SESSION 1-3: Decision Making Control Structures

These are statements used in C++ in making decisions or in selecting alternatives depending on a specified condition hence such structures are normally used by programmers when it becomes necessary to specify one or more alternatives from which a choice is to be made during program execution. The following are the main decision making statements in C++.

1-3.1 IF Statement

The simple if statement is used when an action is to be taken if a given condition is true. No action is taken if otherwise and program execution continues with the statement following the if statement. The general format of the simple if statement is as follows:

```
if (ConditionIsTrue)
    StatementIfTrue;
```

where *ConditionIsTrue* is the condition being tested for. *StatementIfTrue* is the action to be taken or the statement to be executed if the condition (*ConditionIsTrue*) is true. Generally, the if statement accepts only one statement, however, it is necessary at times to execute more than one statement when the given condition is satisfied. When this happens, the statements to be executed when the condition is true must be enclosed in a pair of curly brackets ({ and }) to mark the beginning and the end of the if block. The format of the simple IF statement then becomes

```
if (ConditionIsTrue)
{
        statement1;
            :
        statementn
}
```

where *statement1* to *statementn* are the statements to be executed when the condition being tested for returns a true result.

Example:

```
if  (mark > 90)
        grade = 'A';
```

IF ... ELSE STATEMENT

This is a two-way selection if statement. Unlike the simple if statement, this form of the if statement has actions to be taken when the condition being tested for returns true or false result. The general form is;

```
if   (ConditionIsTrue)
        StatementIfTrue;
else
        StatementIfFalse;
```

It is possible to have more than one statement to be executed if the condition being tested for evaluates to true or false. If that is the case the statements involved must be enclosed in a pair of curly brackets. The format in this case will be as follows:

```
if   (ConditionIsTrue)
{
        statement1;
             :
        statementn;
}
else
{
        statement1;
             :
        statementn;
}
```

1-3.2 Multiway or Cascaded IF..ELSE… Statement

The multiway if statement is similar to the if….else statement, however in a multiway if statement there is at least three branches or alternatives to choose from. The general format is as follows:

```
if (condition1IsTrue)
 {
     statements1;
 }
else if (condition2IsTrue)
 {
          statements2;
 }
        :
```

```
else if (conditionnIsTrue)
  {
        statementsn;
   }
else
   {
       else_statements;
   }
```

By way of example, consider the following grading system.

Mark	Grade
Above 70	A
60 to 69	B
50 to 59	C
40 to 49	D
below 40	F

Assume in this example, it is required to code an if statement that will allow marks entered into a program to be assigned the correct grade. Here, there are five alternatives to choose from based on a given mark. This can be coded using the multiway if…else statement as follows:

```
if (mark > 69)
    Grade = 'A';
else if (mark > 59)
    Grade = 'B';
else if (mark > 49)
    Grade = 'C';
else if (mark >39)
    Grade = 'D';
else
    Grade = 'F';
```

During program execution, if a mark of 50 say is entered, and the if statement is encountered, the first two conditions will evaluate to false causing no action to be taken other than testing for the next condition. The third condition will evaluate to true hence the variable Grade will be assigned the letter 'C'. Program execution will then continue with the statement immediately following this multiway if statement. Note that it is also possible to use five simple if statements for the above as follows:

```
if (mark > 69)
        Grade = 'A';
 if (mark > 59 && mark<70)
```

```
            Grade = 'B';
if (mark > 49 && mark<60)
            Grade = 'C';
if (mark >39 &&mark<50)
            Grade = 'D';
if(mark<40)
            Grade = 'F';
```

However, program execution may be a little slow as compared to the cascade if statements. Note also that additional comparisons using the && (and) are needed for some of the if statements else the if statements will not work. For example, if a mark entered at runtime is say 70 then the multiway if statement will make only one comparison as the first test will evaluate to true while with the simple if statements even when the first evaluates to true, the compiler will still have to make unnecessary tests thus slowing program execution. You must therefore be analytical when choosing between a series of simple if statements and a multiway if statements.

