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 5 major times in 2011, 2014, 2017, 2020, and 2023 to C++11, C++14, C++17, C++20, and C++23.

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++ support classes and objects, while C does not.

C++ Syntax

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

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

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.

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++ Statements

A **computer program** is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called **statements**.

The following statement "instructs" the compiler to print the text "Hello World" to the screen:

Example

```
cout << "Hello World!";
```

It is important that you end the statement with a semicolon ;

If you forget the semicolon (;), an error will occur and the program will not run:

C++ Output (Print Text)

The cout object, together with the << operator, is used to output values/print text:

Example

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

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

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;
}
```

You can also use another << operator and place the \n character after the text, like this:

Example

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

int main() {
  cout << "Hello World!" << "\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).

This example uses a single-line comment before a line of code:

Example

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

This example uses a single-line comment at the end of a line of code:

Example

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

C++ Multi-line Comments

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

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

Example

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

C++ Variables

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;
```

You can also declare a variable without assigning the value, and assign the value later:

Example

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

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

Example

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

Other Types

A demonstration of other data types:

Example

```
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)
```

Display Variables

The cout object is used together with the << operator to display variables.

To combine both text and a variable, separate them with the << operator:

Example

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

Add Variables Together

To add a variable to another variable, you can use the + operator:

Example

```
int x = 5;
int y = 6;
int sum = x + y;
cout << sum;
```

Declare Many Variables

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

Example

```
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

C++ 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;
```

Notes On Constants

When you declare a constant variable, it must be assigned with a value:

Example

Like this:

```
const int minutesPerHour = 60;
```

This however, **will not work**:

```
const int minutesPerHour;
minutesPerHour = 60; // error
```

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
```

NOTE:

- ✓ 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

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

C++ Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + **operator** to add together two values:

Example

```
int x = 100 + 50;
```

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

Example

```
int sum1 = 100 + 50;        // 150 (100 + 50)
int sum2 = sum1 + 250;      // 400 (150 + 250)
int sum3 = sum2 + sum2;     // 800 (400 + 400)
```

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:

Operator	**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++ String Concatenation

The + operator can be used between strings to add them together to make a new string. This is called **concatenation**:

Example

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName + lastName;
cout << fullName;
```

In the example above, we added a space after firstName to create a space between John and Doe on output. However, you could also add a space with quotes (" " or ' '):

Example

```
string firstName = "John";
string lastName = "Doe";
string fullName = firstName + " " + lastName;
cout << fullName;
```

Append

A string in C++ is actually an object, which contain functions that can perform certain operations on strings. For example, you can also concatenate strings with the append() function:

Example

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName.append(lastName);
cout << fullName;
```

C++ uses the + operator for both **addition** and **concatenation**.

Numbers are added. Strings are concatenated.

C++ String Length

To get the length of a string, use the length() function:

Example

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << "The length of the txt string is: " << txt.length();
```

Tip: You might see some C++ programs that use the size() function to get the length of a string. This is just an alias of length(). It is completely up to you if you want to use length() or size():

Example

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << "The length of the txt string is: " << txt.size();
```

C++ Access Strings

You can access the characters in a string by referring to its index number inside square brackets [].

This example prints the **first character** in **myString**:

Example

```
string myString = "Hello";
cout << myString[0];
// Outputs H
```

Note: String indexes start with 0: [0] is the first character. [1] is the second character, etc.

This example prints the **second character** in **myString**:

Example

```
string myString = "Hello";
cout << myString[1];
// Outputs e
```

To print the last character of a string, you can use the following code:

Example

```
string myString = "Hello";
cout << myString[myString.length() - 1];
// Outputs o
```

Change String Characters

To change the value of a specific character in a string, refer to the index number, and use single quotes:

Example

```
string myString = "Hello";
myString[0] = 'J';
cout << myString;
// Outputs Jello instead of Hello
```

The at() function

The <string> library also has an at() function that can be used to access characters in a string:

Example

```
string myString = "Hello";
cout << myString; // Outputs Hello
```

```
cout << myString.at(0);  // First character
cout << myString.at(1);  // Second character
cout << myString.at(myString.length() - 1);  // Last character

myString.at(0) = 'J';
cout << myString;  // Outputs Jello
```

User Input Strings

It is possible to use the extraction operator >> on cin to store a string entered by a user:

Example

```
string firstName;
cout << "Type your first name: ";
cin >> firstName; // get user input from the keyboard
cout << "Your name is: " << firstName;

// Type your first name: John
// Your name is: John
```

However, cin considers a space (whitespace, tabs, etc) as a terminating character, which means that it can only store a single word (even if you type many words):

Example

```
string fullName;
cout << "Type your full name: ";
cin >> fullName;
cout << "Your name is: " << fullName;

// Type your full name: John Doe
// Your name is: John
```

From the example above, you would expect the program to print "John Doe", but it only prints "John".

That's why, when working with strings, we often use the getline() function to read a line of text. It takes cin as the first parameter, and the string variable as second:

Example

```
string fullName;
cout << "Type your full name: ";
getline (cin, fullName);
cout << "Your name is: " << fullName;

// Type your full name: John Doe
// Your name is: John Doe
```

C++ String Namespace

Omitting Namespace

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

Example

```
#include <iostream>
#include <string>
// using namespace std; - Remove this line

int main() {
  std::string greeting = "Hello";
  std::cout << greeting;
  return 0;
}
```

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> Library

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++ Conditions and If Statements

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";
}
```

Example explained

In the example above we use two variables, **x** and **y**, to test whether x is greater than y (using the > operator). As x is 20, and y is 18, and we know that 20 is greater than 18, we print to the screen that "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."
```

Example explained

In the example above, time (20) is greater than 18, so the condition is false. Because of this, we move on to the else condition and print to the screen "Good evening". If the time was less than 18, the program would print "Good day".

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
}
```

Example

```
int time = 22;
if (time < 10) {
  cout << "Good morning.";
} else if (time < 20) {
  cout << "Good day.";
} else {
  cout << "Good evening.";
}
// Outputs "Good evening."
```

Example explained

In the example above, time (22) is greater than 10, so the **first condition** is false. The next condition, in the else if statement, is also false, so we move on to the else condition since **condition1** and **condition2** is both false - and print to the screen "Good evening".

C++ Switch

Use the switch statement to select one of many code blocks to be executed.

Syntax

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

The example below uses the weekday number to calculate the weekday name:

Example

```
int day = 4;
switch (day) {
  case 1:
    cout << "Monday";
    break;
  case 2:
    cout << "Tuesday";
    break;
  case 3:
    cout << "Wednesday";
    break;
  case 4:
    cout << "Thursday";
    break;
  case 5:
    cout << "Friday";
    break;
  case 6:
    cout << "Saturday";
    break;
  case 7:
    cout << "Sunday";
    break;
}
// Outputs "Thursday" (day 4)
```

The break Keyword

When C++ reaches a break keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

The default Keyword

The default keyword specifies some code to run if there is no case match:

Example

```
int day = 4;
switch (day) {
  case 6:
    cout << "Today is Saturday";
    break;
  case 7:
    cout << "Today is Sunday";
    break;
  default:
    cout << "Looking forward to the Weekend";
}
// Outputs "Looking forward to the Weekend"
```

C++ While Loop

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" (also known as ranged-based for loop), which is used exclusively to loop through elements in an array (or other data structures):

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";
}
```

Break and Continue in While Loop

You can also use break and continue in while loops:

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++;
}
```