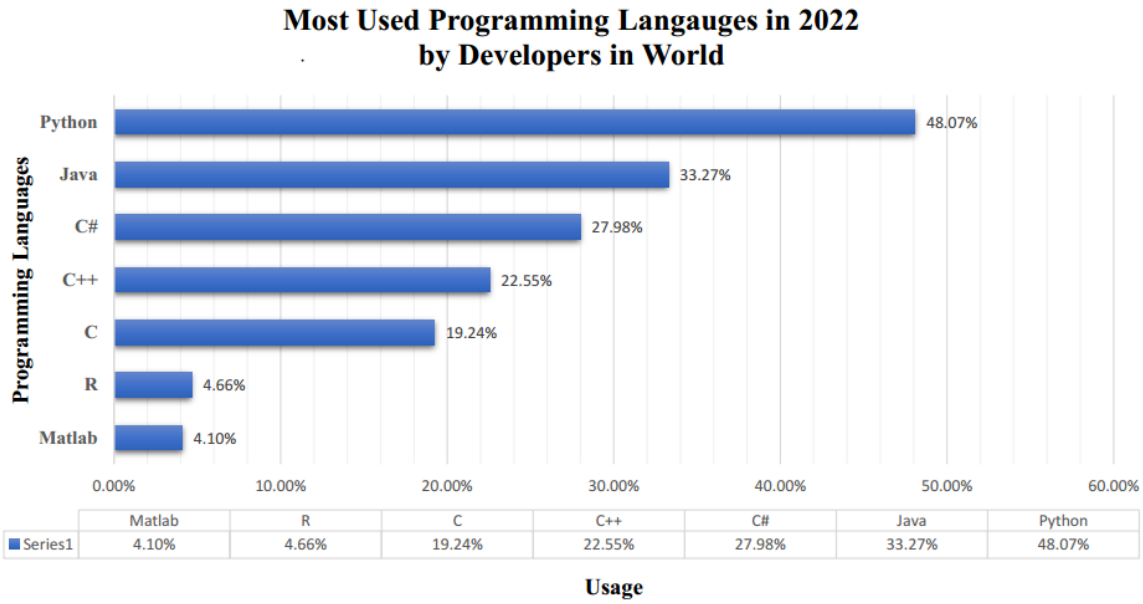
Instructor Name: Engr. Zia Ur Rehman
Department: Computer Engineering

Introduction to
C++ Programming Language

Most Used Programming Languages by developers worldwide in 2022



The statista.com conducted a worldwide online survey in July 2022 of 71,547 software developers. The chart was developed by myself in Microsoft excel on the basis of data obtained from statista.com.

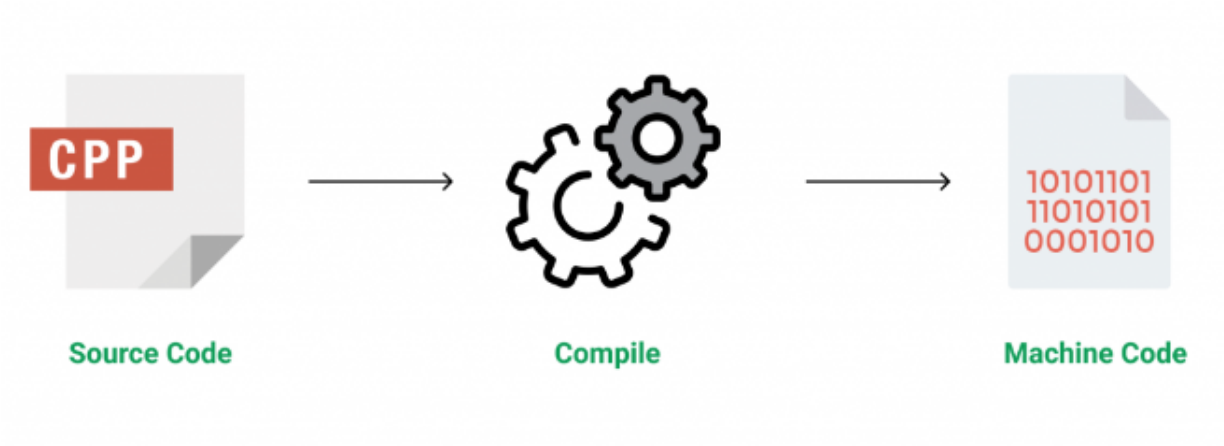
What is C++?

