Notes On C++

C++ is a middle-level programming language developed by Bjarne Stroustrup starting in 1979 at Bell Labs. C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.
This reference will take you through simple and practical approach while learning C++ Programming language.

C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.

C++ is regarded as a **middle-level** language, as it comprises a combination of both high-level and low-level language features.

C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.

C++ is a superset of C, and that virtually any legal C program is a legal C++ program.

Note: A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time.

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 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 **using namespace std;** tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++.

- The next line **// main() is where program execution begins.** It is a single-line comment available in C++. Single-line comments begin with // and stop at the end of the line.
- The line **int main()** is the main function where program execution begins.
- The next line **cout << "This is my first C++ program.";** causes the message "This is my first C++ program" 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.

Compile & Execute C++ Program:

Let's look at how to save the file, compile and run the program. Please follow the steps given below:

- Open a text editor and add the code as above.
- Save the file as: hello.cpp
- Open a command prompt and go to the directory where you saved the file.
- Type 'g++ hello.cpp ' and press enter to compile your code. If there are no errors in your code the command prompt will take you to the next line and would generate a.out executable file.
- Now, type ' a.out' to run your program.
- You will be able to see ' Hello World ' printed on the window.

```
$ g++ hello.cpp
$ ./a.out
Hello World
```

Make sure that g++ is in your path and that you are running it in the directory containing file hello.cpp.
You can compile C/C++ programs using makefile. For more details, you can check Makefile Tutorial.

Semicolons & Blocks in C++:

In C++, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.
For example, following are three different statements:

```
x = y;
y = y+1;
add(x, y);
```

A block is a set of logically connected statements that are surrounded by opening and closing braces. For example:

```
{
  cout << "Hello World"; // prints Hello World
  return 0;
}
```

C++ does not recognize the end of the line as a terminator. For this reason, it does not matter where on a line you put a statement. For example:

```
x = y;
```

```
y = y+1;
add(x, y);
```
is the same as
```
x = y; y = y+1; add(x, y);
```

C++ Identifiers:

A C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9). C++ does not allow punctuation characters such as @, $, and % within identifiers. C++ is a case-sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in C++.
Here are some examples of acceptable identifiers:

```
mohd       zara    abc   move_name  a_123
myname50   _temp   j     a23b9      retVal
```

C++ Keywords:

The following list shows the reserved words in C++. These reserved words may not be used as constant or variable or any other identifier names.

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

Whitespace in C++:

A line containing only whitespace, possibly with a comment, is known as a blank line, and C++ compiler totally ignores it.

Whitespace is the term used in C++ to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the

compiler to identify where one element in a statement, such as int, ends and the next element begins. Therefore, in the statement,

```
int age;
```

there must be at least one whitespace character (usually a space) between int and age for the compiler to be able to distinguish them. On the other hand, in the statement

```
fruit = apples + oranges;   // Get the total fruit
```

no whitespace characters are necessary between fruit and =, or between = and apples, although you are free to include some if you wish for readability purpose.

Comments in C++

Program comments are explanatory statements that you can include in the C++ code that you write and helps anyone reading it's source code. All programming languages allow for some form of comments.

C++ supports single-line and multi-line comments. All characters available inside any comment are ignored by C++ compiler.

C++ comments start with /* and end with */. For example:

```
/* This is a comment */

/* C++ comments can  also
 * span multiple lines
 */
```

A comment can also start with //, extending to the end of the line. For example:

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

main()
{
   cout << "Hello World"; // prints Hello World

   return 0;
}
```

When the above code is compiled, it will ignore **// prints Hello World** and final executable will produce the following result:

```
Hello World
```

Within a /* and */ comment, // characters have no special meaning. Within a // comment, /* and */ have no special meaning. Thus, you can "nest" one kind of comment within the other kind. For example:

```
/* Comment out printing of Hello World:

cout << "Hello World"; // prints Hello World

*/
```

C++ Data Types

While doing programming in any programming language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

Primitive Built-in Types:

C++ offer the programmer a rich assortment of built-in as well as user defined data types. Following table lists down seven basic C++ data types:

Type	Description
bool	Stores either value true or false.
char	Typically a single octet(one byte). This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.
wchar_t	A wide character type.

Several of the basic types can be modified using one or more of these type modifiers:

- signed
- unsigned
- short
- long

The following table shows the variable type, how much memory it takes to store the value in memory, and what is maximum and minimum value which can be stored in such type of variables.

The sizes of variables might be different from those shown in the above table, depending on the compiler and the computer you are using.

Following is the example, which will produce correct size of various data types on your computer.

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

int main()
{
  cout << "Size of char : " << sizeof(char) << endl;
  cout << "Size of int : " << sizeof(int) << endl;
  cout << "Size of short int : " << sizeof(short int) << endl;
  cout << "Size of long int : " << sizeof(long int) << endl;
  cout << "Size of float : " << sizeof(float) << endl;
  cout << "Size of double : " << sizeof(double) << endl;
  cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;
  return 0;
}
```

This example uses **endl**, which inserts a new-line character after every line and << operator is being used to pass multiple values out to the screen. We are also using **sizeof()** operator to get size of various data types.
When the above code is compiled and executed, it produces the following result which can vary from machine to machine:

```
Size of char : 1
Size of int : 4
Size of short int : 2
Size of long int : 4
Size of float : 4
Size of double : 8
Size of wchar_t : 4
```

C++ Variable Types

A variable provides us with named storage that our programs can manipulate. Each variable in C++ has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C++ is case-sensitive:
There are following basic types of variable in C++ as explained in last chapter:

C++ also allows to define various other types of variables, which we will cover in subsequent chapters like **Enumeration, Pointer, Array, Reference, Data structures,** and **Classes**.
Following section will cover how to define, declare and use various types of variables.

Variable Definition in C++:

A variable definition means to tell the compiler where and how much to create the storage for the variable. A variable definition specifies a data type, and contains a list of one or more variables of that type as follows:

```
type variable_list;
```

Here, **type** must be a valid C++ data type including char, w_char, int, float, double, bool or any user-defined object, etc., and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

The line **int i, j, k;** both declares and defines the variables i, j and k; which instructs the compiler to create variables named i, j and k of type int.
Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

```
type variable_name = value;
```

Some examples are:

```
extern int d = 3, f = 5;    // declaration of d and f.
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables is undefined.
Variable Declaration in C++:
A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable. A variable declaration has its meaning at the time of compilation only, compiler needs actual variable declaration at the time of linking of the program.

A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use **extern** keyword to declare a variable at any place. Though you

can declare a variable multiple times in your C++ program, but it can be defined only once in a file, a function or a block of code.

Example

Try the following example where a variable has been declared at the top, but it has been defined inside the main function:

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

// Variable declaration:
extern int a, b;
extern int c;
extern float f;

int main ()
{
  // Variable definition:
  int a, b;
  int c;
  float f;

  // actual initialization
  a = 10;
  b = 20;
  c = a + b;

  cout << c << endl ;

  f = 70.0/3.0;
  cout << f << endl ;

  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
30
23.3333
```

Same concept applies on function declaration where you provide a function name at the time of its declaration and its actual definition can be given anywhere else. For example:

```
// function declaration
int func();

int main()
{
   // function call
   int i = func();
```

```
}

// function definition
int func()
{
   return 0;
}
```

Lvalues and Rvalues:

There are two kinds of expressions in C++:

- **lvalue :** Expressions that refer to a memory location is called "lvalue" expression. An lvalue may appear as either the left-hand or right-hand side of an assignment.
- **rvalue :** The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and can not appear on the left-hand side. Following is a valid statement:

```
int g = 20;
```

But following is not a valid statement and would generate compile-time error:

```
10 = 20;
```

Variable Scope in C++

A scope is a region of the program and broadly speaking there are three places, where variables can be declared:

- Inside a function or a block which is called local variables,
- In the definition of function parameters which is called formal parameters.
- Outside of all functions which is called global variables.

