CYNERGY CODING CLUB

C++ Beginners Guide

ACKNOWLEDGEMENT:

All the notes are referred from the following websites.

1. https://www.programiz.com/cpp-programming
2. https://beginnersbook.com/2017/08/c-plus-plus-tutorial-for-beginners/

Please refer these sites for additional information and relevant problems on the topics.

C++ Tutorial – Learn C++ Programming

C++ language is a direct descendant of C programming language with additional features such as type checking, object oriented programming, exception handling etc. You can call it a “better C”. It was developed by **Bjarne Stroustrup**.
C++ is a general purpose language language, when I say general purpose it simply means that it is designed to be used for developing applications in a wide variety of domains.

C++ Tutorial

To learn C++ programming, refer these tutorials in the given order. These tutorials are written for beginners so even if you have no prior knowledge in C++, you won’t face any difficulty understanding these tutorials.

Hello World Program in C++

```
/*
* Multiple line
* comment
*/
#include<iostream>

//Single line comment
using namespace std;

//This is where the execution of program begins
int main()
{
   // displays Hello World! on screen
   cout<<"Hello World!";

   return 0;
}
```

Output:

```
Hello World!
```

Let’s discuss each and every part of the above program.

1. Comments – You can see two types of comments in the above program

```
// This is a single line comment
/* This is a multiple line comment
* suitable for long comments
*/
```

Comments as the names suggests are just a text written by programmer during code development. Comment doesn't affect your program logic in any way, you can write whatever you want in comments but it should be related to the code and have some meaning so that when someone else look into your code, the person should understand what you did in the code by just reading your comment.

For example:

```
/* This function adds two integer numbers
 * and returns the result as an integer value
 */
int sum(int num1, int num2) {
   return num1+num2;
}
```

Now if someone reads my comment he or she can understand what I did there just by reading my comment. This improves readability of your code and when you are working on a project with your team mates, this becomes essential aspect.

2. #include<iostream> – This statements tells the compiler to include iostream file. This file contains pre defined input/output functions that we can use in our program.

3. using namespace std; – A namespace is like a region, where we have functions, variables etc and their scope is limited to that particular region. Here std is a namespace name, this tells the compiler to look into that particular region for all the variables, functions, etc. I will not discuss this in detail here as it may confuse you. I have covered this topic in a separate tutorial with examples. Just follow the tutorial in the given sequence and you would be fine.

4. int main() – As the name suggests this is the main function of our program and the execution of program begins with this function, the int here is the return type which indicates to the compiler that this function will return a integer value. That is the main reason we have a return 0 statement at the end of main function.

5. cout << "Hello World!"; – The `cout` object belongs to the iostream file and the purpose of this object is to display the content between double quotes as it is on the screen. This object can also display the value of variables on screen(don't worry, we will see that in the coming tutorials).

6. return 0; – This statement returns value 0 from the main() function which indicates that the execution of main function is successful. The value 1 represents failed execution.

Variables in C++

A variable is a name which is associated with a value that can be changed. For example when I write `int num=20;` here variable name is num which is associated with value 20, int is a data type that represents that this variable can hold integer values. We will cover the data types in the next tutorial. In this tutorial, we will discuss about variables.

Syntax of declaring a variable in C++

```
data_type variable1_name = value1, variable2_name = value2;
```

For example:

```
int num1=20, num2=100;
```

We can also write it like this:

```
int num1,num2;
num1=20;
num2=100;
```

Types of variables

Variables can be categorised based on their data type. For example, in the above example we have seen integer types variables. Following are the types of variables available in C++.

int: These type of of variables holds integer value.

char: holds character value like ‘c’, ‘F’, ‘B’, ‘p’, ‘q’ etc.

bool: holds boolean value true or false.

double: double-precision floating point value.

float: Single-precision floating point value.

Types of variables based on their scope

Before going further lets discuss what is scope first. When we discussed the Hello World Program, we have seen the curly braces in the program like this:

```
int main {

//Some code

}
```

Any variable declared inside these curly braces have scope limited within these curly braces, if you declare a variable in main() function and try to use that variable outside main() function then you will get compilation error.

Now that we have understood what is scope. Lets move on to the types of variables based on the scope.

