Error_message.cpp Reason: The array size is too small
```

4.6.5 Which function

Define three external functions: sum, difference and product. The user gives the number 1 if she/he wants to count the sum of two natural numbers, 2 for the difference and 3 for the product. Then gives two integers for which the operation should be performed. The program informs the user what function was used and gives the result of the operation. If the first of the numbers given by the user is different from 1, 2 and 3, the program displays the message "Enter the number 1, 2 or 3".

```
Input : 1 4 6
Output: I use the function sum 10
```

```
Input : 2 1 15
Output: I use the function difference -14
```

Conditional Compilation

Chapter 5

5.1 Conditional compilation

🕮 5.1.1

In order to define a conditional compilation, the following directives are used: #ifdef, #ifndef, #if, #endif, #else and #elif.

The #ifdef and #ifndef directives will be discussed first. They are used to exclude or include part of the code depending on whether the macro specified as a parameter has been defined (for #ifdef) or not (#ifndef).

For example, consider two fragments of code:

```
#ifdef DISCOUNT
cout<<"Your discount is"<< DISCOUNT;
#endif
```

The discount amount will be displayed on the screen, provided it has been defined. If the discount has not been defined, this line of code will be omitted.

```
#ifndef DISCOUNT
#define DISCOUNT 0.2
#endif
```

The above fragment defines a discount of 20%, provided it has not been defined before. If the discount was defined earlier, it leaves its value as it was.

🕮 5.1.2

Now the directives #if, #else and #elif will be discussed. They are used to determine which fragment of code should be compiled depending on certain conditions. To define a condition, we can use constants and macros.

For example, consider the following fragment of code:

```
#if DISCOUNT>0.5
cout<<"Your discount is large";
#elif DISCOUNT<0.1
#undef DISCOUNT
#define DISCOUNT 0.3
cout<<"Your discount has been increased";
#else
cout<<"Your discount has remained the same as it was";
#endif
```

The above fragment of the code for a discount bigger than 50% displays information on the screen that it is large. The Discount of less than 10% is increased to 30%. The discount between 10% and 50% is left as it was.

5.1.3

In order to some fragment of code will be executed on the condition that the macro TEST is not defined, we will use

- #ifndef TEST
- #ifdef TEST
- #undef TEST
- #define TEST

5.1.4

We have the following code:

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

#define VALUE 5
#if VALUE<2
#undef VALUE
#define VALUE 3
#elif VALUE==3
cout<<"Hello";
#else
#undef VALUE1
#define VALUE1 6
#endif

int main(){
cout<<VALUE;
return 0;
}
```

What is the value that will be displayed on the screen.

- 5
- 6
- 3

- 2

5.1.5

Complete the fragment of code so that the macro LEGS will be equal to 4 provided it has not been defined before.

```
_____ LEGS
_____ LEGS 4
_____
```

- #end
- #else
- #ifndef
- #define
- #if
- #endif

5.1.6

Select the incorrect fragments of code

A

```
#if VALUE>0
#define VALUE 0
```

B

```
#if VALUE>0
#undef VALUE
#define VALUE 0
```

C

```
#if VALUE>0
#define VALUE 0
#endif
```

D

```
#if VALUE>0
cout<<"Value is 0";
```

```
#endif
```

- A
- B
- C
- D

5.1.7

At the end of the if statement of conditional compilation we will use

- #endif
- }
- endif
- #end

5.2 Conditional compilation (programs)

5.2.1 Version of C

The __cplusplus macro is defined for C ++ and adopts the following values depending on the standard: __cplusplus = 199711 for C++98, __cplusplus = 201103 for C++11, __cplusplus = 201402 for C++14. Using a conditional compilation, write the code which for the year provided by the user gives information whether the standard in which the program is created is younger, older or exactly from the given year.

Use phrases: "The C ++ standard is younger than the year you specified.", "The C ++ standard is older than the year you specified.", "This is the year of your C ++ standard.". If the program is not written in C ++, display the information "This is not C++".

```
Input : 1994
Output: The C ++ standard is younger than the year you
specified.
```

```
Input : 2005
Output: The C ++ standard is older than the year you
specified.
```

⌨ 5.2.2 Not always available predefined macros

Write a program that, depending on the value of the natural number given by the user, writes the following information on the screen:

1: whether the program is written in C standard compatible with the ANSI or newer

2: is multithreading allowed or the macro specifying multithreading is not defined

3: whether indicator security is required or the macro specifying indicator security is not defined

