

# Algorithms & Programming

p.2.1 – Intro, conditionals, loops



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

# What is C++?

- C++ is a cross-platform language that can be used to create high-performance applications.
- C++ was developed by Bjarne Stroustrup, as an extension to the C language.
- C++ gives programmers a high level of control over system resources and memory.



# C++ standards

Year	C++ Standard	Informal name
1998	ISO/IEC 14882:1998	C++98
2003	ISO/IEC 14882:2003	C++03
2011	ISO/IEC 14882:2011	C++11, C++0x
2014	ISO/IEC 14882:2014	C++14, C++1y
2017	ISO/IEC 14882:2017	C++17, C++1z
2020	ISO/IEC 14882:2020	C++20, C++2a

# Why Use C++

- C++ is one of the world's most popular programming languages.
- C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems.
- C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.
- C++ is portable and can be used to develop applications that can be adapted to multiple platforms.
- As C++ is close to C# and Java, it makes it easy for programmers to switch to C++ or vice versa

# Let's start!

- To start using C++, you need two things:
- A text editor, like Notepad, to write C++ code
- A compiler, like GCC, to translate the C++ code into a language that the computer will understand
- There are many text editors and compilers to choose from. We will use an IDE.



# C++ Install IDE

- An IDE (Integrated Development Environment) is used to edit AND compile the code.
- Popular IDE's include
  - Code::Blocks,
  - Eclipse,
  - Visual Studio,
  - CLion.
- These are all free, and they can be used to both edit and debug C++ code.

# Code::Blocks

- You can find the latest version of Code::Blocks at <https://www.codeblocks.org/downloads/binaries/>
- Choose the **codeblocks-20.03mingw-setup.exe**, which will install the text editor with a compiler.



