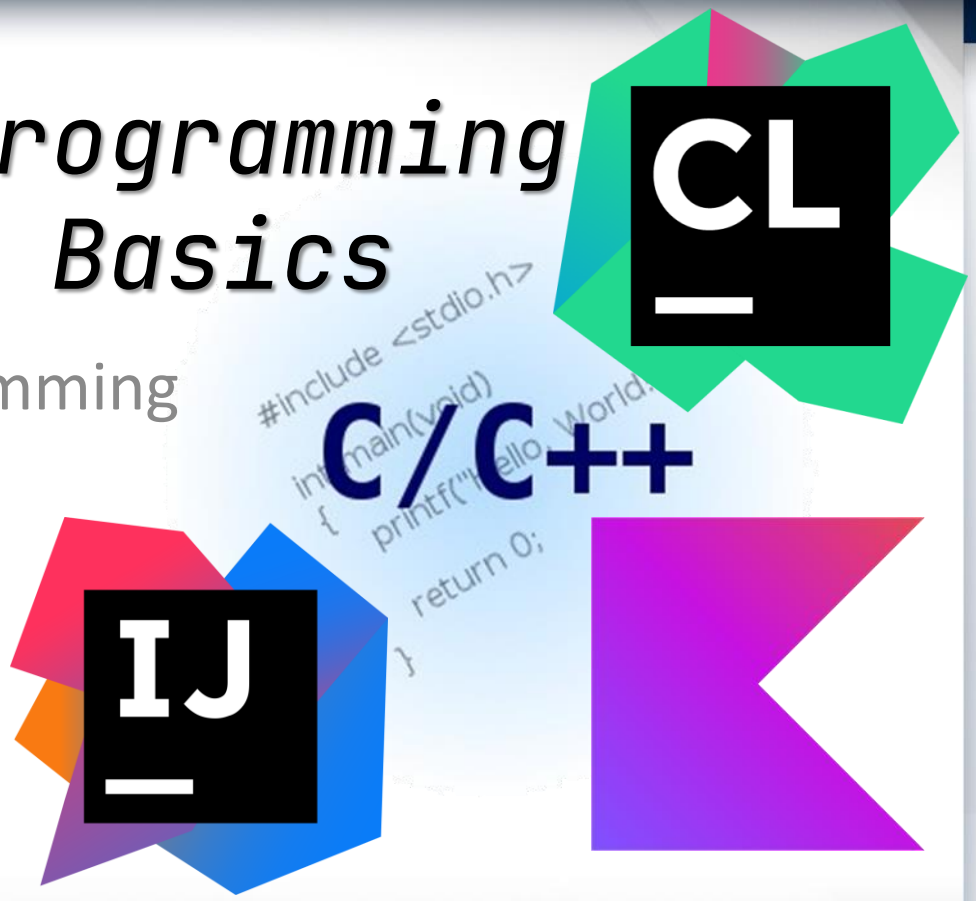
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming
Programming Basics

C/C++/Kotlin programming

(p.4 – Files)



Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Files

- In computing, a file is a unit of data that is stored on a computer or other electronic device.
- A file can be thought of as a collection of information, which can be text, images, music, programs, or any other type of data.

- Files are often organized into directories or folders to help users manage and find them more easily.
- The file name typically provides a brief description of the file's contents, while the file extension indicates the type of file and its format.

Files

- Files can be opened and edited by various software applications, and they can be shared, copied, moved, and deleted.
- Files can be stored on a variety of storage devices, such as hard drives, flash drives, optical discs, or cloud storage services.

Files

- Files with the same name cannot be in the same directory.
- A file name is not only as its name, but also as an extension, for example: **file.txt** and **file.dat** are different files, although they have the same names.
- There is such a thing as the full name of the files - this is the full path to the file directory with the file name, for example: **`D:\docs\file.txt`**.