We will learn what is a function and it's parameter in subsequent chapters. Here let us explain what are local and global variables.

Local Variables:

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables:

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

int main ()
```

```
{
  // Local variable declaration:
  int a, b;
  int c;

  // actual initialization
  a = 10;
  b = 20;
  c = a + b;

  cout << c;

  return 0;
}
```

Global Variables:

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables:

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

// Global variable declaration:
int g;

int main ()
{
  // Local variable declaration:
  int a, b;

  // actual initialization
  a = 10;
  b = 20;
  g = a + b;

  cout << g;

  return 0;
}
```

A program can have same name for local and global variables but value of local variable inside a function will take preference. For example:

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

// Global variable declaration:
int g = 20;

int main ()
{
  // Local variable declaration:
  int g = 10;

  cout << g;

  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
10
```

Initializing Local and Global Variables:

When a local variable is defined, it is not initialized by the system, you must initialize it yourself. Global variables are initialized automatically by the system when you define them as follows:

Data Type	Initializer
int	0
char	'\0'
float	0
double	0
pointer	NULL

It is a good programming practice to initialize variables properly, otherwise sometimes program would produce unexpected result.

C++ Constants/Literals

Constants refer to fixed values that the program may not alter and they are called **literals**.

Constants can be of any of the basic data types and can be divided into Integer Numerals, Floating-Point Numerals, Characters, Strings and Boolean Values.

Again, constants are treated just like regular variables except that their values cannot be modified after their definition.

Integer literals:

An integer literal can be a decimal, octal, or hexadecimal constant.

Floating-point literals:
A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.
Boolean literals:
There are two Boolean literals and they are part of standard C++ keywords:

- A value of **true** representing true.
- A value of **false** representing false.

You should not consider the value of true equal to 1 and value of false equal to 0.

Character literals:
Character literals are enclosed in **single** quotes.

There are certain characters in C++ when they are preceded by a backslash they will have special meaning and they are used to represent like newline (\n) or tab (\t).

String literals:
String literals are enclosed in **double** quotes.

Defining Constants:
There are two simple ways in C++ to define constants:

- Using **#define** preprocessor.
- Using **const** keyword.

The #define Preprocessor:
Following is the form to use #define preprocessor to define a constant:

```
#define identifier value
```

Following example explains it in detail:

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

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main()
{

   int area;

   area = LENGTH * WIDTH;
   cout << area;
   cout << NEWLINE;
   return 0;
```

```
}
```
When the above code is compiled and executed, it produces the following result:
```
50
```

The const Keyword:
You can use **const** prefix to declare constants with a specific type as follows:
```
const type variable = value;
```
Following example explains it in detail:
```
#include <iostream>
using namespace std;

int main()
{
   const int  LENGTH = 10;
   const int  WIDTH  = 5;
   const char NEWLINE = '\n';
   int area;

   area = LENGTH * WIDTH;
   cout << area;
   cout << NEWLINE;
   return 0;
}
```
When the above code is compiled and executed, it produces the following result:
```
50
```
Note that it is a good programming practice to define constants in CAPITALS.

Operators in C++

Operators are special type of functions that takes one or more arguments and produces a new value. For example : addition (+), subtraction (-), multiplication (*) etc., are all operators. Operators are used to perform various operations on variables and constants. The data value on which an operation is performed using an operator is called an **OPERAND.**

Types of operators

1. Assignment Operator
2. Mathematical Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators

6. Shift Operators
7. Unary Operators
8. Ternary Operator
9. Comma Operator

Assignment Operator (=)

Operates '=' is used for assignment, it takes the right-hand side (called rvalue) and copy it into the left-hand side (called lvalue).

Mathematical Operators

There are operators used to perform basic mathematical operations. Addition (+) , subtraction (-) , division (/) multiplication (*) and modulus (%) are the basic mathematical operators.

C++ and C also use a shorthand notation to perform an operation and assignment at same type. *Example*,

```
int x=10;
x += 4 // will add 4 to 10, and hence assign 14 to X.
x -= 5 // will subtract 5 from 10 and assign 5 to x.
```

Relational Operators

These operators establish a relationship between operands. The relational operators are : less than (<) , greater than (>) , less than or equal to (<=), greater than equal to (>=), equivalent (==) and not equivalent (!=).

You must notice that assignment operator is (=) and there is a relational operator, for equivalent (==). These two are different from each other, the assignment operator assigns the value to any variable, whereas equivalent operator is used to compare values, like in if-else conditions, *Example*

```
int x = 10;  //assignment operator

x=5;        // again assignment operator
if(x == 5)   // here we have used equivalent relational operator, for comparison
```

Logical Operators

The logical operators are AND (&&) and OR (||). They are used to combine two different expressions together.

If two statement are connected using AND operator, the validity of both statements will be considered, but if they are connected using OR operator, then either one of them must be valid. These operators are mostly used in loops (especially while loop) and in Decision making(if else).

Bitwise Operators

They are used to change individual bits into a number. They work with only integral data types like char,int and long and not with floating point values.

- Bitwise AND operators &
- Bitwise OR operator |
- And bitwise XOR operator ^
- And, bitwise NOT operator ~

Shift Operators

Shift Operators are used to shift Bits of any variable. It is of two types,

1. Left Shift Operator <<
2. Right Shift Operator >>

Unary Operators

These are the operators which work on only one operand. There are many unary operators, but increment++ and decrement -- operators are most used.

The increase operator (++) and the decrease operator (–) are used to increase or reduce the value
stored in the variable by one.

Example: A++; is the same as A+=1; or A= A + 1;

Ternary Operator

The ternary if-else (**? :**)is an operator which has three operands.

```
int a = 10;

a > 5 ? cout << "true" : cout << "false"
```

Comma Operator

This is used to separate variable names and to separate expressions. In case of expressions, the value of last expression is produced and used.

