Assignment - 3

What is functions ? explain the difference between user defined and library functions

Function :

- A function (including matrix) is an independent module that will be called to do a specific task
- A called function receives control from calling function when called
- When the called function complete its task, it returns control back to the calling function. It may or may not return a value to the caller

Library functions	User defined Functions
• Predefined by c to libraries	• Functions created by the programmer based on specific needs
• Already written and stored in libraries files	• Written and implemented by the user in the source code
• Call directly after including the respective header files	• Defined and implemented explicitly in the program by the user
• Universal and can be reused across different programs without modification	• Limited to the program in which they are written unless shared
Universal and can be used across different programs without modification	

• Simple to use as they are pre-tested and require no additional coding	• Requires design, logic, and debugging by the user
• Requires inclusion of specific header files for use	• Does not rely on predefined headers, but headers can be created
• No additional compilation needed; part of standard	• Compiled with the rest of the user's source file
• Ex: printf( ), scanf( ), sqrt( )	• Custom function like int sum(int a, int b) or void greet( )

2 Differentiate between Call by value and Call By Refrence.

A

Call by Value	Call by Refrence
• Copies the actual value of arguments into the function's formal parameter's	• Passes the address of arguments allowing the function to modify them.
• The original data remains unchanged after the function call	• The original data can be modified by the function
• Only the values of variables are passed to the function	• Memory address (pointers) of variables are passed to the function
• Use normal variables as parameters (eg:- void func(int a)).	• Use pointer variables as parameter (ex: void func(int * a)).
• More memory is required because values are copied	• Less memory is used as no copying of data is there
• Slower due to the overhead of copying values	• Faster as only addresses are passed

```
#include <stdio.h>
#include <conio.h>
void swap (int , int);
void main ( )
{
 int i,j
printf (" enter i & j values : ");
scanf (" %d%d ", &i ,&j);
printf (" Before swapping : %d %d\n",
                    i,j );

swap (i,j)
printf ("After swapping: %d %d\n",i,j);
getch ( );
}
void swap ( int a , int b)
{
```

```
#include <stdio.h>
#include <conio.h>
swap-to void swap (int* , int* )
void main ( )
{
int i = 10 , j = 20 ;
printf (" Before swapping : %d %d", i,j);
swap ( &i , &j );
printf (" After swapping : %d %d\n", i,j);
}
void swap ( int *a , int *b)
{
 int = temp ;
 temp = *a ;
*a = *b ;
*b = temp ;
}
```

3 Define recursion . WACP to calculate the GCD using recursion function

A) Recursion

Process of solving a problem by reducing it to smaller versions of itself

Recursive definition

- Definition in which a problem is expressed is expressed in terms of a smaller version of itself
- Has one or base cases

Recursive solution to GCD solution

```
# include < stdio.h >
int gcd (int , int);
void main (void);
{
 int n1 , n2 ;
 printf (" enter two integers : ");
 scanf (" %d %d ", &n1 , &n2 );
 printf (" GCD of %d and %d is %d ", n1 n2 , gcd (n1 , n2));
 return 0 ;
}
int gcd ( int n1 , int n2 )
 {
 if (n2 != 0 )
 return gcd (n2 , n1 (%. n2 ));)
 else
 return n1 ;
 }
```

4. Write a C program using recursive function for finding factorial of a number and to find the n^{th} term of fibonacci series

A

Finding factorial of a number

```
# include <stdio.h>
int factorial (int);
void main (void)
{
int n, res;
printf ("enter any number : ");
scanf ("%d", &n);
res= factorial (n);
printf (" factorial of %d is %d", n, res);
}
int factorial (int num)
{
int i, fact = 1;
for (i=1; i<= num ; i++)
fact= fact * i;
return fact ;
}
```

return 5 * factorial (4) = 120

return 4 * factorial (3) = 24

return 3 * factorial (2) = 6

return 2 * factorial (1) = 2

return 1 * factorial (0) = 1

1 * 2 * 3 * 4 * 5 = 120

WACP to find the nth term of fibonacci series

```
#include <stdio.h>
int main()
{
int i,n;
int t1=2,t2=4;
int nextterm = t1+t2;
printf("enter the no of terms : ");
scanf("%d",&n);
printf("fibonacci series : %d,%d,",t1,t2);
for(i=3;i<=n;++i)
{
printf("%d,",nextterm);
t1=t2
t2 = nextterm
nextterm = t1+t2;
}
return 0;
}
```

OUTPUT

enter the no of terms : 5

Fabonacci series: 2,4,6,10,16

5 Explain types of arguments of passing techniques with example

A Parameters passing in functions

There are two ways of passing the parameters to a function

They are

1. Call by Value
2. Call by Reference

Call by Value

In this method value of the actual arguments in the calling function are copied into the ~~computer~~ corresponding parameter in the called function

Ex:

```
# include <stdio.h>
# include <conio.h>
void swap (int, int);
void main ()
{
int i,j;
printf (" enter i and j values: ");
scanf ("%d %d", &i, &j);
printf (" Before swapping = %d %d \n", i,j);
swap (i,j);
printf (" After swapping = %d %d \n", i,j);
getch ();
}
```

void swap (int a, int b)

{

OUTPUT

Enter i and j values : 10 20

Before swapping : 10 20

After swapping : 10 20

2 Call by Refrence

In this method the address of the actual parameters are copied to the corresponding parameter in the called function. Hence any modification done to formal parameters in the called function causes the actual parameter to change

Eg:

```
#include <stdio.h>
swap(int*, int*);
main()
{
int i,j;
printf("enter i and j values:");
scanf("%d%d",&i,&j);
printf("Before swapping: %d %d \n",i,j);
swap(&i, &j);
printf("After swapping: %d%d\n", i,j);
}
```

```
void swap (int *a, int *b)
{
int = temp ;
temp = *a ;
*a = *b;
*b = temp ;
}
```

OUTPUT.

Enter i & j values : 10 20

Before swapping : 10 20

After swapping : 20 10

6 Distinguish between actual & formal parameters

A)

Actual Parameters	Formal Parameters
• The values or variables are passed to function during a call	• The variable defined in a function's declaration or definition
• Appear in the function call	• Appears in the function header (declaration or definition)
• Provide input to the function	• Receive the input and hold the values passed by actual parameters

• Can be constant, variables or expressions	• Must be variables or place holders declared in the function signature
• Uses the memory allocated for the variables or literals passed	• Allocated memory temporarily during the execution of the function
• Dependent on the calling function	• Dependent of on the called function
• Provide input to the function	• Serve as place holders to work with the input values
• Cannot have default values (they are explicitly passed)	• Can have default values defined in the function signature
• ex: sum (5, 10) here 5 and 10 are actual parameters	• ex: int add (int x, int y); here x and y are formal parameters