Files

- For working with files, you need to include a header file <fstream>.
- The header file <fstream> defines several classes and includes header files:
  - <ifstream> - file input and
  - <ofstream> - file output.

Files

- File I/O is similar to standard I/O, the only difference is that I/O is not done to the screen, but to a file.
- If input/output to standard devices is performed using the **cin** and **cout** objects, then to organize file I/O, it is enough to create your own objects that can be used in the same way as **cin** and **cout**.

Files

- For example, if we need to create text file and write string ”`Working with files in C++`” in it.
- Then we should do next steps:

1. Create object of ofstream class;
2. Associate this object with file for writing;
3. Write string to file;
4. Close file.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Files

```
// create an object to write to a file
ofstream /*name of object*/; // object of ofstream class
```

Let's name will be – fout:

```
ofstream fout;
```

What is the object for?

- The object is required to be able to write to the file.
- The object has already been created, but is not associated with the file to which the string needs to be written.

```
fout.open("example.txt");
// Associate object with file
```

Files

```
ofstream fout;
fout.open("example.txt");
```

- With the dot operation, we get access to the open() class method, in parentheses of which we specify the file name.
- The specified file will be created in the current directory with the program.
- If a file with the same name exists, then the existing file will be replaced by the new one.

```
fout << "Working with files in C++
// write string to the file
```

Files

```
ofstream fout;
fout.open("example.txt");
fout << "Working with files in C++";
```

Since it is no longer necessary to change the contents of the file, it must be closed, that is, the object should be separated from the file.

```
fout.close(); // closing file
```

Outcome - a file with a string is created

Steps 1 and 2 can be combined, that is, in one line, create an object and associate it with a file:

```
ofstream fout("example.txt");
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Files

As result we get a such program:

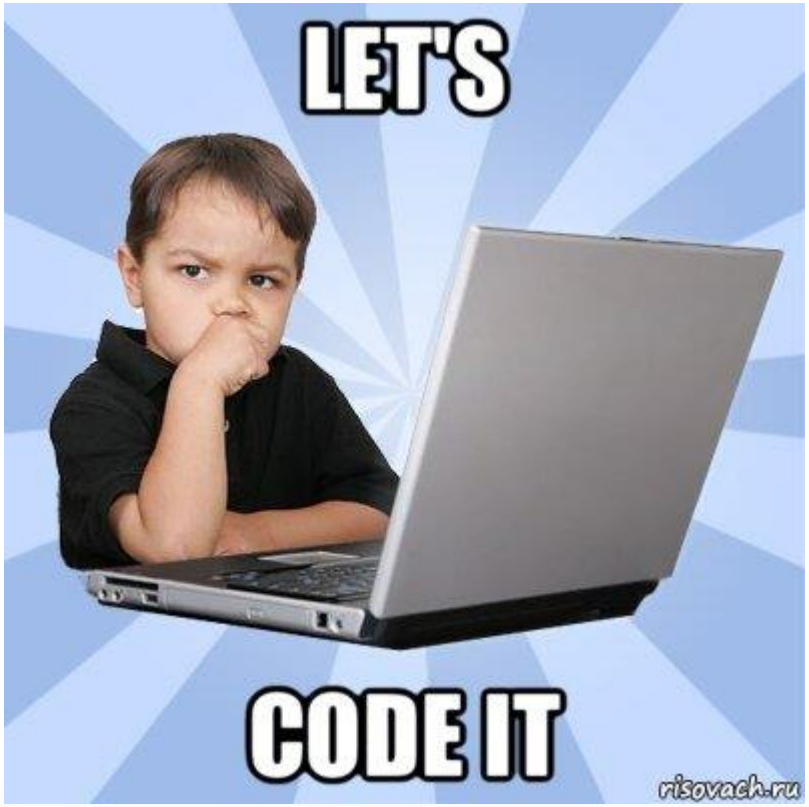
```
#include <fstream>
using namespace std;

int main()
{
    ofstream fout("example.txt");
    fout << "Working with files in C++";
    fout.close();
    return 0;
}
```

It remains to check if program ran correctly, and for this we open the file example.txt:

```
Working with files in C++
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Demo



Files

In order to read the file, you will need to follow the same steps as when writing to a file with minor changes:

- Create an object of the ifstream class and associate it with the file to be written to;
- Read file;
- Close the file.

