**BCA-311: OBJECT ORIENTED PROGRAMMING IN C++**

**UNIT: 2**

**Instructions :** Instructions are the commands in the program that instructs the compiler to do certain action. Basically it gives the instruction to the compiler on how to achieve the goal of the program.

## There are three types of instructions:

### Type Declaration Instructions

These instructions inform the compiler about the type of variables used. That means, whenever a variable is used in the program, we have to specify what types of data it can hold – like integer, float, double, character etc.

int a;

### Arithmetic Instructions

These instructions are used to perform some arithmetic calculation within the program. It uses arithmetic operators like +, -, X, /, %, =, ++,–, +=,-+ etc.

float A=1+2-3%6;

### Control Instructions

Like the name suggests, these instructions are used to control the flow of the program execution. They maintain certain order in which program needs to be executed. These order of execution may be based on certain conditions or may be based on certain values – may be input values or some result values.

## There are four types of control instructions

### *3.a. Sequence Control Instructions*

These instructions are responsible for executing the instructions one after the other, as they appear in the program. There are not any checks on the condition or values to control the execution.

### *3.b Decision / Selection Control Instructions*

This will have set of conditions to execute the instructions within it. If the condition is true, then it will execute the instructions, else it will execute other set of instructions. This type of instructions use if or while statements to make the decision.

### *3.c. Loop Control Instructions*

There will be certain need for executing a set of instruction for certain number of times. This is done by using loop control statement. These loops will have limited number of iterations or will have certain conditions which in turn will give iterations for the instructions to be executed.

### *3.d. Case Control Instructions*

This is similar to decision control instructions where condition will be checked for the execution of the instructions. But this instructions will use switch case statements to check the conditions.

So in Unit 2 we will understand about Decision / Selection Control Instructions and Loop Control Instructions in detail with programming exercise.

# Decision making in C++ - if, if else and else if clause

Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. C++ handles decision-making by supporting the following statements,

* *if* statement
* *switch* statement
* conditional operator statement

The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are,

1. Simple *if* statement

if(condition)

{

statement-inside;

}

statement-outside;

* 1. *if....else* statement

if(condition)

{

statement-block1;

}

else

{

statement-block2;

}

* 1. Nested *if....else*statement

if(condition)

{

if(condition 1)

{

statement-block1;

}

else

{

statement-block2;

}

}

else

{

statement-block3;

}

* 1. *else if* statement

if(condition 1)

{

statement-block1;

}

else if(condition 2)

{

statement-block2;

}

else if(condition 3 )

{

statement-block3;

}

else

default-statement;

**Conditional Operator Statement**

Sayntax:

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.

### The ? is called a ternary operator because it requires three operands and can be used to replace if-else statements.

### Iteration statements (Loops)

Loops repeat a statement a certain number of times, or while a condition is fulfilled. They are introduced by the keywords **while, do\_while, and for**.

#### The while loop

The simplest kind of loop is the while-loop. Its syntax is:  
  
while (expression)

{

statement  
}  
The while-loop simply repeats statement while expression is true. If, after any execution of statement, expression is no longer true, the loop ends, and the program continues right after the loop. For example, let's have a look at a countdown using a while-loop:

// custom countdown using while

#include <iostream>

int main ( )

{

int n = 0;//initialization

while (n>0) // condition

{

cout << n << ", ";// statements

--n;// decremrnt

}

cout << "liftoff!\n";

}

The first statement in main sets n to a value of 10. This is the first number in the countdown. Then the while-loop begins: if this value fulfills the condition n>0 (that n is greater than zero), then the block that follows the condition is executed, and repeated for as long as the condition (n>0) remains being true.

#### The do-while loop

A very similar loop is the do-while loop, whose syntax is:  
  
do {

Statement1;

Statement2;

} while (condition);  
  
It behaves like a while-loop, except that condition is evaluated after the execution of statement instead of before, guaranteeing at least one execution of statement, even if condition is never fulfilled. For example, the following example program echoes any text the user introduces until the user enters goodbye:

// echo machine

#include <iostream.h>

#include <string.h>

int main ()

{

string str;

do {

cout << "Enter text: ";

getline (cin,str);

cout << "You entered: " << str << '\n';

} while (str != "goodbye");

}

The do-while loop is usually preferred over a while-loop when the statement needs to be executed at least once, such as when the condition that is checked to end of the loop is determined within the loop statement itself. In the previous example, the user input within the block is what will determine if the loop ends. And thus, even if the user wants to end the loop as soon as possible by entering goodbye, the block in the loop needs to be executed at least once to prompt for input, and the condition can, in fact, only be determined after it is executed.