4: whether wchar_t encoding may differ from the standard or the macro specifying wchar_t encoding is not defined

for other values, the program should write "Enter a value from 1 to 4".

```
Input : 1
Output: C compilation compatible with the ANSI or newer
standard.
```

```
Input : 2
Output: Multithreading is allowed.
```

⌨ 5.2.3 Current line number

Write a code with eight functions named from F1 to F8. They are executed incrementally starting from the value selected by the user. When the function is executed, the text "Current line is: " and some numbers should be displayed on the screen. To do this, define a macro named CURRENT_LINE that displays the correct line number for standards C++11 and C++14. The standard C++98 displays the correct line number plus 10. The C standard displays the correct line number minus 10.

```
Input : 8
Output: Current line is: 60
```

```
Input : 4
Output: Current line is: 44
Current line is: 48
Current line is: 52
Current line is: 56
Current line is: 60
```

Which_line.cpp

```
#include<|iostream>
using namespace std;
//Add the code here
void F1()
  //It should be in line 20
{     CURRENT_LINE; }

void F2() {
  CURRENT_LINE; }
void F3() {
  CURRENT_LINE; }
void F4() {
  CURRENT_LINE; }
void F5() {
  CURRENT_LINE; }
void F6() {
  CURRENT_LINE; }
void F7() {
  CURRENT_LINE; }
void F8() {
  CURRENT_LINE; }
int main(){
  int temp;
  cin>>temp;
  //Add the code here
  return 0;
}
```

5.2.4 Access code

The access code, which is the number from 1 to 1000, has been defined as a macro. Define the GUIDE macro, which tells you whether the access code is in the first quarter (it is a number from 1 to 250), the second quarter (it is a number from 250 to 500), the third or fourth quarter. A code that checks if the user has guessed the code is given. If so, it writes "Congratulations, you broke the code", if not, it gives the user a guide.

```
Input : 125
Output: Third quarter.
```

```
Input : 567
Output: Congratulations, you broke the code.
```

Access_code.cpp

```
#include<|iostream>
using namespace std;
#define ACCESS_CODE 567
//Add the code here

int main(){
int temp;
cin>>temp;
if(temp==ACCESS_CODE)
{
   cout<|<|"Congratulations, you broke the code";
   return 0;
}
    GUIDE;
  return 0;
}
```

⌨ 5.2.5 Average height

The average height is 180 cm. Using a conditional compilation, write a program that calculates the difference between the given height (h) and the average height (AVGHEIGHT). For people above the average display "You are ? cm higher than the average.", for others display "You are ? cm lower than the average." If the macro AVGHEIGHT is not defined, the program should display "The average height has not been defined.".

```
Input : 200
Output: You are 20 cm higher than the average.
```

```
Input : 160
Output: You are 20 cm lower than the average.
```

Macro assert()

Chapter 6

6.1 Macro assert()

🕮 6.1.1

Assert() is a macro that validates the operation of the program.

Within *Assert()* the expression is checked, if its value is true (1) then the program will continue to run if the result in assert will be false (0), the program will be terminated and the user will receive information about where the error occurred (in which place the error occurred).

To be able to use the assert macro, you need to import:

```
#include <assert.h>
```

An example of simple use of the *assert()* macro:

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

int main()
{
    int a;
    cin >> a;
    assert(a!=0);
    cout << a;
    return 0;
}
```

In the case when the value of variable *a!=0* as a result of the program run, we will get the value an on the console. For the value of *a==0*, the program will be terminated, with a message on the console:

```
Assertion failed: a!=0, line 9
```

The macro *assert()* can be used wherever we can expect errors in the operation of the program. The macro *assert()* allows us to catch errors before the program will be able to damage anything.

However, remember to get rid of expressions with the macro *assert()* in the final version.

6.1.2

To use the *assert()* macro it is necessary to import:

- #include <|assert.h>
- #include <|assert>
- #include <|asert.h>
- #define <|assert.h>

6.1.3

Will the following program be terminated by calling the *assert()* macro:

```
#include <iostream>
#include <assert.h>
using namespace std;
int main()
{
    int val_1 = 1;
    int val_2 = 0;
    assert(val_1 == val_2);
    cout << val_1;
    return 0;
}
```

- True
- False

6.1.4

In the final version of the program, you must get rid of the *assert()* macro. You can do this by defining the *NDEBUG* macro

The sample code with the *assert()* macro disabled will look like this:

```
#define NDEBUG
#include <assert.h>

int main()

