1 What is user defined function? Explain with example. Define the syntax of function in C.

- A **function** is a block of code that performs a specific task.
- The functions which are created by programmer are called **user-defined functions**.
- The functions which are in-built in compiler are known as **system functions**.
- The functions which are implemented in header libraries are known as **library functions**.
- It has a unique name and it is **reusable** i.e. it can be called from any part of a program.
- **Parameter** or argument passing to function is **optional**.
- It is optional to return a value to the calling program. Function which is not returning any value from function, their return type is **void**.
- While using function, three things are important
  1) **Function Declaration:**
     - Like variables, all the functions must be declared before they are used.
     - The function declaration is also known as function prototype or function signature. It consists of four parts,
       a) Function type (return type).
       b) Function name.
       c) Parameter list.
       d) Terminating semicolon

     Syntax: <return type> FunctionName (Argument1, Argument2, Argument3……) ;
     Example: int sum (int , int) ;
     - In this example, function return type is **int**, name of function is sum, 2 parameters are passed to function and both are integer.
  2) **Function Definition:**
     - Function Definition is also called function implementation.
     - It has mainly two parts.
       a) **Function header:** It is same as function declaration but with argument name.
       b) **Function body:** It is actual logic or coding of the function
  3) **Function call:**
     - Function is invoked from main function or other function that is known as **function call**.
     - Function can be called by simply using a function name followed by a list of **actual** argument enclosed in parentheses.

- **Syntax or general structure of a Function:**

```
<return type>  FunctionName  (Argument1, Argument2, Argument3……)
{
    Statement-1;
    Statement-2;
    Statement-3;
}
```

- **An example of function:**

```
#include<stdio.h>
int sum(int, int);              \\ Function Declaration or Signature
void main()
{
   int a, b, ans;
   scanf("%d%d", &a, &b);
   ans = sum(a, b);             \\ Function Calling
   printf("Answer = %d", ans);
   }
int sum (int x, int y)              \\ Function Definition
{
   int result;
   result = x + y;
   return (result);
}
```

2 Explain different categories of functions.

- Functions can be classified in one of the following category based on whether arguments are present or not, whether a value is returned or not.
  1) Functions with no arguments and no return value \\ void printline(void)
  2) Functions with no arguments and return a value \\ int printline(void)
  3) Functions with arguments and no return value \\ void printline(int a)
  4) Functions with arguments and one return value \\ int printline(int a)
  5) Functions that return multiple values using pointer \\ void printline(int a)

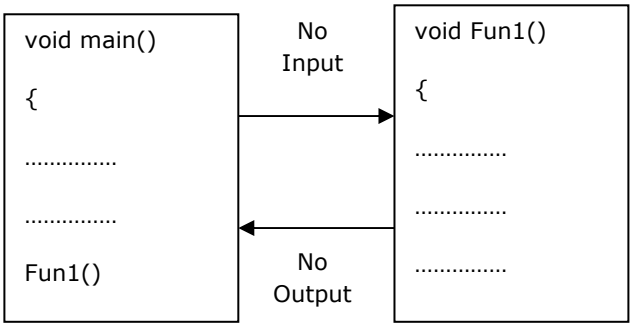
1. Function with no argument and no return value:

- When a function has no argument, it does not receive any data from calling function.
- When it does not return a value, the calling function does not receive any data from the called function.
- In fact there is no data transfer between the calling function and called function.

Example:

```
#include<stdio.h>
void   printline (void);    // No argument – No return value
void main()
{
      clrscr();
      printline();
      printf("\n GTU \n");
}
void  printline (void)
{
      printf("Hello");

}
```

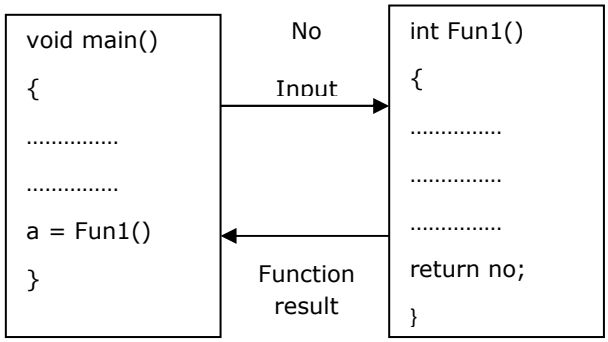


2. Function with no arguments and return a value:

- When a function has no argument, it does not receive data from calling function.
- When a function has return value, the calling function receives one data from the called function.