# JetBrains CLion

- You can find the latest version of CLion at <https://www.jetbrains.com/clion/>



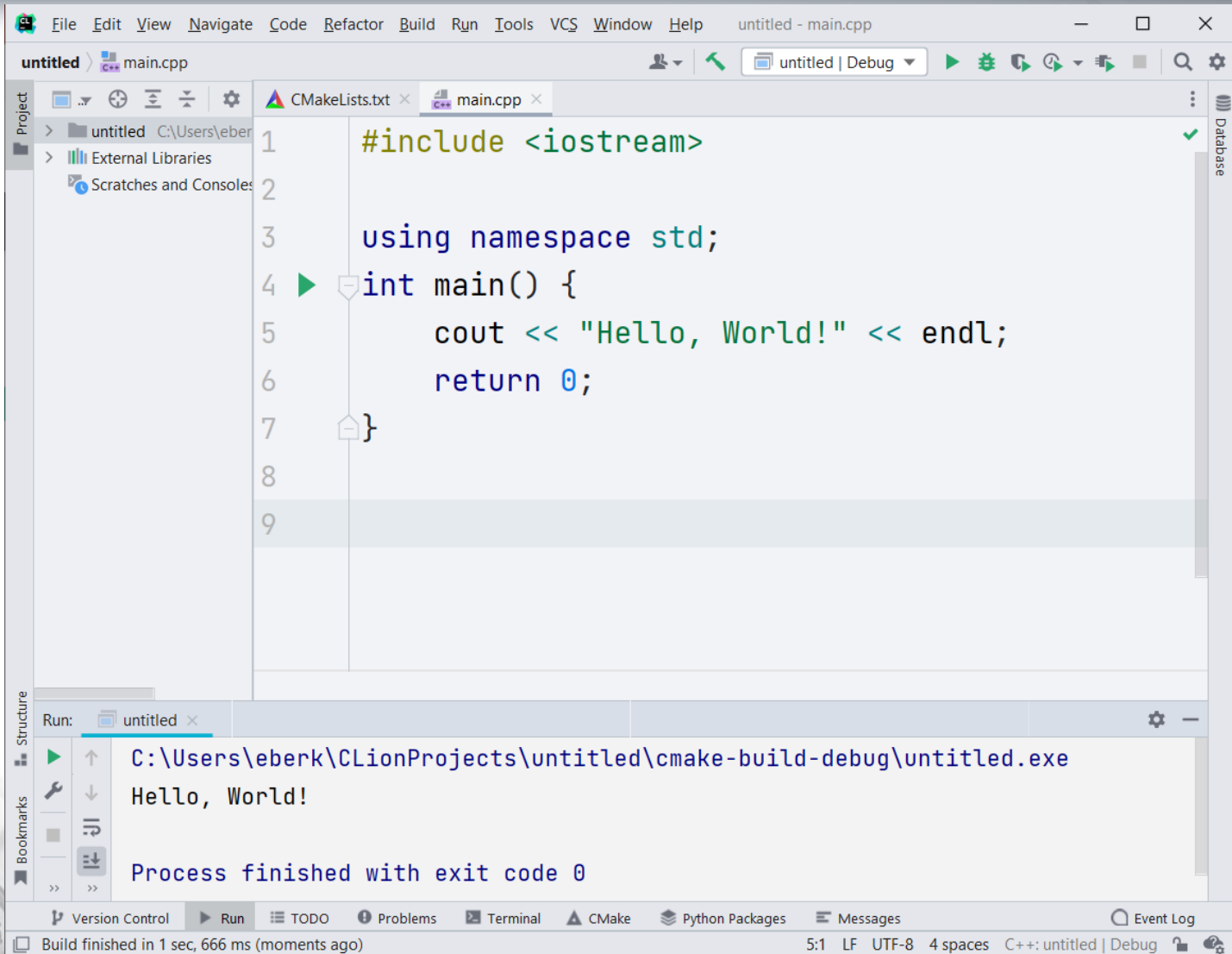


# First Program in C++

- Let's create our first C++ program:

```
1      #include <iostream>
2
3      using namespace std;
4  ►  int main() {
5          cout << "Hello, World!" << endl;
6          return 0;
7      }
```

# Running...



```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help  untitled - main.cpp
untitled  C++ main.cpp
Project
  > untitled C:\Users\eberk\
  > External Libraries
  > Scratches and Consoles
CMakeLists.txt  main.cpp
1  #include <iostream>
2
3  using namespace std;
4  int main() {
5      cout << "Hello, World!" << endl;
6      return 0;
7  }
8
9
Run:  untitled
C:\Users\eberk\CLionProjects\untitled\cmake-build-debug\untitled.exe
Hello, World!

Process finished with exit code 0
Version Control  Run  TODO  Problems  Terminal  CMake  Python Packages  Messages  Event Log
Build finished in 1 sec, 666 ms (moments ago)  5:1  LF  UTF-8  4 spaces  C++: untitled | Debug
```

# C++ Syntax

Let's break up the following code to understand it better:

```
1      #include <iostream>
2
3      using namespace std;
4  ►  int main() {
5          cout << "Hello, World!" << endl;
6          return 0;
7      }
```

- *Line 1:* `#include <iostream>` is a header file library that lets us work with input and output objects, such as `cout` (used in line 5).
- Header files add functionality to C++ programs.

# C++ Syntax

Let's break up the following code to understand it better:

```
1      #include <iostream>
2
3      using namespace std;
4  ►  int main() {
5          cout << "Hello, World!" << endl;
6          return 0;
7      }
```

- *Line 2:* A blank line. C++ ignores white space. But we use it to make the code more readable.

# C++ Syntax

Let's break up the following code to understand it better:

```
1      #include <iostream>
2
3      using namespace std;
4  ►  int main() {
5          cout << "Hello, World!" << endl;
6          return 0;
7      }
```

- *Line 3:* `using namespace std` means that we can use names for objects and variables from the standard library.

# C++ Syntax

Let's break up the following code to understand it better:

```
1      #include <iostream>
2
3      using namespace std;
4  ►  int main() {
5          cout << "Hello, World!" << endl;
6          return 0;
7      }
```

- *Line 4:* Another thing that always appear in a C++ program, is `int main()`. This is called a function.
- Any code inside its curly brackets `{}` will be executed.

# C++ Syntax

Let's break up the following code to understand it better:

```
1      #include <iostream>
2
3      using namespace std;
4  ►   int main() {
5          cout << "Hello, World!" << endl;
6          return 0;
7      }
```

- *Line 5:* cout (pronounced "see-out") is an object used together with the insertion operator (<<) to output/print text. In our example it will output "Hello World".

***Note: Every C++ statement ends with a semicolon ;***

# Some notes

**Note:** The body of `int main()` could also be written as:

```
int main () { cout << "Hello World! "; return 0; }
```

**Remember:** The compiler ignores white spaces. However, multiple lines makes the code more readable.



# C++ Syntax

Let's break up the following code to understand it better:

```
1      #include <iostream>
2
3      using namespace std;
4  ►  int main() {
5          cout << "Hello, World!" << endl;
6          return 0;
7      }
```

*Line 6:* `return 0` ends the main function.

*Line 7:* Do not forget to add the closing curly bracket `}` to actually end the main function.

# Omitting Namespace

```
1  #include <iostream>
2
3  ► int main() {
4      std::cout << "Hello, World!" << std::endl;
5      return 0;
6  }
7
```

# C++ Output (Print Text)

```
1      #include <iostream>
2      using namespace std;
3
4  ►   int main() {
5          cout << "Hello World!";
6          cout << "I am learning C++";
7          return 0;
8      }
```

