Introduction to C++

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.

The language was updated 4 major times in 2011, 2014, 2017, and 2020 to C++11, C++14, C++17, C++20.

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.

C++ is fun and easy to learn!

As C++ is close to C, C# and Java, it makes it easy for programmers to switch to C++ or vice versa.

Difference between C and C++
C++ was developed as an extension of C, and both languages have almost the same syntax.

The main difference between C and C++ is that C++ supports classes and objects, while C does not.

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

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

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

Example explained
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.

Line 2: using namespace std means that we can use names for objects and variables from the standard library.

Don't worry if you don't understand how #include <iostream> and using namespace std works. Just think of it as something that (almost) always appears in your program.

Line 3: A blank line. C++ ignores white space. But we use it to make the code more readable.

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.

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 ;.

Note: The body of int main() could also been written as:
int main () { cout << "Hello World! "; return 0; }

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

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
You might see some C++ programs that runs without the standard namespace library. The using namespace std line can be omitted and replaced with the std keyword, followed by the :: operator for some objects:

Example
```
#include <iostream>

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

C++ Output (Print Text)
The cout object, together with the << operator, is used to output values/print text:

You can add as many cout objects as you want. However, note that it does not insert a new line at the end of the output:

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

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

New Lines
To insert a new line, you can use the \n character:

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

int main() {
  cout << "Hello World! \n";
  cout << "I am learning C++";
  return 0;
}
```
Tip: Two \n characters after each other will create a blank line:

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

int main() {
  cout << "Hello World! \n\n";
  cout << "I am learning C++";
  return 0;
}
```
Another way to insert a new line, is with the endl manipulator:

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

int main() {
  cout << "Hello World!" << endl;
  cout << "I am learning C++";
  return 0;
}
```
Both \n and endl are used to break lines. However, \n is most used.

But what is \n exactly?
The newline character (\n) is called an escape sequence, and it forces the cursor to change its position to the beginning of the next line on the screen. This results in a new line.

Examples of other valid escape sequences are:

Escape Sequence - Description	
\t	Creates a horizontal tab
\\	Inserts a backslash character (\)
\"	Inserts a double quote character

C++ Comments

Comments can be used to explain C++ code, and to make it more readable. It can also be used to prevent execution when testing alternative code. Comments can be singled-lined or multi-lined.

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

Any text between // and the end of the line is ignored by the compiler (will not be executed).

C++ Multi-line Comments
Multi-line comments start with /* and ends with */.

Any text between /* and */ will be ignored by the compiler:

C++ Variables
Variables are containers for storing data values.

In C++, there are different types of variables (defined with different keywords), 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
string - stores text, such as "Hello World". String values are surrounded by double quotes
bool - stores values with two states: true or false

Declaring (Creating) Variables
To create a variable, specify the type and assign it a value:

Syntax
```
type variableName = value;
```
Where 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.

To create a variable that should store a number, look at the following example:

Example
Create a variable called myNum of type int and assign it the value 15:
```
int myNum = 15;
cout << myNum;
```

Example
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;
```

One Value to Multiple Variables
You can also assign the same value to multiple variables in one line:
Example
```
int x, y, z;
```

```
x = y = 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:
Example

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

The general rules for naming variables are:

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

Constants
When you do not want others (or yourself) to change existing variable values, use the const keyword (this will declare the variable as "constant", which means unchangeable and read-only):

Example

```
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:

Example

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

C++ User Input
You have already learned that cout is used to output (print) values. Now we will use cin to get user input.

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

In the following example, the user can input a number, which is stored in the variable x. Then we print the value of x:

Example

```
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++ Data Types
As explained in the Variables chapter, a variable in C++ must be a specified data type:

Example
```
int myNum = 5;               // Integer (whole number)
float myFloatNum = 5.99;     // Floating point number
double myDoubleNum = 9.98;   // Floating point number
char myLetter = 'D';         // Character
bool myBoolean = true;       // Boolean
string myText = "Hello";     // String
```
Basic Data Types
The data type specifies the size and type of information the variable will store:

Data Type	Size	Description
boolean	1 byte	Stores true or false values
char	1 byte	Stores a single character/letter/number, or ASCII values
int	2 or 4 bytes	Stores whole numbers, without decimals
float	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 6-7 decimal digits
double	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits

Numeric Types
Use int when you need to store a whole number without decimals, like 35 or 1000, and float or double when you need a floating point number (with decimals), like 9.99 or 3.14515.

int
```
int myNum = 1000;
cout << myNum;
```

float
```
float myNum = 5.75;
cout << myNum;
```

double
```
double myNum = 19.99;
cout << myNum;
```
float vs. double

The precision of a floating point value indicates how many digits the value can have after the decimal point. The precision of float is only six or seven decimal digits, while double variables have a precision of about 15 digits. Therefore it is safer to use double for most calculations.