NESTED IF STATEMENT

Sometimes, it becomes necessary to have an if statement within an if statement. Such an if structure is usually referred to as nested if. For a nested if, the closing bracket, } of an inner if must come before that of an outer one. A typical structure of a nested if statement is as follows:

For a nested simple if statement we have

```
if (condition1IfTrue)
        if (condition2fTrue)
           StatementIfConditionsAreTrue;
```

The above if statement will cause the statement *StatementIfConditionsAreTrue* to be executed only when the two if statements return true results. Note that if either of the if statements has more than one statement to be executed when the if condition is true then the statements must be enclosed in a pair of brackets.

For a nested if…else statement we have

```
if (condition1IfTrue)
        if (condition2fTrue)
           StatementIfConditionsAreTrue;
        else
                StatementIfInnerConditionsIsFalse;
else
        StatementIfOuterConditionsIsFalse;
```

1-3.3 The SWITCH Statement

The case or switch statement strictly speaking allows you to rewrite code that uses a lot of if else or multiway if statements. Note that using a number of if…else statements make program logic much harder to read and understand as such the use of the switch statement may be used to make program logic much easier to understand. The format of the switch statement is as follows:

```
switch (controllingExpression)
 {
      case value1:
           action1;
           break;
      case value2:
           action2;
           break;
              :
      case value_n:
           action_n;
           break;
     default:
           OtherwiseAction;
}
```

where *ControllingExpression* is an expression or variable whose content is being tested for. The *value1* to *value_n* specify some of the possible values that the *controllingExpression* can evaluate to or assume. The *action1* to *action_n* also specify the corresponding actions to be taken depending on the value of the *controllingExpression*. First, the value of *controllingExpression* is evaluated and compared to the values for a match. When a match is found, the action associated with the matched value is taken. It then exits the loop on executing the break statement. If no match is found, the *controllingExpression* is assumed to evaluate to the default (same as otherwise) and hence the *OtherwiseAction* is executed. Note the breaks in all the cases. The *break* statement is very important and any omission of a break statement in the above format can result in some wrong actions being taken. If a break statement is omitted the compiler will not issue an error message as the switch statement will be syntactically correct. Note that a pair of brackets is not required if any of the actions consist of more than one statement.

If in the above format the first break statement is omitted, then should the *controllingExpression* evaluates to value1 the action1 will be taken but the compiler will continue to execute action2. The break statement tells the computer to leave the switch statement. This means that when the compiler starts executing a case statement, it will not stop executing subsequent statements until a break statement or the end of the switch statement is encountered.

At times, it is possible to have a particular statement executed for different values of the controllingExpression. When this happens and assuming that the first five case values require the same action to be taken then the action against the first four case values need not be specified, however, the action must be specified for the fifth case value followed by the break statement.

Let us consider the following problem:

In a game of crossword SCRABBLE, words are formed by players by using small tiles with a letter and a face value on them. The face value of the letters varies from one letter to letter depending on the rarity of the letters. The face values for the 26 letters are,

1 for each of A, E, I, L, N, O, R, S, T and U,
2 for each of D and G,
3 for each of B, C, M and P,
4 for each of F, H, V, W and Y,
5 for K, 8 for J and X,
and 10 for Q and Z.

You are required to write a switch statement for the above. Assume the characters in a word will be entered one at a time.

Note that we could have used the if statement for example but then that will require at least 25 if statements. That will surely make the program very lengthy. However, we can use a switch statement for the comparisons as follows. Let us assume that the controlling variable is letter and that the value of a letter is to be added to the current value of the variable score.

```
Switch (letter)
{
   case ‘D’:
   case ‘G’:   score =score + 2;break;
   case ‘B’:
   case ‘C’:
   case ‘M’:
   case ‘P’:   score = score +3;break;
   case ‘F’:
   case ‘H’:
   case ‘V’:
   case ‘W’:
   case ‘Y’:   score = score + 4;break;
   case ‘K’:   score = score + 5;break;
   case ‘J’:
   case ‘X’:   score = score + 8;break;
   case ‘Q’:
   case ‘Z’:   score = score+10;break;
 default:      score = score + 1;
```

```
}
```

In the above coding, the default will be for the letters A, E, I, L, N, O, R, S, T and U. If we had used the multiway if statement, the coding would have covered at least two pages. A more compact way of writing the above coding may be as follows:

```
 Switch (letter)
{
   case 'D':   case 'G':
            score =score + 2;break;
   case 'B':   case 'C':    case 'M':    case 'P':
             score = score +3;break;
   case 'F':    case 'H':   case 'V':     case 'W':   case 'Y':
             score = score + 4;break;
   case 'K':
            score = score + 5;break;
   case 'J':   case 'X':
              score = score + 8;break;
   case 'Q':  case 'Z':
              score = score+10;break;
  default:
             score = score + 1;
}
```

1-3.4 The conditional operator (?)