- **Foundation:** 1980, by Bjarne Stroustrup at bell laboratories, located in the USA.
- **Category:** Imperative, Object Oriented.
- C++ is a cross-platform language that can be used to create high-performance applications.
- Since C++ is an attempt to add object-oriented features (plus other improvements) to C, earlier it was called "C with Objects", and later on the name C++ was suggested.
- C++ gives programmers a high level of control over system resources and memory.

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!

Compiler Based Language



A **compiler** is a special program that translates a programming language's source code into machine code, bytecode or another programming language. The source code is typically written in a high-level, human-readable language such as Java or C++.

C++ Get Started

To start using C++, you need two things:

- Text Editor, where you were able to write your code.
- Compiler, to translate the C++ code into a language that the computer will understand.

C++ Install IDE

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

- **Note:** Web-based IDE's can work as well, but functionality is limited.
- We will use DEV C++, which we believe is a good place to start.

You can find the latest version: https://sourceforge.net/projects/orwelldevcpp/

C++ Quickstart

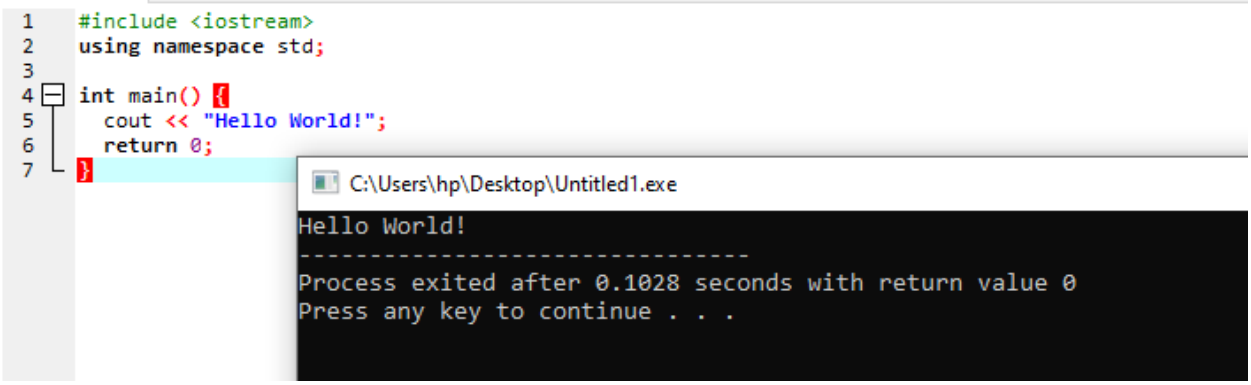
- Let's create our first C++ file.
- Open DEV C++ and go to File ¿ New ¿ Source File.
- Write the following C++ code and save the file as myfirstprogram.cpp (File ¿ Save File as):

Example 01: myfirstprogram.cpp

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

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

Don't worry if you don't understand the code above - we will discuss it in detail in later chapters. For now, focus on how to run the code. **In order to run the code press F11 to compile and execute.**



Congratulations! You have now written and executed your first C++ program.

C++ Syntax

Example 01: myfirstprogram.cpp

```cpp
#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 be written as: int main () { cout << "Hello World! "; return 0; }
- **Remember:** The compiler ignores white spaces. However, multiple lines make 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.

C++ Output (Print Text)

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

Example 02

```
#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:

Example 03

```cpp
#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 04

```cpp
#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 05

```cpp
#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 the 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.

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 06

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

C++ Multi-line Comments

Multi-line comments start with /* and ends with */. Any text between /* and */ will be ignored by the compiler:

Example 07

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

Single or multi-line comments?

It is up to you which you want to use. Normally, we use // for short comments, and /* */ for longer.

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 08

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:

```
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

A demonstration of other data types:

Example 09

```
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 10

```
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 11

```
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 12

```
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:

```
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 13

```
// 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 white spaces 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 14

```
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

- 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 15

```
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 (>>)

Creating a Simple Calculator

In this example, the user must input two numbers. Then we print the sum by calculating (adding) the two numbers:

Example 16

```
int x, y;
int sum;
cout << "Type a number: ";
cin >> x;
cout << "Type another number: ";
cin >> y;
sum = x + y;
cout << "Sum is: " << sum;
```

C++ Data Types

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

Example 17

```
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

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

Example 18

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

float myNum = 5.75;
cout << myNum;

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.

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**.