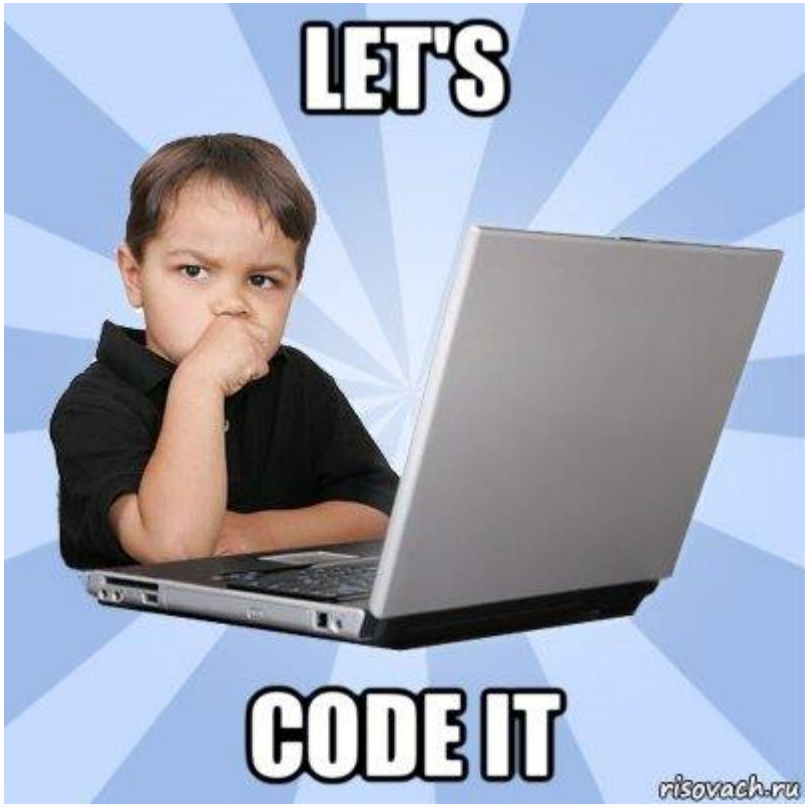
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Files

```
#include <fstream>
#include <iostream>

using namespace std;

int main()
{
    char buff[50];
    ifstream fin("example.txt");
    fin >> buff;
    cout << buff << endl;
    fin.getline(buff, 50);
    fin.close();
    cout << buff << endl;
    return 0;
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Demo



Files

- The program worked correctly, but this is not always the case, even if everything is in order with the code.
- For example, the name of a non-existent file was passed to the program, or an error was made in the name.
- In this case, nothing will happen at all.
- The file will not be found, which means that it is not possible to read it.
- Therefore, the compiler will ignore the lines where the file is being manipulated.
- As a result, the program will exit correctly, but nothing will be shown on the screen.

Files

- A simple user will not understand what is the matter and why the line from the file did not appear on the screen.
- To respond to this situation, C++ provides a special function - is_open(), which returns integer values:
  - 1 - if the file was successfully opened,
  - 0 - if the file has not been opened.
- Let’s improve the program with the opening of the file, in such a way that if the file is not opened, a corresponding message is displayed.

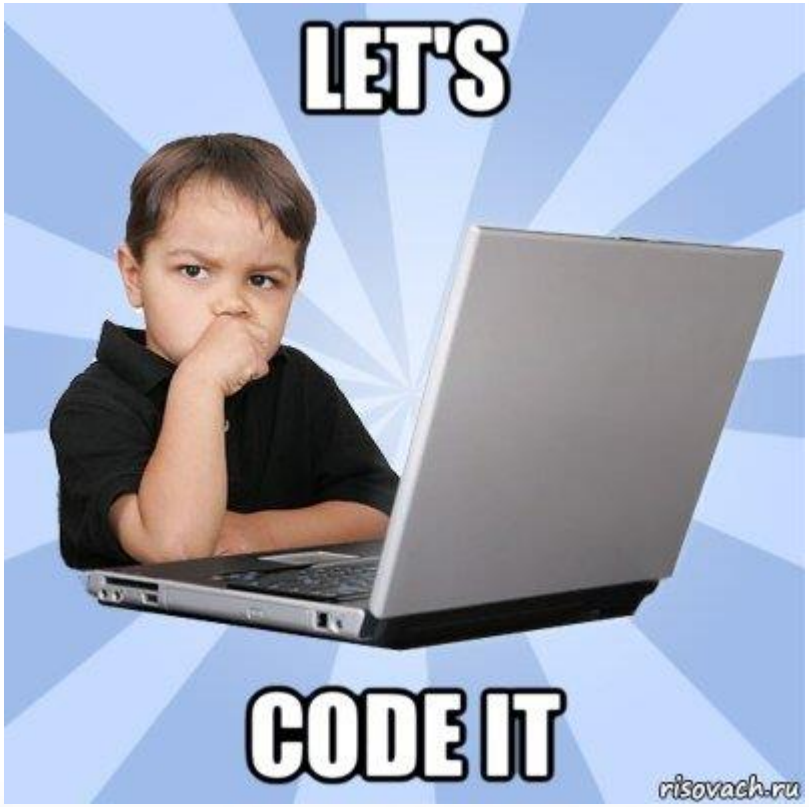
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Files