{
    int a=1;
```

```
    assert(a==0);
    return 0;
}
```

As a result of the program, the line with the *assert()* command will be omitted, this will happen by defining *NDEBUG*.

📖 6.1.5

The *assert()* macro should not be used for expressions that affect the rest of the program, such as changing the value of a variable. It should be remembered that in the final version of the program the code inside the *assert()* macro will not be executed.

Such *assert()* usage are incorrect:

```
assert(a= 4);
assert(b= a+1);
assert(c++);
```

however, such uses are correct:

```
assert(a > 0);
assert(a == b);
assert(b);
```

📝 6.1.6

Which of the following assert macros are valid (according to the rules of using *assert()* macro)

- assert(val == 4);
- assert(val = 4);
- assert(fopen("file.txt","r"))
- assert(sum(a,b) == 0);

📝 6.1.7

In the final version of the program, you can get rid of the *assert()* macro by defining a macro:

- NDEBUG

- NASSERT
- FINALL
- DASSERT

6.2 Macro assert() (programs)

6.2.1 Division by zero

Write a function that returns the result of dividing (total) of two integers received as function parameters. The program should be terminated using the *assert()* macro if the second of the given numbers is equal to 0.

```
Input : 4 2
Output: 2
```

```
Input : 6 0
Output: Assertion failed: ..., line ...
```

Compilation and Linking

Chapter 7

7.1 Compilation and linking, make

📖 7.1.1

Very often, the concept of code compilation is the translation of code written by a programmer into a machine and creating a file that can be run on a computer. In fact, the build process is just one of the processes that the program is prepared to run.

For C / C ++, the most popular compiler is GCC. It is a set of compilers for various programming languages developed under the GNU project and made available under the GPL and LGPL licenses. GCC is just an interface, a controller. GCC (as a package) consists of many programs, GCC is one of them, it easily manages the entire process of building a program from beginning to end. Each build phase has its own application.

In addition to compiling from the IDE environment, we can also compile our programs from the command line. The easiest way to do this is to invoke the command:

```
gcc program_name.c
```

or

```
gcc program_name.cpp
```

Stages of program built:

- Preprocessing
- Compilation
- Optimization
- Assembling
- Linking
- Make

📝 7.1.2

The first stage of the program built is:

- Preprocessing
- Compilation
- Cleaning
- Optimization

7.1.3

To compile a program written in C++ from the command line use the command:

- gcc
- Gcc
- cpp
- gpp

7.1.4

Preprocessing

The first stage of building the program is the stage called preprocessing (or precompilation).

This stage is responsible for creating the final program code. At this stage, files declared with the #include and #define directives, as well as definitions and macros, are attached. Instead of calling the directive, the code and its contents will simply be pasted.

Blank lines and comments are deleted

The precompilation process itself can be performed by running the command from the command line:

```
gcc –E filename.cpp
    or
gcc -E filename.c
```

7.1.5

Compilation

At this stage, the type code that has been precompiled is converted into assembler code. In this stage, errors in the source code are also caught and reported to the user. An assembler file with the extension .s is created. At this stage, elements such as loops are removed and descended with the usual conditions and goto instructions. In addition, all constant values are inserted into the code. During compilation, the code is also adapted to the selected architecture and environment, i.e. whether it is to be 32 or 64-bit machines, and specific processor technologies

To stop the build process at this level, run the command at the command prompt:

```
gcc -S filename.cpp
    or
gcc -S filename.c
```

📖 7.1.6

Optimization

At this stage, the assembly code is optimized to improve efficiency. Improving efficiency is aimed at improving performance and reducing the size of the program. In this stage, unused code fragments are eliminated, the allocation of registers is optimized, and the process of calculating relative addresses is improved.

📖 7.1.7

Linking

This stage consists of three activities. The first is to search for system libraries or those indicated by the programmer that has not been defined in the source files. Simply put, as a result of linking you combine all intermediate files into one program. In the next stage, the machine code is assigned to the address set. The last step is to create an ELF (Executable and Linking File) binary executable file

📖 7.1.8

Make

A system shell program that automates the process of compiling programs that consists of many dependent files. The program processes the Makefile rules file and on this basis determines which source files require compilation. This saves a lot of time when creating the program because as a result of changing the source file only those files that are dependent on this file are compiled. Thanks to this, there is no need to compile the entire project. Make is also suitable for other work that requires the processing of multiple files that depend on each other.

The make program call has the following form:

