UNIT-II CONTROL STRUCTURES AND FUNCTIONS

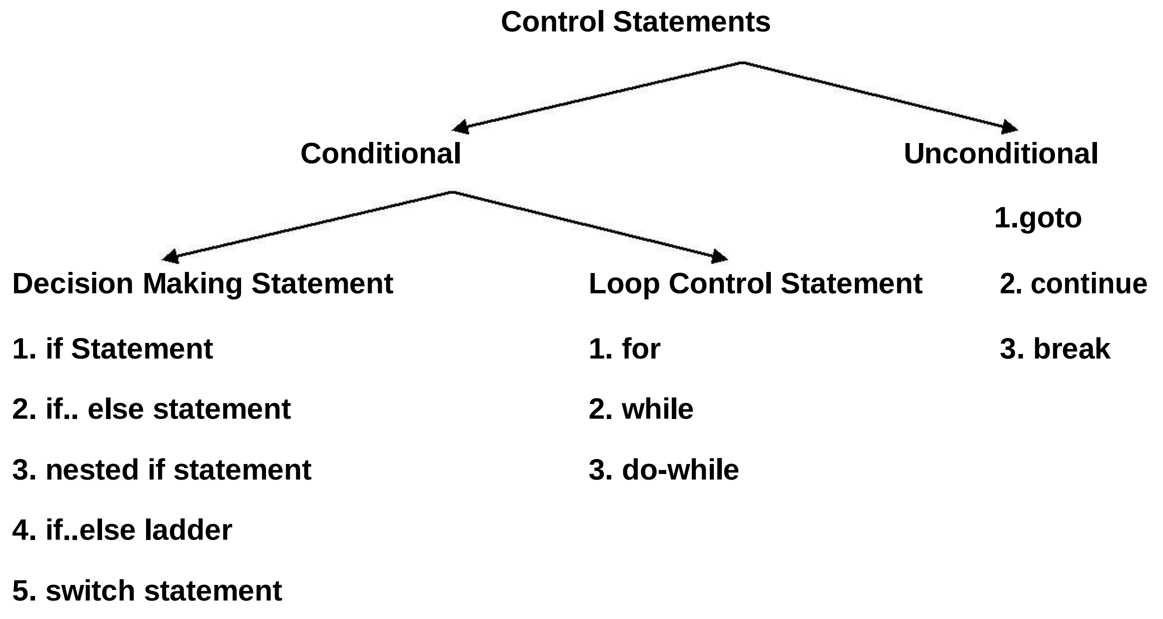
Contents

Control Structures- Conditional Statements, Looping Statements

Functions-Library Functions, User defined Functions, Function Prototype, Function Definitions, Types of Functions, Functions with and without arguments, Functions with no return and with Return Values - Nested Functions - Recursion.

CONTROL STRUCTURES

CONTROL STRUCTURE

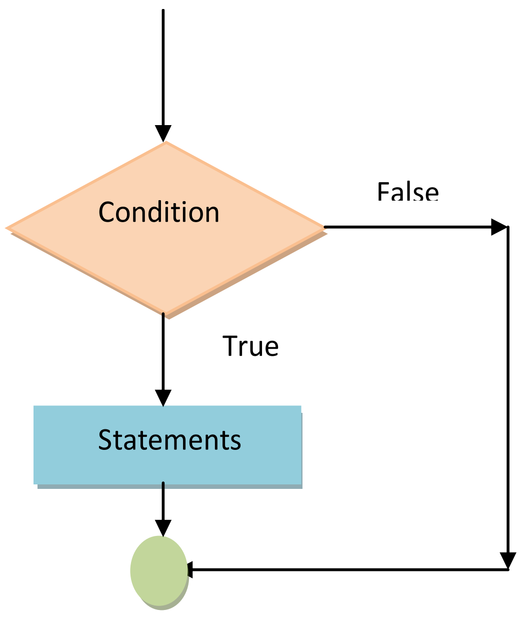


CONDITIONAL STATEMENT

Decision Making Statement

If Statement:

- The if statement is a decision making statement.
- It is used to control the flow of execution of the statement and also used to the logically whether the condition is true or false
- It is always used in conjunction with condition.



Syntax:

```
If(condition)
{

     True statements;
}
```

- If the condition is true, then the true statements are executed.
- If the condition is false then the true statements are not executed, instead the program skips past them.
- The condition is given by relational operators like ==,<=,>=,!=,etc.

Example 1: //program to check whether the entered number is less than 25

```
#include<stdio.h>
#include<conio.h>
void main()
 {
 int i;
 clrscr();
 printf("Enter one value");
 scanf("%d",&i);
 if(i<=25)
     printf("The entered no %d is < 25",i);
 getch();
 }
```

Output:

```
Enter one value 5
The entered no 5 is < 25
```

Example 2: //program to calculate the sum and multiplication using if Statement

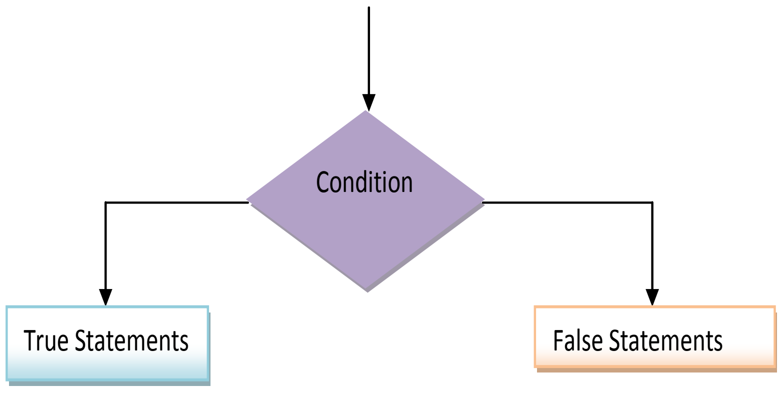
```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a,b,n;
clrscr();
printf("Enter two values");
n=scanf("%d%d",&a,&b);
if(n==2)
{
printf("the sum of two numbers : %d",a+b);
printf("the product of two numbers:%d",a*b);
}
getch();
}
```

Output:

```
Enter two value 5  10
the sum of two numbers : 15
the product of two numbers : 50
```

if.. else statement:

- It is basically two way decision making statement and always used in conjunction with condition.
- It is used to control the flow of expression and also used to carry the logical test and then pickup one of the two possible actions depending on the logical test.
- If the condition is true, then the true statements are executed otherwise false statements are executed.
- The true and false statements may be single or group of statements.



Syntax:

```
If (condition)

        True statements;
else
        False statements;
```

Example 1: //program to find the greatest of two number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b;
        printf("Enter two value");
        scanf("%d%d",&a,&b);
        if(a>b)

                printf("The given no %d is greatest",a);
        else
                printf("The given no %d is greatest",b);
}
```

Output:

```
Enter two value 5 10
The given no 10 is greatest
```

Nested if..else Statement:

When a series of if_else statements are needed in a program, we can write an entire if_else statement inside another if and it can be further nested. This is called nesting if.

Syntax:

```
if(condition 1)
{
        if(condition 2)
        {
                True statement 2;
        else
                False statement 2;
        }
else
False statement 1;
}
```

Example 1: //program to find the greatest of three numbers.

