AMRITA
VISHWA VIDYAPEETHAM

Second Year B.Tech

(Computer Science and Engineering)

Design and Analysis of Algorithms

TASK – 1

Name: Shaik Kolimi Dada Hussain

College Name: Amrita vishwa Vidyapeetham

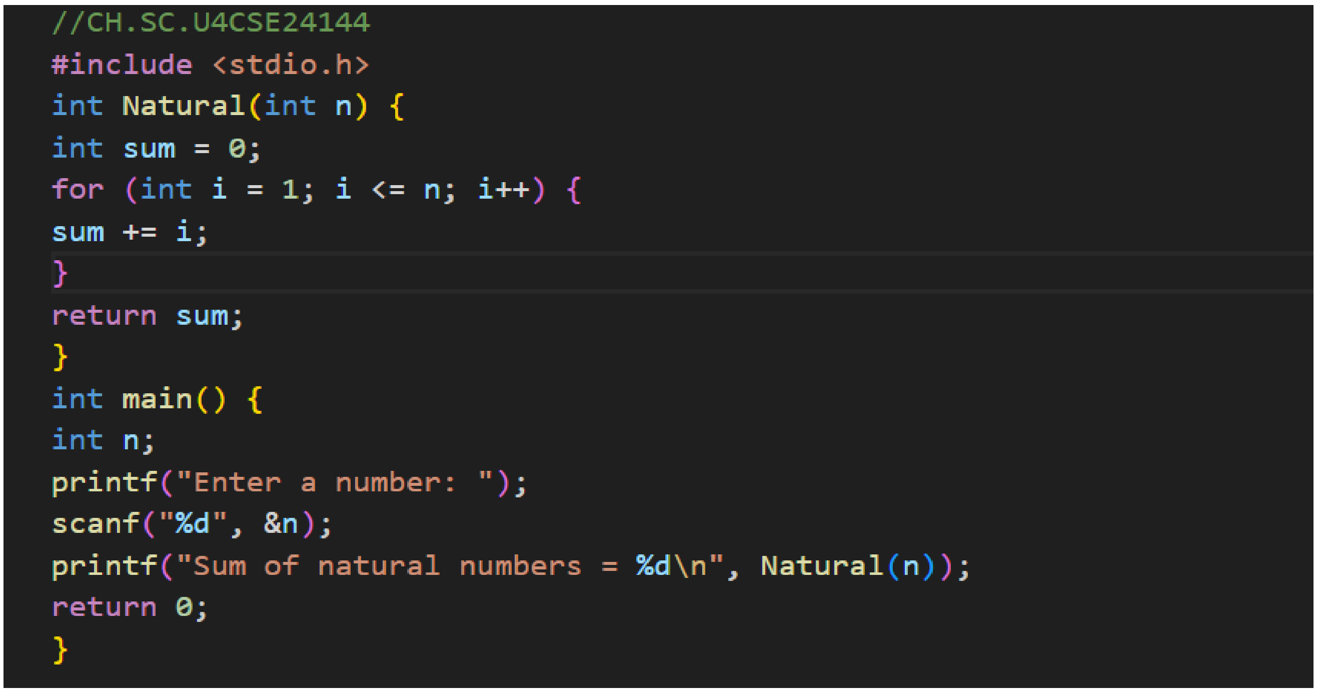
Roll No : ch.sc.u4cse24144

Department: CSE-B

Academic Year : 2024-2028

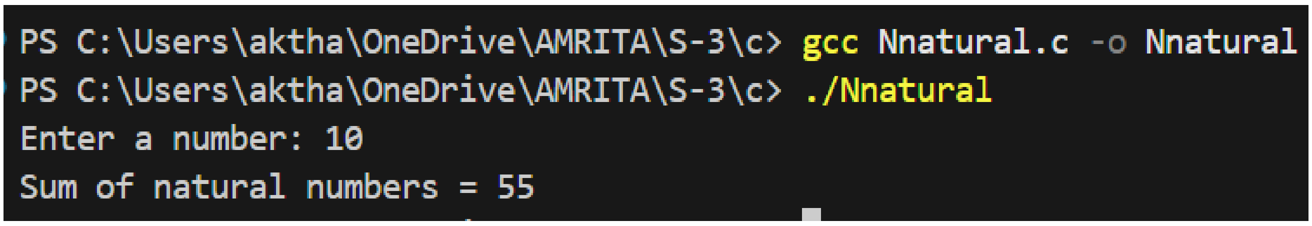
1) Write a code to find the Sum of first n natural numbers with user defined function and its space complexity with justification.

