UNIT 3 DECISION AND LOOP CONTROL STATEMENTS

Structure

3.0 INTRODUCTION

A *program* consists of a number of statements to be executed by the computer. Not many of the programs execute all their statements in sequential order from beginning to end as they appear within the program. A *C program* may require that a logical test be carried out at some particular point within the program. One of the several possible actions will be carried out, depending on the outcome of the *logical test*. This is called ***Branching***. In the ***Selection*** process, a set of statements will be selected for execution, among the several sets available. Suppose, if there is a need of a group of statements to be executed repeatedly until some logical condition is satisfied, then ***looping*** is required in the program. These can be carried out using various control statements.

These ***Control statements*** determine the "*flow of control*" in a program and enable us to specify the order in which the various instructions in a program are to be executed by the computer. Normally, high level procedural programming languages require three basic control statements:

- Sequence instruction
- Selection/decision instruction
- Repetition or Loop instruction

Sequence instruction means executing one instruction after another, in the order in which they occur in the source file. This is usually built into the language as a default action, as it is with C. If an instruction is not a control statement, then the next instruction to be executed will simply be the next one in sequence.

Selection means executing different sections of code depending on a specific condition or the value of a variable. This allows a program to take different courses of action depending on different conditions. C provides three selection structures.

- *if*
- *if…else*
- *switch*

Repetition/Looping means executing the same section of code more than once. A section of code may either be executed a fixed number of times, or while some condition is true. C provides three looping statements:

- *while*
- *do…while*
- *for*

This unit introduces you the decision and loop control statements that are available in C programming language along with some of the example programs.

3.1 OBJECTIVES

After going through this unit you will be able to:

- work with different control statements;
- know the appropriate use of the various control statements in programming;
- transfer the control from within the loops;
- use the ***goto***, ***break*** and ***continue*** statements in the programs; and
- write programs using branching, looping statements.

3.2 DECISION CONTROL STATEMENTS

In a C program, a decision causes a one-time jump to a different part of the program, depending on the value of an expression. Decisions in C can be made in several ways. The most important is with the ***if...else*** statement, which chooses between two alternatives. This statement can be used without the ***else***, as a simple ***if*** statement. Another decision control statement, ***switch***, creates branches for multiple alternative sections of code, depending on the value of a single variable.

3.2.1 The *if* Statement

It is used to execute an *instruction* or sequence/*block of instructions* only if a *condition* is fulfilled. In ***if*** statements, expression is evaluated first and then, depending on whether the value of the expression (relation or condition) is "*true"* or "*false"*, it transfers the control to a particular statement or a group of statements.

Different forms of implementation *if*-statement are:

- Simple *if* statement
- *If-else* statement
- *Nested if-else* statement
- *Else if* statement

Simple *if* statement

It is used to execute an instruction or block of instructions only if a condition is fulfilled.

The syntax is as follows:

if(condition)
statement;

where c*ondition* is the expression that is to be evaluated. If this *condition* is ***true***, *statement* is executed. If it is ***false***, *statement* is ignored (not executed) and the program continues on the next instruction after the conditional statement.

This is shown in the Figure 3.1 given below:

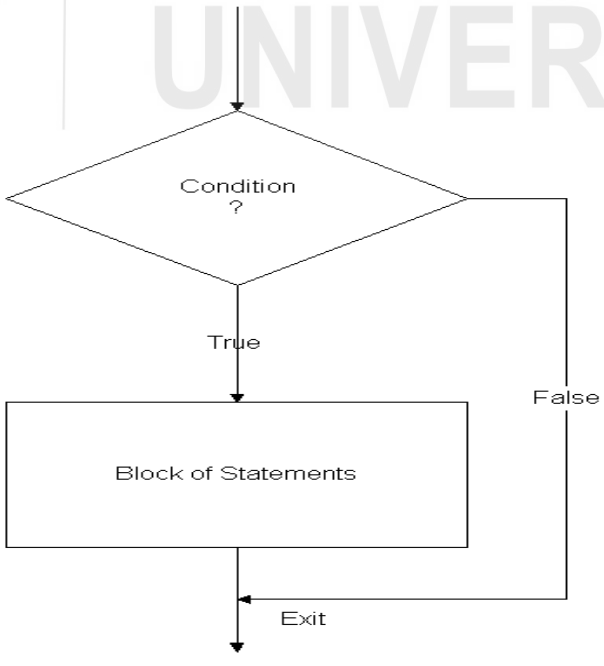

Figure 3.1: Simple *if* statement

If we want more than one statement to be executed, then we can specify a block of statements within the curly bracets { }. The syntax is as follows:

```
if(condition)
  {
    block of statements;
  }
```

Example 3.1

Write a program to calculate the net salary of an employee, if a tax of 15% is levied on his gross-salary if it exceeds Rs. 10,000/- per month.

```
/*Program to calculate the net salary of an employee */

#include<stdio.h>
main()
{
float gross_salary, net_salary;

printf("Enter gross salary of an employee\n");
scanf("%f ",&gross_salary );

if(gross_salary <10000)
      net_salary= gross_salary;
if(gross_salary >= 10000)
      net_salary = gross_salary- 0.15*gross_salary;

printf("\nNet salary is Rs.%.2f\n", net_salary);
}
```

OUTPUT

```
Enter gross salary of an employee
9000
Net salary is Rs.9000.00

Enter gross salary of any employee
10000
Net salary is Rs. 8500.00
```

***If … else* statement**

If…else statement is used when a different sequence of instructions is to be executed depending on the logical value *(True / False)* of the condition evaluated.