```
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;

   /* check the boolean condition */
   if( a == 100 )
   {
       /* if condition is true then check the following */
       if( b == 200 )
       {
        /* if condition is true then print the following */
          printf("Value of a is 100 and b is 200\n" );
       }
   }
   printf("Exact value of a is : %d\n", a );
   printf("Exact value of b is : %d\n", b );

   return 0;
}
```

Output:

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

If_else Ladder:

- Nested if statements will become complex, if several conditions have to be checked.
- In such situations we can use the else if ladder .

Syntax:

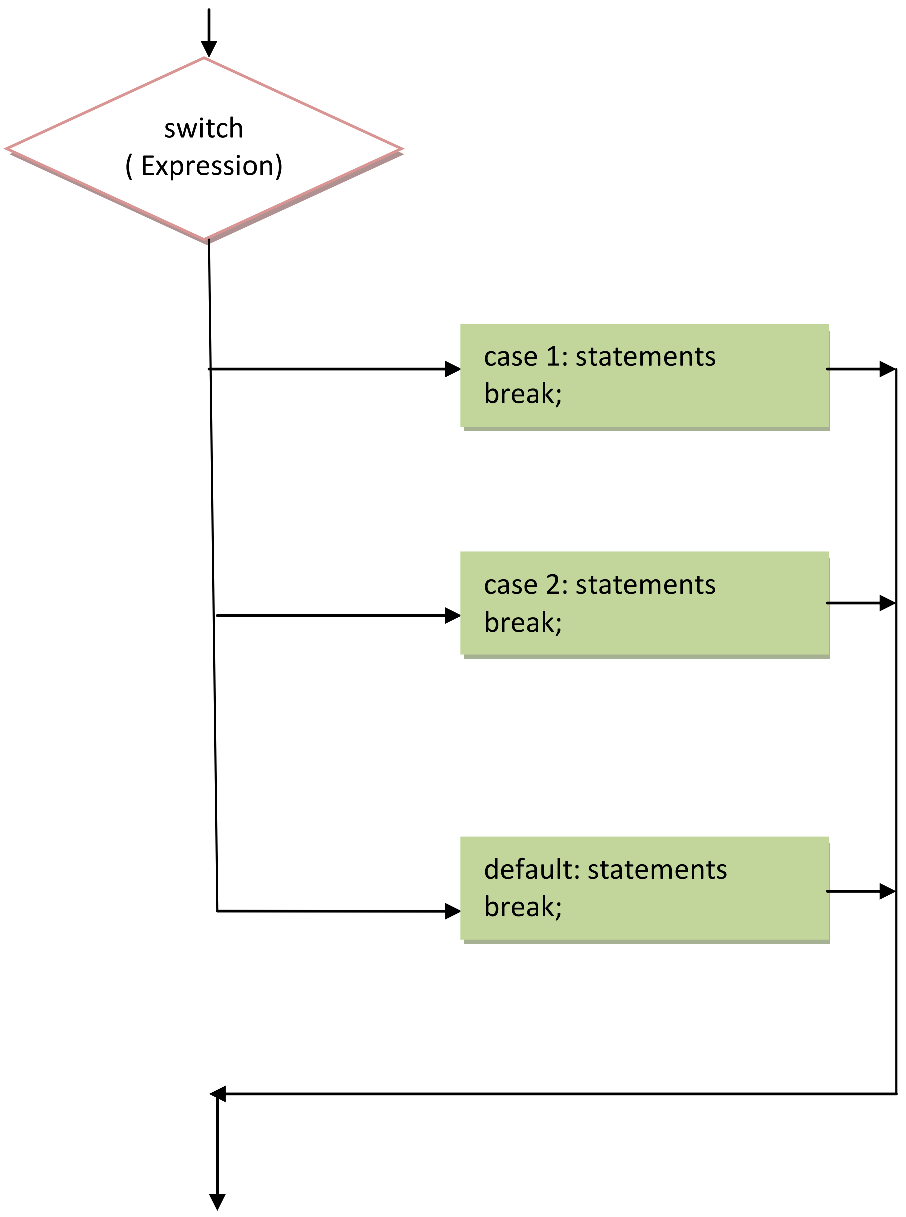
```
if(condition 1)
{
        if(condition 2)
        {
                True statement 2;
        }
        elseif(condition 3)
        {
                True statement 3;
        else
                False statement 3;
        }
else
        False statement 1;
}
```

Switch Statement

- The switch statement is used to execute a particular group of statements from several available groups of statements.
- It allows us to make a decision from the number of choices.
- It is a multi-way decision statement.

Rules for writing switch () statement.

- The expression in switch statement must be an integer value or a character constant.
- No real numbers are used in an expression.
- Each case block and default block must be terminated with break statement.
- The default is optional and can be placed anywhere, but usually placed at end.
- The ‘case’ keyword must terminate with colon(:).
- Cases should not be identical.
- The values of switch expression is compared with the case constant expression in the order specified i.e., from top to bottom.



Syntax:

```
switch(expression)
{
        case 1:
                state
                ment;
                break;
        case 2:
                state
                ment;
                break;
        default: statement;
                break;
}
```

// program to print the give number is odd / even using switch case statement.

```
#include<stdio.h>
#include<conio.h> void main()
        {
        int a,b,c;
        printf("Enter one value"); scanf("%d",&a);
        switch(a%2)
                {
                case 0:
                 printf("The given no %d is even", a);
                 break;
                default :
                 printf("The given no %d is odd", a);
                 break;
                }
        }
```

Output:

```
Enter one value 5
The given no 5 is odd
```

Unconditional statement

Break statement

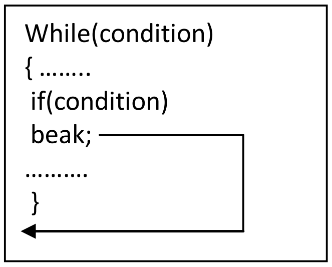
- The break statement is used to terminate the loop.
- When the keyword break is used inside any loop, control automatically transferred to the first statement after the loop.

Syntax:

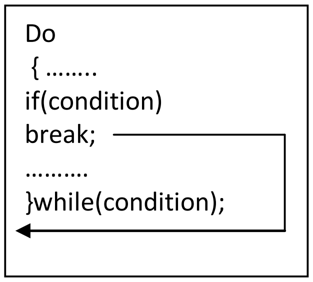
```
break;
```

//program to print the number upto 5 using break statement

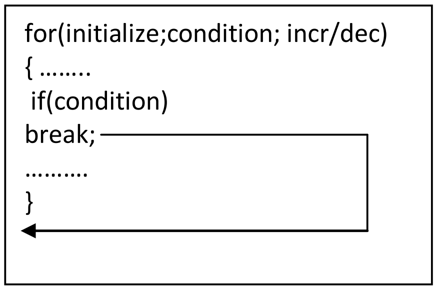
```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
for(i=1;i<=10;i++)
{
if(i==6)
break;
printf("%d",i);
}
}
Output:
1  2    3    4    5
```



```
While(condition)
{ ……..
 if(condition)
 beak;
……….
 }
```



```
Do
 { ……..
if(condition)
break;
……….
}while(condition);
```



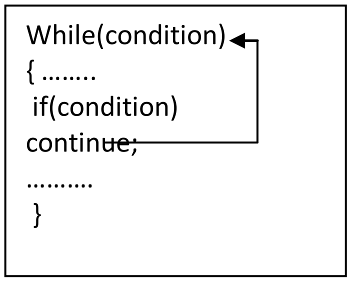
```
for(initialize;condition; incr/dec)
{ ……..
 if(condition)
break;
……….
}
```

Continue Statement

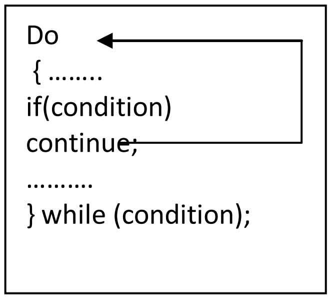
- In some situation, we want to take the control to the beginning of the loop, bypassing the statement inside the loop which have not been executed, for this purpose the continue is used.
- When the statement continue is encountered inside any loop, control automatically passes to the beginning of the loop.

Syntax:

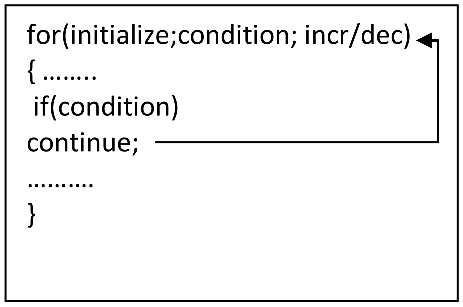
```
        continue;