```
#include <fstream>
#include <iostream>

using namespace std;

int main()
{
    char buff[50];
    ifstream fin("example.txt");
    if (!fin.is_open()) // if file is not opened
        cout << "File can't be opened!\n";
    else {
        fin >> buff;
        cout << buff << endl;
        fin.getline(buff, 50);
        fin.close();
        cout << buff << endl;
    }
    return 0;
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Demo



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Modes (flags) for files

Const	Description
ios_base::in	Open file for read
ios_base::out	Open file fo write
ios_base::ate	move pointer to end of file when opening
ios_base::app	open file for writing to end of file
ios_base::trunc	remove the contents of the file if it exists
ios_base::binary	open a file in a binary mode

Modes (flags) for files

File opening modes can be set when creating an object or when you call a function open()

```
// open file to add information
// at the end of file
ofstream fout("example.txt", ios_base::app);
fout.open("example.txt", ios_base::app);
```

File opening modes can be combined using the bitwise logical operation "or“ - |, for example: `ios_base::out | ios_base::trunc` – open file for writing, and clear it before this.

Default modes

- Objects of the **ofstream** class, when associated with files, by default contain file opening modes **ios_base::out** | **ios_base::trunc**.
- That is, the file will be created if it does not exist.
- If the file exists, then its contents will be deleted, and the file itself will be ready for recording.

- Objects of the **ifstream** class, when associated with a file, have by default the file open mode **ios_base::in** - the file is opened for reading only.
- The file open mode is also called “flag”.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Files in Kotlin

There are several ways to write text files in Kotlin:

- Direct writing:
  - *writeText*
  - *writeBytes*
- *Write, using Writers objects:*
  - *printWriter*
  - *bufferedWriter*

writeText

Maybe the simplest extension method of class File: **writeText** *takes the content as a String argument and writes it directly to the specified file.*

