Lecture 1: Introduction to C++

- C++ is a compiler-based language. That is C++ programs used to be compiled and their executable file is used to run it. Due to which C++ is a relatively faster language than Java and Python.
- C++ allows us to allocate the memory of a variable or an array in run time. This is known as **Dynamic Memory Allocation.**
- C++ is a case-sensitive programming language.

C++ Program Structure

Let us look at a simple code that would print the words *Hello World*.

```
#include <iostream>
using namespace std;
// main() is where program execution begins.
int main() {
   cout << "Hello World"; // prints Hello World
   return 0;
}
```

Let us look at the various parts of the above program –

- The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header **<iostream>** is needed.
- The line **int main()** is the main function where program execution begins.

- The next line **cout << "Hello World";** causes the message "Hello World" to be displayed on the screen.
- The next line **return 0;** terminates main( ) function and causes it to return the value 0 to the calling process.

C++ Variables

Variables are containers for storing data values.

In C++, there are different **types** of variables (defined with different keywords), for example:

Basic Data Types

The data type specifies the size and type of information the variable will store:

Data Type	Description
int	Stores whole numbers, without decimals
float	Stores fractional numbers, containing one or more decimals. Sufficient for storing 7 decimal digits
double	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits
boolean	Stores true or false values
char	Stores a single character/letter/number, or ASCII values
string	Stores text, such as "Hello World". String values are surrounded by double quotes

Declaring (Creating) Variables

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

Syntax
type variable = value;

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)
```

Example

Create a variable called **myNum** of type `int` and assign it the value **15**:

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

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

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;
```

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:

The general rules for constructing names for variables (unique identifiers) 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.

Example

```
// Good
int minutesPerHour = 60;

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

Constants

When you do not want others (or yourself) to override 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'
```

```
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
```

C++ Operators

C++ divides the operators into the following groups:

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

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

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**:

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.

Note: The return value of a comparison is either true (1) or false (0).

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

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
```

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++ Conditions and If Statements

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
}
```

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
}
```

Example

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

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;
  case y:
    // code block
    break;
  default:
    // code block
}
```

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.

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"
```

The default Keyword

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

C++ Loops

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

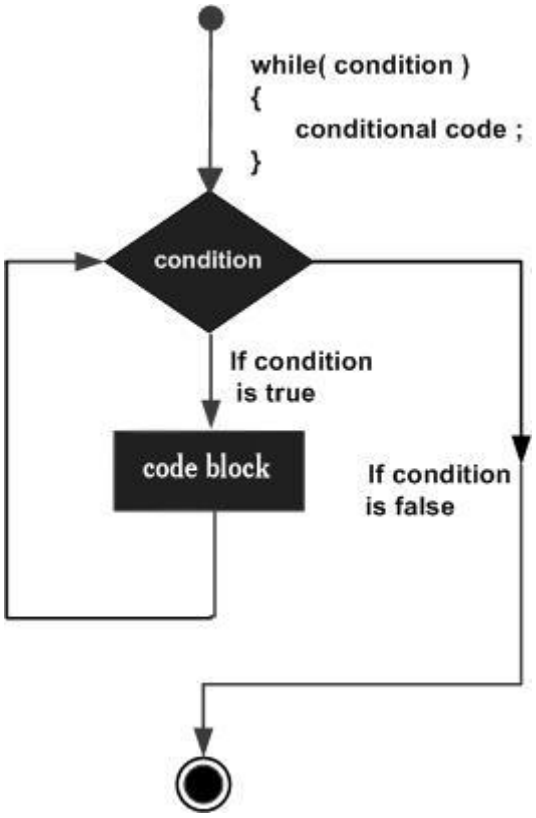
Loops are handy because they save time, reduce errors, and they make code more readable.

No	Loop Type & Description

1	while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	for loop Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do...while loop Like a 'while' statement, except that it tests the condition at the end of the loop body.
4	nested loops You can use one or more loop inside any another 'while', 'for' or 'do..while' loop.

C++ While Loop

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



Syntax

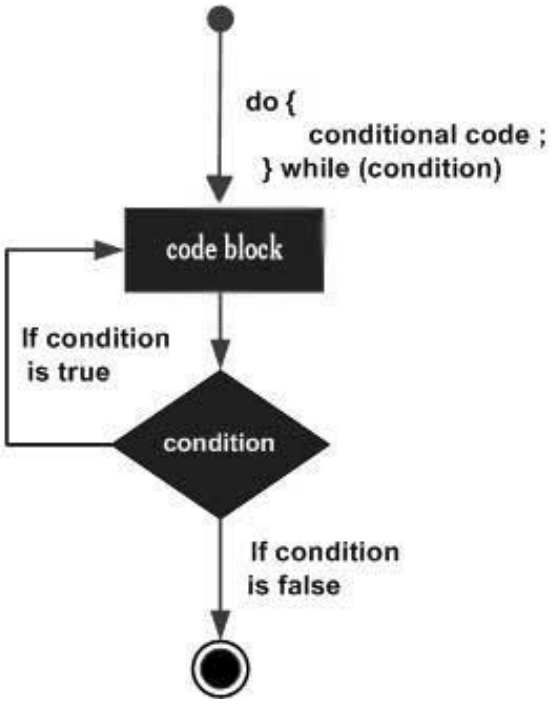
```
while (condition) {
  // code block to be executed
}
```

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);
```