The *conditional operator*, also known as the *ternary operator* or *arithmetic if*, can be used in C++ as a decision making statement. The ? is a conditional operator and hence why we are treating it under decision making statements. The general syntax is as follows:

```
(conditionalTest)?TrueAction:FalseAction;
```

where *conditionalTest* defines the condition being tested for. If it evaluates to true then *TrueAction* is taken and that will be the value of the conditional expression as a whole. If the condition evaluates to false then the *FalseAction* is taken. Example

```
larger=(num1>num2)?large:small;
```

where the above means that if the value of num1 is greater than that of num2 then assign large to larger otherwise assign small to larger. The equivalent of the above statement using the if statement will be as follows:

```
if (num1 > num2)
   larger=num1;
else
  larger = num2;
```

Example: Let us assume that in a certain school students are made to take three exams in a term and the best two marks are used in computing a student's term average. It been established that every student get his/her highest mark in the third exams. If x, y and z hold the first, second and the third exams score respectively, then write a program using the conditional operator, ? to display the average mark of a student.

Program listing:

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

     int main()
     {
          int x,y,z;
          double average;
          cout <<"Please enter marks of a student ";
          cin >>x >>y >>z;
          average=(x>y)?(x+z)/2.0:(y+z)/2.0;
          cout.showpos;
          cout <<"The average of best 2 marks is " <<average;
          cout <<endl;
          return 0;
     }
```

Program run:

```
Please enter marks of a student 25 20 30
The average of best 2 marks is 27.5
Press any key to continue
```

When the program runs, 25 and 20 are compared. Since 25 is greater than 20, 25 and 30 are used for the average. The average of 25 and 30 is 27.5 and hence why the program displays 27.5.

Self Assessment 1-3

1. What output will be produced by the following code when executed within a program

```
int x=3;
if(x>2)
   if(x>6)
        cout<<"you have to welcome me "<<x<<" times";
esle
  cout<<"it appears I am not welcomed ";
cout<<"thanks for the time";
```

2. Rewrite the following statement using the if statements
luck=(first=(x<y)?x:y==x)?2*x:3*y;

3. Using the expression in question (2) or your answer to question (2), what will be the values of *first* and *luck* when the statement is executed given that x=3 and y=4?

Suggested Answers

2.

```
if(x<y)
        first=x;
   else
        first=y;
 if(first==x)
   luck=2*x;
else
   luck=3*y;
```

1. The values that will be stored in first and luck are 3 and 6 respectively .

SESSION 2-1: LOOPING CONTROL STRUCTURE

Iterative or looping statements are those statements that allow for one or more statements to be executed a number of times. C++ has three main looping statements and these are the while, do… while and the for statements.

2-3.1 The WHILE Statement

The while loop allows program statement(s) to be executed as long as a given logical expression between a pair of brackets is true. Here, it is not always possible to specify the exact number of times instructions are to be repeated. As such, the while statement is best used when the number of times the instructions are to be executed are not known in advance but rather when instructions are to be executed as long as a given condition evaluates to true. The structure of the while statement is,

```
while  (ConditionIsTrue)
{
```

```
        ProgramStatement1;
         :
        ProgramStatement n
    }
```

or, if only a single program statement is to be executed,

