Name	Lekh Sanatan Nayak
UID no.	2023800068
Experiment No.	6

AIM:	**Apply the concept of recursion to solve a given problem.**
Program 1	
PROBLEM STATEMENT :	Write a recursive function to multiply 2 numbers
ALGORITHM:	1. Start 2. Declare a function “mul” which takes two integer values ‘a’ and ‘b’ as arguments. 3. Define a variable “ans” which will store the product of the two numbers. 4. If ‘b’ is equal to 0, return 0(this is our base case) 5. Else perform the following for the recursive case i) Subtract 1 from the value of ‘b’ (b-1), and call “mul” funtion with arguments ‘a’ and ‘b-1’ (repeat untill b=0) ii) Add the value of the recursive steps to ‘a’ and then store the value in ‘ans’ 6. Return the value of ‘ans’. 7. Declare a function “main” 8. Define variables ‘a’ and ‘b’ 9. Take user input for the values of ‘a’ and ‘b’ 10. Call the function “mul” with arguments ‘a’ and ‘b’ 11. End

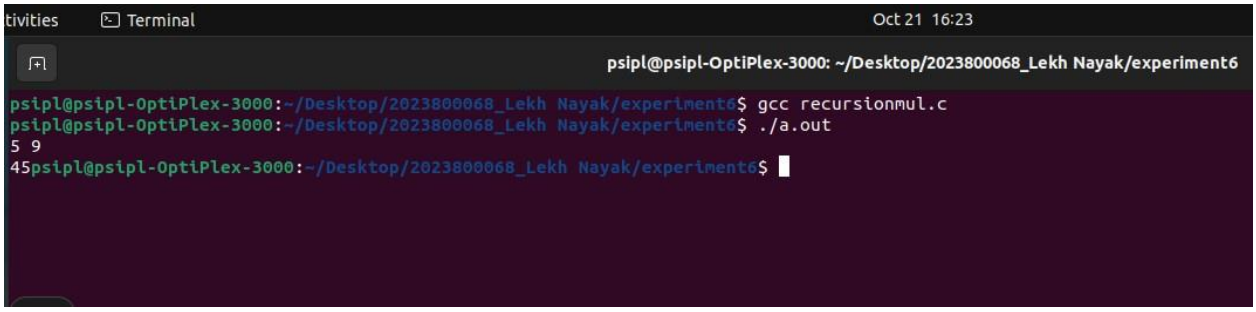
PROGRAM:

```
#include<stdio.h>
int mul(int a, int b){
        int ans;
                if(b==0){
                        return 0;
                }
                else{
                        ans = a + mul(a,b-1);
                }
        return ans;
}
int main(){

        int a,b;
        scanf("%d%d",&a,&b);
        int ans = mul(a,b);
        printf("%d",ans);

        return 0;
}
```

RESULT:



Program 2

PROBLEM STATEMENT :

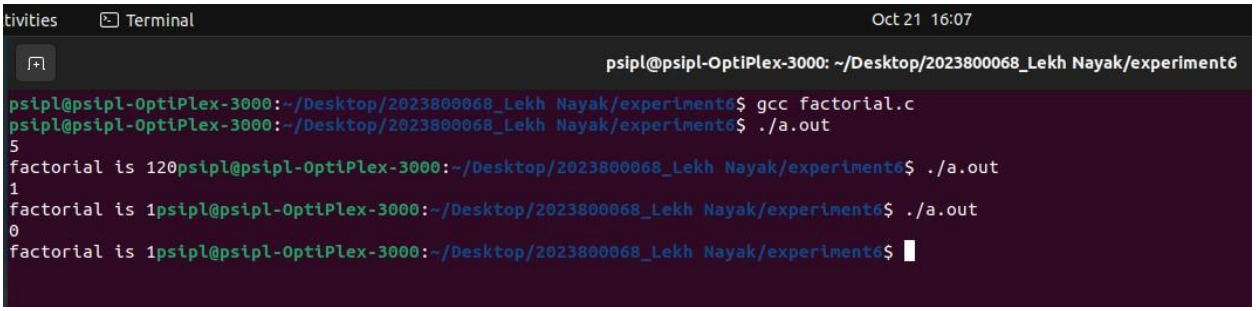
Write a recursive function to find the factorial of a number and test it.