Scientific Numbers
A floating point number can also be a scientific number with an "e" to indicate the power of 10:

Example
```
float f1 = 35e3;
double d1 = 12E4;
cout << f1;
cout << d1;
```

Boolean Types
A boolean data type is declared with the bool keyword and can only take the values true or false.

When the value is returned, true = 1 and false = 0.

Example
```
bool isCodingFun = true;
bool isFishTasty = false;
cout << isCodingFun;  // Outputs 1 (true)
cout << isFishTasty;  // Outputs 0 (false)
```

Character Types
The char data type is used to store a single character. The character must be surrounded by single quotes, like 'A' or 'c':

Example
```
char myGrade = 'B';
cout << myGrade;
```
Alternatively, you can use ASCII values to display certain characters:

Example
```
char a = 65, b = 66, c = 67;
cout << a;
cout << b;
cout << c;
```

String Types
The string type is used to store a sequence of characters (text). This is not a built-in type, but it behaves like one in its most basic usage. String values must be surrounded by double quotes:

Example
```
string greeting = "Hello";
cout << greeting;
```

To use strings, you must include an additional header file in the source code, the <string> library:
Example

```
#include <string> // Include the string library
string greeting = "Hello"; // Create a string variable
cout << greeting; // Output string value
```

C++ Operators
Operators are used to perform operations on variables and values.

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

O	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Assignment Operators
Assignment operators are used to assign values to variables.

In the example below, we use the assignment operator (=) to assign the value 10 to a variable called x:

Example

```
int x = 10;
```

The addition assignment operator (+=) adds a value to a variable:

Example

```
int x = 10;
x += 5;
```

A list of all assignment operators:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operators
Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

The return value of a comparison is either 1 or 0, which means true (1) or false (0). These values are known as Boolean values, and you will learn more about them in the Booleans and If..Else chapter.

In the following example, we use the greater than operator (>) to find out if 5 is greater than 3:
Example

```
int x = 5;
int y = 3;
cout << (x > y); // returns 1 (true) because 5 is greater than 3
```

A list of all comparison operators:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators
As with comparison operators, you can also test for true (1) or false (0) values with logical operators.

Logical operators are used to determine the logic between variables or values:

Op	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

C++ Math
C++ has many functions that allows you to perform mathematical tasks on numbers.

Max and min
The max(x,y) function can be used to find the highest value of x and y:

Example
cout << max(5, 10);
And the min(x,y) function can be used to find the lowest value of x and y:

Example

```
cout << min(5, 10);
```

C++ <cmath> Header

Other functions, such as sqrt (square root), round (rounds a number) and log (natural logarithm), can be found in the <cmath> header file:

Example

```
// Include the cmath library
#include <cmath>

cout << sqrt(64);
cout << round(2.6);
cout << log(2);
```

C++ Booleans

Very often, in programming, you will need a data type that can only have one of two values, like:

YES / NO
ON / OFF
TRUE / FALSE

For this, C++ has a bool data type, which can take the values true (1) or false (0).

Boolean Values

A boolean variable is declared with the bool keyword and can only take the values true or false:

Example

```
bool isCodingFun = true;
bool isFishTasty = false;
cout << isCodingFun;  // Outputs 1 (true)
cout << isFishTasty;  // Outputs 0 (false)
```

From the example above, you can read that a true value returns 1, and false returns 0.

Boolean Expression

A Boolean expression returns a boolean value that is either 1 (true) or 0 (false).

This is useful to build logic, and find answers.

