C++ For beginners

> Patience is tool for programmer’s success.

C++ is one of the most popular programming language in the world . It is used for the high performance applications , video games , device drivers , web browsers , servers , operating systems etc. It is used by Adobe , Google , NASA , Microsoft etc.

C++ is one of the fastest and most efficient language . It has influenced many programming languages.

Some of the popular IDEs for C++ are CLion, DevC++ , TurboC++ etc.

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.

To insert a new line, you can use the `\n` character or `endl` maniplator.

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

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

C++ Variables

Variables are containers for storing data values.

- `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

To create a variable, specify the type and assign it a value:

```
type variableName = value;
```

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

When you don't want to override existing variable values, Use the `const` keyword (this will declare the variable as "constant", which means **unchangeable and read-only**)

```
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
```

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 7 decimal digits
`double`	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits

Arithmetic Operators

Arithmetic operators are used to perform 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

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.

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

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

Operator	Name	Description	Example
||	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)

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;
```

String Concatenation

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

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 + " " + lastName;
cout << fullName;

//----------------or--------------------

fullName = firstName.append(lastName);
cout << fullName;
```

C++ Math

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

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

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
ceil(x)	Returns the value of x rounded up to its nearest integer
cos(x)	Returns the cosine of x
cosh(x)	Returns the hyperbolic cosine of x
exp(x)	Returns the value of Ex
expm1(x)	Returns ex -1
fabs(x)	Returns the absolute value of a floating x
fdim(x, y)	Returns the positive difference between x and y
floor(x)	Returns the value of x rounded down to its nearest integer
hypot(x, y)	Returns sqrt(x2 +y2) without intermediate overflow or underflow
fma(x, y, z)	Returns x*y+z without losing precision
fmax(x, y)	Returns the highest value of a floating x and y
fmin(x, y)	Returns the lowest value of a floating x and y
fmod(x, y)	Returns the floating point remainder of x/y
pow(x, y)	Returns the value of x to the power of y
sin(x)	Returns the sine of x (x is in radians)
sinh(x)	Returns the hyperbolic sine of a double value
tan(x)	Returns the tangent of an angle
tanh(x)	Returns the hyperbolic tangent of a double value

Boolean Values

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

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

C++ Conditions and If Statements

C++ supports the usual logical conditions from mathematics.

```
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
}
```

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

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

C++ Loops

Loops can execute a block of code as long as a specified condition is reached.

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

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

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

C++ Break

The `break` statement is used to jump out of a **loop**.

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

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++ Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type, specify the name of the array followed by **square brackets** and specify the number of elements it should store:

```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
int myNum[3] = {10, 20, 30};
```

You can loop through the array elements with the `for` loop.

```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < 4; i++) {
  cout << cars[i] << "\n";
}
```

Multi-Dimensional Arrays

A multi-dimensional array is an array of arrays.

To declare a multi-dimensional array, define the variable type, specify the name of the array followed by square brackets which specify how many elements the main array has, followed by another set of square brackets which indicates how many elements the sub-arrays have:

```
string letters[2][4] = {
  { "A", "B", "C", "D" },
  { "E", "F", "G", "H" }
};
cout << letters[0][2];  // Outputs "C"
```

C++ Structures

Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a **member** of the structure.

Create a Structure

To create a structure, use the `struct` keyword and declare each of its members inside curly braces.

```
struct {             // Structure declaration
  int myNum;         // Member (int variable)
  string myString;   // Member (string variable)
} myStructure;       // Structure variable
```

To access members of a structure, use the dot syntax (`.`):

```
struct {
  int myNum;
  string myString;
} myStructure;

// Assign values to members of myStructure
myStructure.myNum = 1;
myStructure.myString = "Hello World!";

// Print members of myStructure
cout << myStructure.myNum << "\n";
cout << myStructure.myString << "\n";
```

You can use a comma (`,`) to use one structure in many variables:

```
struct {
  string brand;
  string model;
  int year;
} myCar1, myCar2; // We can add variables by separating them with a comma here

// Put data into the first structure
myCar1.brand = "BMW";
myCar1.model = "X5";
myCar1.year = 1999;

// Put data into the second structure
myCar2.brand = "Ford";
myCar2.model = "Mustang";
myCar2.year = 1969;
```

```
// Print the structure members
cout << myCar1.brand << " " << myCar1.model << " " << myCar1.year << "\n";
cout << myCar2.brand << " " << myCar2.model << " " << myCar2.year << "\n";
```

Named Structures

By giving a name to the structure, you can treat it as a data type. This means that you can create variables with this structure anywhere in the program at any time.

```
struct myDataType { // This structure is named "myDataType"
  int myNum;
  string myString;
};
```