ALGORITHM:

1. Start
2. Declare a function “factorial” with one integer argument ‘a’.
3. Define a variable ‘ans’
4. If a=1 or a=0, then return 1(this is our base case)
5. Else perform the following for the recursive case
    i. Subtract 1 from ‘a’ (a-1) and call the function “factorial”

	recursively with arguments(a-1) ii. Multiply the result of the recursive step(factorial of a-1) with 'a' and store it in 'ans' 6. Return the value of 'ans' 7. Declare a function "main" 8. Define a variable 'a' 9. Take user input for the value of 'a' 10. Call the function "factorial" with integer argument 'a' 11. End

PROGRAM:	#include<stdio.h>
	
	int factorial(int a){
	int ans;
	if(a==1 || a==0){
	return 1;
	}
	else{
	ans=a*factorial(a-1);
	}
	
	}
	int main(){
	
	int a;
	scanf("%d",&a);
	int ans=factorial(a);
	printf("factorial is %d",ans);
	return 0;
	}

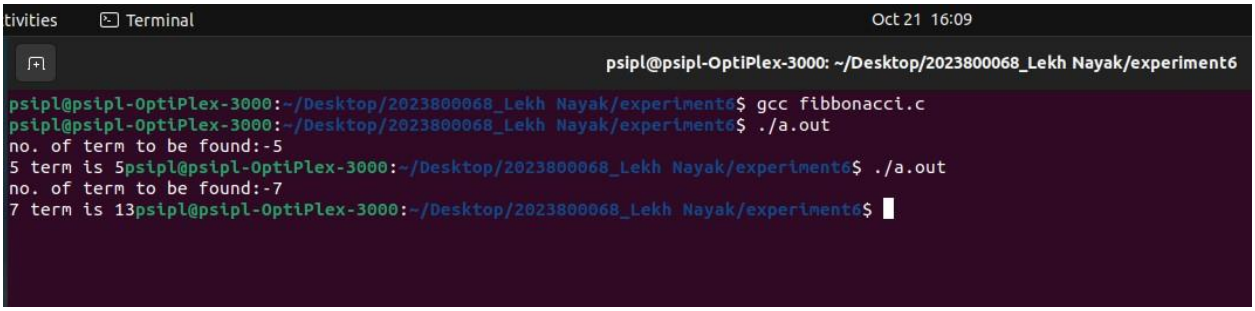
RESULT:  	
Program 3.A	

PROBLEM STATEMENT:	Write a recursive function which returns the nth term of the fibonacci series.
ALGORITHM:	1. Start 2. Declare a funtion “fibbo” that takes 1 integer value ‘n’ as argument. 3. Define a variable ans and initialize it to 0 4. If the value of n is 1 or 2 then return 1 to the function(this is our base case) 5. Else perform the following for recursive cases i) Subtract 1 and 2 from ‘n’ i.e. (n-1) and (n-2) and call the function “fibbo” recursively with arguments (n-1) and (n-2). j) Call the function “fibbo” twice with arguments (n-1)(n-2)and add them both and store their sum in ‘ans’ variable. 6. Return the value of ans 7. Declare a function “main” 8. Define a variable ‘input’ 9. Take user input for the value of ‘input’ and print it 10. Call the function fibbo with ‘input’ as argument and store it in a variable ‘ans’ 11. Return 0 to the function 12. End

PROGRAM:	#include<stdio.h>
	
	int fibbo(int n){
	int ans=0;
	if(n==1 || n==2){
	return 1;
	}
	else{
	ans=fibbo(n-2)+fibbo(n-1);
	}
	
	return ans;
	}
	
	int main(){
	int input;
	printf("no. of term to be found:-");
	scanf("%d",&input);

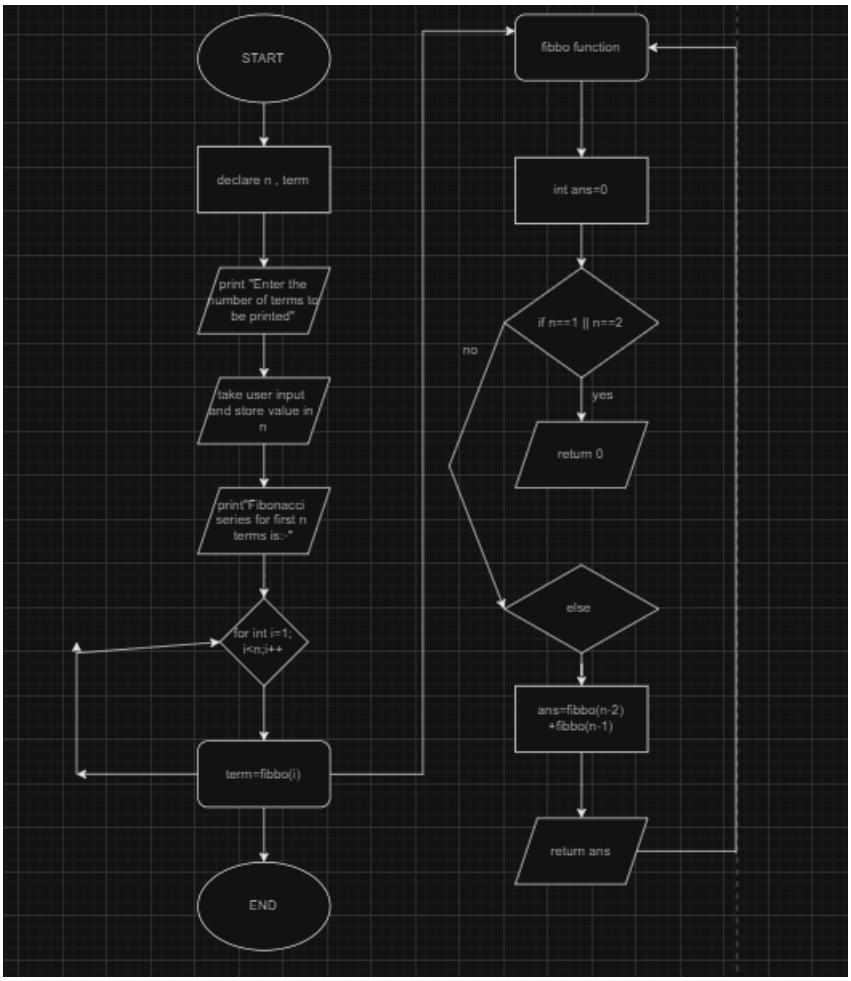
```
        int ans = fibbo(input);
        printf("%d term is %d",input,ans);
        return 0;
}
```

