UNIT-3 DECISION AND CONTROL STRUCTURES

UNIT STRUCTURE

3.1 LEARNING OBJECTIVES

After going through this unit, you will able to :

- learn about Input/Output functions
- describe Conditional Statements
- describe Iterative Statements
- define *break, continue* and *goto* statements

3.2 INTRODUCTION

We have seen that the C language is accompanied by a collection of library functions, which includes a number of input/output functions. Moreover the instructions were executed in the same order in which they appeared within a program. Each instructions was executed once and only once. But the C language provides the facilities to carry out logical test at some particular point within the program and

repeated execution of a group of instructions. In this unit we will discuss about various input/output library functions and various control statements available in C language.

3.3 INPUT/OUTPUT FUNCTIONS

C language simply has no provision for receiving data from any of the input devices (say keyboard, floppy) or for sending data to the output devices (say VDU, floppy etc.). It means that 'C' compiler does not provide any I/O (Input/Output) statements. In programming practice, many programs may require data which needs to be read into the variable names.Moreover, sometimes data stored in variable names need to be shown externally. That is why, in C, there are numerous library functions available for I/O, through which data can be read from or written to files or standard I/O devices. These library functions can be classified into three broad categories:

a. *Console I/O functions* - functions to receive input from keyboard and write output to VDU.
b. *Disk I/O functions* - functions to perform I/O operations on a floppy disk or hard disk.
c. *Port I/O functions* - functions to perform I/O operations on various ports.

We will now deal with the Console I/O functions.Console I/O functions can be further classified as shown in the Table 1.

Console Input/Output functions

Formatted functions			Unformatted functions		
Type	***Input***	***Output***	***Type***	***Input***	***Output***
char	scanf()	printf()	char	getch()	putch()
				getchar()	putchar()
				getche( )	
int	scanf()	printf()	int	-	-
float	scanf()	printf()	float	-	-
string	scanf()	printf()	string	gets()	puts()

The basic difference between formatted and unformatted I/O functions is that the formatted functions allow the input read from the keyboard or the output displayed on the VDU to be formatted as per

our requirements. For example, if values of average marks and percentage marks are to be displayed on the screen, then the details like where this output would appear on the screen, how many spaces would be present between the two values, the number of places after the decimal points etc., can be controlled using formatted functions.

The library implements a simple model of text input and output. A text consists of a sequence of lines, each ending with a newline character. If the system doesn't operate that way, the library does whatever is necessary to make it appear as if it does. For instance, the library might convert carriage return and linefeed to newline on input and back again on output.

Unformatted Console I/O Functions : There are several standard library function available under this category. These functions deals with a single character or with a string of characters. Let us first look at the functions that can handle one character at a time.

a) Single Character Input - the *getchar* Function :

Single character can be entered in to the computer using the C library function getchar. This function reads one character from the keyboard after the new-line character is received (when press Enter key). The function does not require any arguments, though a pair of empty parentheses must follow the word *getchar*.

In general terms, getchar function is written as

character variable = getchar();

Where *character variable* refers to some previously declared character variable.

e.g.

```
char ch;
ch = getchar ( ) ;
```

Program 1 : Demonstration of getchar() function

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

{
 char key;

 printf("\n Type your favourite keyboard character:");
 key=getchar();
 printf("Your favourite character is %c!\n",key);
}
```

RUN:

```
Type your favourite keyboard character: 1
Your favourite character is 1!
```

b) Single Character Output - the *putchar* Function :

Single character can be displayed using the C library function putchar. *putchar()*, the opposite of getchar(), is used to put exactly one character on the screen. Putchar requires as an argument the character to put on the screen.
In general the putchar function is written as

putchar(character variable);

where *character variable* refers to some previously declared character variable

e.g.