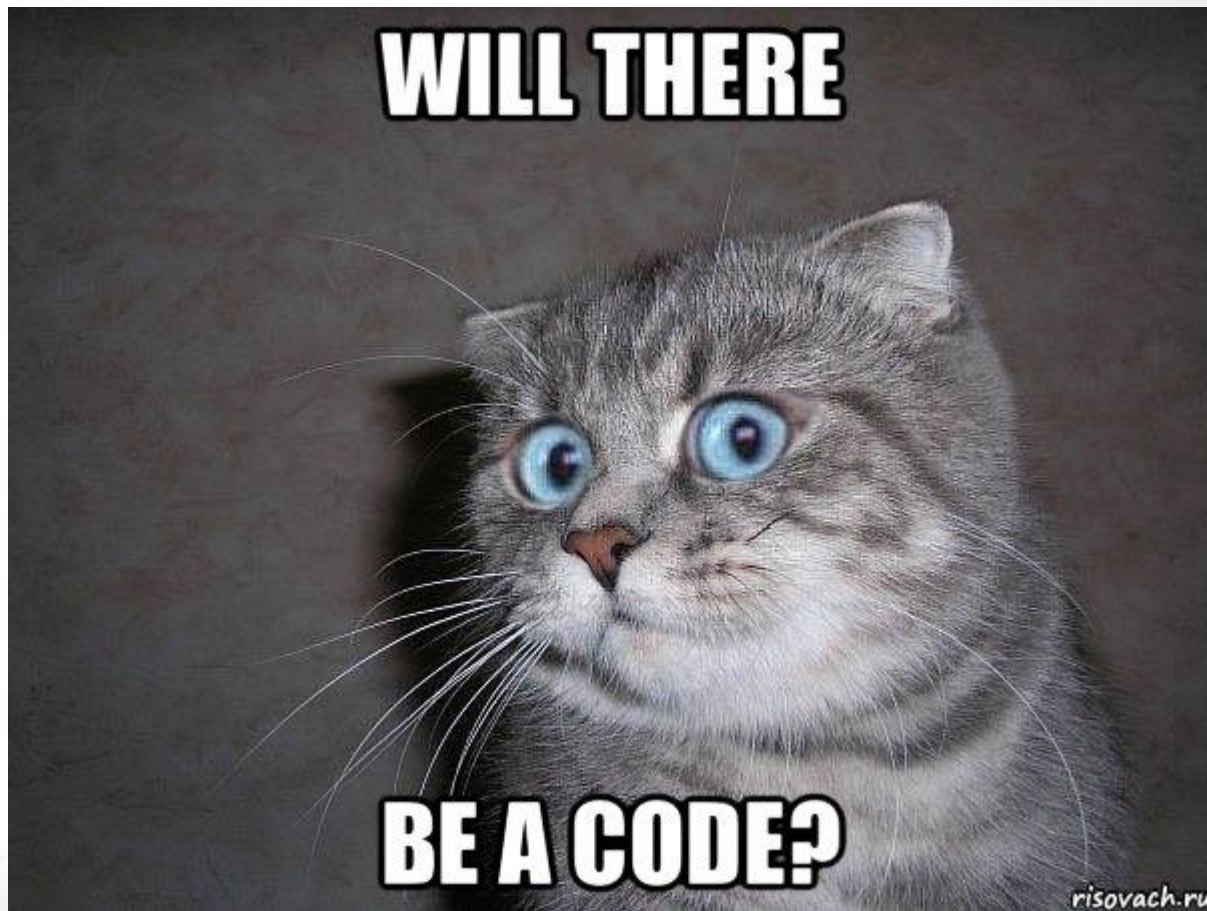
# New Lines

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

int main() {
    cout << "Hello World! \n";
    cout << "I am learning C++";
    return 0;
}
```

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

int main() {
    cout << "Hello World!" << endl;
    cout << "I am learning C++";
    return 0;
}
```



# C++ Comments

- Single-line comments start with two forward slashes (//)

```
cout << "Hello World!"; // This is a comment
```

- Multi-line comments start with /\* and ends with \*/

```
/* The code below will print the words Hello  
World! to the screen, and it is amazing */  
cout << "Hello World!";
```

# C++ Variables

- In C++, there are different types of variables, for example:
- `int` - stores integers (whole numbers), without decimals, such as 123 or -123
- `double` - stores floating point numbers, with decimals, such as 19.99 or -19.99
- `char` - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- `bool` - stores values with two states: true or false
- `string` - stores text, such as "Hello World". String values are surrounded by double quotes

# Declaring (Creating) Variables

*type variableName = value;*

Here **type** is one of C++ types (such as **int**), and **variableName** is the name of the variable (such as **x** or **myName**). The equal sign is used to assign values to the variable.

```
int myNum = 15;  
cout << myNum;
```



# Declaring (Creating) Variables

```
int myNum;  
myNum = 15;  
cout << myNum;
```

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

```
int myNum = 15; // myNum is 15  
myNum = 10;    // Now myNum is 10  
cout << myNum; // Outputs 10
```

# Other Types / Display Variables

```
int myNum = 5;           // Integer (whole number without decimals)
double myFloatNum = 5.99; // Floating point number (with decimals)
char myLetter = 'D';     // Character
string myText = "Hello"; // String (text)
bool myBoolean = true;   // Boolean (true or false)
```

The **cout** object is used together with the **<<** operator to display variables. To combine both text and a variable, separate them with the **<<** operator:

```
int myAge = 35;
cout << "I am " << myAge << " years old.";
```

# Declare Many Variables

- To declare more than one variable of the same type, use a comma-separated list:

```
int x = 5, y = 6, z = 50;  
cout << x + y + z;
```

