C++

"C makes it easy to shoot yourself in the foot;
C++ makes it harder, but when you do it, it blows your whole leg off."

~ Bjarne Stroustrup~

What is C++?

C++ is the successor to the C programming language and was invented by Bjarne Stroustrup in the early 80's.

C++ is a multi-paradigm programming language. It extends C with the object oriented concepts of classes, inheritance and dynamic (run-time) binding. C++ is a strongly typed language which means that all entities int a program must be typed. A declaration must be made to the compiler to state the kind of information which they store.

C++ standards so far:

- cpp98
- cpp03
- cpp11
- cpp14
- cpp17

The number indicates the year when the sandard was release.

The question is: why would I want to learn C++? So, C++ is pretty much the most used language when you need to write fast codes that performs well or if you're writing for a weird architecture or platform and you need to the code run natively. If you want to control the hardware C++ is for you. Game industry uses C++ as well, for example game engines as Unity, Unreal or Frostbite are all written in C++. The biggest reason to use C++ is the direct control over the hardware.

How C++ works?

You write your code in C++ and then you pass your code into a compiler and that compiler will output machine code for your target platform. Machine code is the actual instructions that youre device's CPU will

actually perform. So using C++ we can literaly control every instruction that the CPU executes. C++ runs almost on every platform (Windows, macOS, Unix, iOS, Android, XBox, PS, Nintendo), you just need a compiler that would output machine code for that platform.

Note: just because your code is native doesn't mean it it's going to be fast. If you write bad code in C++ it's gonna be slow.

The basic workflow of writing a C++ program is you have a serious source files which you write actual text in and then you pass it to the compiler which compiles it into some kind of binary. Now that binary can be some sort of library or it can be an actual executable program.

Let's take a look at a very basic C++ program

```
#include <iostream>

int main()
{
    std::cout << "Hello World" << std::endl;
}
```

First of all the `#include <iostream>` statement.

Now this is something called a **preprocessor statement**, anything that begins with a hash (**#**) is a preprocessor statement. The first thing that a compiler does when it receives the source file is it preprocesses all of your preprocessor statement. That's why they called preprocessor statements, because they happen just before just before the actual compilation. What `include` will do is find a file and take all of the contents of that file and just paste it into this current file. These files that you include are typically called **header files**. The reason to include the `iostream` is because we need a declaration for function called `cout` , which let us print stuff to our console.

```
int main() {    }
```

It is very important. The main() function is called the **entry** point. It is the entry point for our application. That means when we run our application our computer starts executing code that begins in this function. As the program is running the computer will execute the lines of code we type in order.

Those who are familiar with functions you might notice that the return type of `main()` is actually an int, however we not returning an integer. That's because the `main()` function is actually a special case, you don't have to return any kind of value from the `main()` function. If you don't return anything it will assume that you returning zero this only applies to the `main()` function.

Lets take a look at the next line -

```
std::cout << "Hello World" << std::endl;
```

These left angular brackets (**<<**) which look kinda bit shift left operator are actually just an overloaded operator. So you need to think of them as a function. In C++ operators are just functions. So what we actually doing here that is pushing this **"Hello World"** string into this cout and then we pushing the endl. It tells our console to advance to the next line.

How do we get from this text file an executable binary file? Basically we go trough a few stages.

First the `#include <iostream>` once our preprocessor statements have been evaluated our file gets compiled. This is the stage where the compiler transforms all of this C++ code into a machine code. **Header files do not get compiled just cpp files**. Every cpp file will compiled into something called an **object** file. After that we need some way to stich them together into one exe file and that's is where the linker comes in. It takes all of the obj files and links them together.

Variables in C++

We want to be able to use data most of what programming is actually using and changing data, we want to be able to read and write from datas so we need to store this data in something called a variable. They allow us to name a piece of data that we store in memory. When we create a variable it is gonna be stored in memory in one of three places **stack**, **heap** or the **static/global** store.

When declaring a variable, an identifier or name must be provided so that the storage can be referenced. C++ requires that all variable are declared before they are used.

```
#include <iostream>

int main()
{
    int i = 1;
    std::cout << "My number is " << i << std::endl;
}
```

C++ has some built in variable like

- `int` - integers
- `float` and `double` - floating point numbers
- `char` - characters
- `bool` - boolean
- `string` - strings

Store in variable

You can store your own specific value in a variable like -

```
#include iostream

int main()
{
    // greet the user
    std::string name;
    std::cout << "Give me your name: ";
    std::cin >> name;
    std::cout << "Welcome, " << name << std::endl;
}
```

Note that the `std::cin` will read your input only for the first white space character. When you want to read a whole line use the following syntax -

`std::getline(std::cin, name);`

Operators in C++

Basic arithmetic operators are:

- +
- -
- /
- *

To know more about operators in C++, please visit this link

Expressions

Expression is anything that yields a value. Most statements in C++ are expressions.

Expression describe the action to be performed and consist of an operator and one or more operands.

```
#include <iostream>

int main()
{
    int x;
    int y;
    int res;

    std::cout << "Enter 2 numbers: ";
    std::cin >> x;
    std::cin >> y;

    result = x + y;

    std::cout << "The sum is: ";
    std::cout << result;
}
```

Lets reduce the lines of this code. How?

```
#include <iostream>

int main()
{
    int x,y; // no res needed, you will see why
    std:: cout << "Enter two numbers: ";
    std::cin >> x >> y; // read as many variable in one line as you want
    std::cout << "The sum is: " << x + y; // x + y will be evaluated first and the
n sent to cout
}
```

Conditional Statement

Conditional Expression

`a > b` or `c == 4` -> evaluates to *true* or *false*.

Note: non-zero always converts to *true*, zero converts to *false*.

The `if` statement

```
if (conditional expression)
    statement1; // if the expression is true
else
    statement2; // if the expression is false
```

You do not need brackets until your statements are single commands. You have many options how you will write your `if` statement. For example -

```
if (conditional expression)
    statement1;
else
    statement2;

if (conditional expression)
{
    statement1;
}
else
{
    statement2;
}

if (conditional expression) statement1;
else statement2;

if (conditional expression) { statement1; }
else { statement2; }
```

It is up to you to make your decision. All the four will do the same.

the `if-else` statement

When you want to check more expression.

```
if (conditional expression)
    statement1;
else if (conditional expression)
    statement2;
else if (conditional expression)
    statement3;
.
.
.
else
    statementN;
```

The ternary operator. It is the same as a normal `if-else` , it just have different syntax.

```
conditional expression ? true brach : false brach;
```