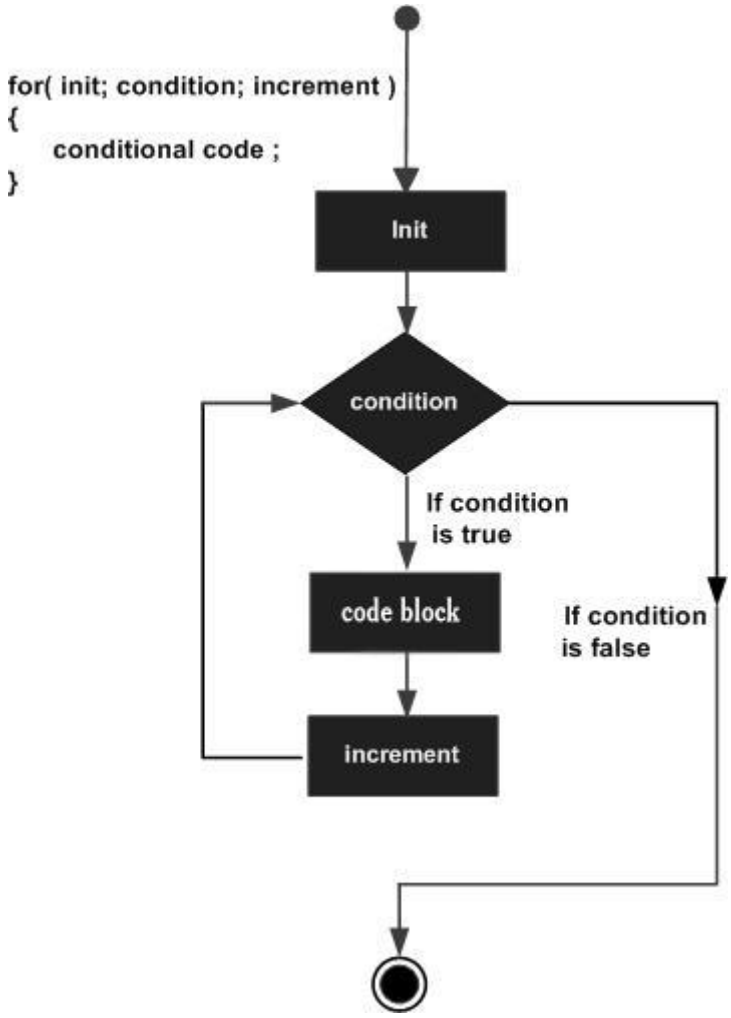
The example below uses a `do/while` loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

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.

Example

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

C++ nested loops

Syntax

The syntax for a **nested for loop** statement in C++ is as follows

```
for ( init; condition; increment ) {
   for ( init; condition; increment ) {
      statement(s);
   }
   statement(s); // you can put more statements.
}
```

The syntax for a **nested while loop** statement in C++ is as follows −

```
while(condition) {
   while(condition) {
      statement(s);
   }
   statement(s); // you can put more statements.
}
```

The syntax for a nested **do...while loop** statement in C++ is as follows −

```
do {
   statement(s); // you can put more statements.
   do {
      statement(s);
   } while( condition );

} while( condition );
```

Example

The following program uses a nested for loop to find the prime numbers from 2 to 100

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

int main () {
   int i, j;

   for(i = 2; i<100; i++) {
      for(j = 2; j <= (i/j); j++)
         if(!(i%j)) break; // if factor found, not prime
         if(j > (i/j)) cout << i << " is prime\n";
   }

   return 0;
}
```