RESULT:



Program 3.B

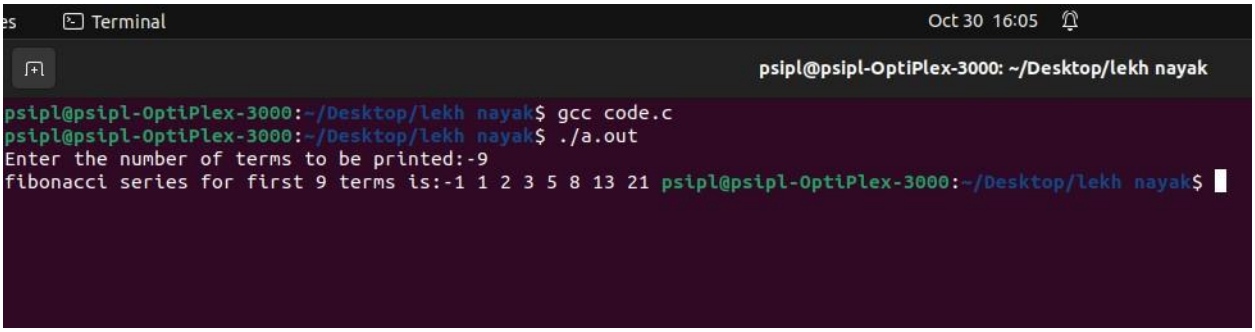
PROBLEM STATEMENT:	Call it from main() to find the 1st n numbers of the fibonacci series.

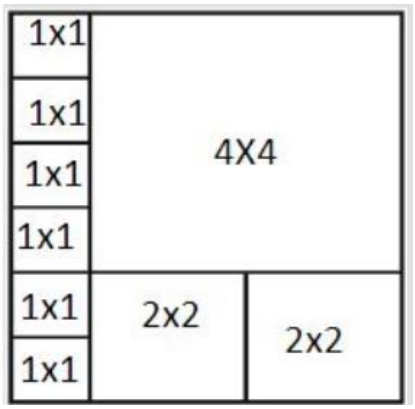
FLOWCHART:	
ALGORITHM:	1. Start 2. Declare a funtion "fibbo" that takes 1 integer value 'n' as argument. 3. Define a variable ans and initialize it to 0 4. If the value of n is 1 or 2 then return 1 to the function(this is our base case) 5. Else perform the following for recursive cases i) Subtract 1 and 2 from 'n' i.e. (n-1) and (n-2) and call the function "fibbo" recursively with arguments (n-1) and (n-2). j) Call the function "fibbo" twice with arguments (n-1)(n-2)and add them both and store their sum in 'ans' variable. 6. Return the value of ans 7. Declare a function "main" 8. Define variables 'n' and 'term'. 9. Print the statement "Enter the number of terms to be printed:' 10. Take user input for the number of terms to be printed 11. Create a 'for' loop and initialize integer variable 'i' to 1, define the

	condition ‘i<n’ for executing the code block, and set the statement 3 to ‘i++’. 12. Call the function “fibbo” and store its value in ‘term’ variable and print the value of ‘term’ inside the “for” loop. 13. Return 0 to show that the program has executed itself . 14. End.