```
make [options] [macrodefinitions] [-f control_file] [target]
```

In the simplest case, the command line only contains the word make. The program then tries to read commands from the control file with the standard name: makefile,

Makefile or MakeFile. If there is no file with this name in the current directory, an error is reported. If the control file has a different name, use the following call:

```
make -f control_file_name
```

Makefile Rules file for make. Describes the relationship between program source files. This allows you to process only those files that have changed since the last build and the files that depend on them. This significantly reduces the time needed to generate the resulting file. The file format varies depending on the implementation of the make program, but the basic rules are the same for all variations of make.

7.1.9

Translation of assembler code into machine code takes place in the stage:

- Assembling
- Linking
- Compilation
- Preprocessing

7.1.10

To perform the *preprocessing* step itself from the command line, use the command

```
gcc -E filename.cpp
```

- True
- False

7.1.11

Blank lines and comments are removed from the code at the stage of:

- Preprocessing
- Compilation
- Assembling
- Linking

7.2 Static and dynamic libraries

📖 7.2.1

Libraries are files containing code that can be used by other programs. By using libraries, programs become more modular. The use of libraries also allows for easier changes and corrections.

An example of libraries is, for example, standard C language libraries. In standard libraries, we have many different functions such as *printf()* etc. Through the use of standard libraries do not need to write part of the code singly, but we can use effective solutions delivered with the language.

In addition to standard libraries, the programmer can also create his own libraries.

The basic division of libraries is:

- Static libraries - on Windows systems have the extension **.lib** or **.obj**., On Unix systems **.a** or **.o**
- Dynamic libraries, on Windows systems **.dll**, on Unix systems **.so**

Examples of standard C language libraries:

- #include <cstdlib>
- #include <cstdio>
- #include <cstring>

Examples of standard C++ language libraries:

- #include <string>
- #include <new>
- #include <vector>

📝 7.2.2

Static libraries on Windows systems have an extension

- .lib
- .a
- .dll
- .cpp

7.2.3

Dynamic libraries on Unix systems have an extension

- .so
- .obj
- .dll
- .a

7.2.4

Library *<cstring>* is a standard language library

- C
- C++

7.2.5

Static libraries

Static libraries are attached to the program at the compilation stage. In order to use the static library code, information about the library header file that we want to use should be placed in the code of our program. In the next step, such a library (actually a code file) should be attached at the stage of compiling the code. The content of the library is usually located in a binary file, attached during the compilation of the code.

Load static libraries via the #include directive, for example

```
#include <cmath>
```

7.2.6

In order to attach a static string library to the program, the following directive should be added to the program

- #include <|string>
- #include "string"
- include <|string>
- include "string"

7.2.7

The content of the static library is attached to the program code at the compilation stage.

- True
- False

7.2.8

Dynamic libraries

Dynamic libraries in terms of construction are no different from static libraries, their content is also located in binary files. This means that if you are preparing your own library, you must compile it into a binary file.

The difference is that dynamic libraries do not have to be loaded when the program is started. Dynamic libraries can be loaded into the program at any time when the program is needed. This requires that you write the program properly to handle library load requests. Dynamic library loading allows creating plugins for the program.

Dynamic libraries are loaded using the API called dl (dynamic loading). This interface, which is also a library, contains functions for loading, searching and removing from the memory of libraries. You can load the library by inserting the #include <dlfnc.h> header file into your code

The dl interface provides four functions for working with shared objects. They are *dlopen*, *dlclose*, *dlsym*, *dlerror*.

An example of loading a dynamic library:

```
Module = dlopen ("library.so", RTLD_LAZY)
```

The first parameter of the function is the name of the binary file with the library, the second is the so-called flag.

Sample flags:

- **RTLD_LAZY -** Perform lazy binding. Only resolve symbols as the code that references them is executed.

- **RTLD_NOW -** If this value is specified, or the environment variable **LD_BIND_NOW** is set to a non-empty string, all undefined symbols in the library are resolved before **dlopen**() returns.
- **RTLD_GLOBAL -** The symbols defined by this library will be made available for symbol resolution of subsequently loaded libraries.

7.2.9

Dynamic libraries can be loaded into the program at any time during its operation.

- True
- False

7.3 Libraries (programs)

7.3.1 Add library

Import the *math.h* and *iostream* library into the code. So that the program will compile and run.

Additional Topics

Chapter 8

8.1 Bit operators, bit arrays

🕮 8.1.1

Bitwise operators operate on integer bits. For example, the left shift operator shifts bits to the left, and the bit negation operator changes each bit with a value of 1 to 0, and each bit with a value of 0 to 1. In C, there are six-bit operators <<, >>, ~, &, | , ^.

🕮 8.1.2

The shift operator

The left shift operator has the following syntax:

Value << offset

The value field indicates the integer whose bits are to be shifted, and the shift field specifies the number of shift bits.