```
char ch;
ch = getchar ( ) ;  /* input a character from kbd*/
putchar (ch) ;      /* display it on the screen */
```

Program 2 : Demonstration of putchar() function

```
#include <stdio.h>
void main()
{
    char x = 'A'
    putchar(x);
    putchar('B');
}
```

RUN:

```
AB
```

c) String Input and String Output Function :

gets( ) - The gets( ) function receives a string from the keyboard. The scanf( ) function has some limitations while receiving a string of characters because the moment a blank character is typed, scanf( ) assumes that the end of the data is being entered. So it is possible to enter only one word string using scanf( ). To enter multiple words in to the string, the gets( ) function can be used. Spaces and tabs are perfectly accepted as part of the string. It is terminated when the

enter key is hit.

In general terms, gets function is written as

gets(variable name);

Where *variable name* will be a previously declared variable.

puts( ) - The *puts( )* function works exactly opposite to *gets( )* function. It outputs a string to the screen. Puts( ) can output a single string at a time.

In general terms, gets function is written as

puts(variable name);

Where *variable name* will be a previously declared variable.

Program 3 : Demonstration of gets() and puts() function

```
#include<stdio.h>
void main( )
    {
        char name[40];
        puts("Enter your name");
        gets(name);
        puts("Your name is");
        puts(name);
    }
```

Formatted Console I/O Functions : In order to write a user interactive program in C language, we would need input and output functions that are also called routines. The two functions used for this purpose are : **printf()** and **scanf()**

scanf( ) - scanf( ) allows us to enter data from the keyboard that will be formatted in a certain way. The general form of scanf( ) statement is as follows:

scanf(control string, arg1, arg2.....argn);

Where control string refers to a string containing certain required formatting information, and arg1, arg2,....argn are arguments that represent the individual data items.

Note that we are sending the addresses of variables (addresses are obtained by using & - 'address of' operator) to scanf( ) function. This is necessary because the values received from keyboard must be dropped into variables corresponding to these addresses. The values that are supplied through the keyboard must be separated by either blank(s), tab(s), or newline(s). Do not include these escape sequences in the format string.

scanf("%d %f %c",&c,&a,&ch);

printf() - The output function printf() translates internal values to character.

printf(control string, arg1, arg2, ...argn)

The format string can contain:

- Characters that are simply printed as they are.
- Conversion specification that begins with a % sign.
- Escape sequences that begins with a \ sign.

Printf() converts, formats, and prints its arguments on the standard output under the control of the format. It returns the number of characters printed. The format string contains two types of objects - ordinary characters, which are copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive argument to printf(). Each conversion specification begins with % and ends with a conversion character. Between the % and the conversion character there may be, in order:

- A minus sign, which specifies left adjustment of the converted argument.
- A number that specifies the minimum field width. The converted arguments will be printed in a field at least as wide as the specified minimum. If necessary, it will be padded on the left (or right, if left adjustment is called for) to make up the field width.
- A period (.) separates the field width from the precision.
- A number, i.e., the precision that specifies the maximum number of characters to be printed from a string, or the number of digits after the decimal point of a floating-point value, or the minimum number of digits for an integer.

If the character after the % is not a conversion specification, the behavior is undefined.

Data type		**Conversion character**
Integer	short signed	%d or %I
	short unsigned	%u
	long signed	%ld
	long unsigned	%lu
	unsigned hexadecimal	%x
	unsigned octal	%o
Real	float	%f
	double	%lf
Characters	signed char	%c
	unsigned char	%c
string		%s

Program 4 : Demonstration of scanf() and printf() function

```
#include <stdio.h>
void main()
        {
              int i,j;
              printf("Please type in 2 numbers: ");
              scanf("%d %d",&i,&j);
             printf("you typed %d and %d\n",i,j);
        }
```

CHECK YOUR PROGRESS

1. A C program contains the following statements

```
#include<stdio.h>
char a, b, c;
```

a) Write appropriate getchar statements that will allow values for a, b, c to be entered into the computer.

b) Write appropriate putchar statement that will allow the current value of a, b, c to be written out of the computer.

2. A C program contains the following statements