While(condition)
{
……..
if(condition)
continue;
……….
}
```



```
While(condition)
{ ……..
 if(condition)
continue;
……….
 }
```



```
Do
 { ……..
if(condition)
continue;
……….
} while (condition);
```



```
for(initialize;condition; incr/dec)
{ ……..
 if(condition)
continue;
……….
}
```

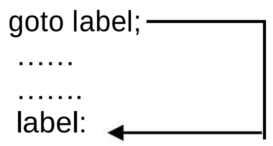
Difference between break and continue

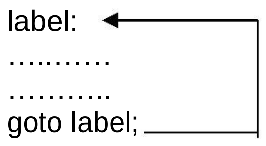
Break	Continue
Break statement takes the control to the outside of the loop	Continue statement takes the control to be beginning of the loop
It is also in switch statement	This can be used only in loop statements
Always associated with if condition in loop	This is also associated with if condition

Goto Statement:

- C provides the goto statement to transfer control unconditionally from one place to another place in the program.
- A goto statement can change the program control to almost anywhere in the program unconditionally.
- The goto statement require a label to identify the place to move the execution.
- The label is a valid variable name and must be ended with colon(:).

Syntax:

1. 

2. 

/* program to print the given both number is equal or not*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
printf("Enter the numbers");
scanf("%d%d",&a,&b);
if(a==b)
    goto equal;
else
 {
 printf("%d and %d are not equal",a,b);
 exit(0);
 }
equal: printf("%d and %d are equal",a,b);
}
```

Output:

```
Enter the numbers   4      5
4 and   5 are not equal

Enter the numbers   5      5
5 and   5 are equal
```

LOOPING STATEMENTS

A loop statement allows us to execute certain block of code repeatedly until test condition is false.

There are 3 types of loops in C programming:

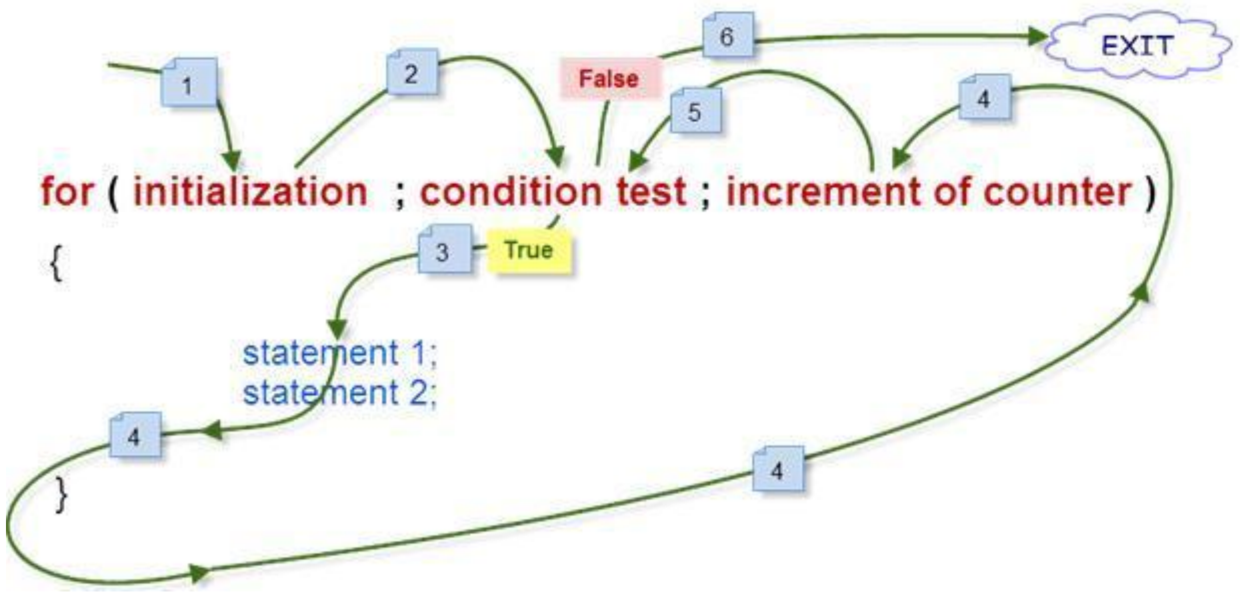
1. for loop
2. while loop
3. do...while loop

for loop:

The syntax for a for loop is

```
for ( variable initialization; condition; variable update )
  {
Code to execute while the condition is true
}
```

The initialization statement is executed only once at the beginning of the for loop. Then the test expression is checked by the program. If the test expression is false, for loop is terminated. But if test expression is true then the code/s inside body of for loop is executed and then update expression is updated. This process repeats until test expression is false.


1
2
6
EXIT
False
5
4
for (initialization ; condition test ; increment of counter)
{
3
True
statement 1;
statement 2;
4
}
4

for loop example

Write a program to find the sum of first *n* natural numbers where n is entered by user. Note: 1,2,3... are called natural numbers.

```
#include <stdio.h>
void main(){
int n, count, sum=0;
printf("Enter the value of n.\n");
scanf("%d",&n);
for(count=1;count<=n;++count)  //for loop terminates if count>n
{
    sum+=count;    /* this statement is equivalent to
    sum=sum+count */
}
printf("Sum=%d",sum);

 }
```

Output

```
Enter the value of
n. 19
Sum=190
```

In this program, the user is asked to enter the value of *n*. Suppose you entered 19 then, count is initialized to 1 at first. Then, the test expression in the for loop,i.e., (count<= n) becomes true. So, the code in the body of for loop is executed which makes *sum* to 1. Then, the expression ++count is executed and again the test expression is checked, which becomes true. Again, the body of for loop is executed which makes *sum* to 3 and this process continues. When count is 20, the test condition becomes false and the for loop is terminated.

/* C program to check whether a number is prime or not. */

```
#include <stdio.h>
int main()
{
int n, i, flag=0;
printf("Enter a positive integer: ");
scanf("%d",&n);
for(i=2;i<=n/2;++i)
 {
 if(n%i==0)
  {
  flag=1;
  break;
  }
 }
if (flag==0)
 printf("%d is a prime number.",n);
else
 printf("%d is not a prime number.",n);
return 0;
}
```

Output

```
Enter a positive integer: 29
29 is a prime number.
```

This program takes a positive integer from user and stores it in variable *n*. Then, for loop is executed which checks whether the number entered by user is perfectly divisible by *i* or not starting with initial value of *i* equals to 2 and increasing the value of *i* in each iteration. If the number entered by user is perfectly divisible by *i* then, *flag* is set to 1 and that number will not be a prime number but, if the number is not perfectly divisible by *i* until test condition i<=n/2 is true means, it is only divisible by 1 and that number itself and that number is a prime number.

Different Types of For Loop in C Programming

For loop can be implemented in different ways

1. Single Statement inside For Loop
2. Multiple Statements inside For Loop
3. No Statement inside For Loop
4. Semicolon at the end of For Loop
5. Multiple Initialization Statement inside For
6. Missing Initialization in For Loop
7. Missing Increment/Decrement Statement
8. Infinite For Loop
9. Condition with no Conditional Operator.

Single Statement inside For Loop:

```
for(i=0;i<5;i++)
  printf("sathyabama");
```

1. Above code will print sathyabama word 5 times.
2. We have single statement inside for loop body.
3. No need to wrap printf inside opening and closing curly block.
4. Curly Block is Optional.

Multiple Statements inside For Loop

```
for(i=0;i<5;i++)
 {
 printf("Statement 1"); printf("Statement 2");
 printf("Statement 3"); if(condition)
 {
  --------
  --------
 }
 }
```

If we have block of code that is to be executed multiple times then we can use curly braces to wrap multiple statement in for loop

No Statement inside For Loop

```
for(i=0;i<5;i++)
   {

   }
```

It is bodyless for loop. It is used to increment value of “i”.This are not used generally. At the end ,for loop value of i will be 5.