PROGRAM:	#include<stdio.h>
	
	int fibbo(int n){
	int ans=0;
	if(n==1 || n==2){
	return 1;
	}
	else{
	ans = fibbo(n-2) + fibbo(n-1);
	}
	
	return ans;
	}
	
	int main(){
	
	int n, term;
	printf("Enter the number of terms to be printed:-");
	scanf("%d",&n);
	
	printf("fibonacci series for first %d terms is:-", n);
	
	int i;
	for(i=1; i<n ;i++){
	term = fibbo(i);
	printf("%d ",term);
	}
	
	return 0;
	}

RESULT:	



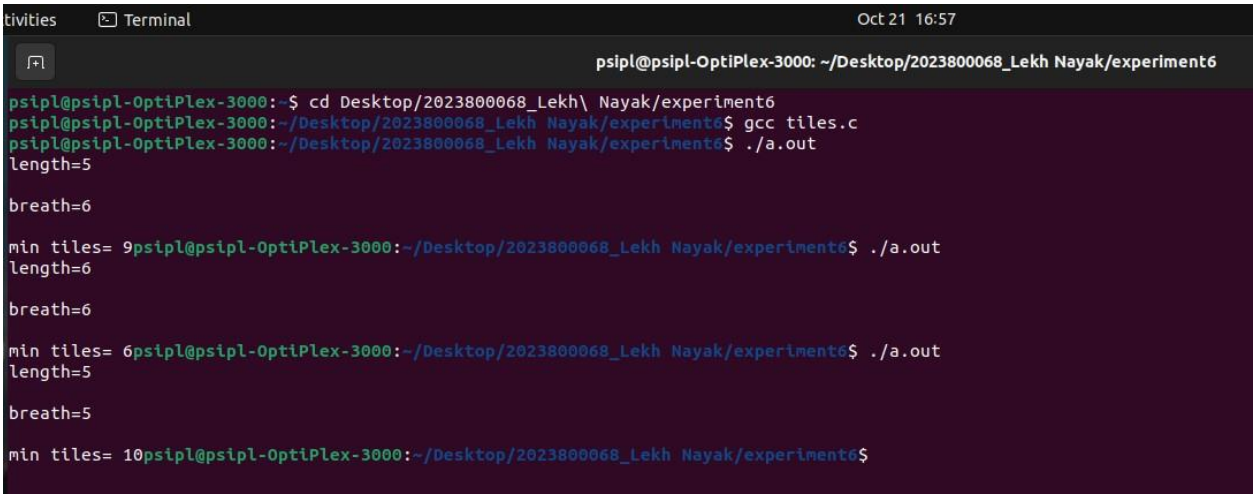
Program 4	
PROBLEM STATEMENT:	Given a room of area L x B. You have an infinite number of tiles of size 2n x 2n , where n = 0, 1,2,... so on. The task is to find the minimum number of square tiles required to fill the given area with tiles. 
ALGORITHM:	1. Declare a function “tile” that takes two parameters, l for the length and b for the breadth of the floor. 2. Initialize a variable ‘ans’ to 0, which will store the minimum number of tiles required. 3. Check if either the length ‘l’ or breadth ‘b’ is 0. If either of them is 0, return 0, indicating that no tiles are needed. 4. Check if both l and b are even numbers. If they are, recursively call the tile function with half of l and half of b and assign the result to ans. 5. If both l and b are odd numbers, calculate l + b - 1, which represents the minimum number of tiles required to cover the floor diagonally, and then recursively call the tile function with half of l and half of b. Add this result to ans. 6. If l is even and b is odd, add l to ‘ans’ and recursively call the tile function with half of l and half of b.

	7. If l is odd and b is even, add b to ‘ans’ and recursively call the tile function with half of l and half of b. 8. Finally, return the ‘ans’ as the minimum number of tiles required to cover the entire floor. 9. Declare a function “main” 10. Take user input for the values of ‘l’ and ‘b’ 11. Call the function tile with arguments l and b and store the value in ‘ans’ variable 12. Print ‘ans’ 13. End

PROGRAM:	#include<stdio.h>
	int tile(int l,int b){
	int ans=0;
	
	if(l==0 || b==0){
	return 0;
	}
	else if(l%2==0 && b%2==0){
	ans= tile(l/2,b/2);
	}
	else if(l%2==1 && b%2==1){
	ans= l+b-1+tile(l/2,b/2);
	}
	else if(l%2==0 && b%2==1){
	ans= l+tile(l/2,b/2);
	}
	else if(l%2==1 && b%2==0){
	ans= b+tile(l/2,b/2);
	}
	return ans;
	}
	
	int main(){
	int l,b;
	printf("length=");
	scanf("%d",&l);
	printf("\nbreath=");
	scanf("%d",&b);

```
        int ans=tile(l,b);
        printf("\nmin tiles= %d",ans);
        return 0;
}
```

RESULT:



CONCLUSION:	In this experiment I learnt the application of recursive functions to solve a given problem