Example :

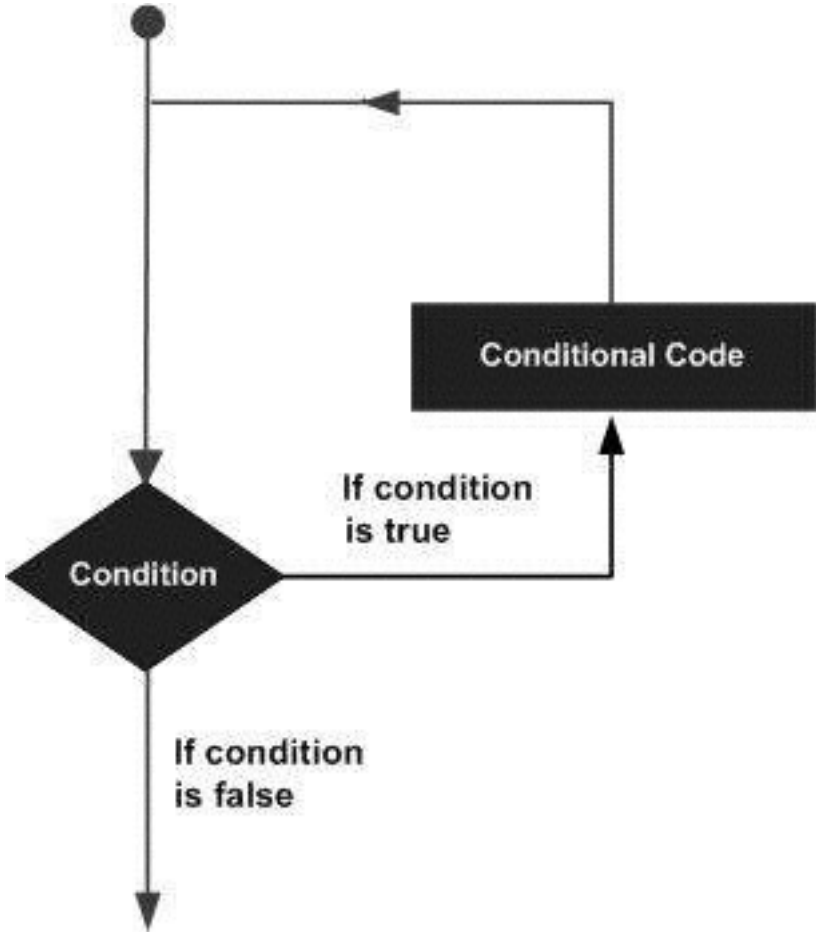
```
int a,b,c; // variables declaration using comma operator
```

C++ Loop Types

There may be a situation, when you need to execute a block of code several number of times. In general statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general from of a loop statement in most of the programming languages:



C++ programming language provides the following types of loop to handle looping requirements. Click the following links to check their detail.

Loop Type

Description

while loop

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

for loop

Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

do...while loop

Like a while statement, except that it tests the condition at the end of the loop body

nested loops

You can use one or more loop inside any another while, for or do..while loop.

Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C++ supports the following control statements. Click the following links to check their detail.

Control Statement

Description
break statement
Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
continue statement
Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

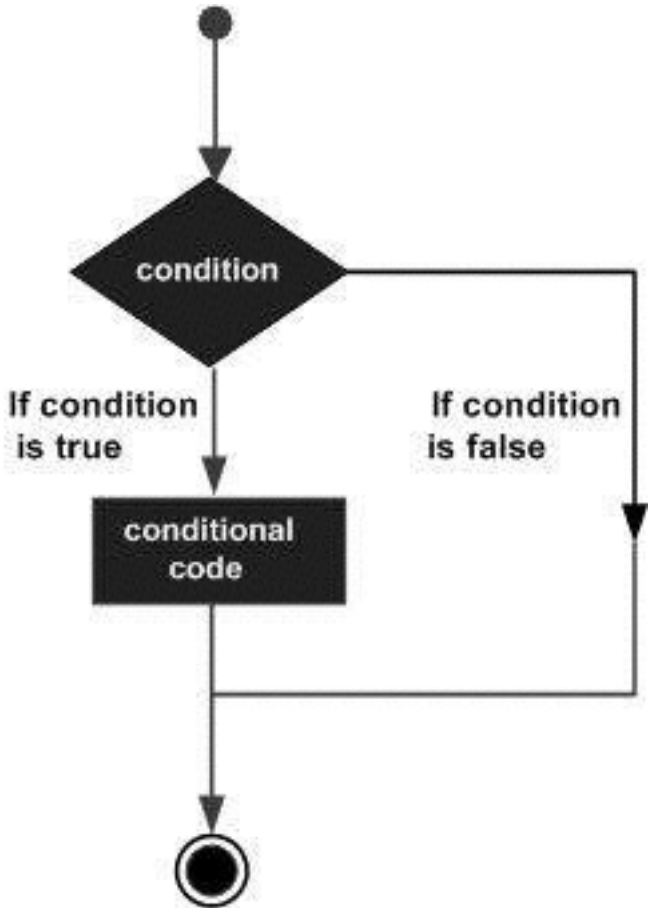
The Infinite Loop:
A loop becomes infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose. Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.

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

int main ()
{

  for( ; ; )
  {
    printf("This loop will run forever.\n");
  }

  return 0;
}
```

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C++ programmers more commonly use the for(;;) construct to signify an infinite loop.
NOTE: You can terminate an infinite loop by pressing Ctrl + C keys.

C++ decision making statements

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
Following is the general from of a typical decision making structure found in most of the programming languages:



C++ programming language provides following types of decision making statements. Click the following links to check their detail.

Statement
Description

if statement

An if statement consists of a boolean expression followed by one or more statements.

if...else statement

An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

switch statement

A switch statement allows a variable to be tested for equality against a list of values.

nested if statements

You can use one if or else if statement inside another if or else if statement(s).

nested switch statements

You can use one switch statement inside another switch statement(s).

The ? : Operator:

We have covered conditional operator ? : in previous chapter which can be used to replace **if...else** statements. It has the following general form:

```
Exp1 ? Exp2 : Exp3;
```

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

The value of a ? expression is determined like this: Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

****

Programs:

C++ if...else and Nested if…else

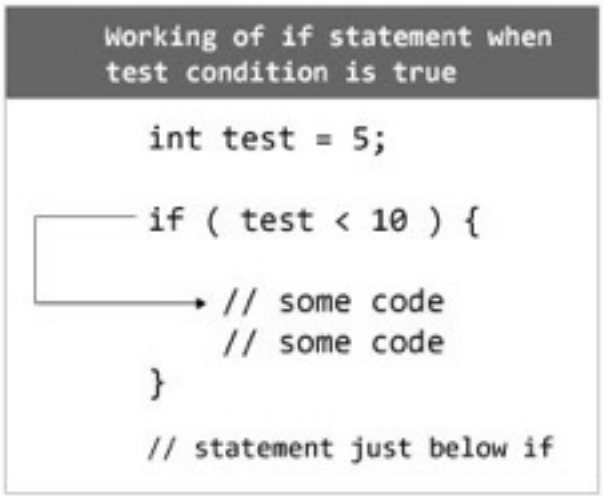
The if, if...else and nested if...else statement are used to make one-time decisions in C ++ Programming, that is, to execute some code/s and ignore some code/s depending upon the test condition. Without decision making, the program runs in similar way every time. Decision making is an important feature of every programming language using C++ programming. Before you learn decision making, you should have basic understanding of relational operators.

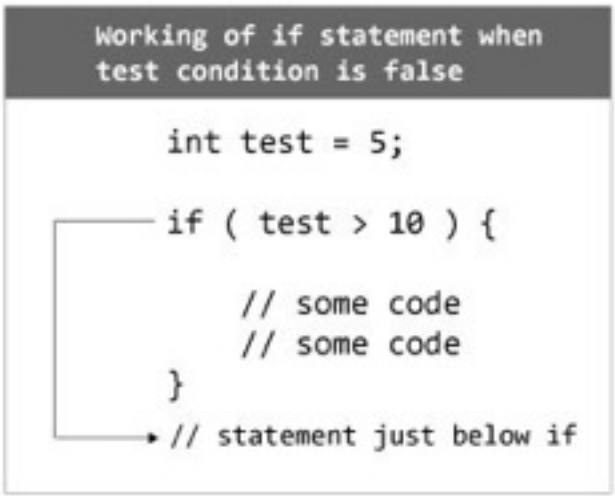
if Statement
if...else Statement
Nested if...else Statement
Conditional Operator

C++ if Statement

The if statement checks whether the test condition is true or not. If the test condition is true, it executes the code/s inside the body of if statement. But it the test condition is false, it skips the code/s inside the body of if statement.

Working of if Statement





The if keyword is followed by test condition inside parenthesis ( ). If the test condition is true, the codes inside curly bracket is executed but if test condition is false, the codes inside curly bracket { } is skipped and control of program goes just below the body of if as shown in figure above.

Flowchart of if

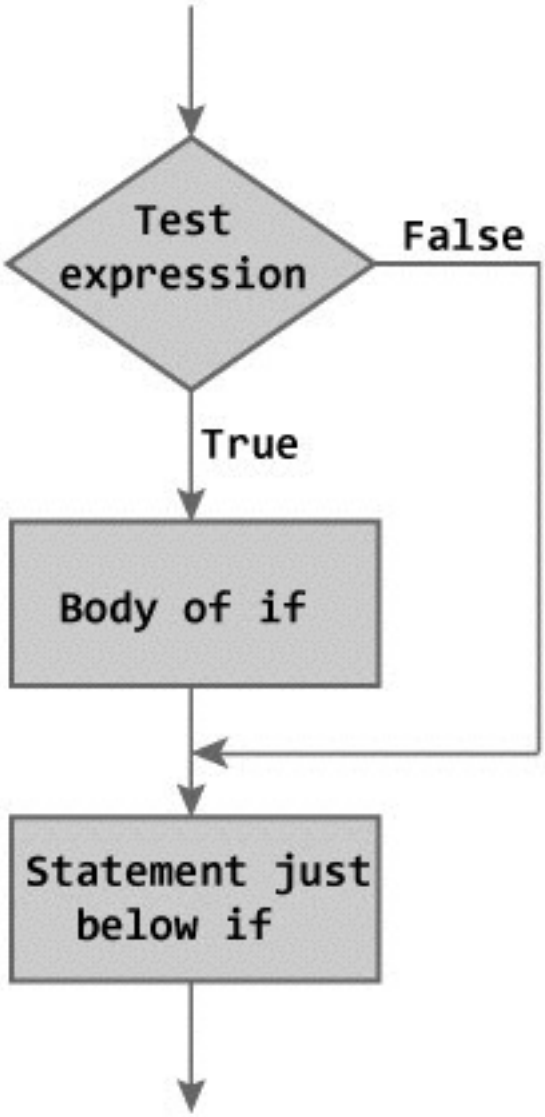


Figure: Flowchart of if Statement

Example 1: C++ if Statement

C++ Program to print integer entered by user only if that number is positive.

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

int main() {
    int number;
    cout<< "Enter an integer: ";
    cin>> number;
```

```
    if ( number > 0) {  // Checking whether an integer is positive or not.
        cout << "You entered a positive integer: "<<number<<endl;
    }

    cout<<"This statement is always executed because it's outside if statement.";
    return 0;

}
```

Output 1

```
Enter an integer: 5
You entered a positive number: 5
This statement is always executed because it's outside if statement.
```

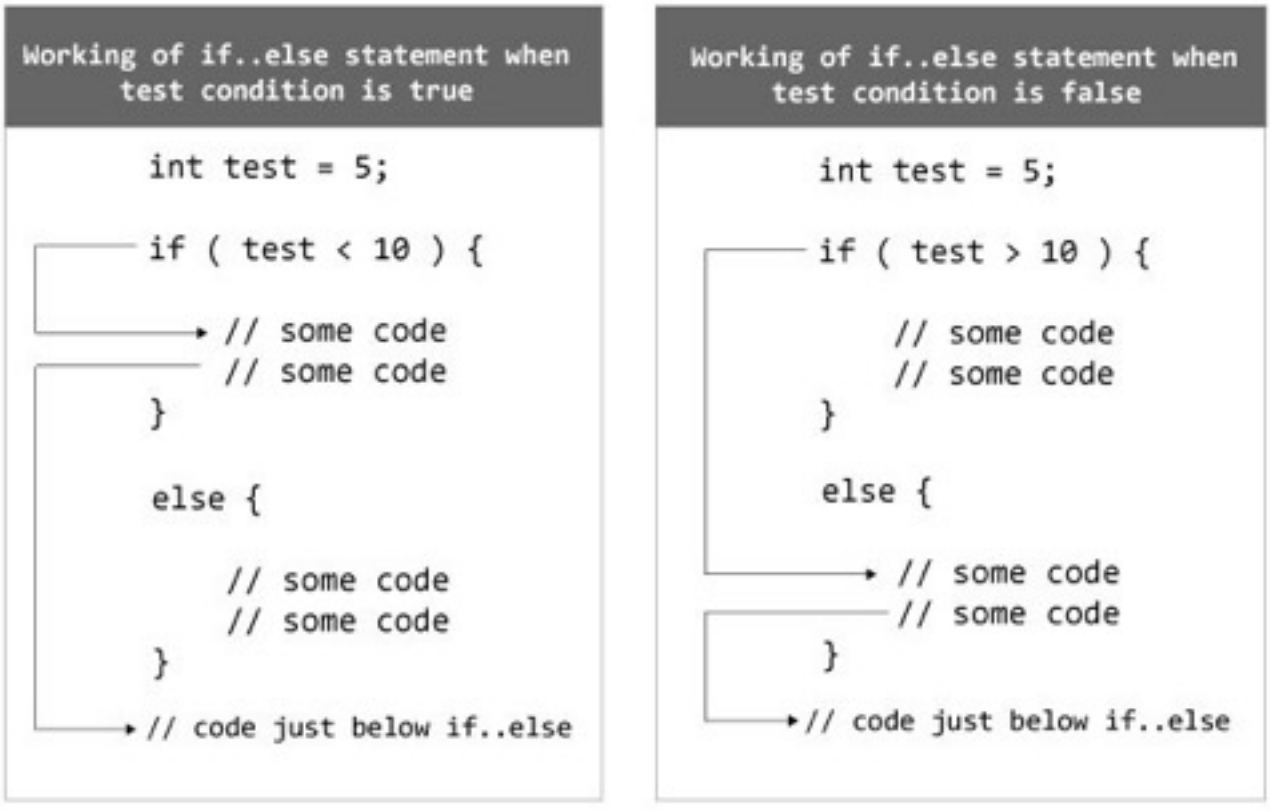
Output 2

```
Enter a number: -5
This statement is always executed because it's outside if statement.
```