```
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':

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

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

// Include the string library
#include <string>

// Create a string variable
string greeting = "Hello";

// Output string value
cout << greeting;
```

C++ Operators

Operators are used to performing operations on variables and values. In the example below, we use the + **operator** to add together two values: 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:

```
int x = 100 + 50;

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

C++ divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators

- Bit-wise operators

Arithmetic operators are used to performing common mathematical operations.

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

```
int x = 10;

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

The addition assignment operator (+=) adds a value to a variable: 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 comparing two values (or variables). This is important in programming because it helps us to find answer 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.

```
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 determining 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++ Strings

Strings are used for storing text. A **string** variable contains a collection of characters surrounded by double quotes. To use strings, you must include an additional header file in the source code, the $< string >$ library:

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

// Create a string variable
string greeting = "Hello";
```

String Concatenation

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

```
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 ' '):*/

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

Adding Numbers and Strings

Warning! C++ uses the + operator for both **addition** and **concatenation**. Numbers are added. Strings are concatenated.

```
// If you add two numbers, the result will be a number:

int x = 10;
int y = 20;
int z = x + y;      // z will be 30 (an integer)

// If you add two strings, the result will be a string concatenation:

string x = "10";
string y = "20";
string z = x + y;   // z will be 1020 (a string)

// If you try to add a number to a string, an error occurs:

string x = "10";
int y = 20;
string z = x + y;
```

String Length

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

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

Access Strings

You can access the characters in a string by referring to its index number inside square brackets [ ]. **Note**: String indexes start with 0: [0] is the first character. [1] is the second character, etc.

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

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

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

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

Strings - Special Characters

Because strings must be written within quotes, C++ will misunderstand this string, and generate an error:

```
string txt = "We are the so-called "Vikings" from the north.";
```

The solution to avoid this problem is to use the **backslash escape character**. The backslash \ escape character turns special characters into string characters:

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

The sequence \" inserts a double quote in a string:

```
string txt = "We are the so-called \"Vikings\" from the north.";

// The sequence \'  inserts a single quote in a string:

string txt = "It\'s alright.";

// The sequence \\  inserts a single backslash in a string:

string txt = "The character \\ is called backslash.";
```

User Input Strings

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

```
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 display a single word (even if you type many words):
*/

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:
*/
```

```
26
27 string fullName;
28 cout << "Type your full name: ";
29 getline (cin, fullName);
30 cout << "Your name is: " << fullName;
31
32 // Type your full name: John Doe
33 // Your name is: John Doe
```

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: And the `min(x,y)` function can be used to find the lowest value of x and y: Other functions, such as sqrt (square root), round (rounds a number) and log (natural logarithm), can be found in the $< cmath >$ header file:

```
1  // Include the cmath library
2  #include <cmath>
3  #include <iostream>
4  using namespace st;
5
6  int main()
7  {
8  cout << max(5, 10);
9  cout << min(5, 10);
10
11
12 cout << sqrt(64);
13 cout << round(2.6);
14 cout << log(2);
15
16 }
```

Function	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x
asin(x)	Returns the arcsine of x
atan(x)	Returns the arctangent of x
cbrt(x)	Returns the cube root of x

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:

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

cout << (10 > 9); // returns 1 (true), because 10 is higher than 9

int x = 10;
cout << (x == 10);  // returns 1 (true), because the value of x is equal to 10

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:
*/
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:
*/

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

Note! that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.

```
// Syntax of if statement

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

/*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:
*/

if (20 > 18) {
  cout << "20 is greater than 18";
}

// We can also test variables:

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

// Syntax of else statement

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
}

// Example

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

Short Hand 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:

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

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

```
}

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. 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

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


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)
```

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.

```
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"
```