# C++ Identifiers

- All C++ variables must be identified with unique names.
- These unique names are called identifiers.
- Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).
- Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

```
// Good
```

```
    int minutesPerHour = 60;
```

```
// OK, but not so easy to understand what m actually is
```

```
    int m = 60;
```

# The general rules for naming

- Names can contain letters, digits and underscores
- Names must begin with a letter or an underscore (`_`)
- Names are case sensitive (`myVar` and `myvar` are different variables)
- Names cannot contain whitespaces or special characters like `!`, `#`, `%`, etc.
- Reserved words (like C++ keywords, such as `int`) cannot be used as names

# C++ Constants

```
const int myNum = 15; // myNum will always be 15  
myNum = 10; // error: assignment of read-only variable 'myNum'
```

You should always declare the variable as constant when you have values that are unlikely to change:

```
const int minutesPerHour = 60;  
const float PI = 3.14;
```

# C++ User Input

- `cin` is a predefined variable that reads data from the keyboard with the extraction operator (`>>`)

```
int x;  
cout << "Type a number: "; // Type a number and press enter  
cin >> x; // Get user input from the keyboard  
cout << "Your number is: " << x; // Display the input value
```



# Good to know



- `cout` is pronounced "see-out". Used for output, and uses the insertion operator (`<<`)
- `cin` is pronounced "see-in". Used for input, and uses the extraction operator (`>>`)





# C++ Math

Function	Description
<code>abs(x)</code>	Returns the absolute value of x
<code>acos(x)</code>	Returns the arccosine of x
<code>asin(x)</code>	Returns the arcsine of x
<code>atan(x)</code>	Returns the arctangent of x
<code>cbrt(x)</code>	Returns the cube root of x
<code>ceil(x)</code>	Returns the value of x rounded up to its nearest integer
<code>cos(x)</code>	Returns the cosine of x
<code>cosh(x)</code>	Returns the hyperbolic cosine of x
<code>exp(x)</code>	Returns the value of $E^x$
<code>expm1(x)</code>	Returns $e^x - 1$
<code>fabs(x)</code>	Returns the absolute value of a floating x
<code>fdim(x, y)</code>	Returns the positive difference between x and y