```
11 << 2
```

Means shifting all 11 value bits 2 positions to the left. Abandoned places are filled with zeros and bits shifted outside the given value are skipped.

Moving one position to the left corresponds to multiplying the value by 2.

So the value of expressing 11 << 2 will be 44.

If you want to change the value of a variable using the offset operator, use the assignment operator:

```
x = x << 3
x <<= 3
```

The right shift operator works in a similar way to shifting the bits to the right. Syntax:

Value >> offset

For example, 14 >> 3 means that all 14 value bits are shifted by 3 places. In the case of unsigned values, the vacated places are filled with zeros, in the case of signed values, the vacated places can be filled with zeros or 1 depending on the character.

Right shift assignment operators:

```
x = x >> 3
x >>= 3
```

8.1.3

Select the right shift operator

- >>
- <|<|

8.1.4

Upon execution of the shift operation given below, the value of variable *x* will be:

```
unsigned char x = 5;
x <<= 3;
```

8.1.5

Are the following offset operations the same?

```
x <<= 2
x = x << 2
```

- True
- False

8.1.6

Bit logical operators

Bitwise logical operators are equivalent to ordinary logical operators, except that they apply to bit-level values, not to integer values.

The **bit negation operator** (~) replaces each bit with its opposite (turns 1 into 0 and 0 into 1).

For example

```
unsigned char x = 3
x = ~ x;
```

We get 252, 3 in a bit write 00000011, after converting each bit to the opposite value we get 11111100, which gives a decimal value of 252.

The bit operator **OR** (|) (bit alternative) results in an integer value that is a combination of two other integer values. Each bit of the new value for which one or both bits have the value 1 is set to 1, if both corresponding bits have the value 0, then the result bit is set to 0.

For example:

```
unsigned char x = 3;
unsigned char y = 4;
unsigned char with;
z = x | y;
```

For the given example, the value z will be 7, because 3 in the bit write is 00000011, 4 in the bit write is 00000100. In the resulting number, the bits for which the input values appeared in at least one case 1 supplement 1, the remaining 0, hence the result actions in bit form are 00000111, i.e. 7.

The bit operator **XOR** (^) (bit symmetric difference) results in a value where each bit for which one (but not both) of the corresponding bits of the original value is equal to 1, is set to 1. If both corresponding bits are equal to 0 or both are equal to 1, the result bit is set to 0.

For example:

```
unsigned char x = 5;
unsigned char y = 4;
unsigned char with;
z = x ^ y;
```

For the given example, the value of z will be 1, because 5 in the bit write is 00000101 and 4 is 00000100, after performing the XOR operation we get in the bit write 00000001, i.e. 1.

The bit operator **AND** (&) (bit conjunction) gives the result a value for which each bit for which both corresponding bits of the original values are equal to 1 is set to 1, the others to 0.

For example:

```
unsigned char x = 5;
unsigned char y = 4;
unsigned char z;
z = x & y;
```

The result of the code from the example *z* will be 4.

8.1.7

To perform the XOR bit operation we will use the operator ^

- True
- False

8.1.8

Bit operation *a* | *b* means

- OR
- AND
- XOR

8.1.9

As a result of the code below, the variable *x* will take its value:

```
int main()
{
    unsigned char x = 11;
    x = ~x;

    return 0;
}
```

8.2 Bits (programs)

8.2.1 Bit shift

Write a bit shifting program to the left. The number to shift and the shift value is to be entered into the program from the keyboard.

```
Input : 13 3
Output: 104
```

```
Input : 8 1
Output: 16
```

⌨ 8.2.2 Bit alternative

Write a program that performs an OR bit operation on two numbers entered from the keyboard.

```
Input : 10 1
Output: 11
```

```
Input : 8 5
Output: 13
```

8.3 Variadic functions

🕮 8.3.1

Functions with a variable number of arguments (**variadic functions**)

There are sometimes cases when at the stage of creating a function we do not know the exact number of arguments we would like to pass to it.

The second option is that you want to create a function to which you can send a different number of arguments. For example, we want to set a maximum value for 2, 4, 10 numbers, or display the values of a certain number of arguments.

We declare variadic functions with an ellipsis (...) in the place of parameters, we put all mandatory parameters before the ellipses, for example:

```
int fun (int * i, ...)
```

It declares a function that can be called with the mandatory first argument of type int * and any number of additional arguments.

An example of a built-in function is *printf*, which can take a different number of arguments.