1. Global variable
2. Local variable

Global Variable

A variable declared outside of any function (including main as well) is called global variable. Global variables have their scope throughout the program, they can be accessed anywhere in the program, in the main, in the user defined function, anywhere.

Lets take an example to understand it:

Global variable example

Here we have a global variable `myVar`, that is declared outside of main. We have accessed the variable twice in the main() function without any issues.

```
#include <iostream>
using namespace std;
// This is a global variable
char myVar = 'A';
int main()
{
   cout <<"Value of myVar: "<< myVar<<endl;
   myVar='Z';
   cout <<"Value of myVar: "<< myVar;
   return 0;
}
```

Output:

```
Value of myVar: A
Value of myVar: Z
```

Local variable

Local variables are declared inside the braces of any user defined function, main function, loops or any control statements(if, if-else etc) and have their scope limited inside those braces.

Local variable example

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

char myFuncn() {
// This is a local variable
char myVar = 'A';
}
int main()
{
   cout <<"Value of myVar: "<< myVar<<endl;
   myVar='Z';
   cout <<"Value of myVar: "<< myVar;
   return 0;
}
```

Output:
Compile time error, because we are trying to access the variable `myVar` outside of its scope. The scope of `myVar` is limited to the body of function `myFuncn()`, inside those braces.

Can global and local variable have same name in C++?

Lets see an example having same name global and local variable.

```
#include <iostream>
using namespace std;
// This is a global variable
char myVar = 'A';
char myFuncn() {
   // This is a local variable
   char myVar = 'B';
   return myVar;
}
```

```
int main()
{
   cout <<"Funcn call: "<< myFuncn()<<endl;
   cout <<"Value of myVar: "<< myVar<<endl;
   myVar='Z';
   cout <<"Funcn call: "<< myFuncn()<<endl;
   cout <<"Value of myVar: "<< myVar<<endl;
   return 0;
}
```

Output:

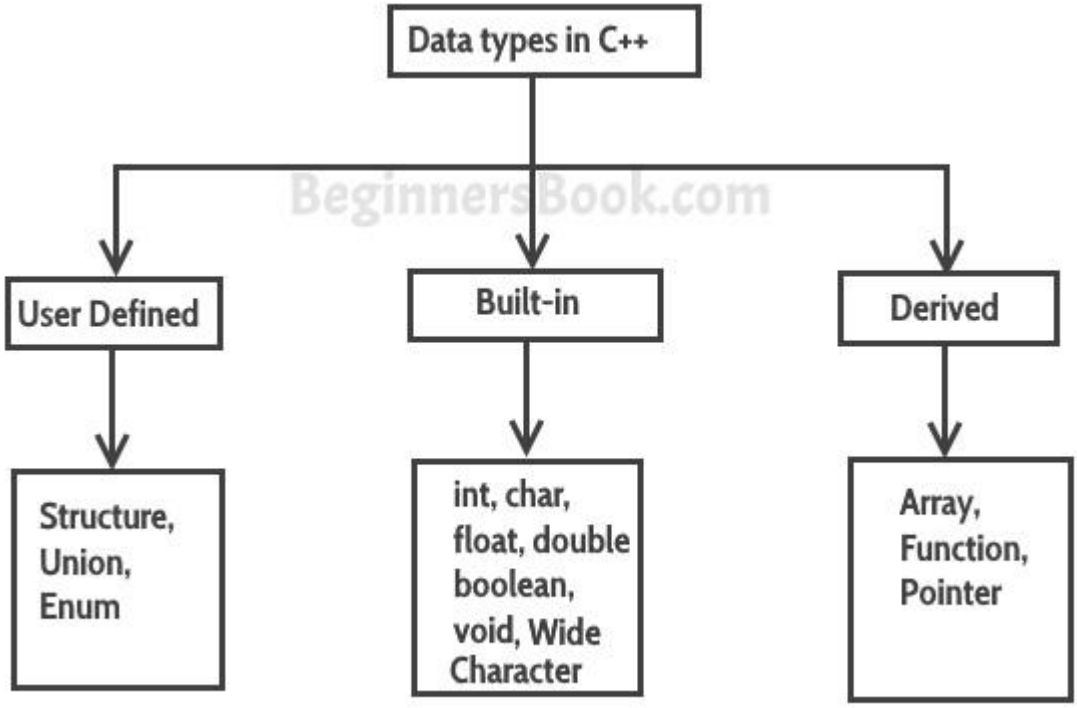
```
Funcn call: B
Value of myVar: A
Funcn call: B
Value of myVar: Z
```

As you can see that when I changed the value of `myVar` in the main function, it only changed the value of global variable `myVar` because local variable `myVar` scope is limited to the function `myFuncn()`.

Data Types in C++

Data types define the type of data a variable can hold, for example an integer variable can hold integer data, a character type variable can hold character data etc.

Data types in C++ are categorised in three groups: **Built-in**, **user-defined** and **Derived**.



Built in data types

char: For characters. Size 1 byte.