C++ if...else

The if...else executes body of if when the test expression is true and executes the body of else if test expression is false.

Working of if...else Statement



The if statement checks whether the test expression is true or not. If the test condition is true, it executes the code/s inside the body of if statement. But it the test condition is false, it executes the code/s inside the body of else.

Flowchart of if...else

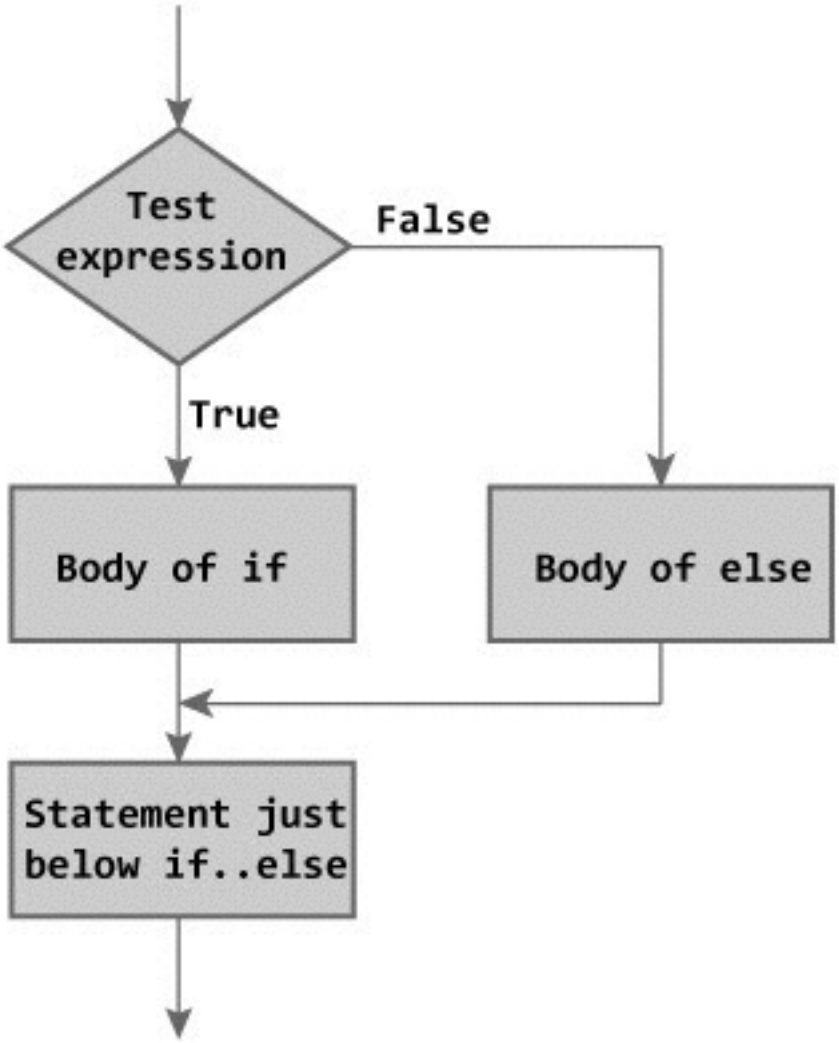


Figure: Flowchart of if...else Statement

Example 2: C++ if...else Statement

C++ Program to check whether integer entered by user is positive or negative (Considering 0 as positive)

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

int main() {
   int number;
   cout<< "Enter an integer: ";
   cin>> number;

   if ( number >= 0) {
      cout << "You entered a positive integer: "<<number<<endl;
   }

   else {
```

```
        cout<<"You entered a negative integer: "<<number<<endl;
    }

    cout<<"This statement is always executed because it's outside if...else statement.";
    return 0;

}
```

Output

```
Enter an integer: -4
You entered a negative integer: -4
This statement is always executed because it's outside if...else statement.
```

C++ Nested if...else

Nested if...else are used if there are more than one test expression.

Syntax of Nested if...else

```
if (test expression1){
    statement/s to be executed if test expression1 is true;
    }
    else if(test expression2) {
        statement/s to be executed if test expression1 is false and 2 is true;
    }
    else if (test expression 3) {
       statement/s to be executed if text expression1 and 2 are false and 3 is true;
    }
        .
        .
        .
    else {
          statements to be executed if all test expressions are false;
     }
```

The nested if...else statement has more than one test expression. If the first test expression is true, it executes the code inside the braces{ } just below it. But if the first test expression is false, it checks the second test expression. If the second test expression is true, if executes the code inside the braces{ } just below it. This process continues. If all the test expression are false, code/s inside else is executed and the control of program jumps below the nested if...else

Example 3: C++ Nested if...else Statement

C++ Program to check whether the integer entered by user is positive, negative or zero.

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

```
int main() {
   int number;
   cout<< "Enter an integer: ";
   cin>> number;

   if ( number > 0) {
      cout << "You entered a positive integer: "<<number<<endl;
   }
   else if (number < 0){
      cout<<"You entered a negative integer: "<<number<<endl;
   }
   else {
      cout<<"You entered 0."<<endl;
   }

   cout<<"This statement is always executed because it's outside nested if..else statement.";
   return 0;

}
```

Output

```
Enter an integer: 0
You entered 0.
This statement is always executed because it's outside nested if..else statement.
```

Conditional/Ternary Operator ?:

Conditional operators are the peculiar case of if...else statement in C++ Programming. Consider this if..else statement:

```
if ( a < b ) {
   a = b;
}
else {
   a = -b;
}
```

The above code can be written using conditional operator as:

```
a = (a < b) ? b : -b;
```

Both codes above check whether *a* is less than *b* or not. If *a* is less than *b*, value of *b* is assigned to *a* if not, -b is assigned to *a*.

C++ Programming
C++ Flow Control

C++ if...else
C++ for Loop
C++ do...while Loop
C++ break & continue
C++ switch Statement

C++ goto Statement
C++ Inheritance

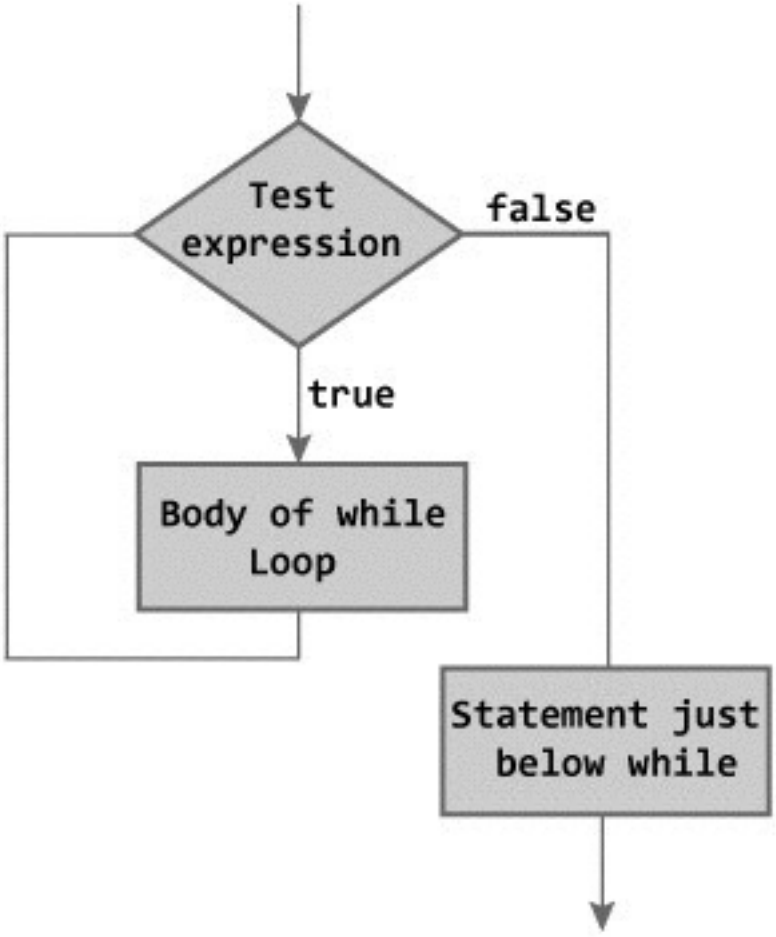


Figure: Flowchart of while Loop

Example 1: C++ while Loop

C++ program to find factorial of a positive integer entered by user. (Factorial of n = 1*2*3…*n)

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

int main() {
   int number, i = 1, factorial = 1;
   cout<< "Enter a positive integer: ";
   cin >> number;

   while ( i <= number) {
      factorial *= i;      //factorial = factorial * i;
      ++i;
   }
```

```
    cout<<"Factorial of "<<number<<" = "<<factorial;
    return 0;
}
```

Output

```
Enter a positive integer: 4
Factorial of 4 = 24
```

C++ do...while Loop

The do...while Loop is similar to while loop with one very important difference. In while loop, check expression is checked at first before body of loop but in case of do...while loop, body of loop is executed first then only test expression is checked. That's why the body of do...while loop is executed at least once.

Syntax of do...while Loop

```
do {
   statement/s;
}
while (test expression);
```

How do...while loop works?

The statement/s inside body of loop is executed at least once, that is, the statement/s inside braces { } is executed at least once. Then the test expression is checked. If the test expression is true, the body of loop is executed. This process continues until the test expression becomes false. Since the body of loop is placed before the test expression in do...while loop, the body of loop is executed at least once.

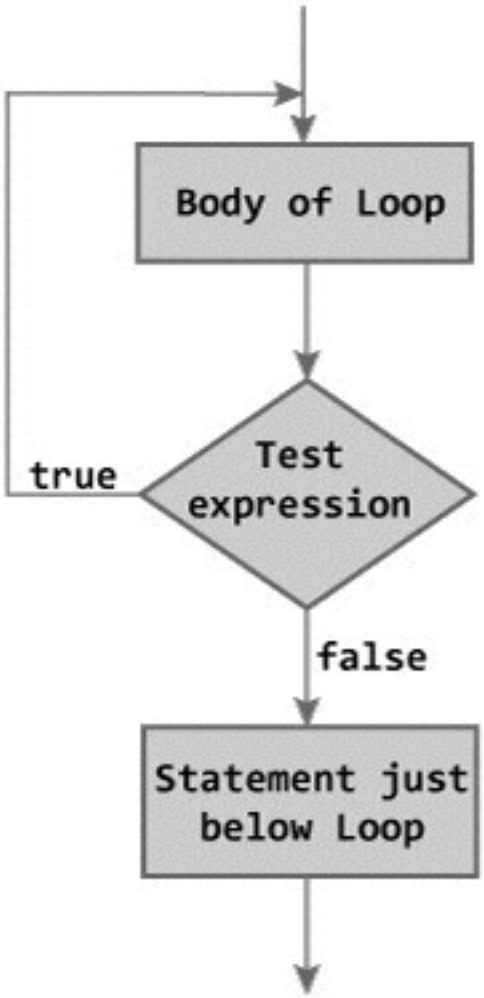


Figure: Flowchart of do...while Loop

Example 2: do...while Loop

C++ program to add numbers entered by user until user enters 0.

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

int main() {
   float number, sum = 0.0;

   do {
      cout<<"Enter a number: ";
      cin>>number;
      sum += number;
   }
   while(number != 0.0);

   cout<<"Total sum = "<<sum;