```
int printf ( const char * format, ... );
```

For example

```
printf ("Characters: %c %c \n", 'a', 65);
```

```
printf ("Decimals: %d %ld\n", 1977, 650000L);
printf ("Preceding with blanks: %10d \n", 1977);
```

8.3.2

The correct variadic function declaration will look like this:

- void sum(char a, ...)
- int sum(... int *)
- int sum(* ...)
- int sum(int ... *)

8.3.3

The *printf()* function can take a different number of arguments.

- True
- False

8.3.4

In C++, it is possible to define functions that do not have a predetermined number of input arguments. To be able to use such functions, the **cstdarg** library must be imported into the code.

To access the arguments of a function from within a function, use the macros:

- *va_start* - enables access to variadic function arguments
- *va_arg* - accesses the next variadic function argument
- *va_copy* - makes a copy of the variadic function arguments
- *va_end* - ends traversal of the variadic function arguments
- *va_list* - holds the information needed by *va_start, va_arg, va_end,* and *va_copy*

Example of variadic function summing arguments given to the function:

```
#include <cstdarg>
int sum_of(int number, ...){
    int sum = 0;
    va_list args;
    va_start(args, number);
```

```
    for (int i = 0; i < number; ++i)     {
        sum += va_arg(args, int);     }
    va_end(args);
    return sum;
}
```

Usage:

```
sum_of(1,2)  // Output 3
sum_of(2,4,6,7) // Output 19
```

Variadic functions may be useful because of their versatility, but the problem is the lack of control over the correctness of arguments passed to the function. The variadic function should be used with caution and only when necessary.

8.3.5

To use the variadic function, you must import a library

- <|cstdarg>
- <|variadic>
- <|cstdvar>
- <|cstd>

8.3.6

va_start - accesses the next variadic function argument

- False
- True

8.3.7

Which macro ends traversal of the variadic function arguments?

- va_end
- va_stop
- va_arg
- va_fin

8.4 Variadic functions (programs)

8.4.1 Average of numbers

Write a variadic function that calculates the average value of the numbers entered.

```
Input : 4 2
Output: 3
```

```
Input : 8 10
Output: 9
```

8.4.2 Connecting strings

Write a variadic function that accepts any number of strings, and as a result returns a joined string consisting of strings given as arguments, separated by a comma, omitting strings beginning with the letter -a-. The first argument of the function is the number of strings to be given in the function-s arguments. Strings are to be entered from the keyboard.

```
Input : "Text1" "Text2"
Output: "Text1,Text2"
```

```
Input : "Text1" "Atext2" "Text3"
Output: "Text1,Text3"
```

Namespaces

Chapter 9

9.1 Namespaces, cin, cout

🕮 9.1.1

The namespace is a feature of the C++ language that allows you to write large programs using code from different sources. We can imagine that we have two packages from different suppliers - Comp_A and Comp_B, in both packages there is the sum() function. If we call the sum() function, the compiler will not know which function we want.

Let's assume now that both companies have placed their functions in namespaces with a name compatible with the name of the company, and in order to call the function provided by Comp_A we will call it as:

```
comp_A::sum();
```

similarly, to call the function provided by a second company we will use:

```
comp_B::sum();
```

The implementation of the namespace is intended to protect against name conflicts (namespace problem) of variables, classes, functions. The implementation of the C++ language of the namespace is intended for better control over the scope of names.

Also, classes, functions and variables that are standard components of the C++ compiler are placed in the namespace, for example, we use iostream.h, with the std namespace, in which we have available for example *std::cout, std::cin, std::endl* .

An example of displaying text using cout:

```
std::cout << "using the std namespace";
```

With frequent use of functions available within the namespace, it becomes troublesome to enter additional information each time from which namespace the given function, class, variable comes from. Therefore, the option of indicating that we will use a given space of naves has been introduced, using:

```
using namespace ...;
```

For example, if you want to use the **std** namespace, you can use:

```
using namespace std;
```

Then the screen display with cout will look like this:

```
cout << "using the std namespace";
```

9.1.2

Indicate correct function call from namespace

- namespace::function();
- namespace:function();
- namespace function();
- namespace..function();

9.1.3

The stream "cout" belongs to the stream namespace

- False
- True

9.1.4

Type the word that should appear in place of dots:

```
... namespace std;
```

9.1.5

In addition to the available namespaces, you can also define your own namespace. Thanks to this solution, names from one namespace do not conflict with identical names declared in other namespaces. The **namespace** keyword is used to create your own namespace.

An example of the declaration of two namespaces is shown below:

```
namespace Ex1{
            int var1;
            void sum(int *tab);
            double mean;
}
namespace Ex2{
            double var1;
            void sum(int *tab){
                ...
            }
            void  view();
```

```
}
```

Namespaces can be extended with an additional element, as well as prototyping functions in them for which the function code can be placed elsewhere in the file with the namespace declaration or even in another file.

For example, to place the code of function *sum(int * tab)*, simply refer to the namespace anywhere and place the function code there.