```
#include<stdio.h>
int i, j, k;
```

Write a printf function for each of the following groups of variable or expressions. Assume all variable represent decimal integers.

a) i, j and k

b) (i+j), (i-k)

c) sqrt (i+j), abs (i-k)

3. Write appropriate scanf function for the above problem to enter automatic values for i, j and k assuming

a) the value for i, j and k will be decimal integers.

b) the value for i will be decimal integer, j an octal integer and k a hexadecimal integer.

3.4 CONDITIONAL STATEMENT

When programming, you will ask the computer to check various kinds of situations and to act accordingly. The computer performs various comparisons of various kinds of statements. These statements come either from you or from the computer itself, while it is processing internal assignments.

In this section we will discuss about some of the conditional statemet used in C language i.e. *if* Statement, *if else* Statement, *Nested if-else* Statement, *switch* Statement, Conditional Operator Statement etc.

3.4.1 *if* Statement

This is the most popular decision making statement. First of all it

checks the *test condition* and then, depending on the result of the test condition it transfers the control to a particular statement or a block of statements.

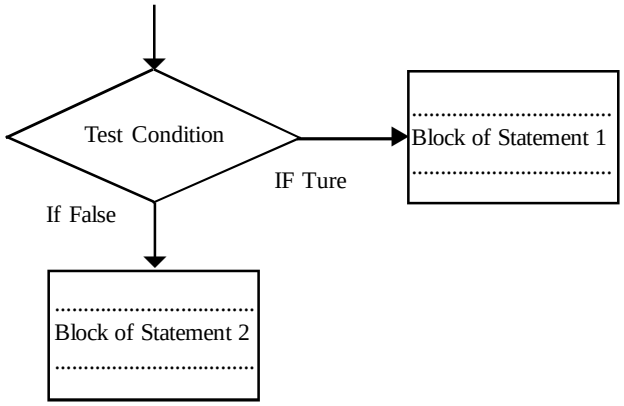

Fig 3.1: Control transfer in if statements

Depending on the complexity of conditions to be tested if statement may be implemented in 4 different ways.The syntax of simple **if** statement is –

```
If(test condition)
{
    - - - - - - - - - - - - - -
    block of statement
    - - - - - - - - - - - - - -
}
```

Example 1: Write a C program to check whether the entered number is positive or negative?

Solution:

```
#include<stdio.h>

void main()
    {
        int a ;
        printf ( " Enter the number " ) ;
        scanf ( " %d " , &a ) ;
        if( a == 0 )
        printf ( "The Number is Zero ") ;
        if ( a > 0 )
        printf (" The Number is Positive " ) ;
        if ( a < 0 )
        printf ( "The number is negative " );
    }
```

Output:

```
Enter the number  3
The Number is Positive
```

3.4.2 *if - else* Statement

This is a bi-directional condition control statement. This type of statement is used to test the condition and take one of two possible actions. If the test condition is evaluated and found to be true then *block of statement 1* will executed, otherwise *block of statement 2* will executed. Syntax is –

```
If (test condition)
   {
     block of statement 1
   }

 else
    {
      block of statement 2
    }
```

Example 3.2 Write a C program to print the largest of two given numbers?

```
#include<stdio.h>

void main()
  {
     int a , b ;
     printf ( " Enter the First number " ) ;
      scanf ( " %d " , &a ) ;
      printf ( " Enter the Second number " ) ;
      scanf ( " %d " , &b ) ;

     if(a>b)
          printf ( " Largest Number is = %d" , a ) ;
                /* This statement will executed if the conditional
                statement is true i.e.  if the value of a is greater than
                the value of b */


     else
          printf ( " Largest Number is=%d" , b ) ;
                /* This statement will executed if the conditional
                statement is false i.e  if the value of a is less than the
                value of b */
  }
```

Output :

```
Enter the First number 12
Enter the Second number 20
Largest  Number is = 20
```

3.4.3 Nested if - else Statement

In certain cases we may use one if else structure within another if else. This is known as nested if else structure. The Syntax is as follows.

```
if(test condition 1)
  {
   if(test condition 2)
    {
        block of statement 1
    }
   else
   {
      block of statement 2
   }
 }
else
 {
    block of statement 3
 }
```

Example 3.3: Example 3.1 can be rewritten as following ways-

```
#include<stdio.h>
void main()
{
        int a ;
        printf ( " Enter the number: " ) ;
        scanf ( " %d " , &a ) ;
          if( a == 0 )
            printf ( "The number is Zero ") ;
          else
            if ( a > 0 )
              printf (" The number is Positive " ) ;
            else
           printf ( "The number is Negative " );
}
```

Output : Enter the number : 5

The number is Positive

3.4.4 *switch* Statement

The main disadvantage of if …else statement is that they are complex to understand,read and debug. If you have a complex set of choices to make, the switch statement is the more powerful alternative. “Pick the matching value and act” is the working strategy of switch statement, rather checking the conditions of nested if…else ladder. The syntax is –

```
switch ( expression )
        {
            case label1 :
                    statement(s) ;
                    break ;
            case label2 :
                    statement(s) ;
                    break ;
            -   - - - - - - - - - - - - -
            -   - - - - - - - - - - - - -
            default:
             statement(s);
                    break ;
        }
```

The switch statement check the value of expression against the list of case labels and when a match is found, the block of statements associated with that case is executed. The break statement at the end of each block signals the end of a particular **case** and causes immediate exit from the switch statement. If the expression does not match any of the case labels then it will execute the default case. Let us consider an example –

Example 3.4: Write a program to convert number to words. For example if you enter 5 the output will be “Five”.

```
#include<stdio.h>
void main()
 {
    int x;
   printf(“Enter a number less than 10 : ” );
    scanf(“%d”,&x) ;
```

```
        switch(x)
        {
        case 1: printf(“ One ”) ;
        break;
        case 2: printf(“ Two ”) ;
        break ;
        case 3: printf(“ Three ”) ;
        break ;
        case 4: printf(“ Four ”) ;
        break ;
        case 5: printf(“ Five ”) ;
        break ;
        case 6: printf(“ Six ”) ;
        break ;
        case 7: printf(“ Seven ”) ;
        break ;
        case 8: printf(“ Eight ”) ;
        break ;
        case 9: printf(“ Nine ”) ;
        break ;
        case 10: printf(“ Ten ”) ;
        break ;
        default : printf( “ Out of range ” ) ;
      }
}
```

Output: Enter a number less than 10 : 5
Five

3.4.5 Conditional Operator Statement

C has a special type of operator which has the two way decision making capability. This operator is known as ternary conditional operator. The syntax is -

Conditional expression ? statement_1 : statement_2

The conditional expression is evaluated first. If the result is true, statement_1 is evaluated and is returned as the value of the conditional expression, otherwise statement_2 is evaluated and its value is returned. Let us again consider example 3.2

```
        void main()
              {
                 int a , b ;
                 printf ( “ Enter the First number ” ) ;
```

```
        scanf ( " %d " , &a ) ;
        printf ( " Enter the Second number " ) ;
        scanf ( " %d " , &b ) ;
        if(a>b)
          printf ( " Largest Number is = %d " , a ) ;
        else
          printf ( " Largest Number is=%d" , b ) ;
    }
```

Can be written as

```
void main()
  {
    int a , b ;
    printf ( " Enter the First number " ) ;
    scanf ( " %d " , &a ) ;
    printf ( " Enter the Second number " ) ;
    scanf ( " %d " , &b ) ;
    printf ( " Largest Number is = %d " , ( a > b ) ? a : b  ) ;
  }
```

CHECK YOUR PROGRESS

4. It is possible to have nested ..... statement in C.
5. A switch statement is used to -
   a) switch between function in a program
   b) switch from one variable to another variable
   c) to choose from multiple possibilities which may arise due to different values of a single variable
   d) to use switching variable
6. Study the following C program

```
#include<stdio.h>
void main()
  {
     int a=7, b=5;
     switch( a/ a%b)
        {
           Case1 : a=a-b;
           Case2 : a=a+b;
           Case3 : a=a*b;
           Case4 : a=a/b;
           default : a=a;
        }
  }
```

On the execution of the above program, what will be the value of the variable a ?

3.5 ITERATIVE STATEMENT

There are three iterative statements available in C, which is also known as Loop control statement. They are-

for statement
while statement
do…while statement

These statements are known as iterative because the block of statement will be executed until the stated condition become false.

3.5.1 *for* Statement

This is the most popular iterative statement. The syntax is –

```
for (
        initialization statement;
        conditional statement ;
        increment/ decrement statement
    )

        {
        .....
        Block of statement
        .....
        }
```

The *for* structure consist of three different statement separated by semicolon, the first statement is the initialization statement, which is executed once before the control entered into the block of statement. The initialization is done using assignment statement. The second statement is the conditional statement. The condition is a relational expression. If the condition is true, the block of statement will executed, otherwise the loop is terminated. The third statement is increment or decrement statement.

The initialization statement initializes the loop variable. Let us take an example -

Example 3.6: Write a C program to print the natural number up to a

given limit?

```
#include<stdio.h>
void main()
  {
      int a, c;
      printf ( " Enter the Limit : " ) ;
      scanf ( "%d", &a ) ;
      for ( c = 0 ; c < a ; c++ )
      printf ( " %d", c ) ;
  }
```

OUTPUT :

```
Enter the Limit : 10
0123456789
```

In the above example the first loop variable c is initialized as zero, then check the condition, is the value of c is less then the value of a, that is 10. For the first execution the condition results true. The control will execute the printf statement which will print the value 0. Next the value of loop variable c is incremented by 1. Thus the current value of loop variable c is now 1. Again check the condition whether 1<10. Since the condition is true therefore the printf statement will executed and print 1. Again increment the loop variable by 1 and repeat the same procedure until the resultant of the condition is false.

3.5.2 *while* Statement

The general form of while statement is-

```
while ( test condition )
   {
        Statement(s);
   }
```

The program will repeatedly execute the *statement(s)* inside the while loop until the condition becomes false. If the condition is initially false, the statement will not be executed. Let us consider an example

Example 3.7: Write a C program to print the natural numbers less

than 10 using while loop.

```
#include<stdio.h>
void main()
  {
    int c ;                    // declaration statement
    c=0 ;                      // initialization statement
    while ( c<10)              // loop statement
       {
            printf ( "%d" , c ) ;      /* block of statement within
            c++;                       while loop */
       }
   }
```

Output :

0 1 2 3 4 5 6 7 8 9

In the above example the first one is a declaration statement, which declares that c is an integer type variable. The second one is an initialization statement which initializes the loop variable c to zero. Next one is the while loop. The loop will be executed until the value of the loop variable c is less than or equal to 9. As soon as the value of c is 10 the outcome of the conditional statement of while loop will false and the loop will be terminated. The execution of while loop for each execution can be tabulated as below –

Execution Number	Value of conditional statement	Result	printf("%d",-c)	value of c (c++)
1	0<10	True	0	
2	1<10	True	1	
3	2<10	True	2	
4	3<10	True	3	
5	4<10	True	4	
6	5<10	True	5	
7	6<10	True	6	
8	7<10	True	7	
9	8<10	True	8	
10	9<10	True	9	
11	10<10	False	will not execute	will not execute

Fig - Execution of while loop for example 3.5

3.5.3 *do…while* STATEMENT

Sometime a *while* loop might not serve your purpose. In such situation you might want to reverse the semantics from “run while this is true” to the subtly different “do this, while this condition remains true”. In other words take the action, and then, after the action is completed, check the condition. Such a loop will always run at least once. To ensure that the action is taken before the condition is tested, use do…while loop. The syntax is-

```
do
   {

        Statement(s);

   } while ( condition ) ;
```

Mind the syntax with the above two. The do…while statement always terminated with a semicolon, whether others two are not. It first executes the statement(s) and then checks whether the condition is true or false. It will repeatedly execute the statement(s) until thecondition become false. Let us take an example –

Example 3.8: Write a C program to print all the even number less than or equal to10 using do…while loop.

```
#include<stdio.h>

void main()
   {
      int c=2;
        do
         {
            printf( “%d  ” , c) ;
            c=c+2;
         } while( c < = 10 );
    }
```

Output:

```
2  4  6  8  10
```

3.6 *break* STATEMENT

The break statement can only appear in a switch body or a loop body. It causes the execution of the current enclosing switch or loop body to terminate.
The general format of the break statement is :

break;

The *break* is a key word in programming language C and a semicolon must be inserted after the word *break*.
break statement with switch - case structure

```
switch ( expression )
        {
            case label1 :
                    statement(s) ;
                    break ;
            case label2 :
                    statement(s) ;
                    break ;
            -   - - - - - - - - - - - - -
            -   - - - - - - - - - - - - -
            default:
             statement(s);
                    break ;
        }
```

The following program shows the use of *break* statement inside *for* loop :

```
#include <stdio.h>

void main()
  {
     int i;

    for (i = 1; i < 10; i++)
      {
        printf ("%d\n", i);