Its form used in conjunction with *if* and the syntax is as follows:

```
if(condition)
        Statement _1;
  else
        Statement_ 2;
statement_3;
```

Or

```
if(condition)
     {
     Statements_1_Block;
    }
else
    {
    Statements_2_Block;
    }
Statements _3_Block;
```

If the *condition* is ***true***, then the sequence of statements (S*tatements_1_Block)* executes; otherwise the *Statements_2_Block* following the *else* part of *if-else* statement will get executed. In both the cases, the control is then transferred to *Statements_3* to follow sequential execution of the program.

This is shown in figure 5.2 given below:

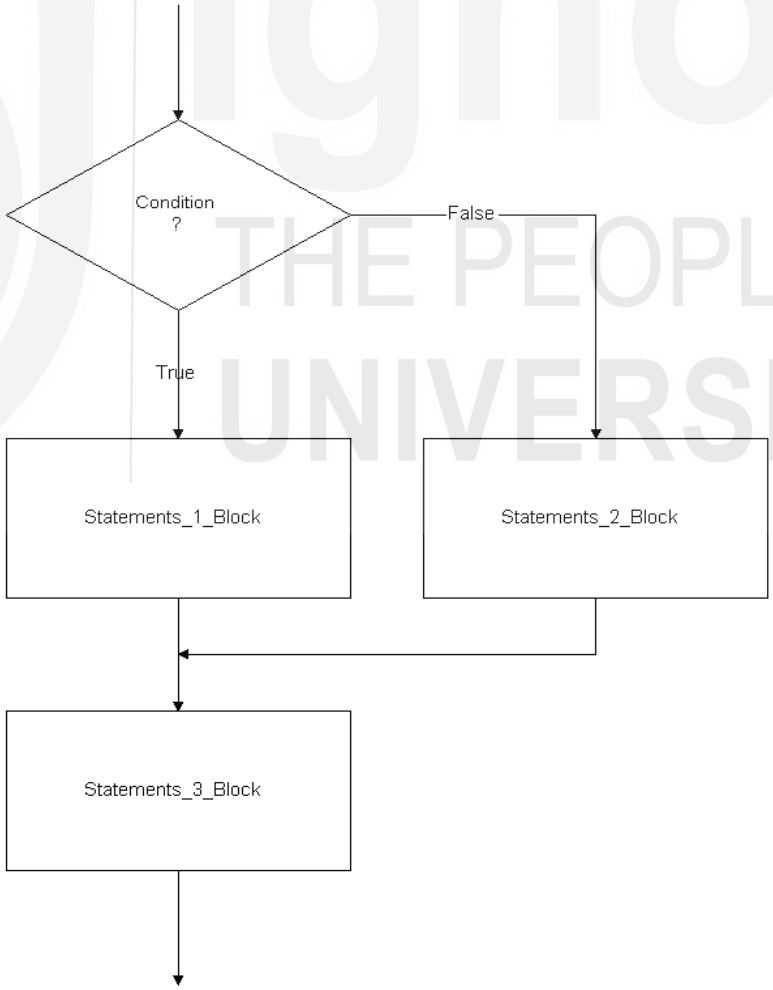

Figure 3.2: *If…else* statement

Let us consider a program to illustrate *if…else* statement,

Example 3.2

Write a program to print whether the given number is even or odd.

```
/* Program to print whether the given number is even or odd*/
#include<stdio.h>
main()
{
int x;
printf("Enter a number:\n");
scanf("%d",&x);
if(x % 2 == 0)
        printf("\nGiven number is even\n");
else
        printf("\nGiven number is odd\n");
}
```

OUTPUT

Enter a number: 6

Given number is even

Enter a number 7

Given number is odd

Conditional expression using Ternary Operator (?:)

There is another way to express an if-else statement is by introducing the **?**: (ternary operator). In a conditional expression the **?:** operator has only one statement associated with the if and the else. The syntax is

variable = expresssion1 ? expression2: expression3;

Example:

```
#include<stdio.h>
main()
{
int x=2;
int y;
y = (x >= 6) ? 6 : x;
printf("y = %d",y);
return 0;
}
```

OUTPUT : y = 2

Nested *if…else* statement

In *nested if... else statement*, an entire *if...else* construct is written within either the body of the ***if*** statement or the body of an *else* statement. The syntax is as follows:

```
if(condition_1)
   {
```

```
    if(condition_2)
      {
        Statements_1_Block;
      }

   else
      {
              Statements_2_Block;
      }
   }

else
     {
       Statements_3_Block;
     }
Statement_4_Block;
```

Here, *condition_1* is evaluated. If it is ***false*** then *Statements_3_Block* is executed and is followed by the execution of *Statements_4_Block*, otherwise if *condition_1* is ***true,*** then *condition_2* is evaluated. *Statements_1_Block* is executed when *condition_2* is ***true*** otherwise *Statements_2_Block* is executed and then the control is transferred to *Statements_4_Block*.