```
Namespace Ex1 {
        void sum (int * tab) {...}
}
```

Calling the function sum(int * tab) for the Ex1 namespace will look like this:

```
Ex1::sum(tab);
```

We can also indicate that we will use functions or variables from a given namespace through the keyword **using**.

```
int main () {
             using Ex1 :: var1;
             cin >> var1; // we refer to the variable from the
Ex1 namespace
}
```

9.1.6

During the namespace declaration, it is forbidden to use function prototypes.

- False
- True

9.1.7

To use a single variable from a namespace without having to precede it each time with what namespace it comes from, you can use the command:

- using namespace::variable;
- namespace::variable;
- using ::variable
- using namespace namespace::variable;

📖 9.1.8

The cin stream

The std::cin is an input stream for loading data into variables. It belongs to the std namespace, to use it you must import the iostram library. In order for data to be read in using std::cin, the operator >> must be used for this, reading the data looks like this:

```
#include <iostream>
...
int a;
std::cin >> a;
```

This way you can load all basic data types.

When using the cin stream, whitespace (space, tab, enter) is ignored, they are treated as data splits, so when you try to write to a string variable using the cin string "Example of using cin" for variables, only the first word will be written ("Example").

The std::cin stream also allows you to enter multiple data using one cin stream, subsequent variables for which the data is to be read are switched by the operator >>, e.g.

```
int a, b, c;
std::cin >> a >> b >> c;
```

Data for input variables can be separated by any white space;

The cin stream also allows error checking, using the fail() method, which I call after loading the data:

```
int a;
std::cin >> a;
cout << std::cin.fail();
```

This method will work when, for example, a program user gives a character instead of a number.

Entering incorrect data and calling an error does not lead to their deletion from the buffer, these data remain in the buffer and will be written to the variable the next time the std::cin stream is called; To remedy this, discard the buffer data using the std::cin.ignore() method.

9.1.9

The cout stream

The std::cout is a stream used to output variable values to the outside (e.g. to console). The cout stream connects to the operator "<<" to output data:

```
int a = 4;
std::cout << a;
```

The values of several variables can be derived using one cout stream:

```
std::cout << a << b << c;
```

The cout stream also has functions and operators for modifying the format of the data displayed, the following inland includes:

```
std::cout.width (10); // sets the display field to 10
std::cout << right << a << endl; // right aligment
std::cout << left << a << endl; // align text to the left
std::cout << internal << a << endl; // align text to the
center
```

9.1.10

Enter the operator that appears after the *cin* stream.

9.1.11

To display the value of several variables using one *cout* command, we will use:

- cout <|<| a <|<| b <|<| c <|<| d;
- cout <|<| a, b, c, d;
- It is not possible.
- cout >> a >> b >> c >> d;
- cout >> a, b, c, d;

9.1.12

To discard data from the buffer, use the command:

- cin.ignore();

- ignore(cin);
- cin.reject();
- reject(cin);

9.2 Namespaces (programs)

⌨ 9.2.1 Own namespace

Write a program in which your own namespace will be declared. Within the created namespace, declare checking if the triangle whose length is given by the user is a right triangle. The result of the function is to be *true* or *false*. Call the function in the main function, display the result of its operation.

```
Input : 6 2.5 6.5
Output: true
```

```
Input : 6 3 7
Output: false
```

String

Chapter 10

10.1 String

🕮 10.1.1

In the current C++ standard, the string class has been added. The string class allows you to store strings not in character arrays, but in objects of this class. The use of the string class to store strings simplified string operations compared to operations on character arrays.

To use the string class, attach the string header file. The string class is part of the std namespace, so you must use the using-declaration or refer to it as std::string.

Initialization of the string object:

```
string author; // create an empty string object
string title = "Introduction to c ++"; // create the
initialized string object
```

The most important features of string objects include:

- string objects can be initialized as array strings are initialized
- you can use the cin object to place keyboard data in the string object
- you can use the cout object to show the contents of a string object
- you can refer to individual characters of the string object using an array notation

🕮 10.1.2

The basic difference between string objects and character arrays is that we create string objects as normal variables, not as arrays. Using the string class also allows you to transfer responsibility for resizing the chain to the program. The program automatically increases the length of the string object as we add more characters to it.