This content is text encoded in UTF-8 (default) or any other specified

```
File(fileName).writeText(fileContent)
```

writeBytes

- Similarly, we can use bytes as input.
- **The writeBytes method takes a ByteArray as an argument and directly writes it to the specified file.**
- This is useful when we have byte array content rather than plain text.

```
File(fileName).writeBytes(fileContentAsArray)
```

printWriter

- If we want to use Java *PrintWriter* , Kotlin provides a method *printWriter* for this purpose.
- With it, we can print formatted representations of objects to the output stream:

```
File(fileName).printWriter()
```

This method returns a new instance of *PrintWriter* .
Then we can use it's method *use* , for write data

```
File(fileName).printWriter().use {
    out -> out.println(fileContent)
}
```

The resource will be closed regardless of whether the function succeeded or not

bufferedWriter

- Similarly, Kotlin also provides a function named *bufferedWriter* , from Java
- With this writer we can more effectively write text to output stream

```
File(fileName).bufferedWriter()
```

As a *PrintWriter* , this function returns a new instance of *BufferedWriter* , that we can use for write a content of file

```
File(fileName).bufferedWriter().use {
    out -> out.write(fileContent)
}
```

Reading text files in Kotlin

There are several ways to read and process text files в Kotlin:

- *forEachLine*
- *useLines*
- *bufferedReader*
- *readLines*
- *inputStream*
- readText

forEachLine

- Reads file line by line, using specified charset (UTF-8 by default) and calls action for each line:

```
fun readFileLineByLineUsingForEachLine(fileName: String)
        = File(fileName).forEachLine { println(it) }
```

useLines

- Calls the block callback giving it a sequence of all the lines in this file and closes the reader once the processing is complete

```
fun readFileAsLinesUsingUseLines(fileName: String): List<String>
        = File(fileName).useLines { it.toList() }
```

bufferedReader

- Returns a new BufferedReader for reading the content of this file.
- When we have a *BufferedReader*, we can read all lines in it:

```
fun readFileAsLinesUsingBufferedReader(fileName: String): List<String>
        = File(fileName).bufferedReader().readLines()
```

readLines

- Reads the file content as a list of lines.

```
fun readFileAsLinesUsingReadLines(fileName: String): List<String>
        = File(fileName).readLines()
```

Do not use this function for huge files.

inputStream

- Constructs a new FileInputStream of this file and returns it as a result.
- When we receive the input stream, we can convert it to bytes and then to a full *String*

```
fun readFileAsTextUsingInputStream(fileName: String) =
    File(fileName)
        .inputStream()
        .readBytes()
        .toString(Charsets.UTF_8)
```

readText

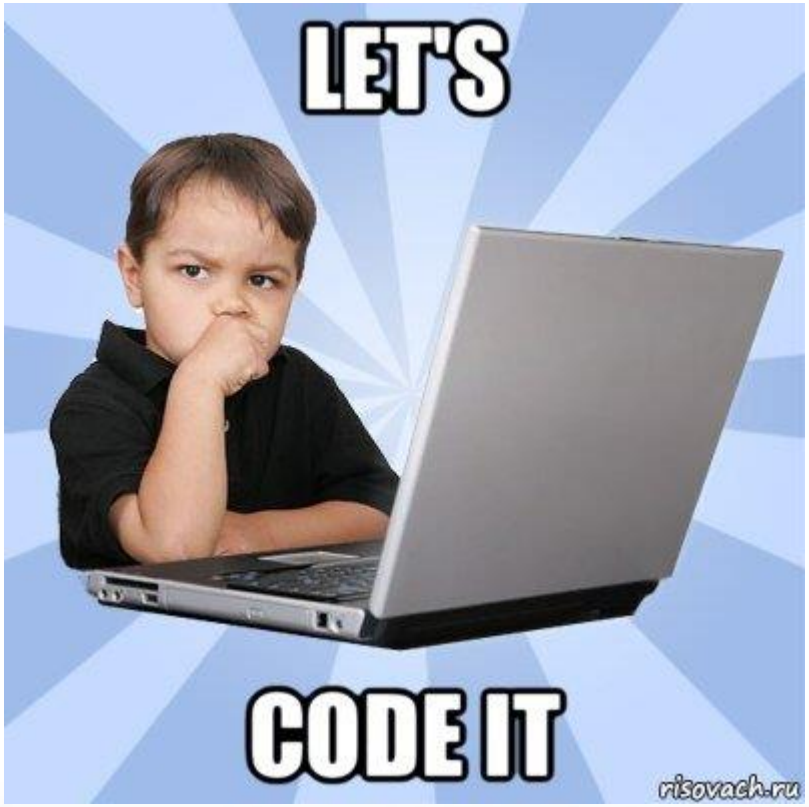
- Gets the entire content of this file as a String using UTF-8 or specified charset.

```
fun readFileDirectlyAsText(fileName: String): String
        = File(fileName).readText(Charsets.UTF_8)
```