This is shown in the figure 3.3 given in the next page:

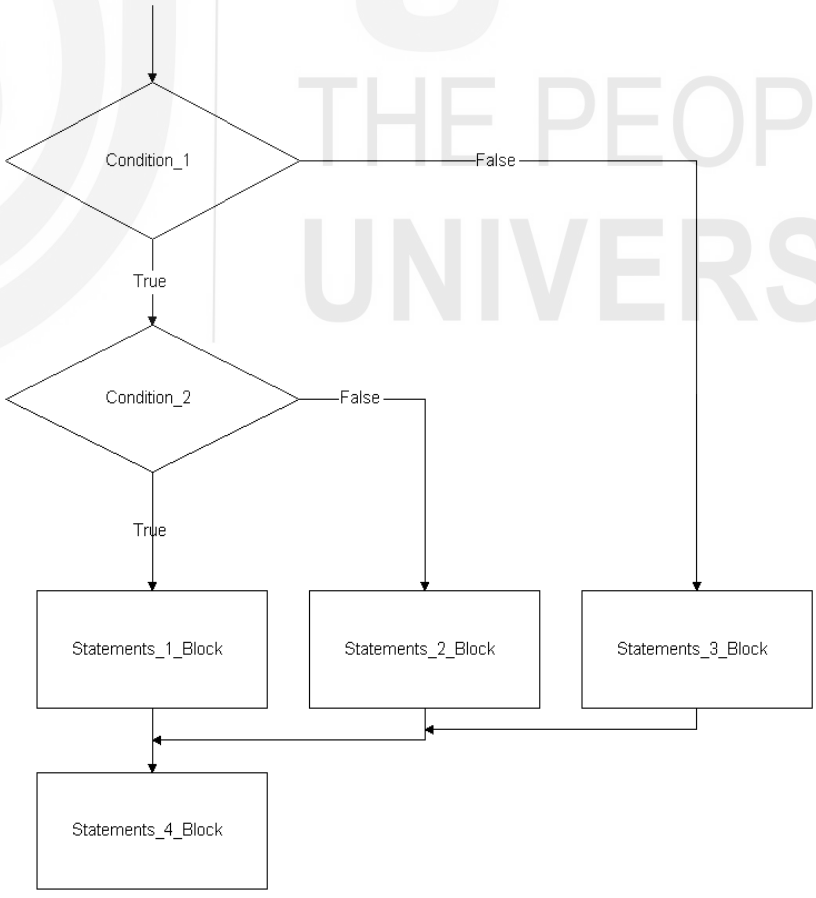

Figure 3.3: Nested *if…else* statement

Let us consider a program to illustrate Nested if…else statement,

Example 3.3

Write a program to calculate an Air ticket fare after discount, given the following conditions:

- If passenger is below 14 years then there is 50% discount on fare
- If passenger is above 50 years then there is 20% discount on fare
- If passenger is above 14 and below 50 then there is 10% discount on fare.

/* Program to calculate an Air ticket fare after discount */

```
#include<stdio.h>
main()
{
int age;
float fare;
printf("\n Enter the age of passenger:\n");
scanf("%d",&age);
printf("\n Enter the Air ticket fare\n");
scanf("%f",&fare);
if(age<14)
fare=fare-0.5*fare;
else
if(age<=50)
  {
     fare=fare-0.1*fare;
  }
     else
  {
     fare=fare-0.2*fare;
  }
printf("\n Air ticket fare to be charged after discount is %.2f",fare);
return 0;
}
```

OUTPUT

Enter the age of passenger 12

Enter the Air ticket fare 2000.00

Air ticket fare to be charged after discount is 1000.00

***Else if* statement**

To show a multi-way decision based on several conditions, we use the ***else if*** statement. This works by cascading of several comparisons. As soon as one of the conditions is true, the statement or block of statements following them is executed and no further comparisons are performed.

The syntax is as follows:

```
if(condition_1)
    {
    Statements_1_Block;
```

```
    }
    else if(condition_2)
        {
            Statements_2_Block;
        }
                ------------
                        else if(condition_n)
                            {
                                Statements_n_Block;
                            }
else
    Statements_x;
```

Here, the *conditions* are evaluated in order from top to bottom. As soon as any condition evaluates to *true*, then the statement associated with the given condition is executed and control is transferred to *Statements_x* skipping the rest of the conditions following it.