#### The for loop

The for loop is designed to iterate a number of times. Its syntax is:  
  
for (initialization; condition; increase)

{

statement;

}  
  
Like the while-loop, this loop repeats statement while condition is true. But, in addition, the for loop provides specific locations to contain an initialization and an increase expression, executed before the loop begins the first time, and after each iteration, respectively. Therefore, it is especially useful to use counter variables as condition.  
  
It works in the following way:

1. initialization is executed. Generally, this declares a counter variable, and sets it to some initial value. This is executed a single time, at the beginning of the loop.
2. condition is checked. If it is true, the loop continues; otherwise, the loop ends, and statement is skipped, going directly to step 5.
3. statement is executed. As usual, it can be either a single statement or a block enclosed in curly braces { }.
4. increase is executed, and the loop gets back to step 2.
5. the loop ends: execution continues by the next statement after it.

// countdown using a for loop

#include <iostream>

using namespace std;

int main ()

{

for (int n=10; n>0; n--) {

cout << n << ", ";

}

cout << "liftoff!\n";

}

This loop will execute 50 times if neither n or i are modified within the loop:  
  
http://www.cplusplus.com/doc/tutorial/control/for_loop.png

**ASSIGNMENT#2**

Q1. Based on all the syntax those given above write programs with output Using C++.

## C++ Switch Statements

Use the switch statement to select one of many code blocks to be executed.

### Syntax

switch(expression) {  
  case 1:  
    *// code block*  
    break;  
  case 2:  
    *// code block*  
    break;  
  default:  
    *// code block*  
}

This is how it works:

* The switch expression is evaluated once
* The value of the expression is compared with the values of each case
* If there is a match, the associated block of code is executed
* The break and default keywords are optional, and will be described later in this chapter

The example below uses the weekday number to calculate the weekday name:

### Example

int day = 3;  
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;

deafault:

cout << "Sorry";  
}

output

Wednesday

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.

The **break** statement has the following two usages in C++

* When the **break** statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
* It can be used to terminate a case in the **switch** statement

#include <iostream>

int main () {

// Local variable declaration:

int a = 10;

// do loop execution

do {

cout << "value of a: " << a << endl;

a = a + 1;

if( a > 15) {

// terminate the loop

break;

}

} while( a < 20 );

cout << "I am out of loop now!"

return 0;

}

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

**Output:**

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

I am out of loop now!

## Example: break statement in for loop

#include <iostream>

int main(){

int var;

for (var =200; var>=10; var--)

{

cout<<"var: "<<var<<endl;

if (var==197) {

break;

}

}

cout<<"Hey, I'm out of the loop";

return 0;

}

**Output:**

var: 200

var: 199

var: 198

var: 197

Hey, I'm out of the loop

## The continue Keyword

The continue statement left one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

### Example

for (int i = 0; i < 10; i+1)

{  
 if (i == 4)

{  
    continue;  
}  
  cout << i << "\n";  
}

### Output

0 1 2 3 5 6 7 8 9

## Use of continue in While loop

#include <iostream>

int main(){

int j=6;

while (j >=0) {

if (j==4) {

j--;

continue;

}

cout<<"Value of j: "<<j<<endl;

j--;

}

return 0;

}

**Output:**

Value of j: 6

Value of j: 5

Value of j: 3

Value of j: 2

Value of j: 1

Value of j: 0

The goto statement

The goto statement is a jump statement which is sometimes also referred to as unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function.

## Syntax

The syntax of a goto statement in C++ is −

goto label;

..

...

.

.

label:

statement1;

statement2;

statement3;

Where **label** is an identifier that identifies a labeled statement.

A labeled statement is any statement that is preceded by an identifier followed by a colon (:).

## Example

#include <iostream>

int main ()

{

// Local variable declaration:

int a = 10;

// do loop execution

LOOP:do {

if( a == 15) {

// skip the iteration.

a = a + 1;

goto LOOP;

}

cout << "value of a: " << a << endl;

a = a + 1;

}

while( a < 20 );

return 0;

}

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

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 16

value of a: 17

value of a: 18

value of a: 19