   return 0;
}
```

Output

```
Enter a number: 4.5
Enter a number: 2.34
Enter a number: 5.63
Enter a number: 6.34
Enter a number: 4.53
Enter a number: 5
Enter a number: 0
Total sum = 28.34
```

C++ break and continue Statement

There are two statements (break; and continue;) built in C++ programming to alter the normal flow of program.

Loops are used to perform repetitive task until test expression is false but sometimes it is desirable to skip some statement/s inside loop or terminate the loop immediately with checking test condition. On these type of scenarios, continue; statement and break; statement is used respectively. The break; statement is also used to terminate switch statement.

C++ break Statement

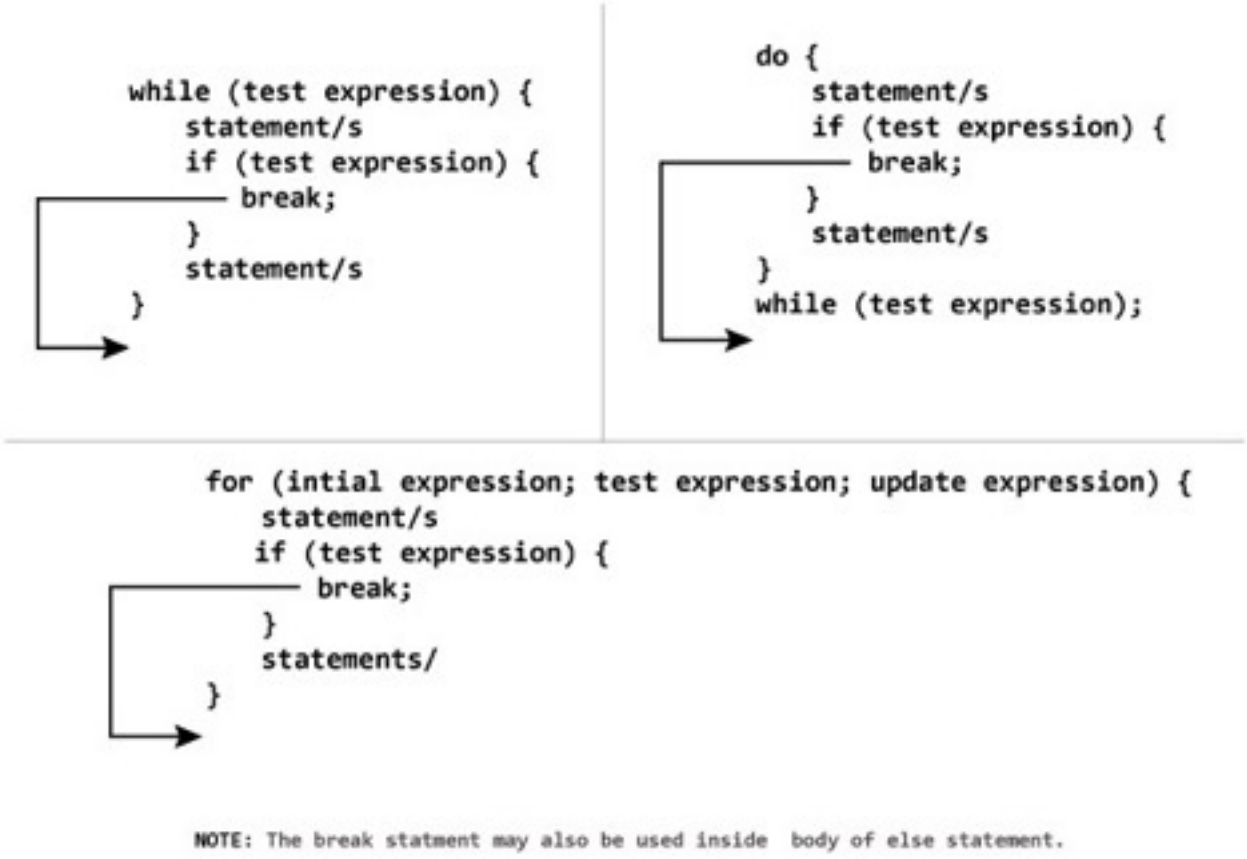
The break; statement terminates the loop(for, while and do..while loop) and switch statement immediately when it appears.

Syntax of break

```
break;
```

In real practice, break statement is almost always used inside the body of conditional statement(if...else) inside the loop.

Working of break Statement



NOTE: The break statment may also be used inside body of else statement.

Example 1: C++ break

C++ program to add all number entered by user until user enters 0.

```
// C++ Program to demonstrate working of break statement

#include <iostream>
using namespace std;
int main() {
   float number, sum = 0.0;

   while (true) {     // test expression is always true
      cout<<"Enter a number: ";
      cin>>number;

      if (number != 0.0) {
         sum += number;
      }
      else {
         break;    // terminating the loop if number equals to 0.0
      }

   }
   cout<<"Sum = "<<sum;
   return 0;
}
```

Output
Enter a number: 4
Enter a number: 3.4
Enter a number: 6.7
Enter a number: -4.5
Enter a number: 0
Sum = 9.6
In this C++ program, the test expression is always true. The user is asked to enter a number which is stored in variable *number*. If the user enters any number other than 0, that number is added to *sum* and stored to it and again user is asked to enter another number. When user enters 0, the test expression inside if statement is false and body of else is executed which terminates the loop. Finally, the sum is displayed.

C++ continue Statement

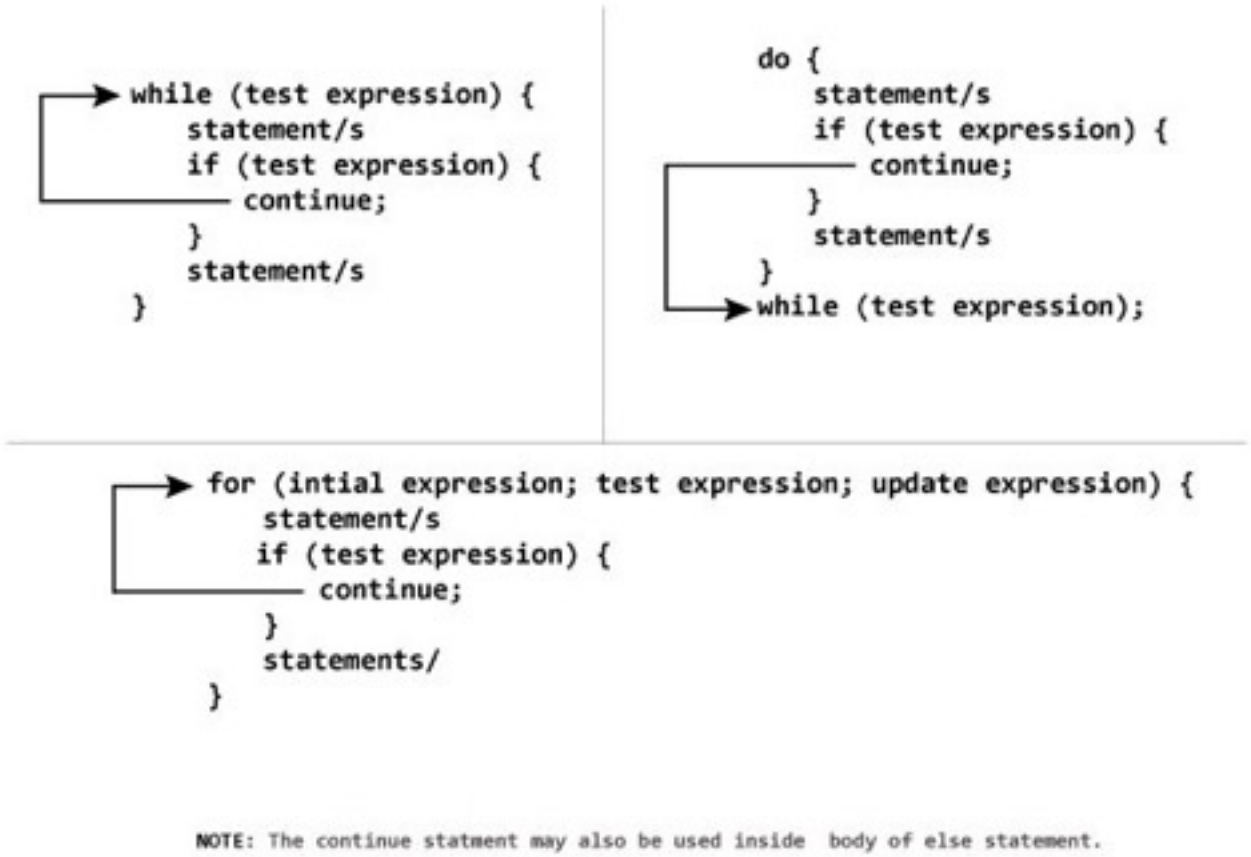
It is sometimes necessary to skip some statement/s inside the loop. In that case, continue; statement is used in C++ programming.

Syntax of continue

```
continue;
```

In practice, continue; statement is almost always used inside conditional statement.

Working of continue Statement



Example 2: C++ continue

C++ program to display integer from 1 to 10 except 6 and 9.