        if (i == 4)
        break;
      }
} // Loop exits after printing 1 through 4
```

Output
1
2
3
4

3.7 *continue* STATEMENT

The continue statement forces the next iteration of the loop to take place, skipping any statement(s) following the continue satement in the body of the loop. The syntax of the continue statement is :

continue;

The use of continue statement in loops is illustrated in the following figure. In *while* and *do - while* loops, continue causes the control to go directly to the test condition and then to continue the iteration process. In the case of *for* loop, the increment section of the loop is executed before the test test-condition is evaluated.

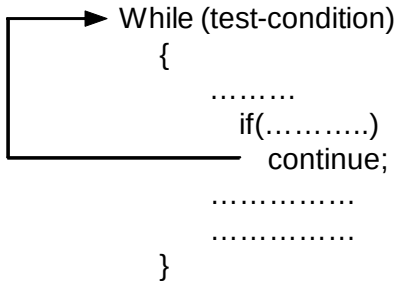

Fig -

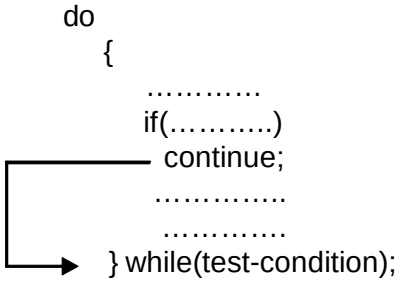

Fig -

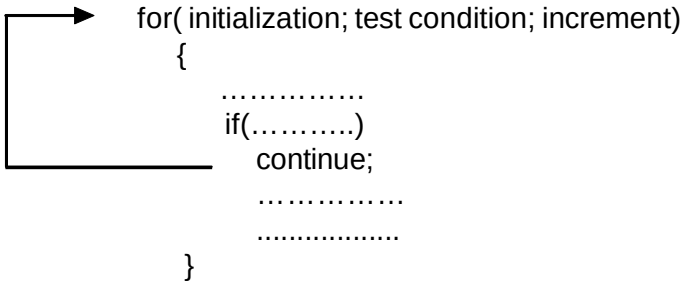

Fig -

The following program shows the use of *continue* statement with the do-while loop :

```
#include<stdio.h>
void main()
  {
        int i, value;
        i = 0;

        do
         {
            printf (" Enter a number :\n");
            scanf ("%d", &value);
            if(value <= 0)
             {
                  printf("Zero or negative value found \n");
                  continue;
              }
               i++;
          }while (i <= 4);
   }
```

Output :

```
Enter a number :
1
Enter a number :
2
Enter a number :
3
Enter a number :
0
Zero or negative value found
Enter a number :
-1
Zero or negative value found
Enter a number :
4
Enter a number :
5
```

3.8 *goto* STATEMENT

The goto statement is used to transfer the control in a program from one point to another point unconditionally. This is alsoo called unconditional branching. The syntax of the goto statement is :

goto label;

where label is a valid identifier used in a programming language C to indicate the destination where a control can be transferred. The syntax of a label is :

label :

The various 'case' statements in a switch construct also serve as labels used to transfer execution control.
The following program shows the use of *goto* statement with the for loop :

```
#include<stdio.h>
void main()
 {
      int i, value;
      for(i=0; i<=10; ++i)
      {
          printf ("Enter a number \n");
          scanf ("%d", &value);
          if (value <= 0)
            {
                  printf ("Error :");
              printf (" Zero or negative value found \n");
              goto error;
             }
       }
      error :
      ;          // Null statement
 }

Output :
Enter a number
1
Enter a number
2
Enter a number
0
Error : Zero or negative value found
```

CHECK YOUR PROGRESS

7. Consider the following code fragment

```
for(digit = 0; digit<9; digit++)
 {
        digit = 2*digit;
        digit --;
 }
```

How many times the loop will be executed -
a) Infinite b) 9 c) 4 d) 0

8. How many times will the following loop be executed-

```
ch = 'b';
while (ch>= 'a' && ch <= 'z')
```

a) 0 b) 25 c) 26 d) 1

9. Write the output of the following program -