```
    while   (ConditionIsTrue)
       ProgramStatement;
```

where *ConditionIsTrue* is the condition controlling the number of times the loop is to be executed. The program statement(s) are executed when the condition evaluates as true. This means that before the loop is executed for the first time, the value of the control variable must be such that it will cause the ConditionIsTrue to evaluate to true result. The while statement is sometimes referred to as a pre-test condition because loop statements are executed after the control expression has been tested. Note that somewhere inside the loop the value of the control variable, that is the variable which is controlling the loop (or the variable in the condition being tested) must change so that the loop can finally exit when the condition evaluates to false.

Note that it is also possible to have a nested while statements. However, the closing bracket, } of an inner while statement must come before that of an outer one. In other words, an inner while statement must start and end within the { and } of an outer while statement.
The following program illustrates the use of the while statement in calculating the factorial of a number.

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

   int main()
   {
      int factorial=1, number, storeNumber;
      cout<< 'Please Enter a number to calculate its factorial';
      cin>> number;
   while (number < 0)
    {
     cout<<'\n Please enter a positive number';
     cin>>number;
    } // end of while statement
   storeNumber=number;
   while (number>1)
   {
      factorial = factorial * number;
      number--
   } // end of while statement
   cout<<'The factorial of ' <<storeNumber <<' is'  << factorial;
```

```
return 0;
} // end of the main function
```

2-3.2 The DO…WHILE Statement

The do… while statement is also used to repeat the execution of loop statements until the expression controlling the loop evaluates to false. The do…while statement is similar to the while loop, however, with the do…while statement, the conditional test occurs at the end of the loop. That is, unlike the while statement in which program statement(s) may not be executed at all if initially the condition being tested for evaluates to false, the program statement(s) constituting the do…while loop body is guaranteed to be executed at least once. The do…while is therefore said to be a post test loop statement. The format is,

```
do
{
    ProgramStatements;
} while (ConditionIsTrue);
```

where *ConditionIsTrue* has the same meaning as above. Somewhere inside the loop the value of the control variable must change so that the loop can finally exit when the condition evaluates to false.

It must be noted that it is also possible to have a nested do…while loop. The while statement of an inner do…while must come before that of the outer do…while.

We will write the factorial program this time using the do…while statement to illustrate how the do...while statement can be used

```
#include<iostream>
using namespace std;
int main()
{
   int factorial=1, number, storeNumber;
do
 {
  cout<<'\n Please enter a positive number';
  cin>>number;
 } while (number < 1);
storeNumber=number;
```

```
do
{
   factorial = factorial * number;
   number--
} while (number>1);
cout<<'The factorial of ' <<storeNumber <<' is'  << factorial;
return 0;
} // end of the main function
```

2-3.3 The FOR Statement

The for statement is used to specify the number of times that a set of instructions is to be executed. It is therefore best used when the number of times that the instructions are to be executed is known in advance. The general format is as follows.

```
for (initialExpression; finalExpression;updateExpression)
{
        Programstatements;
}
```

or for a single loop statement it is

```
for (initialExpression; finalExpression;updateExpression)
        Programstatement;
```

where *initialExpression* is the value the control variable is to assume during the first execution of the for statement. The FinalExpression also specifies the last value that the control variable can assume. As such, the control variable can assume values in the range of *InitialExpression* to *FinalExpression* (inclusive) only. Note that in C++, each time the loop statement is executed, the value of the control variable is either incremented or decremented by a specified value or expression. This is achieved by using updateExpression. The programStatement or programStatements are the loop statements. It is also possible to have a for loop within another for loop, that is nested for do loops.

The value of the control variable must not be changed within the loop else the loop may not be executed the required number of times. Please note that it is allowed for the control variable's value to be changed but it's a bad programming practice. The value of the control variable should only be changed in the update or increment expression.

As an example involving the use of the for statement, we will consider the factorial problem once again.

```
#include<iostream>
using namespace std;
int main()
{
    int factorial=1, number;
    cout<< 'Please Enter a number to calculate its factorial';
    cin>> number;
while (number < 0)
 {
   cout<<'\n Please enter a positive number';
   cin>>number;
 } // end of while statement
storeNumber=number;
for (int num=number; num>1; num--)
    factorial = factorial * num;
cout<<'The factorial of ' <<number <<' is'  << factorial;
return 0;
} // end of the main function
```

THE FOR STATEMENT AND THE OPTIONAL COMMA

The for statement can use an optional comma (,) to specify more than one instruction in any of the three fields such as the initialisation or the update expression. The function of the comma is actually to separate instructions where under normal circumstance only one instruction is required or expected. Consider the following example where two variables are used in the for fields.

for(**m=0, n=10**;n<100;**n++,m++**)

The above for statement performs the following functions:

(i) It initialises the variables m and n to 0 and 10 respectively.