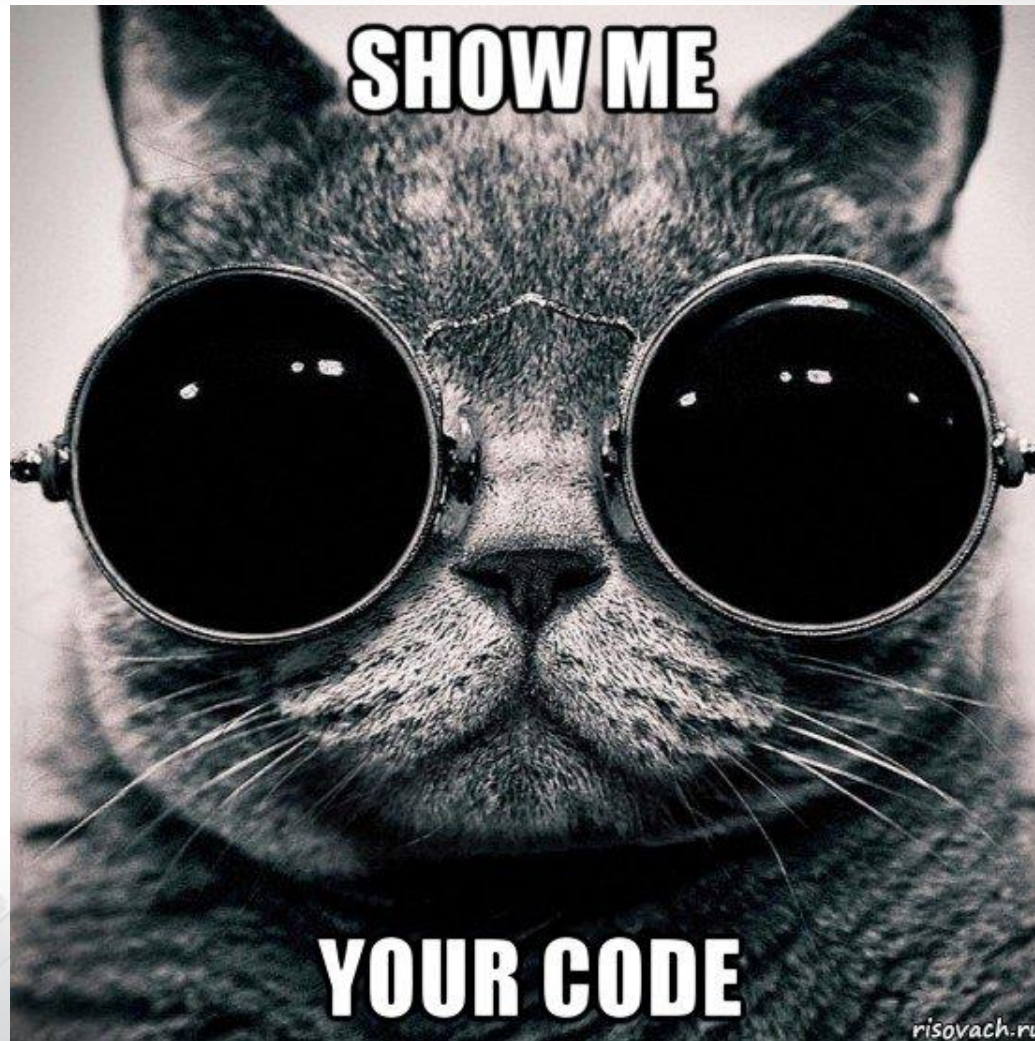
# C++ Math

Function	Description
<code>floor(x)</code>	Returns the value of $x$ rounded down to its nearest integer
<code>hypot(x, y)</code>	Returns $\sqrt{x^2 + y^2}$ without intermediate overflow or underflow
<code>fma(x, y, z)</code>	Returns $x*y+z$ without losing precision
<code>fmax(x, y)</code>	Returns the highest value of a floating $x$ and $y$
<code>fmin(x, y)</code>	Returns the lowest value of a floating $x$ and $y$
<code>fmod(x, y)</code>	Returns the floating point remainder of $x/y$
<code>pow(x, y)</code>	Returns the value of $x$ to the power of $y$
<code>sin(x)</code>	Returns the sine of $x$ ( $x$ is in radians)
<code>sinh(x)</code>	Returns the hyperbolic sine of a double value
<code>tan(x)</code>	Returns the tangent of an angle
<code>tanh(x)</code>	Returns the hyperbolic tangent of a double value



НАЦІОНАЛЬНИЙ  
УНІВЕРСИТЕТ  
КОРАБЛЕБУДУВАННЯ  
ІМЕНІ АДМІРАЛА МАКАРОВА

# Let's code!



# C++ if ... Else

C++ supports the usual logical conditions from mathematics:

- Less than:  $a < b$
- Less than or equal to:  $a \leq b$
- Greater than:  $a > b$
- Greater than or equal to:  $a \geq b$
- Equal to:  $a == b$
- Not Equal to:  $a != b$

# C++ If ... Else

C++ has the following conditional statements:

- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false
- Use **switch** to specify many alternative blocks of code to be executed

# The if Statement

```
if (condition) {  
    // block of code to be executed if the condition is true  
}
```

```
if (condition) {  
    // block of code to be executed if the condition is true  
} else {  
    // block of code to be executed if the condition is false  
}
```

```
if (condition1) {  
    // block of code to be executed if condition1 is true  
} else if (condition2) {  
    // block of code to be executed if the condition1 is false  
    // and condition2 is true  
} else {  
    // block of code to be executed if the condition1 is false  
    // and condition2 is false  
}
```



# Short Hand If...Else (Ternary Operator)

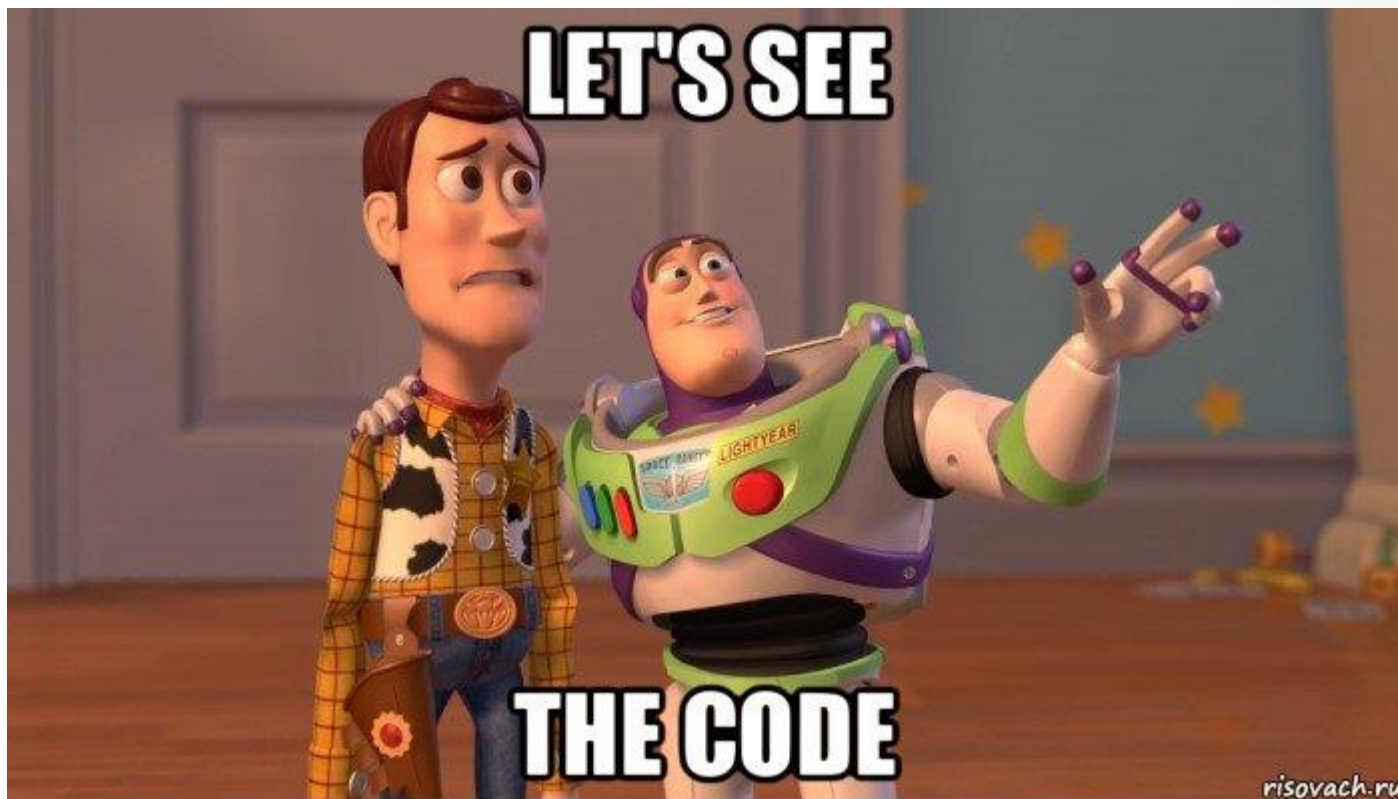
*variable = (condition) ? expressionTrue : expressionFalse;*