Semicolon at the end of For Loop:

```
for(i=0;i<5;i++);
```

- We will not get compile error if semicolon is at the end of for loop.
- This is perfectly legal statement in C Programming.
- This statement is similar to bodyless for loop.

Multiple Initialization Statement inside For:

```
for(i=0,j=0;i<5;i++)
   {
   statement1;
   statement2;
   statement3;
   }
```

Multiple initialization statements must be seperated by Comma .

Missing Increment/Decrement Statement:

```
for(i=0;i<5;)
{
statement1;
statement2;
statement3;
i++;
}
```

we have to explicitly alter the value i in the loop body.

Missing Initialization in For Loop:

```
i = 0;
for(;i<5;i++)
   {
   statement1;
   statement2;
   statement3;
   }
```

we have to set value of ‘i’ before entering in the loop otherwise it will take garbage value of „i".

Infinite For Loop:

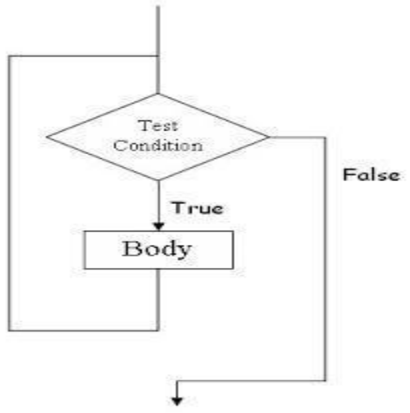
```
 i = 0;
for( ; ; )
   {
   statement1;
   statement2;
   statement3;
   if(breaking condition)
      break;
   i++;
   }
```

Infinite for loop must have breaking condition in order to break for loop. otherwise it will cause overflow of stack.

While Loop

while loop repeatedly executes a target statement as long as a given condition is true.

```
Initialization;
while(condition)
{
----------
---------
----------
------
Increment/decrement;
}
```



- For Single Line of Code – Opening and Closing braces are not needed. while(1) is used for Infinite Loop
- Initialization, Increment/Decrement and Condition steps are on different Line.
- While Loop is also Entry Controlled Loop.[i.e conditions are checked if found true then and then only code is executed]

Examples:

```
#include <stdio.h>

int main()
{
  int y = 0;/* Don't forget to declare variables*/
  while ( y < 10 ) {/* While y is less than 10 */
      printf( "%d\n", y );
      y++; /* Update y so the condition can be met
      eventually */
  }
  getchar();
}
```

C Program to Find Number of Digits in a Number

```
#include <stdio.h>
int main()
{
int n,count=0;
printf("Enter an integer: ");
scanf("%d", &n);
while(n!=0)
{
n/=10;              /* n=n/10 */
++count;
}
printf("Number of digits: %d",count);
}
```

Output:

```
Enter an integer: 34523 Number of digits: 5
```

Types of infinite while loop

Semicolon at the end of while loop

```
#include<stdio.h>
void main()
{
int num=300;
while(num>255); //Note it Carefully
       printf("Hello");
}
```

Output :

Will not print anything

1. In the above program , Condition is specified in the While Loop
2. **Semicolon** at the end of while indicated **while without body**.
3. In the program variable num doesn‟t get incremented , condition remains true forever.
4. As Above program does not have Loop body , It won‟t print anything

Non-Zero Number as a Parameter

```
#include<stdio.h>
void main()
{
while(1)
    printf("Hello");
}
```

Output :

```
Infinite Time "Hello" word
```

1. We can specify any non-zero positive number inside while loop
2. Non zero number is specified in the while loop which means that while loop will remains true forever.

Subscript variable remains the same

```
#include<stdio.h>
void main()
{
int num=20;
  while(num>10) {
      printf("Hello");
  }
}
```

Output :

```
 Infinite Time "Hello C" word
```

Explanation :

1. Condition is specified in while Loop, but terminating condition is not specified and even we haven‟t modified the condition variable.
2. In this case our subscript variable (Variable used to Repeat action) is not either incremented or decremented
3. so while remains true forever.

Character as a Parameter in While Loop

```
#include<stdio.h>
void main()
{
while('A')
    printf("Hello");
}
```

Output :
Infinite Time "Hello" word

Explanation :

1. **Character is Represented** in integer in the form of **ASCII internally**.
2. Any Character is Converted into **Non-zero Integer ASCII value**
3. Any Non-zero ASCII value is **TRUE condition** , that is why Loop executes forever

DO..WHILE

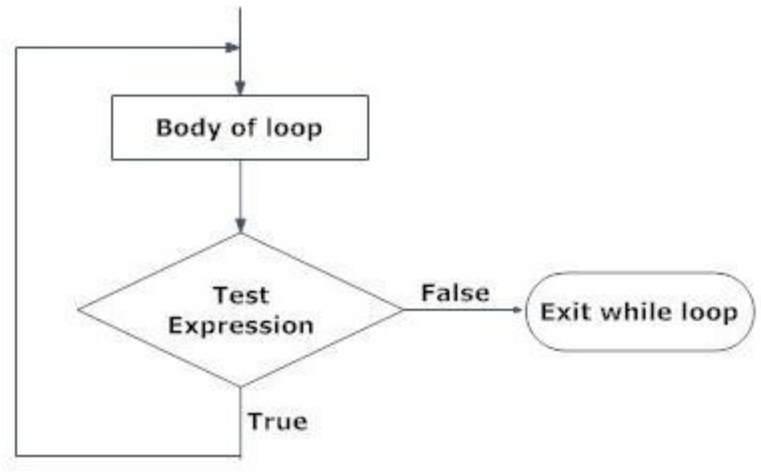
DO..WHILE loops executes the body of the loop atleast once.

The structure is

```
initialization;
do
{
--------------
--------------
incrementation;
}while(condition);
```

The condition is tested at the end of the block instead of the beginning, so the block will be executed at least once. If the condition is true, it go back to the beginning of the block and execute it again. A do..while loop is almost same as a while loop except that the loop body is guaranteed to execute at least once.

- It is **Exit Controlled Loop**.
- Initialization , Incrementation and Condition steps are on **different Line**.
- It is also called **Bottom Tested.**
- Semicolon must be added after the while



Example:

```
#include <stdio.h>

int main()
{
  int z;

  z = 0; do {
       /* " sathyabama is printed at least one time even though the
              condition is false */
      printf( "sathyabama\n" ); } while ( z != 0 );
  getchar();

}
```

C Program to print first 5 Natural Numbers

Using For Loop

```
#include<stdio.h>

void main() { int i = 1;

   for (i = 1; i <= 5; i++) { printf("%d", i);
   }
}
```

```
1
2
3
4
5
```

Using While Loop

```
#include<stdio.h>

void main() { int i = 1;
while (i <= 5) {
printf("%d", i); i++;
   }
}
```

Using Do-While Loop

```
#include<stdio.h>

void main() {
   int i = 1;

   do {
      printf("%d", i);
      i++;
   } while (i <= 5);

}
```

FUNCTIONS

LIBRARY FUNCTIONS

Definition

C Library functions are inbuilt functions in C language which are clustered in a group and stored in a common place called Library. Each and every library functions in C executes explicit functions. In order to get the pre- defined output instead of writing our own code, these library functions will be used. Header file consists of these library functions like Function prototype and data definitions.