(ii) Each time the loop is to be repeated, the values of both m and n are increased by 1 at the same time.

(iii) The loop is repeated until n is greater than or equal to 100.

Question: What will be the output of the following program when run. Explain your answer.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
  int i,j,k;
  cout <<"I\tJ" <<endl;
  cout <<"==========\n";
```

```
 for (i=0,j=2;i<10;j+=2,i++)
  cout <<i <<"\t" <<j <<endl;

    system("PAUSE");
    return 0;
}
```

Program run (Output)

```
I                J
============
0                2
1                4
2                6
3                8
4               10
5               12
6               14
7               16
8               18
9               20
Press any key to continue . . .
```

INFINITE FOR LOOP

In the format of the for loop, either of the three expressions used to specify initialisation, update and terminating conditions can be made optional, that is one need not necessarily have to specify every part of the for loop. Depending on which ones are made optional an infinite loop may be obtained. If the first or the second expression is omitted, the terminating semi-colon must still be specified as shown in the following examples. Even if all expressions are made optional, there should still be two semi-colons. We shall consider omitting each of the expressions one at a time.

(i) If we omit the initialisation of the control variable, the loop will not be executed at all unless the control variable had a value assigned to it before the start of the loop. The main reason why the loop does not get executed at all is that depending on which C++ compiler is used, an uninitialised variable is automatically initialised to a very high or very small random number depending on the data type of the control variable. For example Bloodshed C++ usually initialises integer variables to 575 and double to 1.22015e-311. In the following program, nothing will be displayed by the *for* loop. Visual C++ on the other hand will set variables to -858993460 and -9.25596e+061 for integer and double variables respectively.

```
#include <iostream>
```

```
#include <cstdlib>
using namespace std;
int main()
{
  int i,j,k;
   cout <<i;
   for (;i<10;i++)
        cout <<i endl;
  system("PAUSE");
 return 0;
}
```

Program run:

575 Press any key to continue…

Note that the 575 displayed is the value Bloodshed Dev C++ compiler assigned to i. It is not always that 575 will be displayed for *int* variables.

(ii) If we omit the second expression, that is the terminating condition, then we have an infinite loop. This is because even though the loop variable will be initialised and then updated at each execution of the loop statements, there is no condition to cause or force the loop to be exited. The following example explains this.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
  int i,j,k;
   for (i=0;;i++)
   cout <<i <<endl;
 system("PAUSE");
 return 0;

}
```

If the above program is run, it will display 1, 2, 3,… to infinity.

(iii) Omitting the third expression, that is the update in a for loop statement also results in an infinite looping. This is because the value of the control variable remains constant and hence there is nothing to force the loop to terminate. The program below will print an uncountable number of number 0s (zeroes)

```
#include <iostream>
#include <cstdlib>
```

```
using namespace std;
int main()
{
  int i,j,k;
  cout <<i;
  for (i=0;i<10;)
    cout <<i <<endl;
  system("PAUSE");
  return 0;
}
```

Question: Given that the *break* statement can be used to exit a for loop, write a program that allows the integers 1, 2, 3,… to be added until the sum exceeds 20 million. Your program should also display the last number that was added to the sum.

Program listing:

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
  int i=1,sum=0;
  for (;;)
  {
    sum+=i;
    i++;
    if(sum>20000000) break;
  }
 cout <<"The sum of 1, 2, 3..." <<i
      <<" is " <<sum <<endl;
 system("PAUSE");
 return 0;
}
```

Program run:

```
The sum of 1, 2, 3...6326 is 20005975
Press any key to continue . . .
```

NESTED FOR LOOPS

A nested for loop generally has at least one *for* loop within another *for* loop, in other words there is a loop within another loop. With a nested loop, each loop has its own control variable that determines the number of the times a loop has to be executed. Generally, for a nested loop, each

time the value of the control variable changes, the innermost loop is executed multipl times. Thus, an inner loop is executed more frequent than an outer loop. Consider the following example.