But if all conditions evaluate *false*, then the statement following final ***else*** is executed followed by the execution of *Statements_x*.

This is shown in the figure 5.4 given below:

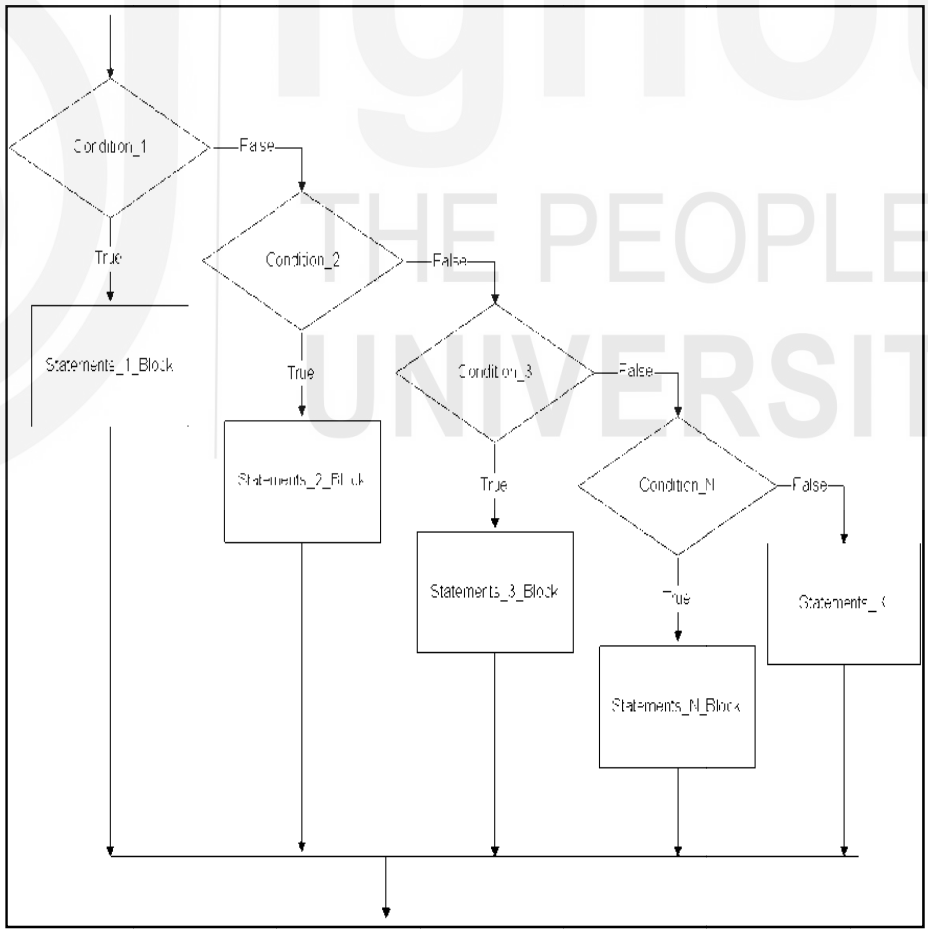

Figure 3.4: *Else if* statement

Let us consider a program to illustrate *Else if* statement,

Example 3.4

Write a program to award grades to students depending upon the criteria mentioned below:

- Marks less than or equal to 50 are given “D” grade
- Marks above 50 but below 60 are given “C” grade
- Marks between 60 to 75 are given “B” grade
- Marks greater than 75 are given “A” grade.

```
/* Program to award grades */
#include<stdio.h>
main()
{
int result;
printf("Enter the total marks of a student:\n");
scanf("%d",&result);
if(result <= 50)
        printf("Grade D\n");
          else if(result <= 60)
              printf("Grade C\n");
            else if(result <= 75)
                printf("Grade B\n");
                  else
                      printf("Grade A\n");
}
```

OUTPUT

```
Enter the total marks of a student:
80
Grade A
```

Check Your Progress 1

1. Find the output for the following program:

```
#include<stdio.h>
main()
{
        int a=1, b=1;
        if(a==1)
            if(b==0)
                printf(“Hi”);
              else
        printf(“Bye”);
}
```

…….

…….

…….

…….

…….

2. Find the output for the following program:

```
#include<stdio.h>
main()
{
                    int a,b=0;
                    if(a=b=1)
                            printf("hello");
                    else
                            printf("world");
                            return 0;
}
```

……

……

……

……

……

3.2.2 The *Switch* Statement

Its objective is to check several possible constant values for an expression, something similar to what we had studied in the earlier sections, with the linking of several ***if*** and ***else if*** statements. When the actions to be taken depending on the value of control variable, are large in number, then the use of control structure ***Nested if…else*** makes the program complex. There ***switch*** statement can be used. Its form is the following:

```
switch(expression){
        case expression 1:
                        block of instructions1
                        break;
        case expression 2:
                        block of instructions2
                        break;
        .
        .
        default:
                default block of instructions
        }
```

It works in the following way: **switch** evaluates expression and checks if it is equivalent to *expression1*. If it is, it executes *block of instructions 1* until it finds the **break** keyword, moment at finds the control will go to the end of the *switch*. If *expression* was not equal to *expression 1* it will check whether *expression* is equivalent to *expression 2*. If it is, it will execute *block of instructions 2* until it finds the **break** keyword.

