1) Write a program to find the sum of n natural using a user defined function

Code:

```c
GNU nano 8.4
#include <stdio.h>

// User-defined function to calculate sum of first n natural numbers
int sumOfNatural(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}

int main() {
    int n;
    printf("Enter the value of n: ");
    if (scanf("%d", &n) != 1) {
        printf("Invalid input.\n");
        return 1;
    }
    if (n < 0) {
        printf("Please enter a non-negative integer.\n");
        return 1;
    }

    int result = sumOfNatural(n);
    printf("Sum of %d natural numbers = %d\n", n, result);
    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~]
└─$ gcc -Wall sum.c -o sum

┌──(kali㉿kali)-[~]
└─$

┌──(kali㉿kali)-[~]
└─$

┌──(kali㉿kali)-[~]
└─$ ./sum

Enter the value of n: 34
Sum of 34 natural numbers = 595

┌──(kali㉿kali)-[~]
└─$ ▮
```
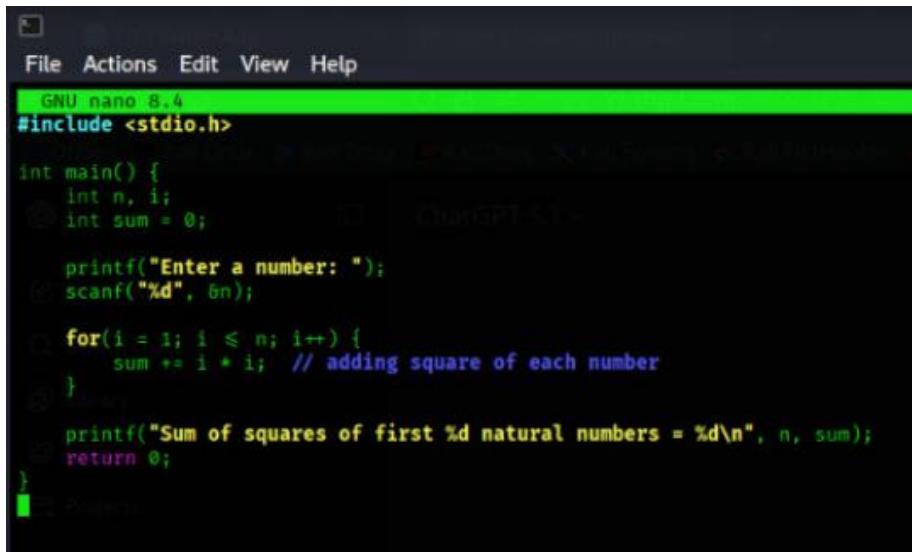
SPACE COMPLEXITY:
 • The loop uses only two variables (s and i), and no extra memory grows with n.
 • So, the memory used stays constant no matter the input size.
 • Therefore, the space complexity is O(1).