- Every input and output operations (e.g., writing to the terminal) and all mathematical operations (e.g., evaluation of sines and cosines) are put into operation by library functions.
- The C library functions are declared in header files (.h) and it is represented as [file_name].h
- The Syntax of using C library functions in the header file is declared as "#include<file_name.h>". Using this syntax we can make use of those library functions.
- #include<filename.h>" command defines that in C program all the codes are included in the header files followed by execution using compiler.
- It is required to call the suitable header file at the beginning of the program in terminal in order to use a library function. A header file is called by means of the pre-processor statement given below,

```
#include<filename.h>
```

Whereas the filename represents the header file name and #include is a pre- processor directive.
To access a library function the function name must be denoted, followed by a list of arguments, which denotes the information being passed to the function.

Example

In case if you want to make use of printf() function, the header file <stdio.h> should be included at the beginning of the C program.

```
#include <stdio.h>
int main()
    {
/* NOTE: Error occurs if printf() statement is written without using
the header file */

        printf(" Hello World");
    }
```

The „main() function" is also a library function which is called at the initial of the program.

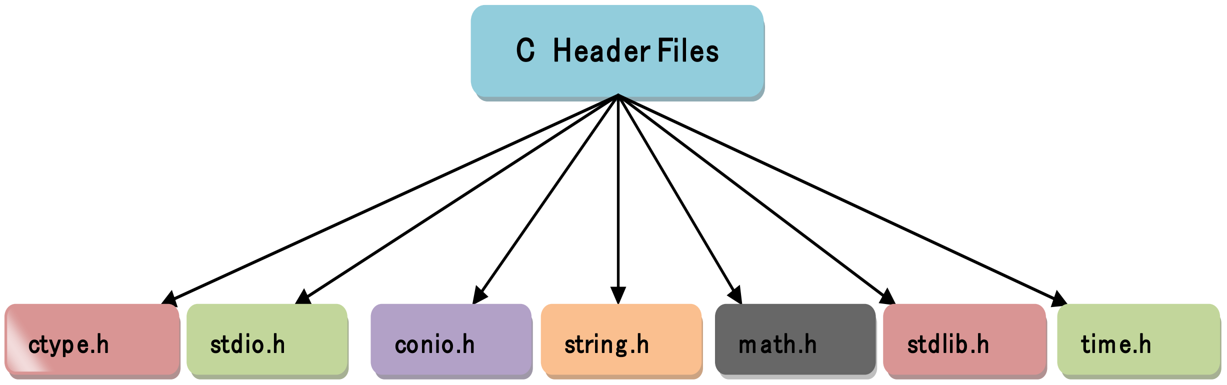
Example

To find the square root of a number we use our own part of code to find them but this may not be most efficient process which is time consuming too. Hence in C programming by declaring the square root function sqrt() under the library function “math.h” will be used to find them rapidly and less time consuming too. Square root program using the library functions is given below:

Finding Square root Using Library Function

```
#include <stdio.h>
#include <math.h>
int main(){
   float num,root;
   printf("Enter a number to find square root.");
   scanf("%f",&num);
   root=sqrt(num); /* Computes the square root of num and stores in
root. */
   printf("Square root of %.2f=%.2f",num,root);
   return 0;
}
```

List of Standard Library Functions in C Programming



Adding User Defined functions in C library:

- In C Programming we can declare our own functions in C library which is called as user-defined functions.
- It is possible to include, remove, change and access our own user defined function to or from C library functions.
- Once the defined function is added to the library it is merely available for all C programs which are more beneficial of including user defined function in C library function
- Once it is declared it can be used anywhere in the C program just like using other C library functions.
- By using these library functions in GCC compilers (latest version), compilation time can be consumed since these functions are accessible in C library in the compiled form.
- Commonly the header files in C program are saved as ”file_name.h” in which all library

functions are obtainable. These header files include source code and this source code is further added in main C program file where we include this header file via “#include <file_name.h>” command.

Steps for adding user defined functions in C library:

Step 1:

For instance, hereby given below is a test function that is going to be included in the C library function. Write and save the below function in a file as “addition.c”

```
addition(int a, int b)
{
int sum;
total =a + b;
return sum;
}
```

Step 2:

Compile “addition.c” file by using Alt + F9 keys (in turbo C).

step 3:

A compiled form of “addition.c” file would be created as “addition.obj”.

Step 4:

To add this function to library, use the command given below (in turbo C).

```
c:\> tlib math.lib + c:\ addition.obj
```

+ represents including c:\addition.obj file in the math library.
We can delete this file using – (minus).

Step 5:

Create a file “addition.h” and declare sample of addition() function like below.

```
int addition (int a, int b);
```

Now “addition.h” file has the prototype of function “addition”.

Note : Since directory name changes for each and every IDE, Kindly create, compile and add files in the particular directory.

Step 6:

Here is an example to see how to use our newly added library function in a C program.

```
# include <stdio.h>
     // User defined function is included here.
# include “c:\\addition.h”
int main ( )
{
        int total;
        // calling function from library
        total = addition (10, 20);
        printf ("Total = %d \n", total);
}

 Output:
Total = 30
```

- Source code checking for all header files can be checked inside “include” directory following C compiler that is installed in system.
- For instance, if you install DevC++ compiler in C directory in our system, “C:\Dev-Cpp\include” is the path where all header files will be readily available.

Mostly used header files in C:

C library functions and header files in which they are declared in conio.h is listed below:

S.No	Header file	Description
1	stdio.h	A standard input/output header file where Input/ Output functions are declared
2	conio.h	Console input/output header file
3	string.h	String functions are defined in this header file
4	stdlib.h	The general functions used in the C program is defined in this header file.
5	math.h	Mathematical related functions are defined in this header file.
6	time.h	Time and clock allied functions are defined in this header file.
7	ctype.h	Every character managing functions are declared in this header file
8	errno.h	This header file contains Error handling functions.
9	assert.h	Diagnostics functions are declared in this header file.

C – conio.h library functions

The entire C programming inbuilt functions that are declared in conio.h header file are given below. The source code for conio.h header file is also given below for your reference.
List of inbuilt conio.h file C functions:

S.no	Function	Description
1	clrscr()	This function is used to clear the output screen.
2	getch()	It reads character from keyboard
3	getche()	It reads character from keyboard and echoes to o/p screen
4	textcolor()	This function is used to change the text colour
5	textbackground()	This function is used to change text background

C – stdio.h library functions

Inbuilt functions of C declared in stdio.h header file are given below.

S.no	Function	Description
1	printf()	This function is used to print the character, string, float, integer, octal and hexadecimal values onto the output screen
2	scanf()	This function is used to read a character, string, numeric data from keyboard.
3	getc()	It reads character from file
4	gets()	It reads line from keyboard
5	getchar()	It reads character from keyboard
6	puts()	It writes line to o/p screen
7	putchar()	It writes a character to screen
8	clearerr( )	Clears the error indicators
9	f open()	All file handling functions are defined in this header file.
10	f close()	closes an opened file
11	getw()	reads an integer from file
12	putw()	writes an integer to file
13	f getc()	reads a character from file
14	putc()	writes a character to file
15	f putc()	writes a character to file
16	f gets()	reads string from a file, per line at a time
17	f puts()	writes string to a file
18	f eof()	finds end of file
19	f getchar	reads a character from keyboard
20	f getc()	reads a character from file
21	f printf()	writes formatted data to a file
22	f scanf()	reads formatted data from a file
23	f getchar	reads a character from keyboard
24	f putchar	writes a character from keyboard
25	f seek()	moves file pointer position to given location
26	SEEK_SET	moves file pointer position to the beginning of the file
27	SEEK_CUR	moves file pointer position to given location
28	SEEK_END	moves file pointer position to the end of file.
29	f tell()	gives current position of file pointer
30	rewind()	moves file pointer position to the beginning of the file
31	putc()	writes a character to file
32	sprint()	writes formatted output to string
33	sscanf()	Reads formatted input from a string
34	remove()	deletes a file
35	fflush()	flushes a file