This method is not recommended on huge files. It has an internal limitation of 2 GB file size.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Demo

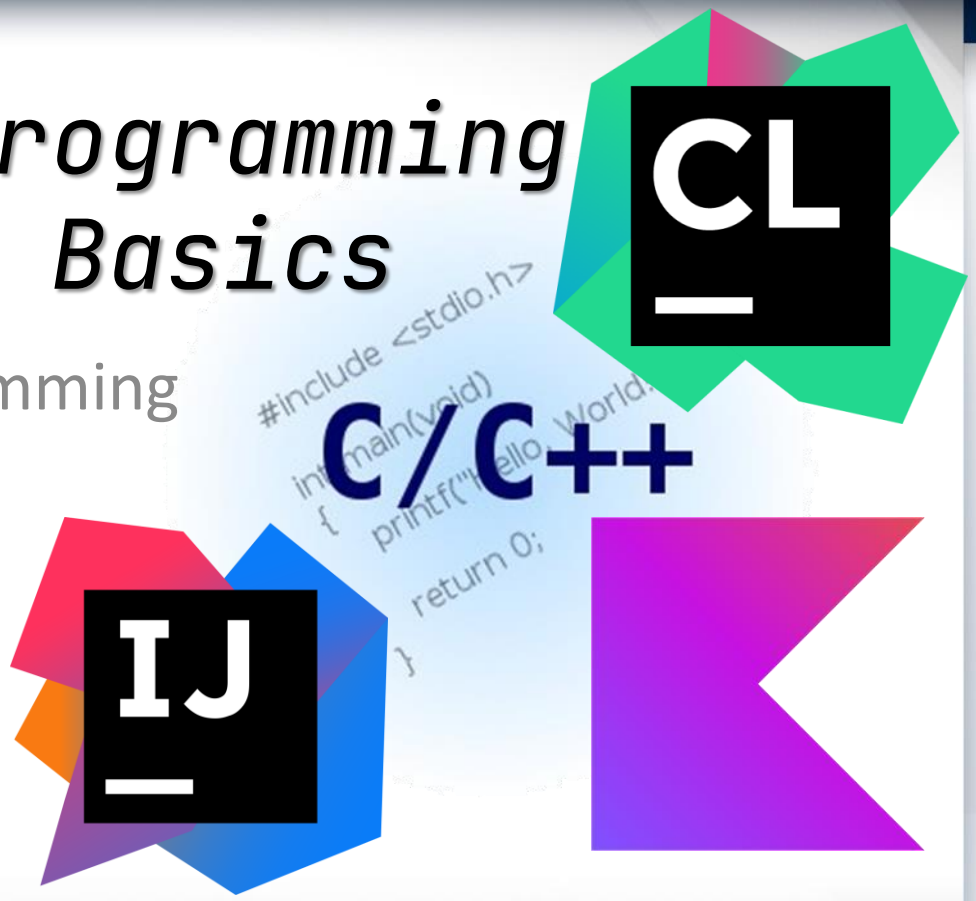


НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming
Programming Basics

C/C++/Kotlin programming

(p.4 – Files)



Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua