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

Difference between C and C++

Introduction to 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

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

Example

```
#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.

Introduction to c++

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

C++ Output (Print Text)

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;
```

Introduction to c++

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

C++ Variables

C++ Variables

Primitive and Non-Primitive Data Types

Primitive Data Types

Data Type	Memory Occupied	Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	Range	0 to 65,535
signed short int	Range	-32768 to 32767
long int	4bytes	-2,147,483,647 to 2,147,483,647
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)

Introduction to c++

double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character

The built-in data types in C++, are known as the Primitive Data Types.

Following are the 7 basic data types in C++.

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

The memory occupied and the range of each of these data types s as follows:

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

Non – Primitive Data Types

Non – Primitive Data Types are the ones that are also known as the ***user-defined data types*** as they can hold the data as per the choice of the programmer. We will read about these Non – Primitive Data Types in detail later on.

Class and structure

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:

Introduction to c++

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 = 10;

cout << myNum;
```

C++ User Input

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
```

Creating a Simple Calculator

Introduction to c++

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

Example

```
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++ Operators

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)
```

Introduction to c++

C++ Operators

In this tutorial, we will learn about the different types of operators in C++ with the help of examples. In programming, an operator is a symbol that operates on a value or a variable.

Operators are symbols that perform operations on variables and values. For example, `+` is an operator used for addition, while `-` is an operator used for subtraction.
Operators in C++ can be classified into 6 types:

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators
6. Other Operators

1. C++ Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on variables and data. For example,

```
a + b;
```

Here, the `+` operator is used to add two variables `a` and `b`. Similarly there are various other arithmetic operators in C++.

Operator	Operation

Introduction to c++

`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`%`	Modulo Operation (Remainder after division)

Example 1: Arithmetic Operators

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

int main() {
    int a, b;
    a = 7;
    b = 2;

    // printing the sum of a and b
    cout << "a + b = " << (a + b) << endl;

    // printing the difference of a and b
    cout << "a - b = " << (a - b) << endl;

    // printing the product of a and b
    cout << "a * b = " << (a * b) << endl;

    // printing the division of a by b
    cout << "a / b = " << (a / b) << endl;

    // printing the modulo of a by b
    cout << "a % b = " << (a % b) << endl;
```

```
    return 0;
}
```

Run Code

Output

```
a + b = 9
a - b = 5
a * b = 14
a / b = 3
a % b = 1
```

Here, the operators `+`, `-` and `*` compute addition, subtraction, and multiplication respectively as we might have expected.

/ Division Operator

Note the operation `(a / b)` in our program. The `/` operator is the division operator.

As we can see from the above example, if an integer is divided by another integer, we will get the quotient. However, if either divisor or dividend is a floating-point number, we will get the result in decimals.

```
In C++,



7/2 is 3

7.0 / 2 is 3.5

7 / 2.0 is 3.5

7.0 / 2.0 is 3.5
```

% Modulo Operator

The modulo operator `%` computes the remainder. When `a = 9` is divided by `b = 4`, the remainder is **1**.

> **Note:** The `%` operator can only be used with integers.

Increment and Decrement Operators

C++ also provides increment and decrement operators: `++` and `--` respectively.

- `++` increases the value of the operand by **1**
- `--` decreases it by **1**

For example,

```
int num = 5;

// increment operator
++num;  // 6
```

Here, the code `++num;` increases the value of `num` by **1**.

Example 2: Increment and Decrement Operators

```
// Working of increment and decrement operators

#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 100, result_a, result_b;

    // incrementing a by 1 and storing the result in result_a
    result_a = ++a;
    cout << "result_a = " << result_a << endl;
```

```
    // decrementing b by 1 and storing the result in result_b
    result_b = --b;
    cout << "result_b = " << result_b << endl;

    return 0;
}
```

Output

```
result_a = 11
result_b = 99
```

In the above program, we have used the `++` and `--` operators as **prefixes (++a and --b)**. However, we can also use these operators as **postfix (a++ and b--)**.

2. C++ Assignment Operators

In C++, assignment operators are used to assign values to variables. For example,

```
// assign 5 to a
a = 5;
```

Here, we have assigned a value of `5` to the variable `a`.

Operator	Example	Equivalent to
`=`	`a = b;`	`a = b;`
`+=`	`a += b;`	`a = a + b;`
`-=`	`a -= b;`	`a = a - b;`
`*=`	`a *= b;`	`a = a * b;`

/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Example 3: Assignment Operators

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

int main() {
    int a, b;

    // 2 is assigned to a
    a = 2;

    // 7 is assigned to b
    b = 7;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "\nAfter a += b;" << endl;

    // assigning the sum of a and b to a
    a += b;  // a = a +b
    cout << "a = " << a << endl;

    return 0;
}
```

Run Code

Output

```
a = 2
b = 7

After a += b;
```

```
a = 9
```

3. C++ Relational Operators

A relational operator is used to check the relationship between two operands. For example,

```
// checks if a is greater than b
a > b;
```

Here, `>` is a relational operator. It checks if `a` is greater than `b` or not. If the relation is **true**, it returns **1** whereas if the relation is **false**, it returns **0**.