Functions

- A function is a group of statement that is used to perform a specified task which repeatedly occurs in the main program. By using function, we can divide the complex problem into a manageable problem.
- A function can help to avoid redundancy.
- Function can be of two types, there are
    1. Built-in Function (or) Predefined Function (or) Library Function
    2. User defined Function

$$\begin{tikzcd}
 & \textbf{Functions} \arrow[dl] \arrow[dr] & \\
\textbf{Predefined Function} & & \textbf{User-defined Function}
\end{tikzcd}$$

Difference between Predefined and User-defined Functions

Predefined Function	User-defined function
Predefined function is a function which is already defined in the header file (Example: math.h, string.h, etc)	User- Defined function is a function which is created by the user as per requirement of its owner
Predefined Function is a part of a header file, which are called at runtime	User- Defined function are part of the program which are compiled at runtime
The Predefined function name is given by the developer	User- Defined function name created by the user
Predefined Function name cannot be changed	User defined Function name can be changed

User Defined Functions

- The function defined by the users according to their context (or) requirements is known as a user defined function.
- The User defined function is written by the programmer to perform specific task (or) operation, which is repeatedly used in the main program.
- These functions are helpful to break down the large program into a number of the smaller function.
- The user can modify the function in order to meet their requirements.
- Every user define function has three parts namely
  Function Declaration
  Function Calling
  Function Definition

Need for user-defined function

- While it is possible to write any complex program under the function, and it leads to a number of problems, such as
  - ➢ The problem becomes too large and complex.
  - ➢ The user can‟t go through at a glance
  - ➢ The task of debugging, testing and maintenance become difficult.
- If a problem is divided into a number of parts, then each part may be independently coded and later it combined into a single program. These subprograms are called functions, it is much easier to understand, debug and test the program.

Merits of User-Defined Function

- The length of the source program can be reduced by dividing it into smaller functions
- It provides modularity to the program
- It is easy to identify and debug an error
- Once created a user defined function, can be reused in other programs
- Function facilitates top-down programming approach
- The Function enables a programmer to build a customized library of repeatedly used routines
- Function helps to avoid coding of repeated programming of the similar instruction

Elements of User-Defined Function

1. **Function Declaration**
2. **Function Call**
3. **Function Definition**

Function Declaration

- Like normal variable in a program, the function can also be declared before they defined and invoked
- Function declaration must end with semicolon (;)
- A function declaration must declare after the header file
- The list of parameters must be separated by comma.
- The name of the parameter is optional, but the data type is a must.
- If the function does not return any value, then the return type void is must.
- If there are no parameters, simply place void in braces.

- The data type of actual and formal parameter must match.

Syntax:

Return_type function_name (datatype parameter1, datatype parameter2,…);

Description:

Return type	**:**	type of function
Function_name	:	name of the function
Parameter list or argument list	:	list of parameters that the function can convey.

Example:

int add(int x,int y,int z);

Function Call

The function call be called by simply specifying the name of the function, return value and parameters if presence.

Syntax: function_name();
function_name(parameter);
return_value =function_name (parameter);

Description:

function_name	:	Name of the function
Parameter	:	Actual value passed to the calling function

Example

```
fun();
fun(a,b);
fun(10,20);
c=fun(a,b);
e=fun(2.3,40);
```

Function Definition

- It is the process of specifying and establishing the user defined function by specifying all of its element and characteristics.

Syntax:

Return_type function_name (datatype parameter1, datatype parameter2)

Example 1

```
#include<stdio.h>
#include<conio.h>
void add(); //Function Declaration void sub();//Function Declaration
void main()
{
      clrscr();
      add(); //Function call
      sub(); //Function call
      getch();
}
void add()      //Function Definition
{
      int a,b,c;
      printf("Enter two values");
      scanf("%d%d",&a,&b); c=a+b;
      printf(,add=%d‘,c);
}
void sub()     //Function Definition
{
int a,b,c;
printf("Enter two values");
scanf("%d%d",&a,&b);
c=a-b;
printf("sub=%d",c);
}
```

Example 2 :

```
//Program to check whether the given number is odd or even
#include<stdio.h>
#include<conio.h>
void oddoreven()
{
printf("Enter One value");
scanf("%d",&oe);
if(oe%2==0)
printf("The Given Number%d is even");
else
printf("The Given Number %d is odd");
}
void main()
{
clrscr();
oddoreven();
getch();
```

}

Function Parameter

- The Parameter provides the data communication between the calling function and called function.
- There are two types of parameters.
    - **Actual parameter:** passing the parameters from the calling function to the called function i.e the parameter, return in function is called actual parameter
    - **Formal parameter:** the parameter which is defined in the called function i.e. The parameter, return in the function definition is called formal parameter

Example:

```
main()
{
        ……….
        ……….
        Fun(a,b);
        ……….
        ……….
}
Fun(int x,int y)
{
        ………..
        ………..
}
```

```
Where
     a,b are the actual
parameters

x,y are formal parameter
```

Example Program

```
#include<stdio.h>
#include<conio.h>
void add(int,int); //Function Declaration
void sub(float,int);//Function Declaration
void main()
{
     clrscr();
     add(3,4);    //Function call
     sub(2.5,5);  //Function call
     getch();
}
void add(int a,int b)//Function Definition
{
int c;
c=a+b;
printf(“add=%d”,c);
}
void sub(float a, int b)     //Function Definition
{
float c;
c=a-b;
printf(“sub=%f”,c);
}
```

```
Output:

add=7
sub=-2.500000
```

Example 2:

```
//program for factorial of given
number #include<stdio.h>
#include<conio.h> void main()
{
int fact(int);
int f;
clrscr();
printf("Enter one value");
scanf("%d",&f);
printf("The Factorial of given number %d is %d",f,fact(f));
getch();
}
int fact(int f)
{
if(f==1) return 1;
else
return(f*fact(f-1));
}
```

```
Output:
Enter one value  5
The Factorial of given
number 5 is 120
```

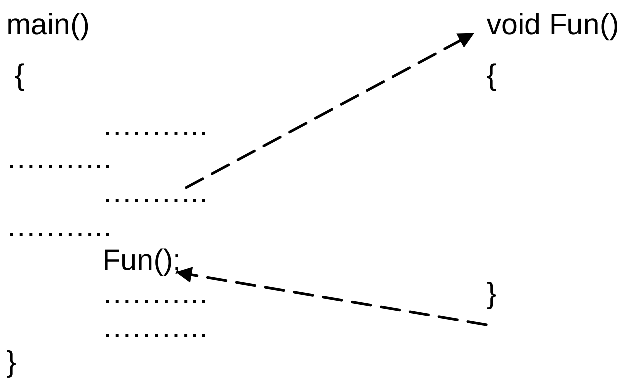
Function Prototype (or) Function Interface

- The functions are classified into four types depends on whether the arguments are present or not, whether a value is returned or not. These are called function prototype.
- In ‘C’ while defining user defined function, it is must to declare its prototype.
- A prototype states the compiler to check the return type and arguments type of the function.
- A function prototype declaration consists of the function’s return type, name and argument. It always ends with semicolon. The following are the function prototypes
  - **Function with no argument and no return value.**
  - **Function with argument and no return value.**
  - **Function with argument and with return value.**
  - **Function with no argument with return value.**

Function with no argument and no return value

- In this prototype, no data transfer takes place between the calling function and the called function. i.e., the called program does not receive any data from the calling program and does not send back any value to the calling program.

Syntax:-



```
main()                                  void Fun()
 {                                      {
          ...........
...........
          ...........
...........
          Fun();
          ...........                   }
          ...........
}
```

The dotted lines indicates that, there is only transfer of control, but no data transfer.

Example program 1

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

Output:

```
Enter two values  6     4
mul=24
```

```
{
      clrscr();
      mul();
      getch();
}
void mul()
{
      int a,b,c;
      printf("Enter two values");
      scanf("%d%d",&a,&b);
      c=a*b;
      printf("mul=%d",c);
}
```

Example program 2

```
//Program for finding the area of a circle using Function with no argument
and no return value
I#include<stdio.h>
#include<conio.h>
void circle();
void main()
{
circle();
}
void circle()
{
int r;
float cir;
printf("Enter radius");
scanf("%d",&r);
cir=3.14*r*r;
printf("The area of circle is %f",cir);
}
```

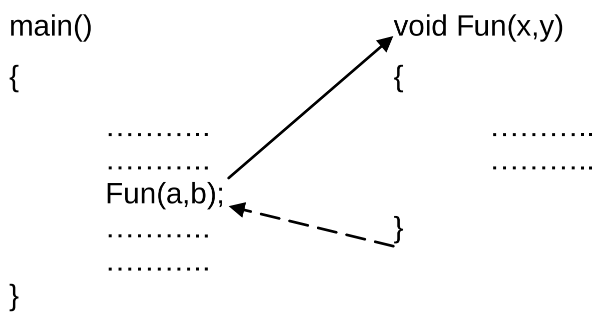
Output:

```
Enter radius 5
The area of circle 78.500000
```

Function with argument and no return value

- In this prototype, data is transferred from the calling function to called function. i.e., the called function receives some data from the calling function and does not send back any values to calling function
- It is one way data communication.

Syntax:-



```
main()                          void Fun(x,y)
{                               {
          ……….                            ……….
          ……….                            ……….
          Fun(a,b);
          ……….                  }
          ……….
}
```

The solid lines indicate data transfer and dotted line indicates a transfer of control.

a and b are the actual parameters

x and y are formal parameters

Example program 1:

```
#include<stdio.h>
#include<conio.h>
void add(int,int);
void main()
{
          clrscr();
          int a,b;
          printf("Enter two values");
          scanf("%d%d",&a,&b);
          add(a,b);
          getch();
}
void add(int x,int y)
{
          int c;
          c=x+y;
          printf("add=%d",c);
}
```

Output:

```
Enter two values  6     4
add=10
```

Example program 2:

//Program to find the area of a circle using Function with argument and no return value

```
#include<stdio.h>
#include<conio.h>
void circle(int);
void main()
{
        int r;
        clrscr();
        printf("Enter radius");
        scanf("%d",&r);
        circle(r);
}
void circle(int r)
{
        float cir;
        cir=3.14*r*r;
        printf("The area of circle is %f",cir);
        getch();
}
```

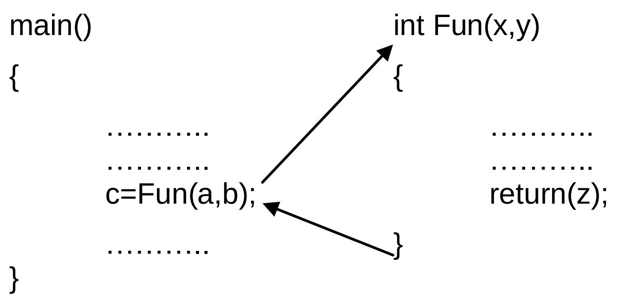
Output:

```
Enter radius 5
The area of circle 78.500000
```

Function with argument and with return value.

- In this prototype, the data is transferred between the calling function and called function. i.e., the called function receives some data from the calling function and sends back returned value to the calling function.
- It is two way data communication

Syntax:-



```
main()                          int Fun(x,y)
{                               {
        ………..                          ………..
        ………..                          ………..
        c=Fun(a,b);                     return(z);
        ………..                  }
}
```

The solid lines indicates data transfer takes place in between thecalling program and called program

a,b are the actual parameter

x,y are formal parameter

Example program 1:

```
#include<stdio.h>
#include<conio.h>
void add(int,int);
```

Output:

```
Enter two values  6     4
Add=10
```

```
void main()
{
      clrscr();
      int a,b,c;
      printf(“Enter two values”);
      scanf(“%d%d”,&a,&b);
      c=add(a,b);
      printf(“Add=%d”,c);
      getch();
}
void add(int x,int y)
{
      int m;
      m=x+y;
      return m;
}
```

Example Program 2

```
// Program to find the area of a circle using Function with argument
and with return value
#include<stdio.h>
#include<conio.h>
float circle(int);
void main()
{
      int r;
      clrscr();
      printf("Enter radius");
      scanf("%d",&r);
      printf("the area of circle is %f",circ
      getch();
}
```

Output:

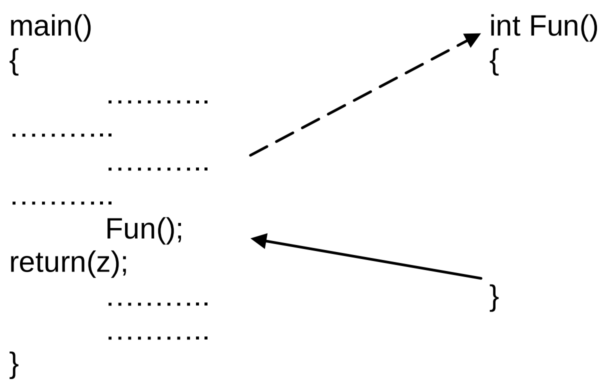
```
Enter radius 5
the area of circle 78.500000
```

```
float circle(int r)
{
      float cir;
      cir=3.14*r*r;
      return cir;
 }
```

Function with no argument with return value

- In this prototype, the calling function cannot pass any arguments to the called function, but the called program may send some return value to the calling function.
- It is one way data communication

Syntax:-



```
main()                                int Fun()
{                                     {
          ……….
……….
          ……….
……….
          Fun();
return(z);
          ……….                        }
          ……….
}
```

The dotted line indicates a control transfer to the called program and the solid line indicates data return to the calling program

Example program 1

```
#include<stdio.h>
#include<conio.h>
int add();
void main()
{
      clrscr();
      int z;
      z=add();
      printf("Add=%d",z);
      getch();
}
int add()
{
```

Output:

```
Enter two values  6     4
Add=10
```

```
     int a,b,c;
     printf(“Enter two values”);
     scanf(“%d%d”,&a,&b);
     c=a+b;
     return c;
}
```

Example Program 2

```
// Program to the area of a circle using no argument with a return
value
#include<stdio.h>
#include<conio.h>
float circle();
void main()
{
     clrscr();
     printf("the area of circle is %f",circle());
     getch();
}
float circle()
{
     float cir;
     int r;
     printf("Enter radious");
     scanf("%d",&r);
     cir=3.14*r*r;
     return cir;
}
```

Output:

```
Enter radius 5
the area of circle 78.500000
```

Parameter Passing Methods (or) Passing Arguments to Function

- Function is a good programming style in which we can write reusable code that can be called whenever required.
- Whenever we call a function, the sequence of executable statements gets executed. We can pass some of the information (or) data to the function for processing is called a parameter.
- In 'C' Language there are two ways a parameter can be passed to a function. They are
    - **Call by value**
    - **Call by reference**

Call by Value:

- This method copies the value of the actual parameter to the formal parameter of the function.
- Here, the changes of the formal parameters cannot affect the actual parameters, because formal parameter are photocopies of the actual parameter.
- The changes made in formal arguments are local to the block of the called function. Once control returns back to the calling function the changes made disappears.

Example Program

```
#include<stdio.h>
#include<conio.h>
void cube(int);
int cube1(int);
void main()
{
     int a;
     clrscr();
     printf("Enter one values");
     scanf("%d",&a);
```

Output:

```
Enter one values  3
Value of cube function is  3
Value of cube1 function is 27
```

```
        printf(“Value of cube function is=%d”, cube(a));
        printf(“Value of cube1 function is =%d”, cube1(a ));
        getch();
}
void cube(int x)
{
        x=x*x*x;
        return x;
}
int cube1(int x)
{
        x=x*x*x;
        return x;
}
```

Call by reference

- Call by reference is another way of passing parameter to the function.
- Here the address of the argument is copied into the parameter inside the function, the address is used to access arguments used in the call.
- Hence, changes made in the arguments are permanent.
- Here pointer is passed to function, just like any other arguments.