```
#include<stdio.h>
void main()
 {
    int i=0, sum =0;
     while(i<20)
       {
          if( i%5 = = 0)
            {
               sum += i;
               printf("%d", sum);
             }
               ++i;
          }
        printf ( "\n Sum = %d", sum);
    }
```

10. Write a C program to check whether a given number is a palindrome or not.

A palindrome is a sentence, phrase, name or number that reads the same forwards as it does backwards. A good exaple would be the word ''racecar." R-a-c-e-c-a-r. r-a-c-e-c-a-r or radar. R-a-d-a-r. r-a-d-a-r.
or the number : 16461, 1234321....etc

3.9 LET US SUM UP

1. C library functions used for I/O operations can be classified into three categories i.e. console I/O function, disk I/O function and port I/O function.
2. Console I/O function is classified into two categories : formatted and unformatted function.
3. *scanf* and *printf* are formatted console I/O function.
4. *getch(), getche(), getchar(), gets(), putch(), putchar()* and *puts()* are unformatted console I/O function.
5. Control statements allows a programmer to change the sequence of instructions for execution.
6. The *if* statement is a conditional control statement that test a particular condition. The *if* statement also allows answer for the kind of either-or condition by using an *else* clause.
7. The *switch* statement is available in programming language C for handling multiple choices.
8. The *loop or iterative statement*, directs a program to perform a set of operations again and again until a specifiedcondition is achieved.
9. The *while* and *do-while* loop constructs are more suitable in situations where prior knowledge of the terminating condition is not known.
10. *While* loop evaluates a test expression before allawing entry into the loop, whereas *do-while* loop is executed atleast once before it evaluates the test expression which is available at the end of the loop.
11. The *for* loop construct is appropriate when in advance it is known as to how many times the loop will be executed.
12. The *break* statement causes an immediate *exit* from the innermost loop structure.
13. The *continue* statement causes the loop to be continued with the next iteration after skipping any statement in between.
14. The *goto* statement is used to transfer the control in a program from one point to another point unconditionally.

3.1O FURTHER READINGS

1. Balagurusamy, E: *Programming in ANSI C*, Tata McGraw-Hill publication.
2. Gottfried Byron S: *Programming with C*, Tata McGraw-Hill publication.

3.11 ANSWERS TO CHECK YOUR PROGRESS

1. a)
```
a = getchar();
b = getchar();
c = getchar();
```

b)
```
putchar(a);
putchar(b);
putchar(c);
```

2.
```
a) printf ("%d %d %d",  i, j,k);
b) printf("%d %d", (i+j), (i-k));
c) printf ("%f %d", sqrt(i+j), abs(i-k));
```

3.
```
a) scanf ("%d %d %d", &i, &j &k);
b) scanf ("%d %o %x", &i, &j, &k);
```

4. if - else 5. c) 6. 35

7. a) Infinite 8. b) 25 9. 0 5 15 30
sum = 30

10.
```
#include<stdio.h>
void main()
    {
        long int n, digit, sum = 0, rev = 0;
        long int num;
        printf ("Input the number \n");
         scanf ("%ld", &num);
         n = num;
         do
          {
            digit = num % 10;
            sum += digit;
            rev = rev * 10 + digit;
            num /= 10;
          }
            while (num != 0);
            Printf ("Sum of the digits of the number = %ld\n", sum);
            printf ("Reverse of the number = %ld \n", rev);
```

```
        if (n == rev)
                printf ("The number is a palindrome \n");
          else
                printf ("The number is not a palindrome \n");
}
```

3.12 MODEL QUESTIONS

1. What is the purpose of the getchar function ? How it is used within a program ?
2. What is the purpose of the putchar function ? How it is used within a program ? Compare with the getchar function ?
3. Write down the main purpose of the scanf and printf function. Summarize the meaning of the more commonly used conversion charactes within the control string of a scanf function.
4. Compare the use of gets and puts functions to transfer strings between the computer and the standard input/output devices.
5. What is the purpose of the while statement ? When is the logical expression evaluated ? What is the minimum number of times that a while loop can be executed ?
6. What is the purpose of the do-while statement ? How does it differ from the while statement ?What is the minimum number of times that a while loop can be executed ?
7. How does a for loop differ from a while and do-while statement ?
8. Why mainfunction is special ? Give two reason ?
9. What is the difference between entryh controlled loop and exit controlled loop ?
10. What is the similarity and difference between break and continue statements ?
11. Write a C program to find the factorial of a number ?
12. Write a C program which accepts a number and prints the sum of digits of this number ?
13. Write a program to find the odd and even numbers upto 100.
14. Write a program to convert a binary number to its equivalent decimal number.
15. Write a program to read a string of characters and print out hte following :
    a) Number of uppercase alphabets (A,B,C,.....)
    b) Number of lower case alphabets (a,b,c,...)
    c)Number of special characters (+,-,/,*,....)
16. Write a program to obtain the following output

a)
```
*
* *
* * *
```

b)
```
  *
 * * *
* * * * *
```

c)
```
    1
  2 2 2
3 3 3 3 3
```

```
*****
```