Finally, if the value of *expression* has not matched any of the previously specified constants (you may specify as many **case** statements as values you want to check), the program will execute the instructions included in the **default:** section, if it exists, as it is an optional statement.

Let us consider a program to illustrate *Switch* statement,

Example 3.5

Write a program that performs the following, depending upon the choice selected by the user.

i). calculate the square of number if choice is 1

ii). calculate the square-root of number if choice is 2 and 4

iii). calculate the cube of the given number if choice is 3

iv). otherwise print the number as it is

```
main()
{
int choice,n;
printf("\n Enter any number:\n ");
scanf("%d",&n);
printf("Choice is as follows:\n\n");
printf("1. To find square of the number\n");
printf("2. To find square-root of the number\n");
printf("3. To find cube of a number\n");
printf("4. To find the square-root of the number\n\n");
printf("Enter your choice:\n");
scanf("%d",&choice);
switch(choice)
{
        case 1: printf("The square of the number is %d\n",n*n);
                break;
        case 2:
        case 4: printf("The square-root of the given number is %f",sqrt(n));
                break;
        case 3:  printf(" The cube of the given number is %d",n*n*n);
        default: printf("The number you had given is %d",n);
                 break;
}
}
```

OUTPUT

Enter any number: 4

Choice is as follows:

1. To find square of the number
2. To find square-root of the number
3. To find cube of a number
4. To find the square-root of the number

Enter your choice: 2
The square-root of the given number is 2

In this section we had discussed and understood various decision control statements. Next section explains you the various loop control statements in C.

3.3 LOOP CONTROL STATEMENTS

Loop control statements are used when a section of code may either be executed a fixed number of times, or while some condition is true. C gives you a choice of three types of loop statements, *while*, *do- while* and *for*.

- The *while* loop keeps repeating an action until an associated *condition* returns ***false***. This is useful where the programmer does not know in advance how many times the loop will be traversed.
- The *do while* loop is similar, but the c*ondition* is checked after the loop body is executed. This ensures that the loop body is run at least once.
- The *for* loop is frequently used, usually where the loop will be traversed a fixed number of times.

3.3.1 The *While* Loop

When in a program a single statement or a certain group of statements are to be executed repeatedly depending upon certain test condition, then *while statement* is used. The syntax is as follows:

```
while(test condition)
{
  body_of_the_loop;
}
```

Here, *test condition* is an expression that controls how long the loop keeps running. Body of the loop is a statement or group of statements enclosed in braces and are repeatedly executed till the value of *test condition* evaluates to *true*. As soon as the *condition* evaluates to ***false***, the control jumps to the first statement following the *while* statement. If condition initially itself is ***false***, the body of the loop will never be executed. *While* loop is sometimes called as *entry-control loop,* as it controls the execution of the body of the loop depending upon the value of the *test condition*. This is shown in the figure 5.5 given below:

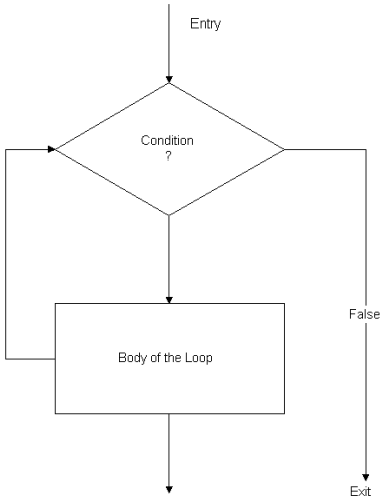

Figure 3.5: The *while* loop statement

Let us consider a program to illustrate *while loop*,

Example 3.6

Write a program to calculate the factorial of a given input natural number.

/* Program to calculate factorial of given number */

```
#include<stdio.h>
#include<math.h>
main()
{
int x;
long int fact = 1;
printf("Enter any number to find factorial:\n");        /*read the number*/
scanf("%d",&x);
while(x > 0)
    {
        fact = fact*x;          /* factorial calculation*/
        x=x-1;
        }
printf("Factorial is %ld",fact);
}
```

OUTPUT

Enter any number to find factorial: 4

Factorial is 24

Here, *condition* in *while* loop is evaluated and body of loop is repeated until *condition* evaluates to ***false*** i.e., when x becomes zero. Then the control is jumped to first statement following *while* loop and print the value of factorial.