```
// C++ Program to demonstrate working of continue statement
```

```
#include <iostream>
using namespace std;
int main() {
   for (int i = 1; i <= 10; ++i) {
      if ( i == 6 || i == 9) {
         continue;
      }
      cout<<i<<"\t";
   }
   return 0;
}
```

Output

```
1       2       3       4       5     78       10
```

In above program, when *i* is 6 or 9, execution of statement cout<<i<<"\t"; is skipped inside the loop using continue; statement.

C++ switch Statement

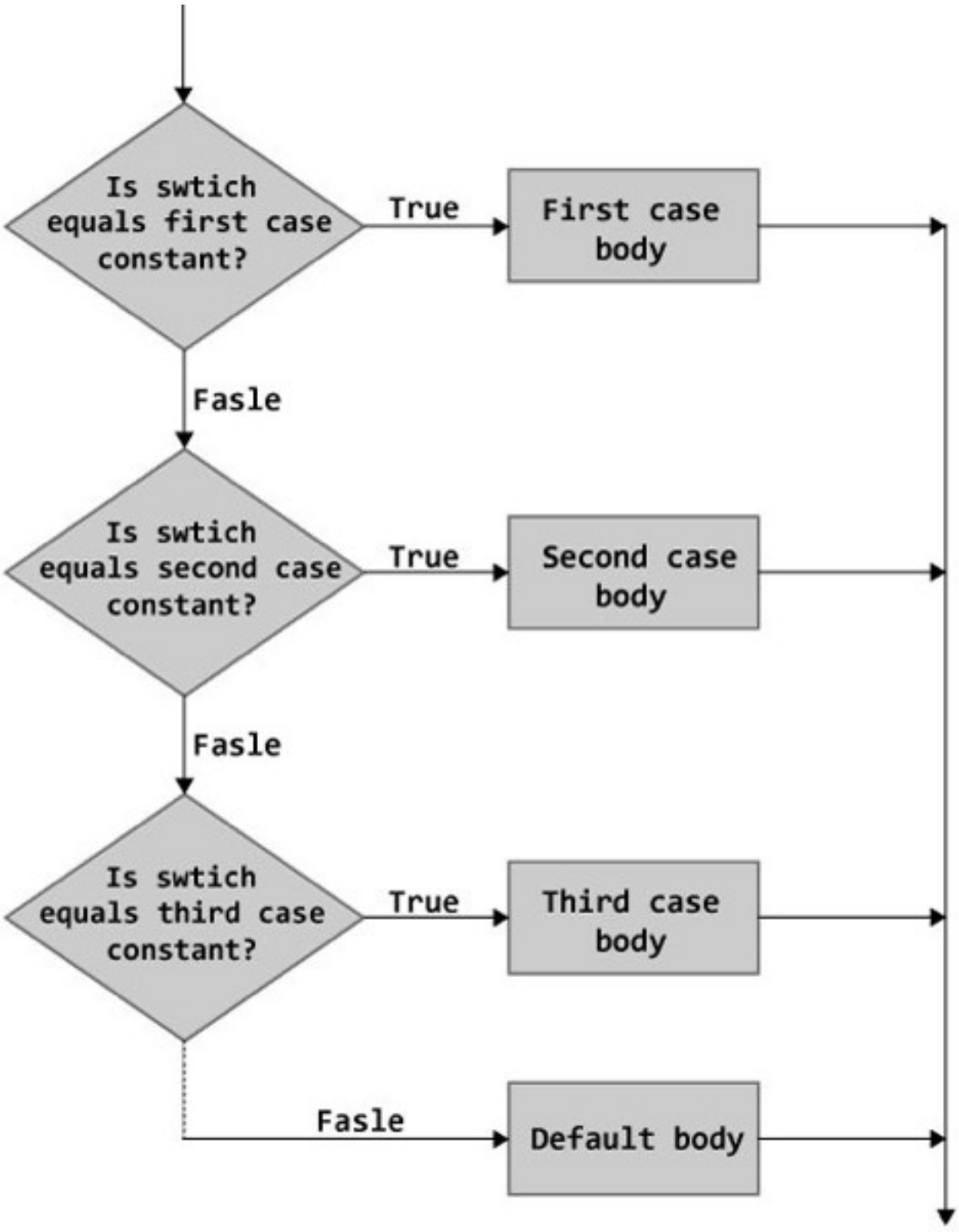
Consider a situation in which, only one block of code needs to be executed among many blocks. This type of situation can be handled using nested if...else statement but, the better way of handling this type of problem is using switch...case statement.

Syntax of switch

```
switch (n) {
case constant1:
  code/s to be executed if n equals to constant1;
  break;
case constant2:
  code/s to be executed if n equals to constant2;
  break;
  .
  .
  .
default:
  code/s to be executed if n doesn't match to any cases;
}
```

The value of *n* is either an integer or a character in above syntax. If the value of *n* matches constant in case, the relevant codes are executed and control moves out of the switch statement. If the *n* doesn't matches any of the constant in case, then the default statement is executed and control moves out of switch statement.

Flowchart of switch Statement



Example 1: C++ switch Statement

```
/* C++ program to demonstrate working of switch Statement */
/* C++ program to built simple calculator using switch Statement */


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

```
int main() {
   char o;
   float num1,num2;
   cout<<"Select an operator either + or - or * or / \n";
   cin>>o;
   cout<<"Enter two operands: ";
   cin>>num1>>num2;

   switch(o) {
      case '+':
         cout<<num1<<" + "<<num2<<" = "<<num1+num2;
         break;
      case '-':
         cout<<num1<<" - "<<num2<<" = "<<num1-num2;
         break;
      case '*':
         cout<<num1<<" * "<<num2<<" = "<<num1*num2;
         break;
      case '/':
         cout<<num1<<" / "<<num2<<" = "<<num1/num2;
         break;
      default:

         /* If operator is other than +, -, * or /, error message is shown */
         printf("Error! operator is not correct");
         break;
   }

   return 0;
}
```

Output

```
Select an operator either + or - or * or /
-
Enter two operands: 2.3
4.5
2.3 - 4.5 = -2.2
```

The break statement at the end of each case cause switch statement to exit. If break statement is not used, all statements below that case statement are also executed.

****