2)Write a program to find the sum of squares of n natural numbers

Code:

```
File  Actions  Edit  View  Help
  GNU nano 8.4
#include <stdio.h>

int main() {
    int n, i;
    int sum = 0;

    printf("Enter a number: ");
    scanf("%d", &n);

    for(i = 1; i ≤ n; i++) {
        sum += i * i;  // adding square of each number
    }

    printf("Sum of squares of first %d natural numbers = %d\n", n, sum);
    return 0;
}
```
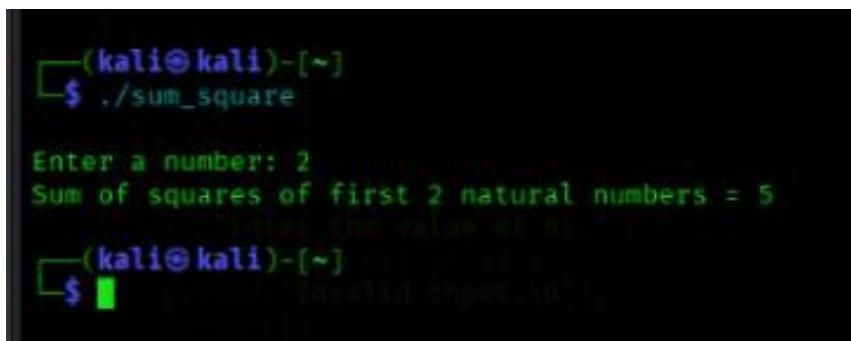
OUTPUT:

```
  ┌──(kali㉿kali)-[~]
  └─$ ./sum_square

Enter a number: 2
Sum of squares of first 2 natural numbers = 5

  ┌──(kali㉿kali)-[~]
  └─$
```
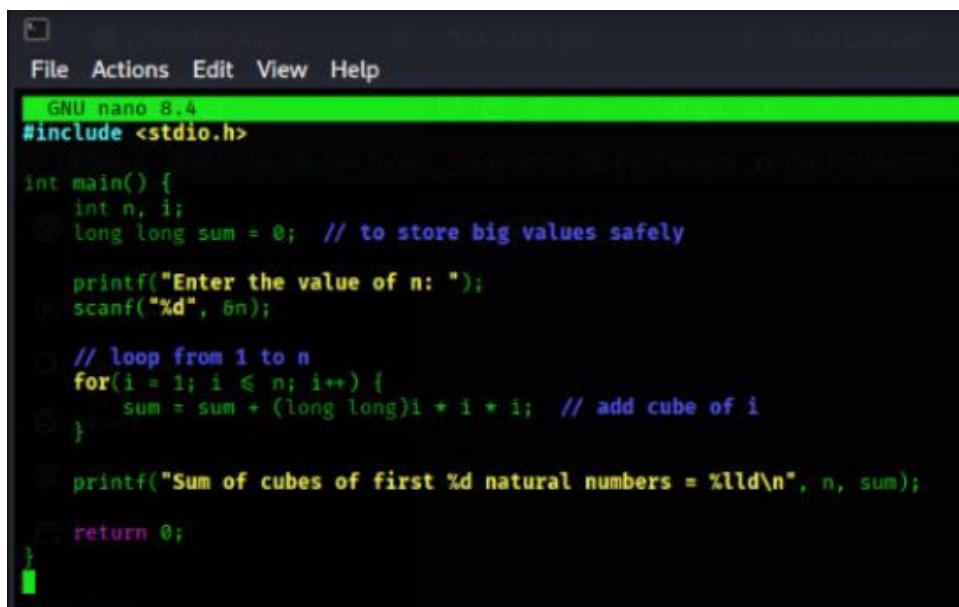
SPACE COMPLEXITY:

• The program uses only a few variables, and this number does not increase when n becomes bigger.

• Since the memory used stays the same all the time, the space complexity is O(1).

3)Write a program to find the sum of cubes of n natural the sum of cubes of n natural numbers

CODE:

```c
#include <stdio.h>

int main() {
    int n, i;
    long long sum = 0;   // to store big values safely

    printf("Enter the value of n: ");
    scanf("%d", &n);

    // loop from 1 to n
    for(i = 1; i <= n; i++) {
        sum = sum + (long long)i * i * i;   // add cube of i
    }

    printf("Sum of cubes of first %d natural numbers = %lld\n", n, sum);

    return 0;
}
```
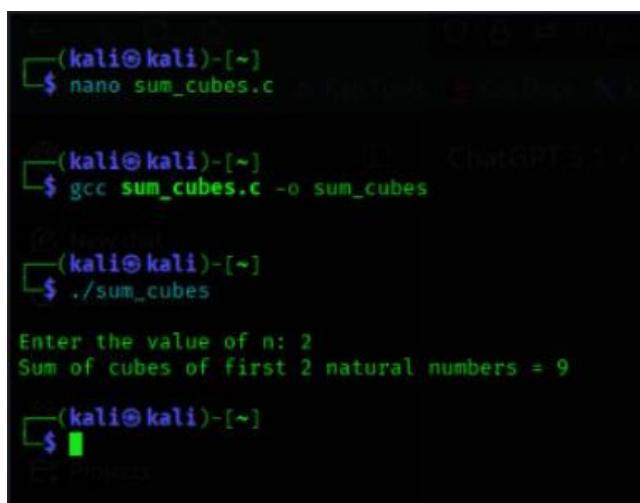
OUTPUT:

```
┌──(kali㊀kali)-[~]
└─$ nano sum_cubes.c

┌──(kali㊀kali)-[~]
└─$ gcc sum_cubes.c -o sum_cubes

┌──(kali㊀kali)-[~]
└─$ ./sum_cubes

Enter the value of n: 2
Sum of cubes of first 2 natural numbers = 9

┌──(kali㊀kali)-[~]
└─$
```
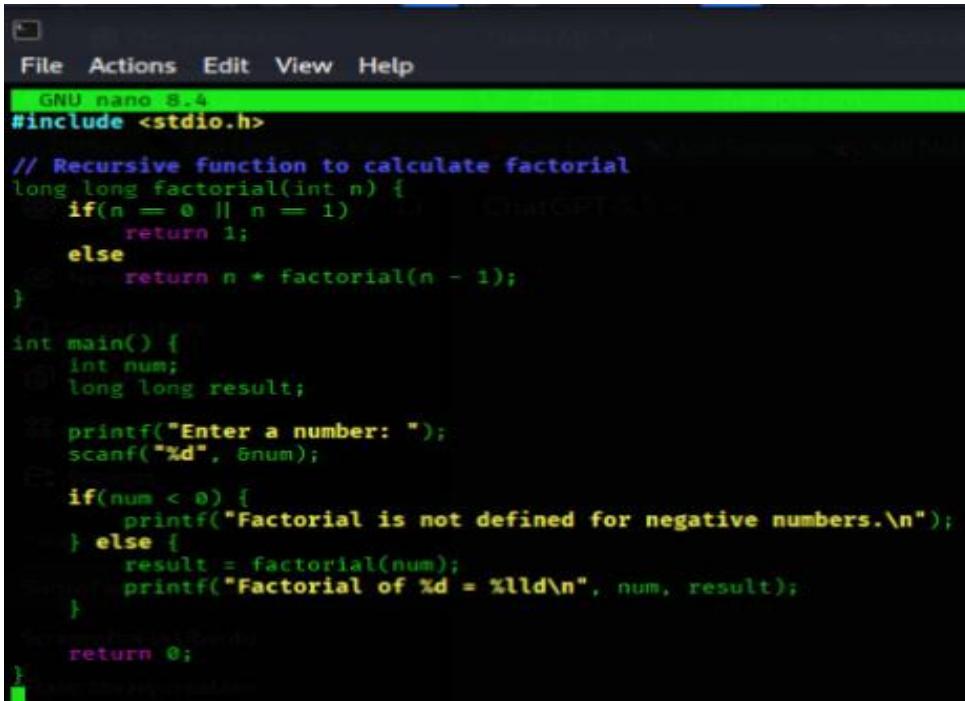
SPACE COMPLEXITY:

• The loop doesn't create new memory again and again — it just reuses the same variables.

• Because the memory doesn't grow when n grows, the space stays constant.

• So it is O(1) space.

4)Write a program to find the factorial of a given number using recursion

CODE:

```c
#include <stdio.h>

// Recursive function to calculate factorial
long long factorial(int n) {
    if(n == 0 || n == 1)
        return 1;
    else
        return n * factorial(n - 1);
}

int main() {
    int num;
    long long result;

    printf("Enter a number: ");
    scanf("%d", &num);

    if(num < 0) {
        printf("Factorial is not defined for negative numbers.\n");
    } else {
        result = factorial(num);
        printf("Factorial of %d = %lld\n", num, result);
    }

    return 0;
}
```
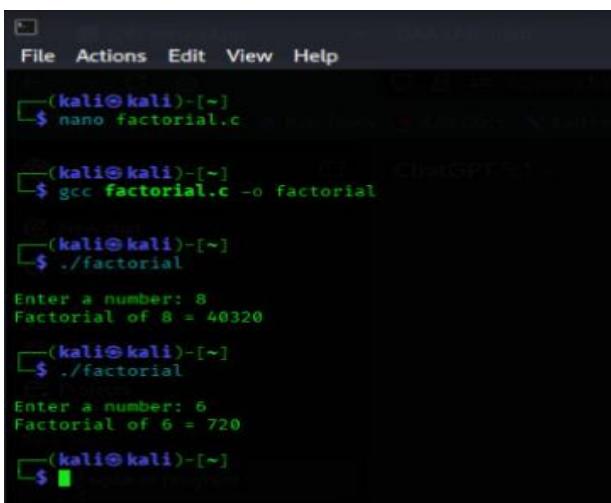
OUTPUT:

```
┌──(kali㉿kali)-[~]
└─$ nano factorial.c

┌──(kali㉿kali)-[~]
└─$ gcc factorial.c -o factorial

┌──(kali㉿kali)-[~]
└─$ ./factorial

Enter a number: 8
Factorial of 8 = 40320
┌──(kali㉿kali)-[~]
└─$ ./factorial

Enter a number: 6
Factorial of 6 = 720

┌──(kali㉿kali)-[~]
└─$ ▮
```

SPACE COMPLEXITY:

• The function calls itself many times, and each call uses some memory.

• More calls mean more memory is used, so memory increases with n.

• That's why the space complexity is O(n).