3.3.2 The *do...while* Loop

There is another loop control structure which is very similar to the *while* statement – called as the ***do.. while*** statement. The only difference is that the expression which determines whether to carry on looping is evaluated at the end of each loop. The syntax is as follows:

```
do
{
  statement(s);
} while(test condition);
```

In *do-while* loop, the body of loop is executed at least once before the *condition* is evaluated. Then the loop repeats body as long as *condition* is ***true***. However, in *while* loop, the statement doesn't execute the body of the loop even once, if *condition* is ***false***. That is why *do-while* loop is also called *exit-control loop*. This is shown in the figure 3.6 given below.

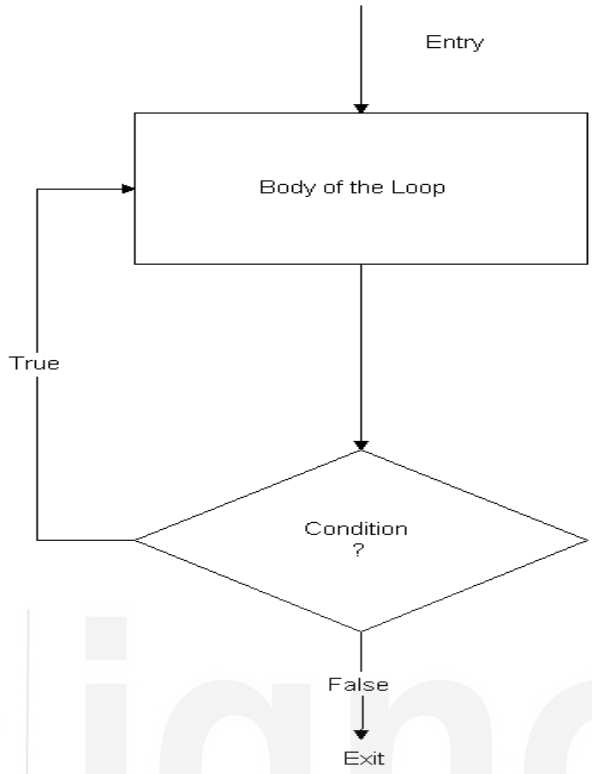

Figure 3.6: The *do…while* statement

Let us consider a program to illustrate do..*while loop*,

Example 3.7

Write a program to print first ten even natural numbers.

```
/* Program to print first ten even natural numbers */
#include<stdio.h>
main()
{
int i=0;
int j=2;
do   {
          printf("%d",j);
           j =j+2;
           i=i+1;   } while(i<10);      }
```

OUTPUT
2 4 6 8 10 12 14 16 18 20

3.3.3 The *for* Loop

for statement makes it more convenient to count iterations of a loop and works well where the number of iterations of the loop is known before the loop is entered. The syntax is as follows:

```
for(initialization; test condition; increment or decrement)
{
      Statement(s);
}
```

The main purpose is to repeat *statement* while *condition* remains true, like the *while* loop. But in addition, ***for*** provides places to specify an *initialization* instruction and an *increment or decrement of the control variable* instruction. So this loop is specially designed to perform a repetitive action with a counter.

The *for* loop as shown in figure 5.7, works in the following manner:

1. *Initialization* is executed. Generally it is an initial value setting for a counter variable. This is executed only once.
2. *Condition* is checked, if it is *true* the loop continues, otherwise the loop finishes and *statement* is skipped.
3. S*tatement(s)* is/are executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }.
4. Finally, whatever is specified in the *increment or decrement of the control variable* field is executed and the loop gets back to step 2.