```
for(i=1;i<5;i++)
   for(j=1; j< 4;j++)
```

The outer loop will be executed four times (that is *i*=1, 2, 3 and 4) and for each value of *i* the inner loop will be executed 3 times (*j*=1, 2 and 3). This means that while the outer loop will be executed four times the inner loop will be executed twelve times. Please note that for a nested loop, an inner loop need not follow the outer loop immediately. This will depend on the nature of the problem at hand. Thus, there may or may not be any statements between the for loops. Where there are statements between the loops, a typical nested loop may be as follows:

```
for(i=1;i<10;i++)
{
   outer loop Statements;
    for (j=1;j<n;j++)
    {
          inner loop statements
    }
    Outer loop statement after inner loop
}
```

Please note that the other loops can also be nested.

2-3.4 The CONTINUE and BREAK Statements

THE BREAK STATEMENT

We have so far seen one use of the break statement and that is when we considered the switch statement. The break statement can be used within any of the three looping structures to exit a loop. The break statement is generally used to end an infinite loop or to force program execution to leave a loop. The break statement will transfer program control to the statement immediately following the last loop statement. The following is a typical example that uses the break statesment. Note here that when the break statement is executed, control will be transferred to the statement cout<<"The sum of the numbers from 1 to " <<i <<" is " <<sum <<endl;

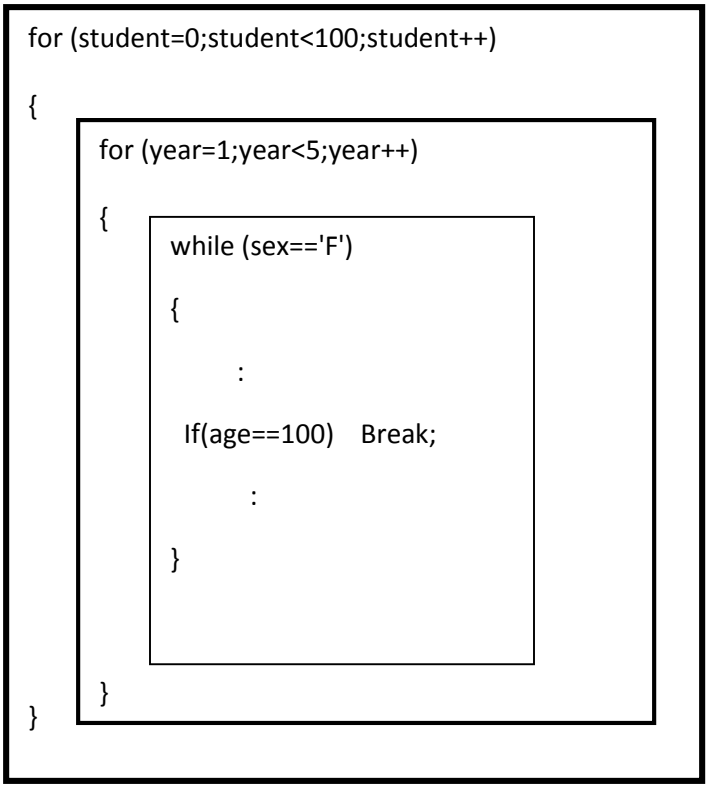
```
#include <iostream>
#include <cstdlib>
using namespace std;