Example Program

```
#include<stdio.h>
#include<conio.h>
void swap(int,int);
void main()
{
        int a=5,b=10;
        clrscr();
        printf(“Before swapping a=%d b=%d”,a,b);
```

Output:

```
Before swapping a=5  b=10
After swapping a=10  b=5
```

```
          swap(&a,&b);
          printf("After swapping a=%d b=%d",a,b);
          getch();
     }
     void swap(int *x,int *y)
     {
          int *t;
          t=*x;
          *x=*y;
          *y=t;
     }
```

Nesting of function call in c programming

If we are calling any function inside another function call, then it is known as Nesting function call. In other words, a function calling different functions inside is termed as Nesting Functions.

Example:

```
// C program to find the factorial of a number.

#include <stdio.h>

 //Nesting of functions
//calling function inside another function
//calling fact inside print_fact_table
function

void print_fact_table(int);         // function declaration

int fact(int);                      // function declaration

void main()                        // main function
{
print_fact_table(5);              // function call
}

void print_fact_table(int n)       // function definition
{
 int i;
 for (i=1;i<=n;i++)
```

```
  printf("%d factorial is %d\n",i,fact(i)); //fact(i)-- function call
}

int fact(int n)                     // function definition
{
 if (n == 1)
  return 1;
 else
  return n * fact(n-1);
}
```

Output:

```
 1 factorial is 1
 2 factorial is 2
 3 factorial is 6
 4 factorial is 24
 5 factorial is 120
```

Recursion

A function calling same function inside itself is called as recursion.

Example: `// C program to find the factorial of a number.`

```
#include <stdio.h>
int fact(int); // function declaration
void main()            // main function
{
printf("Factorial =%d",fact(5));   // fact(5) is the function call
}
int fact(int n)        // function definition
{
if (n==1) return 1; else
return n * fact(n-1);   // fact(n-1) is the recursive function call
}
```

Output:

```
   Factorial = 120
```

Discussion:

For 1! , the functions returns 1, for other values, it executes like the one below:

When the value is 5, it comes to else part and calculates like this,

```
= 5 * fact (5-1)      = 5 * fact (4)

= 5* 4* fact (4-1)    = 5 * 4* fact (3)

= 5* 4* 3* fact (3-1)  = 5 * 4* 3* fact (2)

= 5* 4* 3*  2* fact (2-1)    = 5 * 4* 3* 2* fact (1)

= 5* 4* 3*  2* 1  (if (n==1) then return 1, hence we get 1)
```

=120

Example :

// A program that contains both nested functions and recursion in it.

// Find the maximum number among five different integers using nested function call and recursion.

```
int max(int x,int y)        // function defintion
 {
 return x>y ? x:y;    // condition operator is used (exp1?exp2:exp3)
 }

void main()                  // main function
{
    int m;
    m=max(max(4,max(11,6)),max(10,5)); //nested, recursive call
of function max
    printf("%d",m);
    getch();
}
```

Output: 11

QUESTIONS FOR PRACTICE

1. What would be the output of the following programs:

```
(a) main( )
{
 int a = 300, b, c ; if ( a >= 400 )
 b = 300 ; c = 200 ;
 printf ( "\n%d %d", b, c ) ;
}

(b) main( )
{
 int a = 500, b, c ; if ( a >= 400 )
b = 300 ; c = 200 ;
 printf ( "\n%d %d", b, c ) ;
}

(c) main( )
{
 int x = 10, y = 20 ; if ( x == y ) ;
 printf ( "\n%d %d", x, y ) ;
}

(d) main( )
{
 int x = 3, y = 5 ;
 if ( x == 3 )
   printf ( "\n%d", x ) ;
 else
   printf ( "\n%d", y ) ;
}

(e) main( )

{
int x = 3 ; float y = 3.0 ;
if ( x == y )
 printf ( "\nx and y are equal" ) ;
else
 printf ( "\nx and y are not equal" ) ;
}

(f) main( )
{
 int x = 3, y, z ; y = x = 10 ;
 z = x < 10 ;
 printf ( "\nx = %d y = %d z = %d", x, y, z ) ;
}
```

```
(g) main( )
{
 int k = 35 ;
 printf ( "\n%d %d %d", k == 35, k = 50, k > 40 ) ;
}
```

```
(h) main( )
{
 int i = 65 ; char j = ‘A’ ;
 if ( i == j )
  printf ( “C is WOW” ) ;
 else
  printf( "C is a headache" ) ;
}
```

```
(i) main( )
{
 int a = 5, b, c ; b = a = 15 ;
 c = a < 15 ;
 printf ( "\na = %d b = %d c = %d", a, b, c ) ;
}
```

```
(j) main( )
{

 int x = 15 ;
 printf ( "\n%d %d %d", x != 15, x = 20, x < 30 ) ;
}
```

2. What is the output of this C code (when 1 is entered)?

```
#include <stdio.h> void main()
{
    double ch;
    printf("enter a value btw 1 to 2:");
    scanf("%lf", &ch);
    switch (ch)
    {
    case 1: printf("1"); break;
```

```
        case 2: printf("2"); break;
        }
    }
```
a) Compile time error
b) 1
c) 2
d) Varies

3. What is the output of this C code (When 1 is entered)?

```
    #include <stdio.h> void main()
    {
        int ch;
        printf("enter a value btw 1 to 2:");
        scanf("%d", &ch);
        switch (ch, ch + 1)
        {
        case 1: printf("1\n"); break;
        case 2: printf("2"); break;
        }
    }
```
a) 1
b) 2
c) 3
d) Run time error

4. Find the output for the following Program

```
                  main()
                  {
                  int a,b , c =
                  1000; a = 100;
                  b = 20;
                  go to lab;
                  c = -1000;
                  printf(“\n Dummy”);
             lab:  print(“\n C value is %d”,c);
                  }
```

5. C program to display all prime numbers between Two interval entered by user.
6. C program to reverse a given number.
7. Program to Check Whether Given Number is Perfect Or Not.

8. Program Even Number Pyramid in C

```
2 4
2 4 6
2 4 6 8
```

9. Print prime number Pyramid in C

```
2

3     5
7     11      13
17      19     23     29
31      37     41     43   47
```

10. Write a C program to find the summation of the following series :

a) 1+ 2 + 3 +......+ n.
b) $1^2 + 2^2 + 3^2 + ... + n^2$.

11. The keyword used to transfer control from a function back to the calling function is

a) Switch b) Return c) Goto

12. Write a program to Calculate the Power of a Number Using Recursion

13. How many times the program will print "Sathyabama University" ?

```
#include<stdio.h>

int main()

{

    printf("\nSathyabma University");

    main();

    return 0;

}
```

a) Infinite times b) 32767 times c) Till stack overflows

14. What will be the output of the program?

```
#include<stdio.h>

int f1(int);

int main()

{

    int k=35;

    k = f1(k=f1(k=f1(k)));

    printf("k=%d\n", k);
    return 0;
}
int f1(int k)
{
    k++;
    return k;
}
```

15. Which of the following function declaration is illegal?

a) int 11bhk(int); b) int 11bh2k(int a);
c) int 22bhk2(int*, int []); d) All of the mentioned

16. Which function is not called in the following program?

```
#include <stdio.h>
    void one()
    {
        printf("first");
    }
    void two()
    {
        one();
    }
    void three()
    {
        two();
    }
    void main()
    {
        void (*ptr)();
        ptr = three;
        ptr();
    }
```

a) Function first b) Function second
c) Function third d) None of the mentioned

17. What will be the output of the program?

```
#include<stdio.h>

void fun(int*, int*);

int main()

{

    int i=5, j=2;

    fun(&i, &j);

    printf("%d, %d", i, j);

    return 0;

}

void fun(int *a, int *b)

{

    *a = *a**b;

    *b = *a**b;

}
```