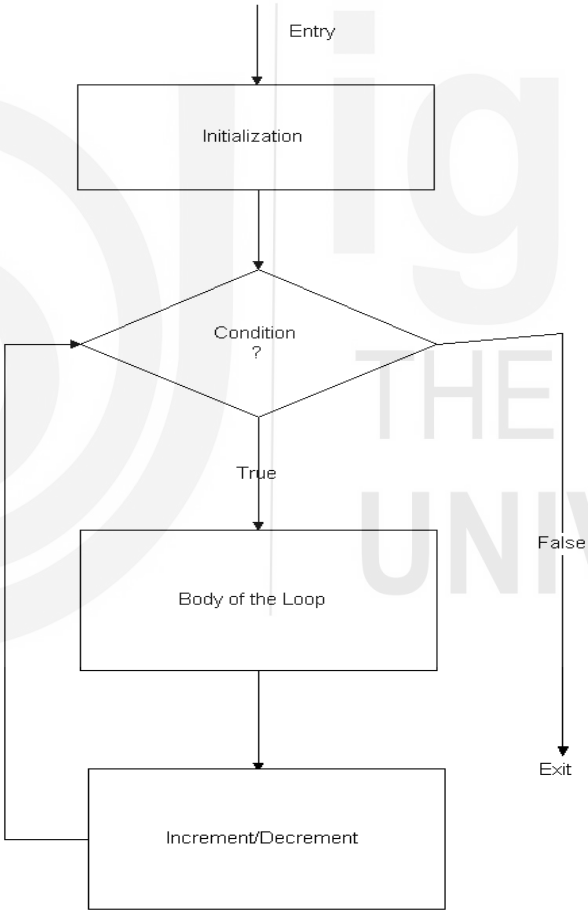

Figure 3.7: The *for* statement

Let us consider a program to illustrate *for loop*,

Example 3.8

Write a program to print first n natural numbers.

```
/* Program to print first n natural numbers */

#include<stdio.h>
main()
{
int i,n;
printf("Enter  value of n \n");
scanf("%d",&n);
printf("\nThe first %d natural numbers are :\n", n);
for(i=1;i<=n;++i)
   {
     printf("%d",i);
    }
}
```

OUTPUT

```
Enter value of n
6
The first 6 natural numbers are:
1 2 3 4 5 6
```

The three statements inside the braces of a *for* loop usually meant for one activity each, however any of them can be left blank also. More than one control variables can be initialized but should be separated by comma.

Various forms of loop statements can be:

a) *for(;condition;increment/decrement)*

body;

A blank first statement will mean no initialization.

b) *for(initialization;condition;)*

body;

A blank last statement will mean no running increment/decrement.

c) *for(initialization;;increment/decrement)*

body;

A blank second conditional statement means no test condition to control the exit from the loop. So, in the absence of second statement, it is required to test the condition inside the loop otherwise it results in an infinite loop where the control never exits from the loop.

d) *for(;;increment/decrement)*

body;

Initialization is required to be done before the loop and test condition is checked inside the loop.

e) *for(initialization;;)*

body;

Test condition and *control variable* increment/decrement is to be done inside the body of the loop.

(f) for(;condition;)

body;

Initialization is required to be done before the loop and control variable increment/decrement is to be done inside the body of the loop.

g) for(;;;)

body;

Initialization is required to be done before the loop, *test condition* and *control variable* increment/decrement is to be done inside the body of the loop.

3.3.4 The Nested Loops

C allows loops to be *nested*, that is, one loop may be inside another. The program given below illustrates the *nesting* of loops.

Let us consider a program to illustrate *nested loops*,

Example 3.9

Write a program to generate the following pattern given below:

```
1
1       2
1       2       3
1       2       3       4
```

```
/* Program to print the pattern */

#include<stdio.h>
main()
{
int i,j;
for(i=1;i<=4;++i)
  {
    printf("%d\n",i);
        for(j=1;j<=i;++j)
                printf("%d\t",j);
        }
}
```

Here, an *inner for loop* is written inside the *outer for loop*. For every value of ***i, j*** takes the value from 1 to ***i*** and then value of ***i*** is incremented and next iteration of outer loop starts ranging ***j*** value from 1 to ***i***.

Check Your Progress 2

1. Predict the output :

```
#include <stdio.h>
main()
{
    int i;
    for(i=0;i<=10;i++)
    printf("%d",i);
   return 0;
}
```

……

……

……

……

……

2. What is the output?

```
#include<stdio.h>
 main()
 {
        int i;
        for(i=0;i<3;i++)
         printf("%d ",i);
 }
```

……

……

……

3. What is the output for the following program?

```
#include<stdio.h>
 main()
 {
  int i=1;
  do
   {
      printf("%d",i);
   }while(i=i-1);
}
```

……

……

……

4. Give the output of the following:

```
#include<stdio.h>
main()
{
   int i=3;
   while(i)
   {
       int x=100;
       printf("\n%d..%d",i,x);
       x=x+1;
       i=i+1;
    }
}
```

……

……

……

3.4 THE *goto* STATEMENT

The ***goto*** statement is used to alter the normal sequence of program instructions by transferring the control to some other portion of the program. The syntax is as follows:

goto label;

Here, ***label*** is an identifier that is used to label the statement to which control will be transferred. The targeted statement must be preceded by the unique label followed by colon.

label: statement;

Although *goto* statement is used to alter the normal sequence of program execution but its usage in the program should be avoided. The most common applications are:

i). To branch around statements under certain conditions in place of use of *if- else* statement,
ii). To jump to the end of the loop under certain conditions bypassing the rest of statements inside the loop in place of *continue* statement,
iii). To jump out of the loop avoiding the use of *break* statement.

goto can never be used to jump into the loop from outside and it should be preferably used for forward jump.

Situations may arise, however, in which the ***goto*** statement can be useful. To the possible extent, the use of the ***goto*** statement should generally be avoided.

Let us consider a program to illustrate *goto* and *label* statements.