```
std::string str1;
std::cin>>str1;
```

Basic operations on string objects:

Concatenation of strings:

```
string str1 = "Example";
string str2 = "string";
```

```
string str3 = str1 + str2;
str1 + = str2;
str3 + = "!";
```

Copying objects of the class string

```
string str4 = str3;
```

Specifying string length

```
int len1 = str3.length();
```

Comparison of chains

```
bool compar = str1 == str2;
```

Checking if the string object is empty:

```
str1.empty ();
```

Returning a character with the given position:

```
str1.at(0); // will return the first character
Or
str1[0];
```

Deleting characters from the string object:

```
str1.clear(); // remove all characters from the string
str2.erase(pos, len) // pos - position of the first character
to resed, len - number of character to erase
```

Finding a substring of characters in a string:

```
str1.find("Ex"); // returns the position of the first
character of the "Ex" substring
str1.find("Ex", 2); // returns the position of the first
character of the "Ex" substring, it begins the search from the
character index 2
```

Swap the value of two strings

```
Str1.swap(Str2);
```

Returning a substring of a text string

```
Str3.substr(pos, linen); // pos - starting position of the substring, length - the length of the sustring
```

10.1.3

Indicate the correct declaration of the string object.

- std::string text;
- std::String text;
- std::string="Some text";
- std::string[] string="Some text";

10.1.4

The string class belongs to the *std* namespace.

- True
- False

10.1.5

You can work on objects of the String class like regular variables.

- True
- False

10.1.6

You can use syntax to combine two strings:

- str1 += str2;
- str1.add(str2);
- str1 = add(str1, str2)
- str1.copy(str2);

10.1.7

As a result of the code presented below, the screen will display:

```
#include <iostream>
using namespace std;
int main()
{
    string text = "Some example text";
    cout << text[0];
    return 0;
}
```

- S
- Error - the program will not compile
- Some example text
- Na

10.1.8

We have the following statement:

```
string text = "Example";
```

After using the command *text.erase(1, 2)*, what value will the string variable *text* store?

10.2 String (programs)

10.2.1 Odd characters

Write a program that will display the given string (given by the user), skipping every second letter for it.

```
Input : "Example"
Output: "Eape"
```

```
Input : "Apple"
Output: "Ape"
```

10.2.2 Is upper

Write the function *first_Upper()*, which will check if the first letter of the string given as the function parameter is uppercase. The function returns *true* if it is, or *false*

otherwise. In the main function call your function, strings are to be typed from the keyboard.

```
Input : "Example"
Output: true
```

```
Input : "example"
Output: false
```

10.2.3 Reverse string

Write a program that reads a string from the user and then creates a string that is the inverse of the given string and displays it on the screen.

```
Input : "Example"
Output: "elpmaxE"
```

```
Input : "apple"
Output: "elppa"
```

10.2.4 Palindrome

Write a program that reads a string from the user and then checks to see if the string is a palindrome. The program displays true if it is or false if not.

A palindrome is a word, number, phrase, or other sequence of characters that reads the same backward as forward,

```
Input : "taco cat"
Output: true
```

```
Input : "Example"
Output: false
```

10.2.5 Count last character

Write a program that reads a string of characters from the user and then displays information about how many times the last character is repeated in that string.

```
Input : "Example"
Output: 2
```

```
Input : "banana"
Output: 3
```

Exercises

Chapter 11

11.1 Exercises

⌨ 11.1.1 Operator overloading - Inventory class

Write a code for the inventory class with an object of three elements: int ID, string name and double value. We assume that the values are given in this order when creating the object. Overload the input and output operators for this class so that the object output will be in the form

```
Input : 4
1000 table 12.5
1002 chair 4.5
1004 bookstand 21.5
1006 box 7.1
Output: <1000,table,12.5euro>;
<1002,chair,4.5euro>;
<1004,bookstand,21.5euro>;
<1006,box,7.1euro>;
```

```
Input :  4
table 1000 12.5
1002 chair 4.5
1004 bookstand 21.5
1006 box 7.1
Output: Error. Incompatibility of types
```

PRISCILLA

priscilla.fitped.eu