You can use a comparison operator, such as the greater than (>) operator, to find out if an expression (or variable) is true or false:

Example

```
int x = 10;
int y = 9;
cout << (x > y); // returns 1 (true), because 10 is higher than 9
```

Or even easier:

Example

```
cout << (10 > 9); // returns 1 (true), because 10 is higher than 9
```
In the examples below, we use the equal to (==) operator to evaluate an expression:

Example
```
int x = 10;
cout << (x == 10);  // returns 1 (true), because the value of x is equal to 10
```
Example
```
cout << (10 == 15);  // returns 0 (false), because 10 is not equal to 15
```
Real Life Example
Let's think of a "real life example" where we need to find out if a person is old enough to vote.

In the example below, we use the >= comparison operator to find out if the age (25) is greater than OR equal to the voting age limit, which is set to 18:

Example
```
int myAge = 25;
int votingAge = 18;

cout << (myAge >= votingAge); // returns 1 (true), meaning 25 year olds are allowed to vote!
```
Cool, right? An even better approach (since we are on a roll now), would be to wrap the code above in an if...else statement, so we can perform different actions depending on the result:

Example
Output "Old enough to vote!" if myAge is greater than or equal to 18. Otherwise output "Not old enough to vote.":

```
int myAge = 25;
int votingAge = 18;

if (myAge >= votingAge) {
  cout << "Old enough to vote!";
} else {
  cout << "Not old enough to vote.";
}

// Outputs: Old enough to vote!
```

C++ Conditions and If Statements
You already know that C++ supports the usual logical conditions from mathematics:
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- Equal to a == b
- Not Equal to: a != b

You can use these conditions to perform different actions for different decisions.

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
- Use the if statement to specify a block of C++ code to be executed if a condition is true.

Syntax
```
if (condition) {
  // block of code to be executed if the condition is true
}
```
Note that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.

In the example below, we test two values to find out if 20 is greater than 18. If the condition is true, print some text:

Example
```
if (20 > 18) {
  cout << "20 is greater than 18";
}
```
We can also test variables:

Example
```
int x = 20;
int y = 18;
if (x > y) {
  cout << "x is greater than y";
}
```

The else Statement
Use the else statement to specify a block of code to be executed if the condition is false.

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

The else if Statement
Use the else if statement to specify a new condition if the first condition is false.

Syntax
```
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
}
```

C++ ShortHand If Else
ShortHand If...Else (Ternary Operator)
There is also a short-hand if else, which is known as the ternary operator because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

Syntax
```
variable = (condition) ? expressionTrue : expressionFalse;
```
Instead of writing:

Example
```
int time = 20;
if (time < 18) {
  cout << "Good day.";
} else {
  cout << "Good evening.";
}
```
You can simply write:

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

C++ Switch Statements
Use the switch statement to select one of many code blocks to be executed.

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

This is how it works:

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 in this chapter

C++ 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++ While Loop

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

Syntax

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

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

Example

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

The 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.

Syntax

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

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

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

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:

Syntax
```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```
Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

The example below will print the numbers 0 to 4:

Example
```
for (int i = 0; i < 5; i++) {
  cout << i << "\n";
}
```
Example explained
Statement 1 sets a variable before the loop starts (int i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5). If the condition is true, the loop will start over again, if it is false, the loop will end.

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

Another Example
This example will only print even values between 0 and 10:

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

Nested Loops
It is also possible to place a loop inside another loop. This is called a nested loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example
```
// Outer loop
for (int i = 1; i <= 2; ++i) {
  cout << "Outer: " << i << "\n"; // Executes 2 times
  // Inner loop
  for (int j = 1; j <= 3; ++j) {
    cout << " Inner: " << j << "\n"; // Executes 6 times (2 * 3)
  }
}
```

The foreach Loop
There is also a "for-each loop" (introduced in C++ version 11 (2011), which is used exclusively to loop through elements in an array (or other data sets):

Syntax

```
for (type variableName : arrayName) {
  // code block to be executed
}
```

The following example outputs all elements in an array, using a "for-each loop":

Example

```
int myNumbers[5] = {10, 20, 30, 40, 50};
for (int i : myNumbers) {
  cout << i << "\n";
}
```

C++ Break
You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

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

This example jumps out of the loop when i is equal to 4:
Example

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

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.

This example skips the value of 4:
Example

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