int main()
{
     int i, sum=0;
```

```
    for(i=1;i<1000000;i++)
    {
     sum=sum+i;
     if(sum>12000000)
       break;
    }
    cout<<"The sum of the numbers from 1 to "
       <<i <<" is " <<sum <<endl;
    system("PAUSE");
    return 0;
}
```

The above program attempts to find the sum of all numbers between 1 and 1,000,000 inclusive as indicated by the for statement. However, the if statement forces the loop to be exited when the sum of the numbers being added exceeds 12,000,000. When the above program is run the output will be as follows:

The sum of the numbers from 1 to 4899 is 12002550
Press any key to continue . . .

It must be noted that for a nested loop, if the break statement is used within the innermost loop, then on executing the break statesment, control will be transferred to the statement immediately following the end of the inner loop. The following figure explains this.



The above figure shows a three level nested loop structure. Each box represents a loop statement or structure. The *while* is nested within the *for… year* loop which in turn is nested within the *for… student* loop. When the break statement is executed and a student age is 100, control is transferred outside the while box (loop) to the statement immediately after the closing bracket of the while loop as indicated above.

THE CONTINUE STATEMENT

The continue statement is used to force program execution to skip the remaining loop statements to jump to the end of the loop and to start the next iteration if the loop has not been executed the required number of times. Let us consider a very simple example so that this can be understood. Suppose we want to write a program to list all even numbers between 1 and 50 inclusive using the for loop statement with incremental value of 1 for the control variable each time the loop is repeated. Thus, if I is the control variable then I should take the values 1, 2, 3,…50. We can code this program as follows:

Program listing

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{    int i, sum=0;
     for(i=1;i<51;i++)
     {
      if(i%2==1) continue;
      cout <<i <<"\t";
      sum=sum+i;
     }
     cout <<"\n The sum of the number is " <<sum <<endl;
     system("PAUSE");
     return 0;
}
```

Program run

```
2    4    6    8    10   12   14   16   18   20
22   24   26   28   30   32   34   36   38   40
42   44   46   48   50
 The sum of the number is 650
Press any key to continue . . .
```

It can be seen from the program run that even though the loop's control variable I takes the values 1, 2, 3,…50 inclusive no odd number is displayed as one of the numbers summed together. Apparently, the if statement test whether or not the current value of the control variable is odd. If it is, the output statement (cout) and the summation statement are skipped and the loop is repeated with the next value of the control variable.

THE SYSTEM STATEMENT

This statement may be considered as one of the most useful statements. If you have taken note or run some of the above programs you would have realised that in programs that the system statement was not used, when the program runs, it quickly returns to the operating system level or to the program editor such that you harldy see the results the program displayed to the screen. Also, there are at times when a program is running and you may want the program to execute a Disk Operating System (DOS) command or run another application software. This is usually achieved by the use of the system statement. We will therefore look at the system statement briefly with respect to some of the DOS commands. The general format of the system statement is as follows:

```
system("command");
```

where *command* specifies a valid DOS command to be executed or an executable program command such as notepad.exe to run the window's notepad program. The following are some of the commands that one can specify when using a system statement in a program. To use a cls command, the required statement is specified as **system("cls");**. If there are any command that have not been given and you intend to use, you may have to try it with a small program to see whether or not the system statement will accept such a command. We are only considering commands that anyone is likely to use in their programs. If you intend to use an EXTERNAL DOS command, then the path leading to where the command is located mut be included. We shall see some examples soon. Please note that the commands can be typed in either lower case, upper case or a combination of the lower and upper cases but the system itself must be typed only in lower cases.

The pause command

The pause command is used in a program to instruct the compiler to pause program execution temporally. When the pause command is executed, program execution continues only when a user presses a key. Usually, when a pause command is executed, the computer displays the message

"Press any key to continue…". There can be as many pause commands as may be required in ny program and may appear anywhere within the main function or a user define function except placing the command after the return statement. Placing the pause command after the return statement in the main function or user defined function will not cause an error or warning message to be displayed however the pause command will be ignored by the Computer as the return statement is used to tell the computer there are no more statements to be executed at this point onwards. Thus, a pause command after a return statement is same as not using a pause command in a program.

The cls command

The cls command is used to clear the screen when necessary. There can be as many cls statements in any program. Just like the pause command, the cls command can appear anywhere in a program that one intends to clear whatever information is likely to be on the screen.