Example:

```
#include<stdio.h>
int  get_number (void);          // No argument
void main()
{
      int m;
      m=get_number();
      printf("%d",m);
}
int  get_number()
{
      int number;
      printf("enter number:");
      scanf("%d",&number);
      return   number;      // Return value
}
```

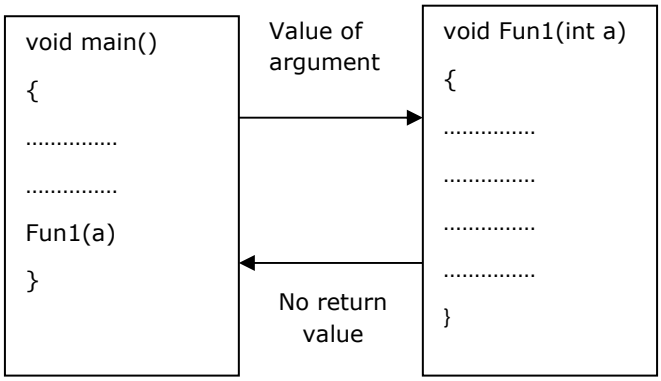


3. Function with arguments and no return values:

- When a function has argument, it receives data from calling function.
- When it does not return a value, the calling function does not receive any data from the called function.

Example:

```
#include<stdio.h>
void sum (int,int);
void main()
{
      int no1,no2;
      printf("enter no1,no2:");
      scanf("%d%d",&no1,&no2);
      sum(no1,no2);
}
void  sum (int  no1,int  no2)
{
      int ans;
      ans=no1+no2;
       prinf("Addition is: %d",ans);
}
```

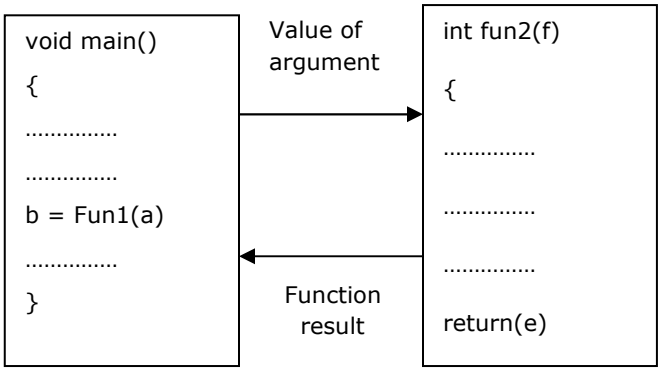


4. Function with arguments and one return value:

- When a function has argument, it receives data from calling function.
- When a function has return value, the calling function receives any data from the called function.

Example:

```
#include<stdio.h>
int sum(int, int);
void main()
{
   int a, b, ans;
   scanf("%d%d", &a, &b);
   ans = sum(a, b);
   printf("Answer = %d", ans);
   }
int sum (int x, int y)
{
   int result;
   result = x + y;
   return result;
}
```



5. Function that returns a multiple value:

- Function can return either one value or zero value. It cannot return more than one value.
- To receive more than one value from function, we have to use pointer.
- So function should be called with reference not with value.

Example:

```
#include<stdio.h>
void  mathoperation (int x, int y, int *s, int *d);
void main()
{
      int x=20,y=10,s,d;
      mathoperation(x,y,&s,&d);
      printf("s=%d \nd=%d", s,d);
}
void mathoperation(int a, int b, int *sum, int *diff)
{
      *sum = a + b;
      *diff = a - b;
}
```

3 Explain actual argument and formal argument with example.

- Arguments passed to the function during function calling are called actual arguments or parameters.
- Arguments received in the definition of a function are called formal arguments or parameters.