Operator	Meaning	Example
`==`	Is Equal To	`3 == 5` gives us **false**
`!=`	Not Equal To	`3 != 5` gives us **true**
`>`	Greater Than	`3 > 5` gives us **false**
`<`	Less Than	`3 < 5` gives us **true**
`>=`	Greater Than or Equal To	`3 >= 5` give us **false**
`<=`	Less Than or Equal To	`3 <= 5` gives us **true**

Introduction to c++

Example 4: Relational Operators

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

int main() {
    int a, b;
    a = 3;
    b = 5;
    bool result;

    result = (a == b);   // false
    cout << "3 == 5 is " << result << endl;

    result = (a != b);  // true
    cout << "3 != 5 is " << result << endl;

    result = a > b;   // false
    cout << "3 > 5 is " << result << endl;

    result = a < b;   // true
    cout << "3 < 5 is " << result << endl;

    result = a >= b;  // false
    cout << "3 >= 5 is " << result << endl;

    result = a <= b;  // true
    cout << "3 <= 5 is " << result << endl;

    return 0;
}
```

Run Code

Output

```
3 == 5 is 0
3 != 5 is 1
3 > 5 is 0
3 < 5 is 1
```

```
3 >= 5 is 0
3 <= 5 is 1
```

> **Note**: Relational operators are used in decision-making and loops.

4. C++ Logical Operators

Logical operators are used to check whether an expression is **true** or **false**. If the expression is **true**, it returns **1** whereas if the expression is **false**, it returns **0**.

Operator	Example	Meaning
`&&`	expression1 **&&** expression2	Logical AND. True only if all the operands are true.
`||`	expression1 **||** expression2	Logical OR. True if at least one of the operands is true.
`!`	**!**expression	Logical NOT. True only if the operand is false.

In C++, logical operators are commonly used in decision making. To further understand the logical operators, let's see the following examples,

```
Suppose,
a = 5
b = 8
```

```
Then,

(a > 3) && (b > 5) evaluates to true
(a > 3)  && (b < 5) evaluates to false

(a > 3) || (b > 5) evaluates to true
(a > 3) || (b < 5) evaluates to true
(a < 3) || (b < 5) evaluates to false

!(a < 3) evaluates to true
!(a > 3) evaluates to false
```

Example 5: Logical Operators

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

int main() {
    bool result;

    result = (3 != 5) && (3 < 5);     // true
    cout << "(3 != 5) && (3 < 5) is " << result << endl;

    result = (3 == 5) && (3 < 5);    // false
    cout << "(3 == 5) && (3 < 5) is " << result << endl;

    result = (3 == 5) && (3 > 5);    // false
    cout << "(3 == 5) && (3 > 5) is " << result << endl;

    result = (3 != 5) || (3 < 5);    // true
    cout << "(3 != 5) || (3 < 5) is " << result << endl;

    result = (3 != 5) || (3 > 5);    // true
    cout << "(3 != 5) || (3 > 5) is " << result << endl;

    result = (3 == 5) || (3 > 5);    // false
    cout << "(3 == 5) || (3 > 5) is " << result << endl;
```

Introduction to c++

```
    result = !(5 == 2);    // true
    cout << "!(5 == 2) is " << result << endl;

    result = !(5 == 5);    // false
    cout << "!(5 == 5) is " << result << endl;

    return 0;
}
```

Run Code

Output

```
(3 != 5) && (3 < 5) is 1
(3 == 5) && (3 < 5) is 0
(3 == 5) && (3 > 5) is 0
(3 != 5) || (3 < 5) is 1
(3 != 5) || (3 > 5) is 1
(3 == 5) || (3 > 5) is 0
!(5 == 2) is 1
!(5 == 5) is 0
```

Explanation of logical operator program

- `(3 != 5) && (3 < 5)` evaluates to **1** because both operands `(3 != 5)` and `(3 < 5)` are **1** (true).
- `(3 == 5) && (3 < 5)` evaluates to **0** because the operand `(3 == 5)` is **0** (false).
- `(3 == 5) && (3 > 5)` evaluates to **0** because both operands `(3 == 5)` and `(3 > 5)` are **0** (false).
- `(3 != 5) || (3 < 5)` evaluates to **1** because both operands `(3 != 5)` and `(3 < 5)` are **1** (true).
- `(3 != 5) || (3 > 5)` evaluates to **1** because the operand `(3 != 5)` is **1** (true).
- `(3 == 5) || (3 > 5)` evaluates to **0** because both operands `(3 == 5)` and `(3 > 5)` are **0** (false).
- `!(5 == 2)` evaluates to **1** because the operand `(5 == 2)` is **0** (false).

- `!(5 == 5)` evaluates to **0** because the operand `(5 == 5)` is **1** (true).

5. C++ Bitwise Operators

In C++, bitwise operators are used to perform operations on individual bits. They can only be used alongside `char` and `int` data types.

Operator	Description
&	Binary AND
|	Binary OR
^	Binary XOR
~	Binary One's Complement
<<	Binary Shift Left
>>	Binary Shift Right

To learn more, visit C++ bitwise operators.

6. Other C++ Operators

Here's a list of some other common operators available in C++. We will learn about them in later tutorials.

Introduction to c++

Operator	Description	Example
`sizeof`	returns the size of data type	`sizeof(int); // 4`
`?:`	returns value based on the condition	`string result = (5 > 0) ? "even" : "odd"; // "even"`
`&`	represents memory address of the operand	`&num; // address of num`
`.`	accesses members of struct variables or class objects	`s1.marks = 92;`
`->`	used with pointers to access the class or struct variables	`ptr->marks = 92;`
`<<`	prints the output value	`cout << 5;`
`>>`	gets the input value	`cin >> num;`