5)Write a program to transpose a 3x3 matrix

CODE:

```c
#include <stdio.h>

int main() {
    int matrix[3][3], transpose[3][3];
    int i, j;

    // Input elements for 3×3 matrix
    printf("Enter 9 elements of the matrix:\n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    // Finding transpose
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }

    // Printing original matrix
    printf("\nOriginal Matrix:\n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    // Printing transpose matrix
    printf("\nTranspose of the Matrix:\n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d ", transpose[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

OUTPUT:

```
┌──(kali㉿kali)-[~]
└─$ ./transpose

Enter 9 elements of the matrix:
1
2
3
4
5
6
7
8
9

Original Matrix:
1 2 3
4 5 6
7 8 9

Transpose of the Matrix:
1 4 7
2 5 8
3 6 9
```
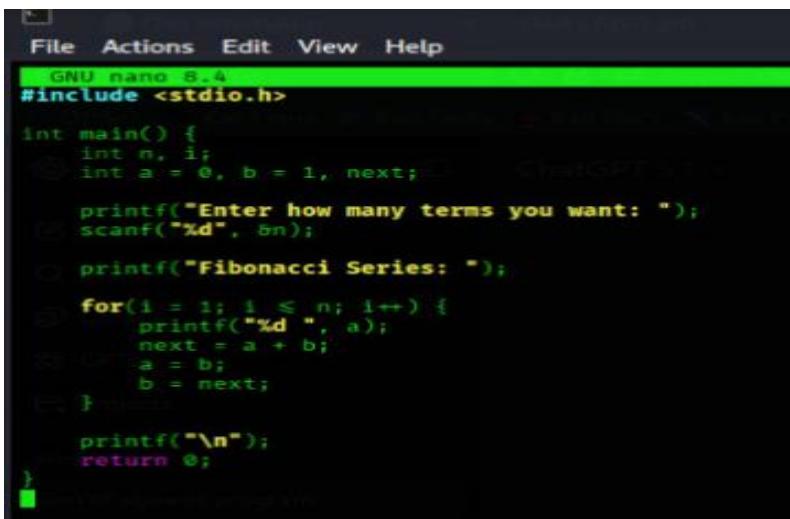
SPACE COMPLEXITY:

• The program creates a 3×3 array, which is fixed in size and does not grow.

• You only use a few extra variables for loops.

• So the space complexity is O(1) 6. Write a program to find the Fibonacci series

6) Write a program to find the Fibonacci series
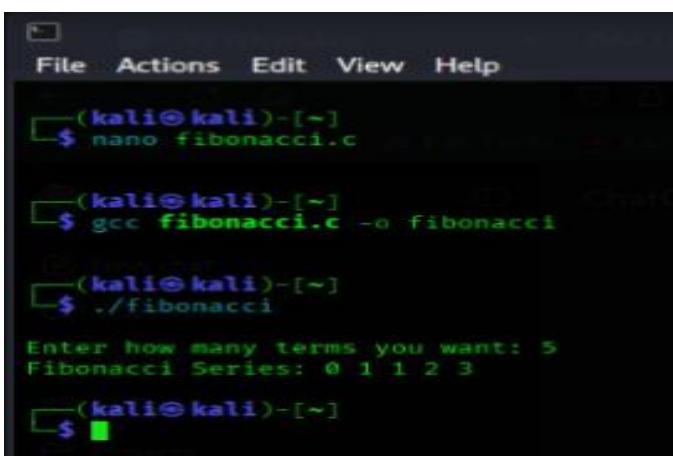
CODE:

```c
#include <stdio.h>

int main() {
    int n, i;
    int a = 0, b = 1, next;

    printf("Enter how many terms you want: ");
    scanf("%d", &n);

    printf("Fibonacci Series: ");

    for(i = 1; i <= n; i++) {
        printf("%d ", a);
        next = a + b;
        a = b;
        b = next;
    }

    printf("\n");
    return 0;
}
```

OUTPUT:

```
(kali@kali)-[~]
$ nano fibonacci.c

(kali@kali)-[~]
$ gcc fibonacci.c -o fibonacci

(kali@kali)-[~]
$ ./fibonacci
Enter how many terms you want: 5
Fibonacci Series: 0 1 1 2 3

(kali@kali)-[~]
$
```

SPACE COMPLEXITY:

• The program only uses a few variables (a, b, sum, temp) and these do not increase when n increases.

• No extra memory grows with the loop.

• So the space complexity is O(1).