The format command

This command is used to format a disk. In using this command, the user must specify the drive holding the disk to be formatted. For example, to format a disk in drive A, the required statement will be as follows:

```
system("format a:");
```

The dir command

This is used to list the files in a specified directory. In specifying files to be displayed, the location of the files must be specified in the command. Note that this command accepts the use of wildcards. To list all files in the default directory in pages, the required statement will be as follows:

```
system("dir  /p");
```

The copy command

This command is used to copy files from one location to another. For example to copy all files that have their name starting with b from the default directory to default directory of drive e, the required statement is as follows:

```
system("copy b*.* e:");
```

Running an application program

As mentioned above the system statement can also be used to run an application such as the notepad, calculator, games, wordprocessors and spreadsheets. For example to run the notepad program which is usally found in windows directory the required command may be as follows:

```
system("c:/windows/notepad.exe");
```

Assume you have the acrobat or pdf reader installed on your computer and that you want to open a pdf file with the name C++HowTo.pdf" in the manuals directory of the root, then the required statement will be as follows:

```
system("c:/manuals/C++HowTo.pdf");
```

Self Assessment 2-3

1. What will be the output when the following codes are executed within a program?

(a)

```
int x=10;
while(x)
 {
  cout<< 2*x-1<<" ";
 x++;
}
```

(b)

```
int x=10;
while(x)
 {
 cout<< 2*x-1<<" ";
x--;
}
```

2. State what the output will be when the following statements are executed in a program

(a)

```
int x=10;
do
 {
  cout<< 2*x-1 <<" ";
 x++;
} while(x<20);
```

(b)

```
int x=10;
do
 {
  cout<< 2*x-1<<" ";
 x--;
} while(x<20);
```

3. State what the output will be when the following statements are executed in a program

```
int x=10;
while (x<20)
  cout<< 2*x-1;
 x++;
```

4. State what the output will be when the following statements are executed in a program

(a)

```
for(int x=0;x=4;x++)
 cout<<x<<" ";
```

(b)

```
for(int x=0;;x++)
 cout<<x<<" ";
```

(c)

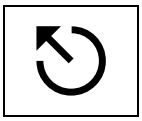
```
for(int x=0;x<4;x++)
 cout<<x<" ";
```

Answers to self assessment 2-3

1. (a) it will display 19 21 23 25 27 etc and run up to infinity. This is because the value of x at each execution of the loop will never be zero (0) making the statement false all the time.
   (b) It will display 19 17 15 13 11 9 7 5 3 1

2. (a) it will display 19 21 23 25 27 29 31 33 35 37
   (b) it will display 19 17 15 13 11 9 7 5 3 1 -1 etc. The loop will be executed infinitely since the value of x will always be less than 20
3. It will display 19 21 23 25 27 29 31 33 35 37
4. (a) It will an infinite number of 4s. This is because when the loop is first entered, the control variable x will be assigned the value zero (0). In an attempt to check the terminating condition the computer will find another assignment statement (x=4) and that will result in x having the value 4. Even if the value of x is modified within the body of the loop and after the output statement it will be reset to 4 whenever the loop is to be executed again. Note that in this case the incremental value will have no effect on the value of x.
   (b) The loop will run infinitely as there is no terminating condition by displaying 0 1 2 3 4 5 etc
   (c) It will display 0 1 2 3

Learning Track Activities



Unit Summary

In this Unit we learnt about the three control structures namely sequence, decision making and looping. The sequence control structures defines the order in which program instructions are executed. Generally, the statements are executed from top to bottom but where there is more than one statement on a line those statements will be executed from left to right. We also learnt that a decision making statement can be used to alter the order of program execution.

The decision control structure is about the different decision making statements that can be used in a program. We have the simple if, the if..else, multiway if and the switch statements. The simple if does not have an else part hence it has actions that need to be taken only when the condition evaluates to true. The if…else and the multiway if are identical in function just that the if…else is used when for any decision only one of the two

possible outcomes's action is to be taken while for a multiway if there are more than two possible outcomes of the decision to be taken and hence the number of actions.

The looping control structures consist of statements that can be used in a program to repeat the execution of a set of instructions a number of times. We have two main types of looping statements and these are pre-test and post-test loops. For the pre-test, the condition controlling the looping is first tested before an attempt is made at executing the loop while for a post test the loop statements are executed before the condition is tested. The while statement is a typical pre-test while the do…while is a post-test statement.

Unit Assignments 3

Making use of a looping and a decision making statement convert the following flowchart into a program segment in C++. Go through your solution and determine the values of x, y and res when the end statement is executed. Hence or otherwise find the function of the program