Example:

```
#include<stdio.h>
int  max(int , int )  // Function Declaration.
void main()
{
      int  a=5,b=3, ans;
      ans = max(a, b);       //a and b are actual arguments.
      printf("max=%d", ans);
}
int max (int x, int y)     // x and y are formal arguments.
{
      if(x>y)
            return x;
      else
            return y;
}
```

4 Explain call by value (pass by value) and call by reference (pass by reference) with example.

- The parameters can be passed in two ways during function calling,
  - Call by value
  - Call by reference

Call by value

- In call by value, the values of actual parameters are copied to their corresponding formal parameters.
- So the original values of the variables of calling function remain unchanged.
- Even if a function tries to change the value of passed parameter, those changes will occur in formal parameter, not in actual parameter.

Example:

```
#include<stdio.h>
void  swap(int, int);
void main()
{
      int x, y;
      printf("Enter the value of X & Y:");
      scanf("%d%d", &x, &y);
      swap(x, y);
      printf("\n Values inside the main function");
      printf("\n x=%d, y=%d", x, y);
```

```
        getch();
}
void swap(int x,int y)
{
        int temp;
        temp=x;
        x=y;
        y=temp;
        printf("\n Values inside the swap function");
        printf("\n x=%d y=%d", x, y);
}
```

Output:

```
Enter the value of X & Y: 3 5
Values inside the swap function
X=5 y=3
Values inside the main function
X=3 y=5
```

Call by Reference

- In call by reference, the address of the actual parameters is passed as an argument to the called function.
- So the original content of the calling function can be changed.
- Call by reference is used whenever we want to change the value of local variables through function.

Example:

```
#include<stdio.h>
void swap(int *, int *);
void main()
{
        int x,y;
        printf("Enter the value of X & Y:
        scanf("%d%d", &x, &y);
        swap(&x, &y);
        printf("\n Value inside the main function");
        printf("\n x=%d y=%d", x, y);
}
void swap(int *x, int *y)
{
        int temp;
        temp=*x;
        *x=*y;
        *y=temp;
        printf("\n Value inside the swap function");
        printf("\n x=%d y=%d", x, y);
}
```

Output:

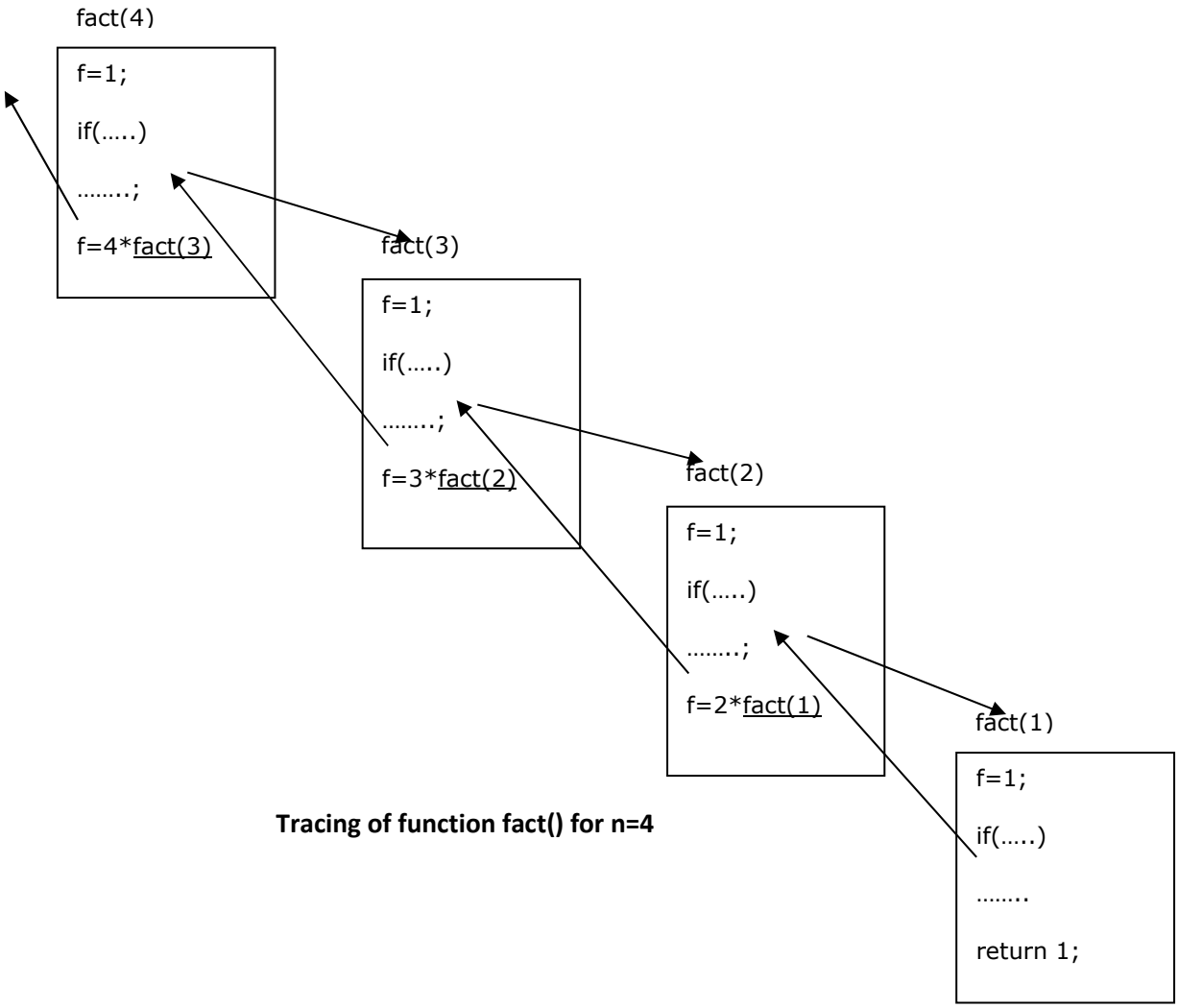
```
Enter the value of X & Y: 3 5
```

Value inside the swap function
x=5 y=3
Value inside the main function
x=5 y=3

- Difference between two program is marked as **bold** in call by reference program

4 What do you mean by recursive function? Explain with example.

- Recursive function is a function that **calls itself**.
- **Pictorial presentation of recursion :**



Tracing of function fact() for n=4

- If a function calls itself then it is known as recursion.
- Recursion is thus the process of defining something in terms of itself.
- Suppose we want to calculate the factorial of a given number then in terms of recursion we can write it as n! = n * (n-1)!. First we have to find (n-1)! and then multiply it by n. the (n-1)! is computed as (n-1)! = (n-1) * (n-2)!. This process end when finally we need to calculate

1!.which is 1.

Ex: 4! =4*3!
=4*3*2!
=4*3*2*1!
=4*3*2*1

- Terminating condition must be there which can terminate the chain of process, otherwise it will lead to infinite number of process.

Example: Find factorial of a given number using recursion.

```
#include<stdio.h>
int  fact (int);
void main()
{
      int f,n;
      printf("enter number:");
      scanf("%d",&n);
      f=fact(n);
      printf("\n factorial=%d",f);
}
int  fact (int n)
{
      int f=1;
      if(n==1)
      {
            return 1;
      }
      else
      {
            f=n*fact(n-1);
            return f;
      }
}
```

Advantages:

- Easy solution for recursively defined solution.
- Complex programs can be easily written with less code.

Disadvantages:

- Recursive code is difficult to **understand** and **debug**.
- Terminating condition is must; otherwise it will go in an **infinite** loop.
- Execution speed decreases because of function call and return activity many times.

5 What is scope, lifetime and visibility of variable?

Scope

- The scope of variable can be defined as that a part of a program where the particular variable is accessible
- In what part of the program the variable is accessible is depends on where the variable is declared.
- Local variables which are declared inside the body of function cannot be accessed outside the body of function.
- Global variables which are declared outside any function definition can be accessible by all the function in a program.

Lifetime

- Lifetime is a time limit during the program execution until which a variable exist in a memory.
- It is also referred to the longevity of variable.

Visibility

- Visibility is the ability of the program to access a variable from the memory.
- If variable is redeclared within its scope of variable, the variable losses the visibility in the scope of variable which is redeclared.