Instead of this

```
int time = 20;
if (time < 18) {
    cout << "Good day.";
} else {
    cout << "Good evening.";
}
```

You can write this

```
int time = 20;
string result = (time < 18) ? "Good day." : "Good evening.";
cout << result;
```





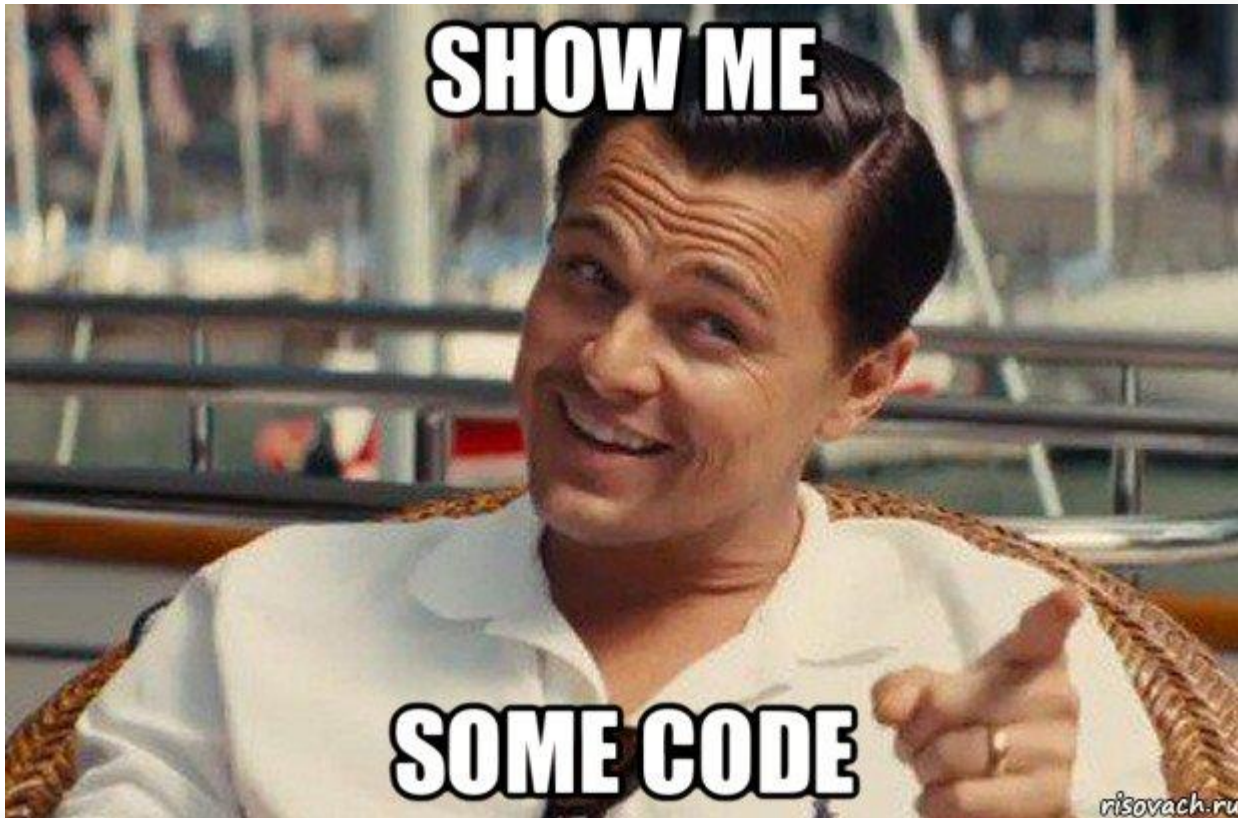
# C++ Switch

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

# C++ Switch

- The switch expression is evaluated once
- The value of the expression is compared with the values of each case
- If there is a match, the associated block of code is executed
- The break and default keywords are optional, and will be described later

# Let's code



# Loops

- Loops can execute a block of code as long as a specified condition is reached.
- Loops are handy because they save time, reduce errors, and they make code more readable.



