UNIT-III

Module – 3: (Loops & Functions)

Iteration and loops: use of while, do while and for loops, multiple loop variables, use of break and continue statements.

Functions: Introduction, types of functions, functions with array, passing parameters to functions, call by value, call by reference, recursive functions.

Iteration Statements or Loop: It is a process of repeating the same set of statements again and again until the specified condition holds true.

Computers execute the same set of statements again and again by putting them in a loop.

The C language provides three iteration statements.

1. for loop
2. while loop
3. do-while loop

In general, loops are classified as:

1. Counter-controlled loops
2. Sentinel-controlled loops

1. Counter-controlled loops the number of iterations to be performed is known in advance. The counter-controlled loop starts with the initial value of the loop counter and terminates when the final value of the loop counter is reached. They are also known as definite repetition loops.

Syntax: for Loop

```
for(Initialization section ; Contition Section ; Manipulation Section) // for header
{
  Statement(s); // for body
}
```

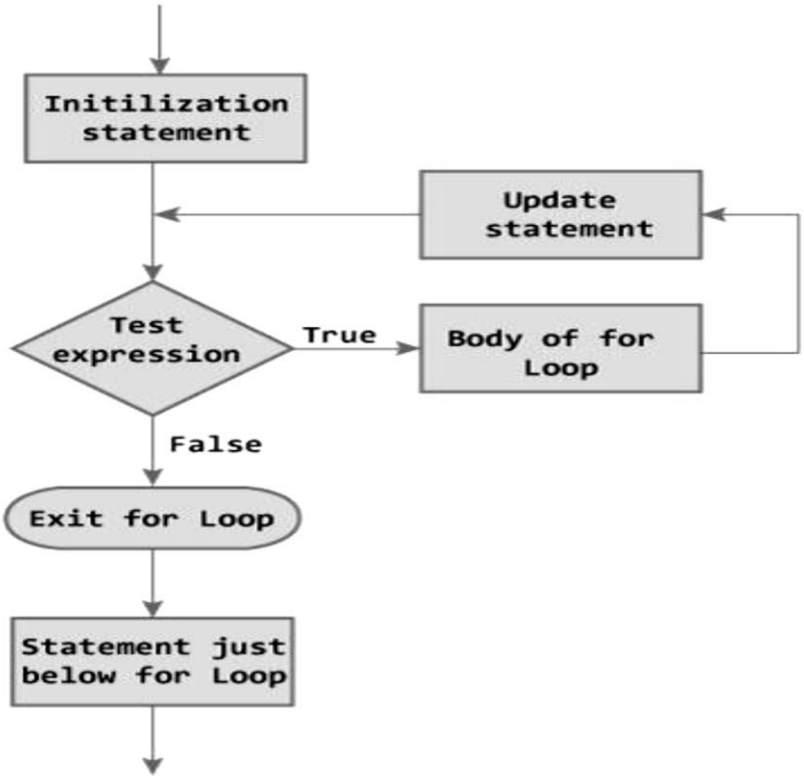


Figure: Flowchart of for Loop

Points about for Loop:

- The for header consists of the keyword for followed by three expressions separated by semicolons and enclosed within parentheses.
- Three sections are named as: initialization section/ condition section and manipulation section.
- Initialization section: It is used to initialize the loop counter.
- Condition section: It tests the value of the loop counter. This section determines whether the body of the loop is to be executed or not.
- Manipulation section: It manipulates the value of the loop
- for header is not terminated with a semicolon.
- The for statement is executed as follows:
  - o a. Initialization section is executed only once at the start of the loop.
  - o b. Condition section is evaluated.
    - If it evaluates *to* true, the body of the loop is executed.
    - If it evaluates to false, the loop terminates and the program control is transferred to the statement present next to the for statement.
- After the execution of the body of the loop, the manipulation expression is evaluated.

Example: Write a program in C to find sum of natural numbers.

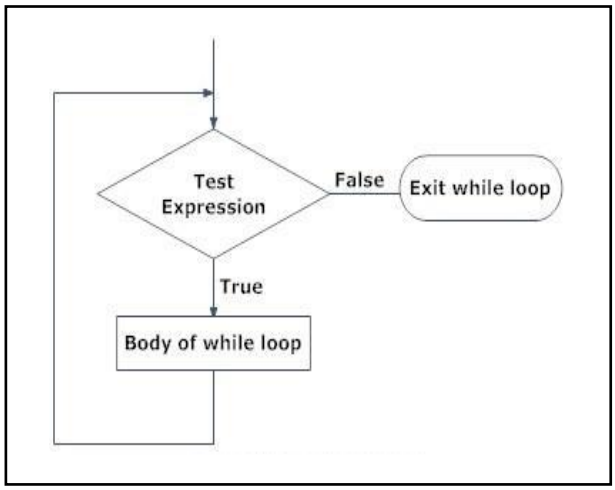
```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,sum=0,n;
clrscr();
printf("Enter the nth term \t ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
sum=sum+i;
}
printf("Sum of natural numbers is %d",sum);
getch();
}
```

Output:
Enter the nth term 10
Sum of natural numbers is 55

The while loop:

Syntax:

```
Initialization Expression;
while (Test Condition)
{
Body of the loop;
Updation Expression;
}
```



- The *while* is an entry – controlled loop statement.
- The test condition is evaluated and if the condition is true, then the body of the loop is executed.
- The execution process is repeated until the test condition becomes false and the control is transferred out of the loop.
- On exit, the program continues with the statement immediately after the body of the loop.
- The body of the loop may have one or more statements.
- The braces are needed only if the body contains two or more statements.
- It is a good practice to use braces even if the body has only one statement.

Example 1: Write a program in C to find sum of natural numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,sum=0,n;
clrscr();
printf("Enter the nth term \t ");
scanf("%d",&n);
i=0;
while(i<=n)
{
sum=sum+i;
i++;
}
printf("Sum of natural numbers is %d",sum);
getch();
}
```

Example 2: Write a program in C to calculate factorial of a given number use while loop.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
long int n, fact =1;
clrscr( ) ;
printf( "\n Enter the Number:");
scanf("%ld", &n);
while(n>=1)
{
fact = fact*n;
n --;
}
printf(" \n Factorial of given number is %d", fact);
getch( );
}
```

Output:
Enter the Number: 5
Factorial of given number is 120

The do-while loop:

- In do-while, the condition is checked at the end of the loop.
- The do-while loop will execute at least one time even if the condition is false initially.
- The do-while loop executes until the condition becomes false.

Syntax:

Initialization Expression;
do
{
Body of the loop
Updation Expression;
} while (Test Condition);

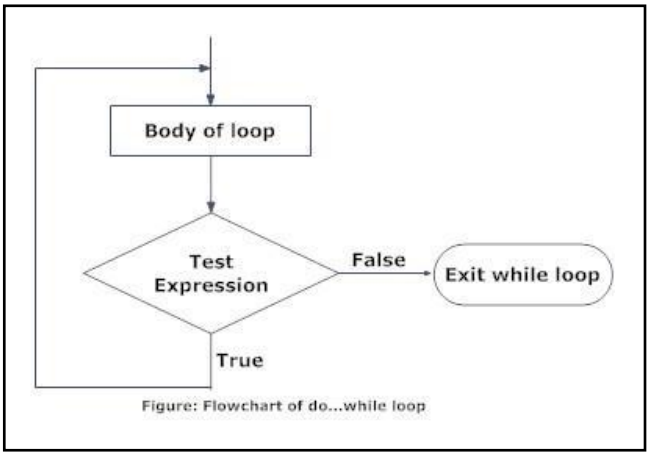


Figure: Flowchart of do...while loop

Example 1: Write a program in C to calculate factorial of a given number use while loop.

```
#include < stdio.h>
#include<conio.h>
void main()
{
int n, fact, i;
clrscr() ;
printf("Input an integer\n");
scanf("%d", &n);
fact=1;
i=1;
{
 fact=fact*i;
 ++i;
} while (i <= n);
printf("Factorial = %d\n", fact);
getch();
}
```

Example 2: Program to check whether the given number is prime or not.

```
# include<stdio.h>
# include<conio.h>
void main( )
 {
     int n, i=2,c=2;
     printf("Enter the number for testing (prime or not: ");
     scanf("%d", &n);
    do
     {
          if(n%i==0)
          {
           c++;
           break;
          }
           i++;
      } while (i<n);
     if(c==2)
     printf("%d is Prime Number",n);
    else
    printf("%d is not Prime number",n);
    getch();
 }
```

Output:
Enter the number for testing (Prime or not): 5
5 is prime.

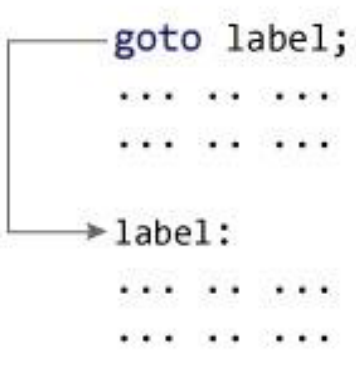
Example 3: Program to add numbers until user enters zero

```
#include <stdio.h>
#include <conio.h>
void main()
  {
    double number, sum = 0;
     do
     {
       printf("Enter a number: ");
       scanf("%lf", &number);
       sum += number;
      } while(number != 0.0);
      printf("Sum = %.2lf",sum);
}
```

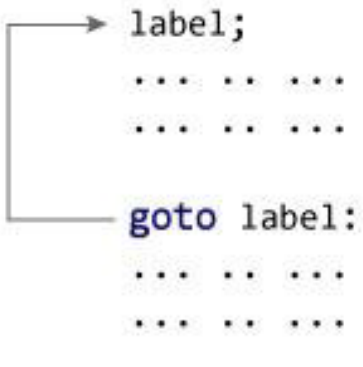
Jumping Statements

goto Statement

- A **goto** statement in C programming language provides an unconditional jump from the goto to a labelled statement in the same function.
- In this syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code below it.



Forward Jump



Backward Jump

This program calculates the average of numbers entered by user

```
# include <stdio.h>
int main(){
float num,average,sum;
int i,n;
printf("Maximum no. of inputs: ");
scanf("%d",&n);
for(i=1;i<=n;++i)
 {
printf("Enter n%d: ",i);
scanf("%f",&num);
if(num<0.0)
goto jump;
sum=sum+num;
}
jump:
average=sum/(i-1);
printf("Average: %.2f",average);
return 0;
}
```

To calculate square root of a given number.

```
#include<stdio.h
#include<conio.h>
#include<math.h>
void main()
{
int x,y;
clrscr();
read:
printf("Enter the number");
scanf("%d",&x);
if(x<0)
   goto read;
y=sqrt(x);
printf("Square root of number is %f",y);
getch();
}
```

break Statement

- The break command allows you to terminate and exit a loop (that is, do, for, and while) or switch command from any point other than the logical end.
- You can place a break command only in the body of a looping command or in the body of a switch command.
- The break keyword must be lowercase and cannot be abbreviated.
- In a looping statement, the break command ends the loop and moves control to the next command outside the loop. Within nested statements, the break command ends only the smallest enclosing do, for, switch, or while commands.
- In a switch body, the break command ends the execution of the switch body and gives control to the next

command outside the switch body.

Example: Write a C program to find prime number.

```
#include <stdio.h>
#include <math.h>
void main()
{
int i, n, prime=1; / / prime is true
printf("Input natural number :");
scanf("%d", &n);
for( i=2; i<= n-1; i++)
{
if( n % i == 0 )
{
prime =0; // prime is now false
break;
}
}
if( prime )
printf("%d is prime number !\n", n);
else
printf("%d isn't prime number!\n", n);
}
```

continue Statement

- The continue statement can be used to skip the rest of the body of an repeated loop.
- It provides a convenient way to force an immediate jump to the loop control statement.

Example

```
#include <stdio.h>
#include <stdlib.h>
void main ()
{
int x;
x = 0;
while (x < 10)
   {
    ++x;
  if (x % 2 == 0)
   {
    continue;
   }
    printf ("%i is an odd number.\n", x);
    }
}
```

Output:

```
1 is an odd number.
3 is an odd number.
5 is an odd number.
7 is an odd number.
9 is an odd number.
```

Nested for loop:

- We can also use loop within loops. i.e. one *for* statement within another *for* statement is allowed in C. (or C allows multiple *for* loops in the nested forms).
- In nested *for* loops one or more *for* statements are included in the body of the loop.
- ANSI C allows up to 15 levels of nesting. Some compilers permit even more.
- Two loops can be nested as follows.

Syntax:

```
for(initialize ; test condition ; updation) /* outer loop */
  {
for(initialize ; test condition ; updation) /* inner loop */
   {
```

```
    Body of loop;
      }
     }
```

- The outer loop controls the rows while the inner loop controls the columns.

Pattern:

Examples: Write a program to display the stars as shown below

```
*
**
***
****
*****

#include<stdio.h>
#include<conio.h>
void main ()
{
int row, column;
for(row=1; row<=5; row++)
{
for(column=1;column<=row;
column++)
{
printf("*");
}
printf( " \n");
}
getch( );
}
```

```
1
12
123
1234
12345

#include<stdio.h>
#include<conio.h>
void main ()
{
int row, column;
for(row=1; row<=5; row++)
{
for(column=1;column<=row;
column++)
{
printf("%d",column);
}
printf( " \n");
}
getch( );
}
```

```
A
AB
ABC
ABCD
ABCDE

#include<stdio.h>
#include<conio.h>
void main ()
{
int row, column;
for(row='A'; row<='E'; row++)
{
for (column ='A'; column<=row;
column++)
{
printf("%c",column);
}
printf( " \n");
}
getch( );
}
```

Example: Write a program to display the pattern as below

```
*
**
***
****
*****
#include<stdio.h>
#include<conio.h>
void main ()
{
int row, column,space;
for(row=1; row<=5; row++)
{
for(space=4;space>row-1;space--)
{
printf(" ");
}
for(column=1;column<=row; column++)
{
printf("*");
}
```

```
*****
****
***
**
*
#include<stdio.h>
#include<conio.h>
void main ()
{
int row, column,space;
for(row=5; row>=1; row--)
{
for(space=5;space>row-1;space--)
{
printf(" ");
}
for(column=1;column<=row; column++)
{
printf("*");
}
printf( "\n");
```

```
printf( "\n");
}
getch( );
}
```

```
}
getch( );
}
```

Write a program to display the pattern as below

```
    *
   ***
  *****
 *******

#include<stdio.h>
#include<conio.h>
void main ()
{
int row, column,space;
for(row=1; row<=5; row++)
{
for(space=5;space>row-1;space--)
{
printf(" ");
}
for(column=1;column<=row*2-1; column++)
{
printf("*");
}
printf( "\n");
}
getch( );
}
```

```
*******
 *****
  ***
   *
#include<stdio.h>
#include<conio.h>
void main ()
{
int row, column,space;
for(row=5; row>=1; row--)
{
for(space=5;space>row-1;space--)
{
printf(" ");
}
for(column=1;column<=row*2-1; column++)
{
printf("*");
}
printf( "\n");
}
getch( );
}
```

Write a program to display the pattern as below

```
A
BA
ABA
BABA
ABABA
#include<stdio.h>
#include<conio.h>
void main ()
{
int row, column,space;
for(row=1; row<=5; row++)
{
for(column=1;column<=row; column++)
{
printf("%c",65+(row+column)%2);
}
printf( "\n");
}
getch( );
}
```

FUNCTIONS

- Functions are subprograms which are used to compute a value or perform a task.
- They cannot be run independently and are always called by the main ( ) function or by some other function.

There are two kinds of functions

1. Library or built-in functions are used to perform standard operations eg: squareroot of a number sqrt(x), absolute value fabs(x), scanf( ), printf( ), and so on. These functions are available along with the compiler and are used along with the required header files such as math.h, stdio. h, string.h and so on at the beginning of the program.

2. User defined functions are self–contained blocks of statements which are written by the user to compute a value or to perform a task. They can be called by the main() function repeatedly as per the requirement.

USES OF FUNCTIONS:

- Functions are very much useful when a block of statements has to be written/executed again and again.
- Functions are useful when the program size is too large or complex. Functions are called to perform each task sequentially from the main program. It is like a top-down modular programming technique to solve a problem
- Functions are also used to reduce the difficulties during debugging a program

USER DEFINED FUNCTIONS:

- In C language, functions are declared to compute and return the value of specific data type to the calling program.
- Functions can also be written to perform a task. It may return many values indirectly to the calling program and these are referred to as void functions.

There are three steps for using a function

(a) Declaration or Prototype-: It is used to tell the compiler that a user defined functions is being used in the program. Generally, it is done before the main function. The basic syntax of declaration or prototype is

Syntax:

return type Func_name (parameter list);

Example: 1

int add(int x, int y);

Example: 2

float square(float x);

(b) Definition-: It is used to code the function or implement the function. The actual code is written in this step.

Syntax

```
return_type function_name( parameter list )
 {
  body of the function
 }
```

Function definition in C programming language consists of a function header and a function body. Here are all the parts of a function:

Return Type: A function may return a value. The return type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

Function Name: This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Function Body: The function body contains a collection of statements that define what the function does.

Example: 1 Addition of two numbers.

```
int add(int x, int y)
 {
    int c;
    c=x + y;
   return c;
  }
```

Example: 2 function returning the max between two numbers

```
int max(int num1, int num2)
  {
      /* local variable declaration */
      int result;
      if (num1 > num2)
      result = num1;
```

```
        else
        result = num2;
        return result;
    }
```

Function main():

- main() is the starting function for any C program. Execution commences from the first statement in the main () function
- It returns nothing when we use void for return.
- It uses no parameter. But it may use two specific parameters
- Recursive call is allowed for main () function also
- The program execution ends when the closing brace of the in main is reached.

Calling-: Whenever we want to use a function, we have to call that function in the program. The syntax for calling of a function are as follows

Function-name(parameter list);

Example

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
a=10; b=20;
add(a,b);
}
```

Complete Example are as follows

```
#include<stdio.h>
#include<conio.h>
int add(int x, int y);
void main()
{
int a,b;
a=10; b=20;
add(a,b);
}
int add(int x, int y)
{
int c;
c=x + y;
printf("answer is %d", c);
return c;
}
```

There are two ways that a C function can be called from a program. They are,

- Call by value
- Call by reference

Call by value:

- In call by value method, the value of the variable is passed to the function as parameter.
- The value of the actual parameter cannot be modified by formal parameter.
- Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.

Note:

- **Actual parameter** – This is the argument which is used in function call.
- **Formal parameter** – This is the argument which is used in function definition

Example 1: Find the larger value from two integer numbers using function.

```
#include< stdio.h>
#include< conio.h>
```

```
int larger(int a,int b); // function declaration
void main()
{
int i,j,k;
clrscr();
i=99;
j=112;
k=larger(i,j); // function call
printf("%d",k);
getch();
}
int larger(int a,int b) // function declaration
{
if(a>b)
return a;
else
return b;
}
```

Example 2: To swap two numbers using function.

```
#include<stdio.h>
#include< conio.h>
void swap(int a, int b); // function prototype, also called function declaration
int main()
{
int m = 22, n = 44;
printf(" values before swap m = %d \nand n = %d", m, n);
swap(m, n); // calling swap function by value
}
void swap(int a, int b)
{
int tmp;
tmp = a;
a = b;
b = tmp;
printf(" \nvalues after swap m = %d\n and n = %d", a, b);
}
```

Call by reference:

- In call by reference method, the address of the variable is passed to the function as parameter.
- The value of the actual parameter can be modified by formal parameter.
- Same memory is used for both actual and formal parameters since only address is used by both parameters.

```
#include<stdio.h>
// function prototype, also called function declaration
void swap(int *a, int *b);
int main()
{
int m = 1, n = 99;
// calling swap function by reference
printf("Values before swap m = %d and n = %d",m,n);
swap(&m, &n);
printf("\n V alues after swap a = %d and b = %d", m, n);
}
void swap(int *a, int *b)
{
int temp;
```

Output:
Values before swap m=1 and n=99
Values after swap m=99 and n=1

```
temp = *a;
*a = *b;
*b = temp;
}
```

Recursion:

A function that calls itself is known as recursive function and this technique is known as recursion in C programming.

Use of recursion function:

- Recursion functions are written less number of statements.
- Recursion is effective where terms are generated successively to compute value.
- Recursion is useful for branching process. Recursion helps to create short code that would otherwise be impossible.

Following are the Example of the recursion where the program is used to compute the factorial of a given number.

Example 1: Write a program in C to find factorial of a number using recursion.

```
#include<stdio.h>
#include<conio.h>
int fact(int x)
{
int y;
if(x==0)
return 1;
y=x*fact(x-1);
return y;
}
void main()
{
int m,n,;
clrscr();
printf("\nEnter the integer");
scanf("%d",&m);
n=fact(m);
printf("\nThe facorial is:-%d",n);
getch();
}
```

Some Important Problem Solved

1. Write a C program to find sum of digits of positive number.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int i,n,digit,sum=0;
printf("\nEnter number");
scanf("%d",&n);
while(n>0)
{
digit=n%10;
sum=sum+digit;
n=n/10;
}
printf("\nSum of digit is %d",sum);
getch( );
}
```

2. Write a C program to find reverse of a given positive number.

```
#include<stdio.h>
#include<conio.h>
```

```
void main ()
{
int i,n,digit,rev=0;
printf("\nEnter number");
scanf("%d",&n);
while(n>0)
{
digit=n%10;
rev=rev*10+digit;
n=n/10;
}
printf("\nReverse of a number is %d",rev);
getch( );
}
```

3. Write a C program to find sum of natural number. (Hint 1,2,3, 4……n)

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int i,n,sum=0;
printf("\nEnter number of terms of series");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
sum=sum+i;
}
printf("\nsum of natural number is %d",sum);
getch( );
}
```

4. Write a C program to find the given number is palindrome or not. (Hint. Reverse of number is same like 121,141,565 etc.)

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int i,n,n1,digit,palin=0;
printf("\nEnter number of terms of series");
scanf("%d",&n);
n1=n;
while(n!=0)
{
digit=n%10;
palin=palin*10+digit;
n=n/10;
}
if(palin==n1)
{
printf("\nNumber is palindrome");
}
else
{
printf("\nNumber is not palindrome");
}
getch( );
```

```
}
```

5. Write a C program to find sum of series

Sum=13+33+53+73+ ……..upto 100 terms

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int i,n,sum=0;
printf("\nEnter number of terms of series");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
sum=sum+pow(i*2-1,3);
}
printf("\nSum is %d",sum);
getch( );
}
```