Example 3.10

Write a program to print first 10 even numbers

```
/* Program to print 10 even numbers */

#include<stdio.h>
main()
{
 int i=2;
 while(1)
       {
          printf("%d ",i);
          i=i+2;
            if(i>=20)
               goto outside;
       }
  outside : printf("over");
  }
```

OUTPUT

2 4 6 8 10 12 14 16 18 20 over

3.5 THE *break* STATEMENT

Sometimes, it is required to jump out of a loop irrespective of the *conditional test value*. ***Break*** statement is used inside any loop to allow the control jump to the immediate statement following the loop. The syntax is as follows:

break;

When nested loops are used, then ***break*** jumps the control from the loop where it has been used. *Break* statement can be used inside any loop i.e., *while*, *do-while*, *for* and also in *switch* statement.

Let us consider a program to illustrate *break* statement.

Example 3.11

Write a program to calculate the first smallest divisor of a number.

/*Program to calculate smallest divisor of a number */

```
#include<stdio.h>
main()
{
int div,num,i;
printf("Enter any number:\n");
scanf("%d",&num);
for(i=2;i<=num;++i)
   {
  if((num % i) == 0)
         {
         printf("Smallest divisor for number %d is %d",num,i);
         break;
         }
   }
}
```

OUTPUT

```
Enter any number:
9
Smallest divisor for number 9 is 3
```

In the above program, we divide the input number with the integer starting from 2 onwards, and print the smallest divisor as soon as remainder comes out to be zero. Since we are only interested in first smallest divisor and not all divisors of a given number, so jump out of the *for* loop using *break* statement without further going for the next iteration of *for* loop.

Break is different from *exit*. Former jumps the control out of the loop while exit stops the execution of the entire program.

3.6 THE *continue* STATEMENT

Unlike *break* statement, which is used to jump the control out of the loop, it is sometimes required to skip some part of the loop and to continue the execution with next loop iteration. ***Continue*** statement used inside the loop helps to bypass the section of a loop and passes the control to the beginning

of the loop to continue the execution with the next loop iteration. The syntax is as follows:

continue;

Let us see the program given below to know the working of the ***continue*** statement.

Example 3.12

Write a program to print first 20 natural numbers skipping the numbers divisible by 5.

```
/*  Program to print first 20 natural numbers skipping the numbers divisible by 5 */

#include<stdio.h>
main()
{
        int i;
        for(i=1;i<=20;++i)
        {
                if((i % 5) == 0)
                        continue;
                printf("%d ",i);
        }
}
```

OUTPUT

1 2 3 4 6 7 8 9 11 12 13 14 16 17 18 19

Here, the printf statement is bypassed each time when value stored in ***i*** is divisible by 5.

Check Your Progress 3

1. How many times will hello be printed by the following program?

```
#include<stdio.h>
 main()
 {
        int i = 5;
        while(i)
          {
        i=i-1;
        if(i==3)
        continue;
        printf("\nhello");
        }
 }
```

……

……

……

2. Give the output of the following program segment:

```
#include<stdio.h>
main()
```

```
{
int num,sum;
for(num=2,sum=0;;)
  {
  sum = sum + num;
   if(num > 10)
      break;
   num=num+1;
  }
   printf("%d",sum);
}
```

……

……

……

3. What is the output for the following program?

```
#include<stdio.h>
main()
{
    int i, n = 3;
    for(i=3;n<=20;++n)
     {
      if(n%i == 0)
      break;
      if(i == n)
       printf("%d\n",i);
     }
}
```

……

……

……

3.7 SUMMARY

A *program* is usually not limited to a linear sequence of instructions. During its process it may require to repeat execution of a part of code more than once depending upon the requirements or take decisions. For that purpose, C provides *control* and looping statements. In this unit, we had seen the different looping statements provided by C language namely ***while, do…while and for.***

Using ***break*** statement, we can leave a loop even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end. The ***continue*** statement causes the program to skip the rest of the loop in the present iteration as if the end of the *statement* block would have reached, causing it to jump to the following iteration.

Using the ***goto*** statement, we can make an absolute jump to another point in the program. You should use this feature carefully since its execution ignores any type of nesting limitation. The destination point is identified by a label,

which is then used as argument for the *goto* instruction. A *label* is made of a valid identifier followed by a colon (**:**).

3.8 SOLUTIONS / ANSWERS

Check Your Progress 1

1 Bye

2 hello

Check Your Progress 2

1 0 1 2 3 4 5 6 7 8 9 10

2 0 1 2

3 1

4 3..100
2..100
1..100
…..
…..
…...
till infinity

Check Your Progress 3

1 4 times

2 65

3 3

3.9 FURTHER READINGS

1. The C programming language, *Brain W. Kernighan, Dennis M. Ritchie*, PHI.
2. Programming with C, Second Edition, *Byron Gottfried*, Tata McGraw Hill, 2003.
3. C,The Complete Reference, Fourth Edition, *Herbert Schildt*, Tata McGraw Hill,
4. 2002.
5. Computer Science: A Structured Programming Approach Using C, Second Edition, *Behrouz A. Forouzan, Richard F. Gilberg,* Brooks/Cole Thomas Learning, 2001.
6. The C Primer, *Leslie Hancock*, *Morris Krieger*, Mc Graw Hill, 1983.