# C++ Loops

- C++ While Loop

```
while (condition) {  
    someOperation();  
}
```

The while loop loops through a block of code as long as a specified condition is true

- Example

In this example, the code in the loop will run, over and over again, as long as a variable (i) is less than 5:

```
int i = 0;  
while (i < 5) {  
    cout << i << "\n";  
    i++;  
}
```

# C++ Loops

- C++ Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
do {  
    // code block to be executed  
}  
while (condition);
```

- Example

```
int i = 0;  
do {  
    cout << i << "\n";  
    i++;  
}  
while (i < 5);
```

# C++ Loops

- C++ For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

```
for (statement1; statement2; statement3) {  
    // code block to be executed  
}
```

- **statement1** is executed (one time) before the execution of the code block.
- **statement2** defines the condition for executing the code block.
- **statement3** is executed (every time) after the code block has been executed.



# C++ Loops

- C++ For Loop Example

The example below will print the numbers 0 to 4:

```
for (int i = 0; i < 5; i++) {  
    cout << i << "\n";  
}
```

Another Example

```
for (int i = 0; i <= 10; i = i + 2) {  
    cout << i << "\n";  
}
```

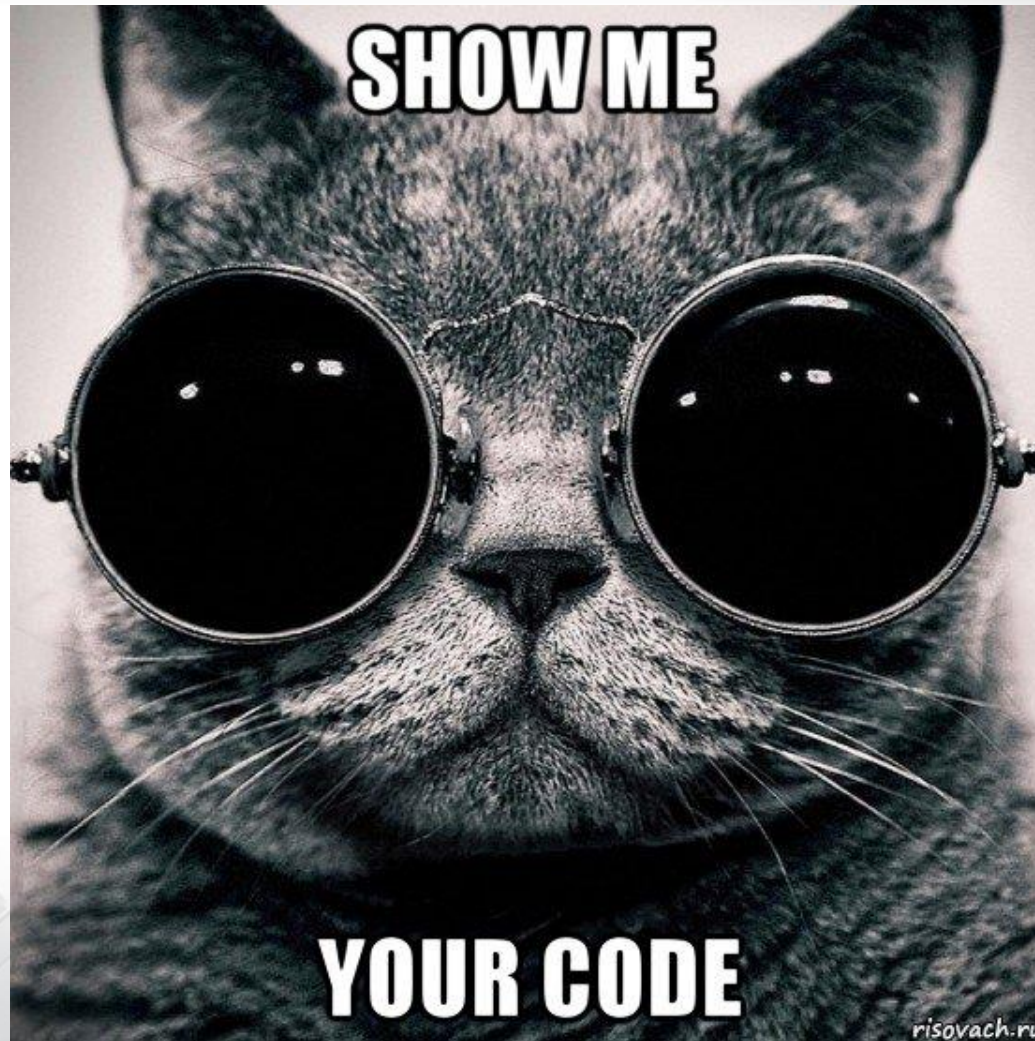
This example will only print even values between 0 and 10





НАЦІОНАЛЬНИЙ  
УНІВЕРСИТЕТ  
КОРАБЛЕБУДУВАННЯ  
ІМЕНІ АДМІРАЛА МАКАРОВА

# Let's code!



# C++ Break and Continue

- C++ Break

You have already seen the break statement used in the previous lecture. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a loop.