```
char ch = 'A';
```

int: For integers. Size 2 bytes.

```
int num = 100;
```

float: For single precision floating point. Size 4 bytes.

```
float num = 123.78987;
```

double: For double precision floating point. Size 8 bytes.

```
double num = 10098.98899;
```
bool: For booleans, true or false.

```
bool b = true;
```
wchar_t: Wide Character. This should be avoided because its size is implementation defined and not reliable.

These are some other types of data types which will be covered in the upcoming sessions.

User-defined data types

We have three types of user-defined data types in C++
1. struct
2. union
3. enum

I have covered them in detail in separate tutorials. For now just remember that these comes under user-defined data types.

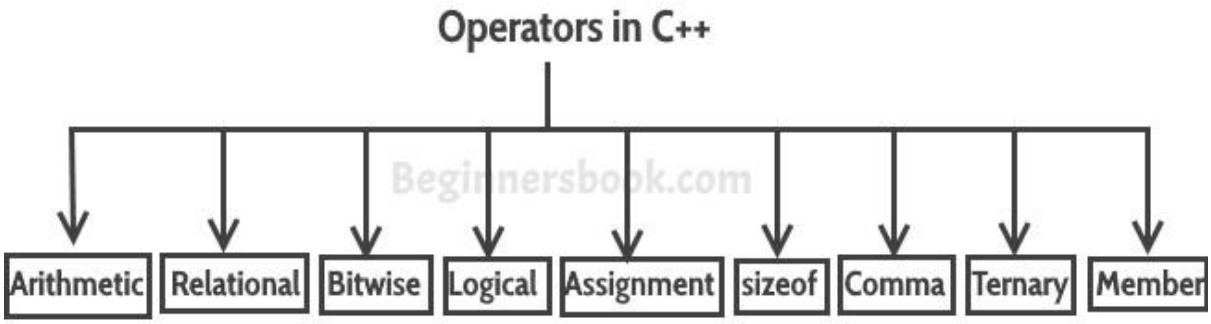
Derived data types in C++

We have three types of derived-defined data types in C++
1. Array
2. Function
3. Pointer

Operators in C++

Operator represents an action. For example + is an operator that represents addition. An operator works on two or more operands and produce an output. For example 3+4+5 here + operator works on three operands and produce 12 as output.

Types of Operators in C++



1) Basic Arithmetic Operators
2) Assignment Operators
3) Auto-increment and Auto-decrement Operators
4) Logical Operators
5) Comparison (relational) operators
6) Bitwise Operators
7) Ternary Operator

1) Basic Arithmetic Operators

Basic arithmetic operators are: +, -, *, /, %
+ is for addition.

– is for subtraction.

***** is for multiplication.

/ is for division.

% is for modulo.
Note: Modulo operator returns remainder, for example 20 % 5 would return 0

Example of Arithmetic Operators

```
#include <iostream>
using namespace std;
int main(){
  int num1 = 240;
  int num2 = 40;
  cout<<"num1 + num2: "<<(num1 + num2)<<endl;
  cout<<"num1 - num2: "<<(num1 - num2)<<endl;
  cout<<"num1 * num2: "<<(num1 * num2)<<endl;
  cout<<"num1 / num2: "<<(num1 / num2)<<endl;
  cout<<"num1 % num2: "<<(num1 % num2)<<endl;
  return 0;
}
```

Output:

```
num1 + num2: 280
num1 - num2: 200
num1 * num2: 9600
num1 / num2: 6
num1 % num2: 0
```

2) Assignment Operators

Assignments operators in C++ are: =, +=, -=, *=, /=, %=

num2 = num1 would assign value of variable num1 to the variable.

num2+=num1 is equal to num2 = num2+num1

num2-=num1 is equal to num2 = num2-num1

num2*=num1 is equal to num2 = num2*num1

num2/=num1 is equal to num2 = num2/num1

num2%=num1 is equal to num2 = num2%num1

Example of Assignment Operators

```
#include <iostream>
using namespace std;
int main(){
 int num1 = 240;
 int num2 = 40;
 num2 = num1;
 cout<<"= Output: "<<num2<<endl;
 num2 += num1;
 cout<<"+= Output: "<<num2<<endl;
 num2 -= num1;
 cout<<"-= Output: "<<num2<<endl;
 num2 *= num1;
 cout<<"*= Output: "<<num2<<endl;
 num2 /= num1;
 cout<<"/= Output: "<<num2<<endl;
 num2 %= num1;
 cout<<"%= Output: "<<num2<<endl;
 return 0;
}
```

Output:

```
= Output: 240
+= Output: 480
-= Output: 240
*= Output: 57600
/= Output: 240
%= Output: 0
```

3) Auto-increment and Auto-decrement Operators

++ and —
num++ is equivalent to num=num+1;

num–- is equivalent to num=num-1;

Example of Auto-increment and Auto-decrement Operators

```
#include <iostream>
using namespace std;
int main(){
  int num1 = 240;
  int num2 = 40;
  num1++; num2--;
  cout<<"num1++ is: "<<num1<<endl;
  cout<<"num2-- is: "<<num2;
  return 0;
}
```

Output:

```
num1++ is: 241
num2-- is: 39
```

4) Logical Operators

Logical Operators are used with binary variables. They are mainly used in conditional statements and loops for evaluating a condition.

Logical operators in C++ are: &&, ||, !

Let's say we have two boolean variables b1 and b2.

b1&&b2 will return true if both b1 and b2 are true else it would return false.

b1||b2 will return false if both b1 and b2 are false else it would return true.

!b1 would return the opposite of b1, that means it would be true if b1 is false and it would return false if b1 is true.

Example of Logical Operators

```
#include <iostream>
using namespace std;
int main(){
   bool b1 = true;
   bool b2 = false;
   cout<<"b1 && b2: "<<(b1&&b2)<<endl;
   cout<<"b1 || b2: "<<(b1||b2)<<endl;
   cout<<"!(b1 && b2): "<<!(b1&&b2);
   return 0;
}
```

Output:

```
b1 && b2: 0
b1 || b2: 1
!(b1 && b2): 1
```

5) Relational operators

We have six relational operators in C++: ==, !=, >, <, >=, <=

== returns true if both the left side and right side are equal

!= returns true if left side is not equal to the right side of operator.

> returns true if left side is greater than right.

< returns true if left side is less than right side.

>= returns true if left side is greater than or equal to right side.

<= returns true if left side is less than or equal to right side.

Example of Relational operators

```
#include <iostream>
using namespace std;
int main(){
   int num1 = 240;
   int num2 =40;
```

```
   if (num1==num2) {
      cout<<"num1 and num2 are equal"<<endl;
   }
   else{
      cout<<"num1 and num2 are not equal"<<endl;
   }
   if( num1 != num2 ){
      cout<<"num1 and num2 are not equal"<<endl;
   }
   else{
      cout<<"num1 and num2 are equal"<<endl;
   }
   if( num1 > num2 ){
      cout<<"num1 is greater than num2"<<endl;
   }
   else{
      cout<<"num1 is not greater than num2"<<endl;
   }
   if( num1 >= num2 ){
      cout<<"num1 is greater than or equal to num2"<<endl;
   }
   else{
      cout<<"num1 is less than num2"<<endl;
   }
   if( num1 < num2 ){
      cout<<"num1 is less than num2"<<endl;
   }
   else{
      cout<<"num1 is not less than num2"<<endl;
   }
   if( num1 <= num2){
      cout<<"num1 is less than or equal to num2"<<endl;
   }
   else{
      cout<<"num1 is greater than num2"<<endl;
   }
   return 0;
}
```

Output:

```
num1 and num2 are not equal
num1 and num2 are not equal
num1 is greater than num2
num1 is greater than or equal to num2
num1 is not less than num2
num1 is greater than num2
```

6) Bitwise Operators

There are six bitwise Operators: &, |, ^, ~, <<, >>

```
num1 = 11; /* equal to 00001011*/
num2 = 22; /* equal to 00010110 */
```

Bitwise operator performs bit by bit processing.
num1 & num2 compares corresponding bits of num1 and num2 and generates 1 if both bits are equal, else it returns 0. In our case it would return: 2 which is 00000010 because in the binary form of num1 and num2 only second last bits are matching.

num1 | num2 compares corresponding bits of num1 and num2 and generates 1 if either bit is 1, else it returns 0. In our case it would return 31 which is 00011111

num1 ^ num2 compares corresponding bits of num1 and num2 and generates 1 if they are not equal, else it returns 0. In our example it would return 29 which is equivalent to 00011101

~num1 is a complement operator that just changes the bit from 0 to 1 and 1 to 0. In our example it would return -12 which is signed 8 bit equivalent to 11110100

num1 << 2 is left shift operator that moves the bits to the left, discards the far left bit, and assigns the rightmost bit a value of 0. In our case output is 44 which is equivalent to 00101100

Note: In the example below we are providing 2 at the right side of this shift operator that is the reason bits are moving two places to the left side. We can change this number and bits would be moved by the number of bits specified on the right side of the operator. Same applies to the right side operator.

num1 >> 2 is right shift operator that moves the bits to the right, discards the far right bit, and assigns the leftmost bit a value of 0. In our case output is 2 which is equivalent to 00000010

Example of Bitwise Operators

```
#include <iostream>
using namespace std;
int main(){
```

```
   int num1 = 11;  /* 11 = 00001011 */
   int num2 = 22;  /* 22 = 00010110 */
   int result = 0;
   result = num1 & num2;
   cout<<"num1 & num2: "<<result<<endl;
   result = num1 | num2;
   cout<<"num1 | num2: "<<result<<endl;
   result = num1 ^ num2;
   cout<<"num1 ^ num2: "<<result<<endl;
   result = ~num1;
   cout<<"~num1: "<<result<<endl;
   result = num1 << 2;
   cout<<"num1 << 2: "<<result<<endl;
   result = num1 >> 2;
   cout<<"num1 >> 2: "<<result;
   return 0;
}
```

Output:

```
num1 & num2: 2
num1 | num2: 31
num1 ^ num2: 29
~num1: -12
num1 << 2: 44 num1 >> 2: 2
```

7) Ternary Operator

This operator evaluates a boolean expression and assign the value based on the result.
Syntax:

```
variable num1 = (expression) ? value if true : value if false
```

If the expression results true then the first value before the colon (:) is assigned to the variable num1 else the second value is assigned to the num1.

Example of Ternary Operator

```
#include <iostream>
using namespace std;
int main(){
  int num1, num2; num1 = 99;
  /* num1 is not equal to 10 that's why
   * the second value after colon is assigned
   * to the variable num2
   */
  num2 = (num1 == 10) ? 100: 200;
  cout<<"num2: "<<num2<<endl;
  /* num1 is equal to 99 that's why
   * the first value is assigned
```

```
   * to the variable num2
   */
  num2 = (num1 == 99) ? 100: 200;
  cout<<"num2: "<<num2;
  return 0;
}
```

Output:

```
num2: 200
num2: 100
```

Miscellaneous Operators

There are few other operators in C++ such as **Comma operator** and **sizeof operator**. We will cover them in detail in a separate tutorial.

Operator Precedence in C++

This determines which operator needs to be evaluated first if an expression has more than one operator. Operator with higher precedence at the top and lower precedence at the bottom.

Unary Operators
++ – – ! ~

Multiplicative
* / %

Additive
+ –

Shift
<< >> >>>

Relational
> >= < <=

Equality
== !=

Bitwise AND

&

Bitwise XOR
^

Bitwise OR
|

Logical AND
&&

Logical OR
||

Ternary
?:

Assignment
= += -= *= /= %= > >= < <= &= ^= |=

If else Statement in C++

Sometimes we need to execute a block of statements only when a particular condition is met or not met. This is called **decision making**, as we are executing a certain code after making a decision in the program logic. For decision making in C++, we have four types of control statements (or control structures), which are as follows:

a) if statement
b) nested if statement
c) if-else statement
d) if-else-if statement

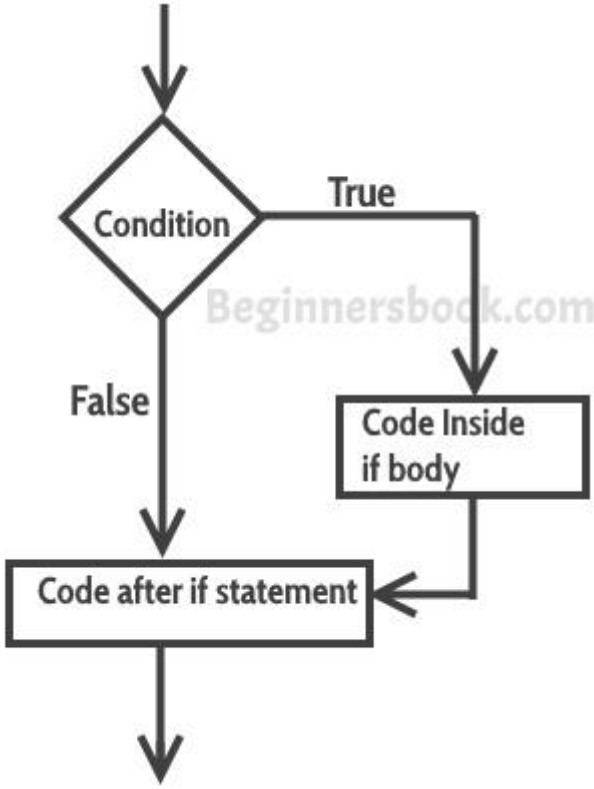
If statement in C++

If statement consists a condition, followed by statement or a set of statements as shown below:

```
if(condition){
  Statement(s);
}
```

The statements inside **if** parenthesis (usually referred as if body) gets executed only when the given condition is true. If the condition is false then the statements inside if body are completely ignored.

Flow diagram of If statement



Example of if statement

```
#include <iostream>
using namespace std;
int main(){
  int num=70;
  if( num < 100 ){
     /* This cout statement will only execute,
      * if the above condition is true
      */
     cout<<"number is less than 100";
  }

  if(num > 100){
     /* This cout statement will only execute,
      * if the above condition is true
      */
     cout<<"number is greater than 100";
  }
  return 0;
}
```

Output:

```
number is less than 100
```

Nested if statement in C++

When there is an if statement inside another if statement then it is called the **nested if statement**.
The structure of nested if looks like this:

```
if(condition_1) {
   Statement1(s);

   if(condition_2) {
      Statement2(s);
   }
}
```

Statement1 would execute if the condition_1 is true. Statement2 would only execute if both the conditions(condition_1 and condition_2) are true.

Example of Nested if statement

```
#include <iostream>
using namespace std;
int main(){
   int num=90;
   /* Nested if statement. An if statement
    * inside another if body
    */
   if( num < 100 ){
      cout<<"number is less than 100"<<endl;
      if(num > 50){
         cout<<"number is greater than 50";
      }
   }
   return 0;
}
```

Output:

```
number is less than 100
number is greater than 50
```

If else statement in C++

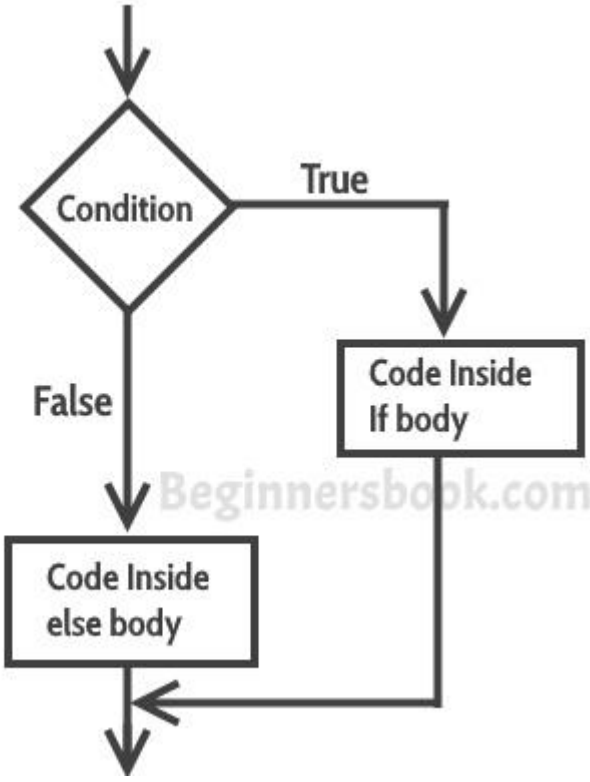
Sometimes you have a condition and you want to execute a block of code if condition is true and execute another piece of code if the same condition is false. This can be achieved in C++ using if-else statement.

This is how an if-else statement looks:

```
if(condition) {
   Statement(s);
}
else {
   Statement(s);
}
```

The statements inside “if” would execute if the condition is true, and the statements inside “else” would execute if the condition is false.

Flow diagram of if-else



Example of if-else statement

```
#include <iostream>
using namespace std;
int main(){
```

```
   int num=66;
   if( num < 50 ){
      //This would run if above condition is true
      cout<<"num is less than 50";
   }
   else {
      //This would run if above condition is false
      cout<<"num is greater than or equal 50";
   }
   return 0;
}
```

Output:

```
num is greater than or equal 50
```

if-else-if Statement in C++

if-else-if statement is used when we need to check multiple conditions. In this control structure we have only one “if” and one “else”, however we can have multiple “else if” blocks. This is how it looks:

```
if(condition_1) {
   /*if condition_1 is true execute this*/
   statement(s);
}
else if(condition_2) {
   /* execute this if condition_1 is not met and
    * condition_2 is met
    */
   statement(s);
}
else if(condition_3) {
   /* execute this if condition_1 & condition_2 are
    * not met and condition_3 is met
    */
   statement(s);
}
.
.
.
else {
   /* if none of the condition is true
    * then these statements gets executed
    */
   statement(s);
}
```

Note: The most important point to note here is that in if-else-if, as soon as the condition is met, the corresponding set of statements get executed, rest gets ignored. If none of the condition is met then the statements inside “else” gets executed.

Example of if-else-if

```
#include <iostream>
using namespace std;
int main(){
   int num;
   cout<<"Enter an integer number between 1 & 99999: ";
   cin>>num;
   if(num <100 && num>=1) {
      cout<<"Its a two digit number";
   }
   else if(num <1000 && num>=100) {
      cout<<"Its a three digit number";
   }
   else if(num <10000 && num>=1000) {
      cout<<"Its a four digit number";
   }
   else if(num <100000 && num>=10000) {
      cout<<"Its a five digit number";
   }
   else {
      cout<<"number is not between 1 & 99999";
   }
   return 0;
}
```

Output:

```
Enter an integer number between 1 & 99999: 8976
Its a four digit number
```

For loop in C++ with example

A loop is used for executing a block of statements repeatedly until a particular condition is satisfied. For example, when you are displaying number from 1 to 100 you may want set the value of a variable to 1 and display it 100 times, increasing its value by 1 on each loop iteration.

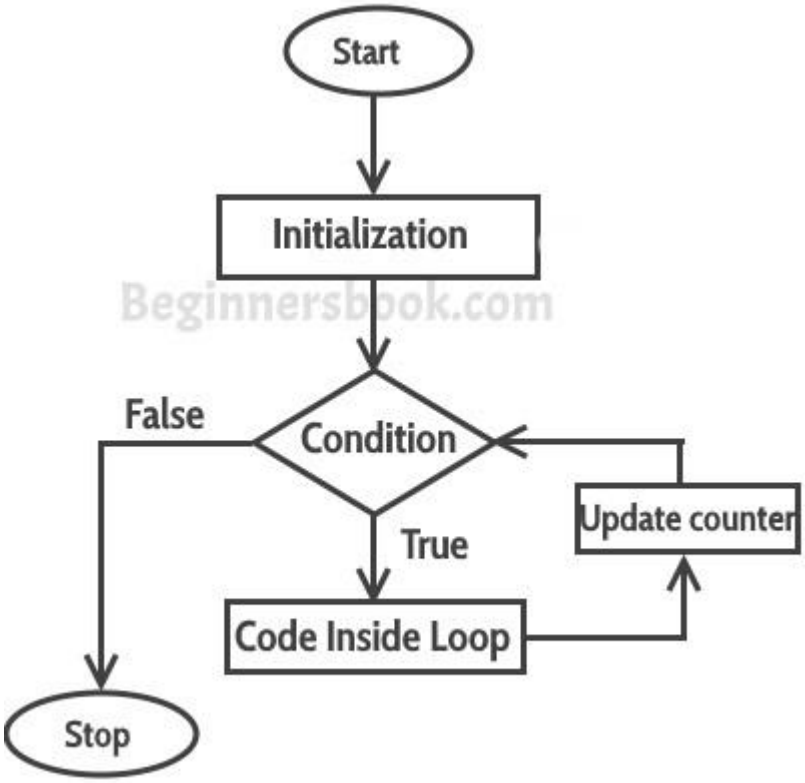
In C++ we have three types of basic loops: for, while and do-while. In this tutorial we will learn how to use “for loop” in C++.

Syntax of for loop

```
for(initialization; condition ; increment/decrement)
{
   C++ statement(s);
}
```

Flow of Execution of the for Loop

As a program executes, the interpreter always keeps track of which statement is about to be executed. We call this the control flow, or the flow of execution of the program.



First step: In for loop, initialization happens first and only once, which means that the initialization part of for loop only executes once.

Second step: Condition in for loop is evaluated on each loop iteration, if the condition is true then the statements inside for for loop body gets executed. Once the condition returns false, the statements in for loop does not execute and the control gets transferred to the next statement in the program after for loop.

Third step: After every execution of for loop's body, the increment/decrement part of for loop executes that updates the loop counter.

Fourth step: After third step, the control jumps to second step and condition is re-evaluated.

The steps from second to fourth repeats until the loop condition returns false.

Example of a Simple For loop in C++

Here in the loop initialization part I have set the value of variable i to 1, condition is i<=6 and on each loop iteration the value of i increments by 1.

```
#include <iostream>
using namespace std;
int main(){
   for(int i=1; i<=6; i++){
      /* This statement would be executed
       * repeatedly until the condition
       * i<=6 returns false.
       */
      cout<<"Value of variable i is: "<<i<<endl;
   }
   return 0;
}
```

Output:

```
Value of variable i is: 1
Value of variable i is: 2
Value of variable i is: 3
Value of variable i is: 4
Value of variable i is: 5
Value of variable i is: 6
```

Infinite for loop in C++

A loop is said to be infinite when it executes repeatedly and never stops. This usually happens by mistake. When you set the condition in for loop in such a way that it never return false, it becomes infinite loop.

For example:

```
#include <iostream>
using namespace std;
int main(){
   for(int i=1; i>=1; i++){
      cout<<"Value of variable i is: "<<i<<endl;
   }
   return 0;
}
```

This is an infinite loop as we are incrementing the value of i so it would always satisfy the condition i>=1, the condition would never return false.

Here is another example of infinite for loop:

```
// infinite loop
for ( ; ; ) {
   // statement(s)
}
```

Example: display elements of array using for loop

```
#include <iostream>
using namespace std;
int main(){
   int arr[]={21,9,56,99, 202};
   /* We have set the value of variable i
    * to 0 as the array index starts with 0
    * which means the first element of array
    * starts with zero index.
    */
   for(int i=0; i<5; i++){
      cout<<arr[i]<<endl;
   }
   return 0;
}
```

Output:

```
21
9
56
99
202
```