```
for (int i = 0; i < 10; i++) {  
    if (i == 4) {  
        break;  
    }  
    cout << i << "\n";  
}
```

# C++ Break and Continue

- C++ Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop

```
for (int i = 0; i < 10; i++) {  
    if (i == 4) {  
        continue;  
    }  
    cout << i << "\n";  
}
```

# C++ Break and Continue

- Break and Continue in While Loop

## Break Example

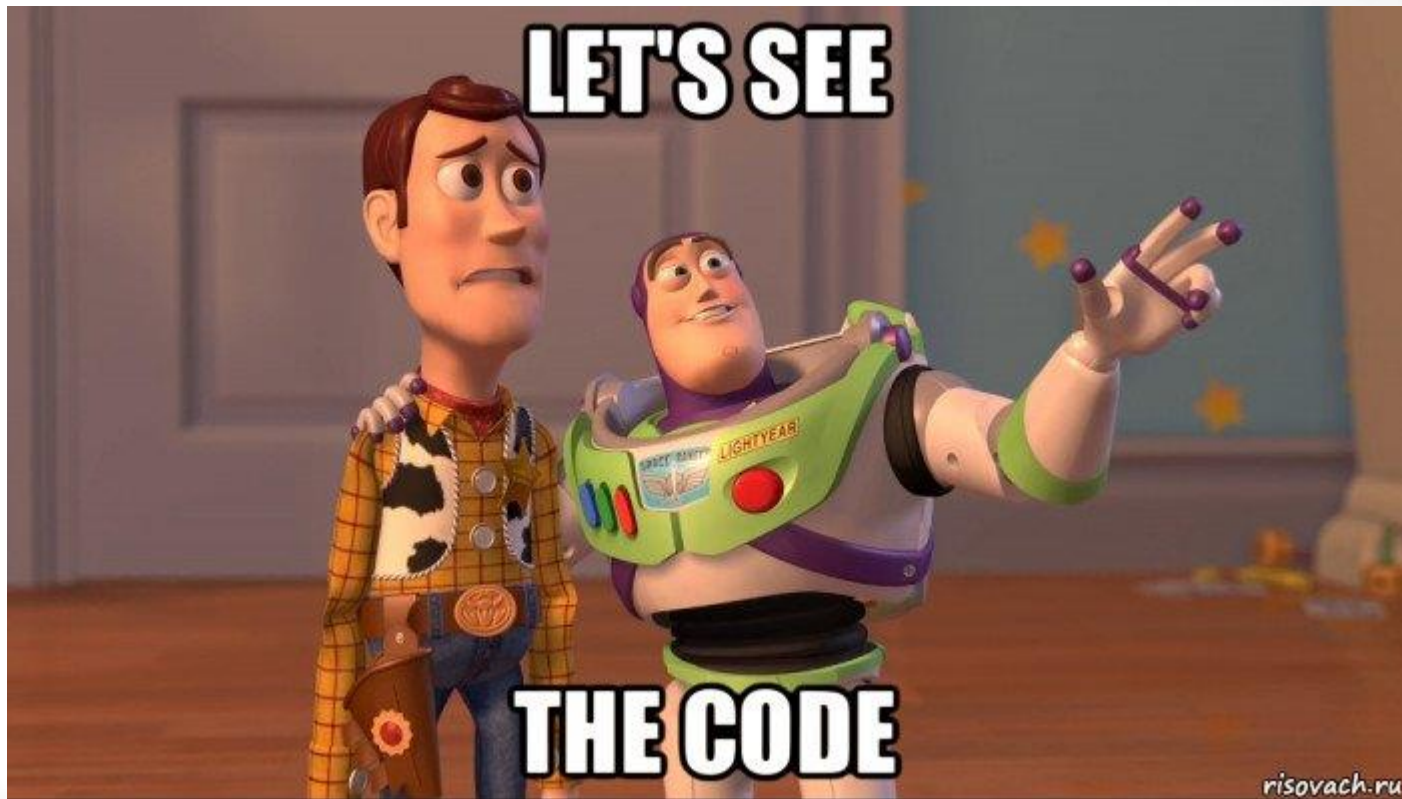
```
int i = 0;
while (i < 10) {
    cout << i << "\n";
    i++;
    if (i == 4) {
        break;
    }
}
```

## Continue Example

```
int i = 0;
while (i < 10) {
    if (i == 4) {
        i++;
        continue;
    }
    cout << i << "\n";
    i++;
}
```



# Let's code





# Q & A



# Algorithms & Programming

p.2.1 – Intro, conditionals, loops



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