Code:



```c
//CH.SC.U4CSE24144
#include <stdio.h>
int Natural(int n) {
int sum = 0;
for (int i = 1; i <= n; i++) {
sum += i;
}
return sum;
}
int main() {
int n;
printf("Enter a number: ");
scanf("%d", &n);
printf("Sum of natural numbers = %d\n", Natural(n));
return 0;
}
```

OUTPUT:



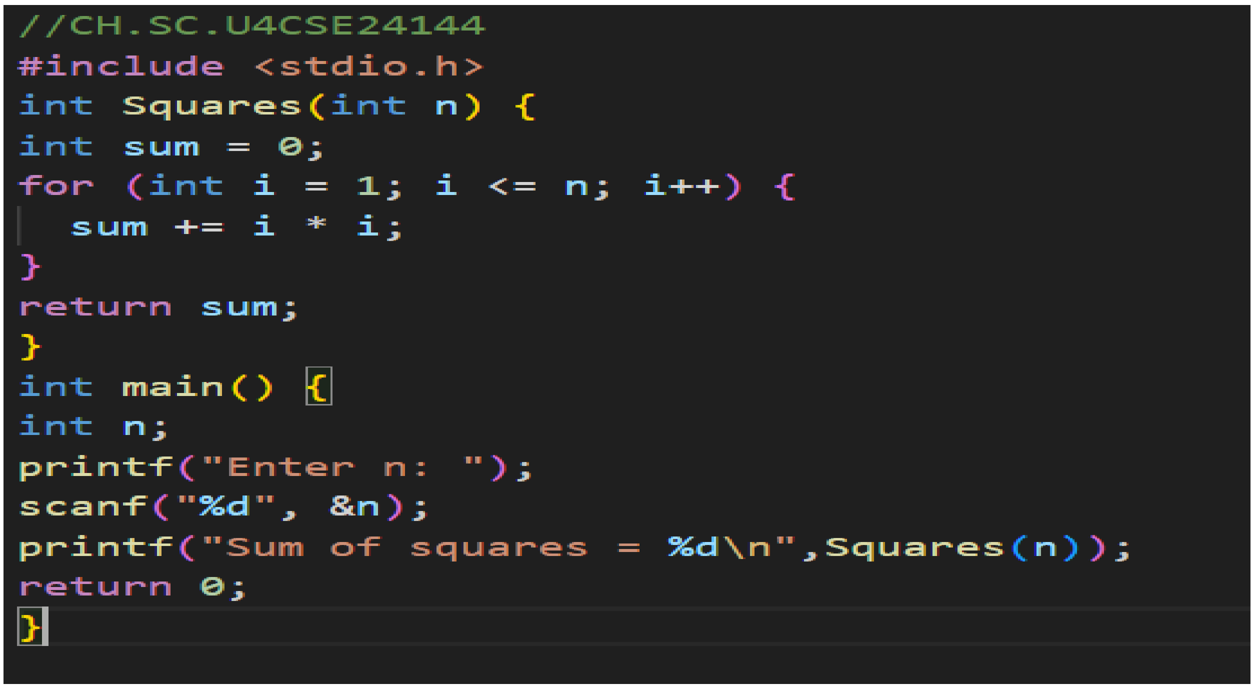
```
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> gcc Nnatural.c -o Nnatural
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> ./Nnatural
Enter a number: 10
Sum of natural numbers = 55
```

Space complexity & Justification:

Space Complexity: O(1)
The function Natural(n) uses only a fixed number of variables: i and sum. This amount of memory does not change based on the value of n.
The loop runs from 1 to n, but it does not create new memory during each iteration; it simply updates the same variables repeatedly.
There is no recursion, so the program does not keep multiple function calls in memory at the same time.
There is no dynamic memory allocation, such as arrays.
Even if n becomes twice as large or 100 times larger, the memory used remains exactly the same, which makes the space complexity O(1).

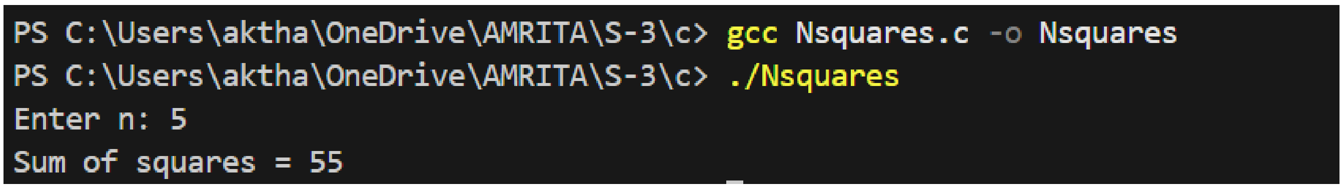
2) Write a code to find the Sum of squares of first n natural numbers and its space complexity with justification.

Code:



```c
//CH.SC.U4CSE24144
#include <stdio.h>
int Squares(int n) {
int sum = 0;
for (int i = 1; i <= n; i++) {
    sum += i * i;
}
return sum;
}
int main() {
int n;
printf("Enter n: ");
scanf("%d", &n);
printf("Sum of squares = %d\n",Squares(n));
return 0;
}
```

OUTPUT:



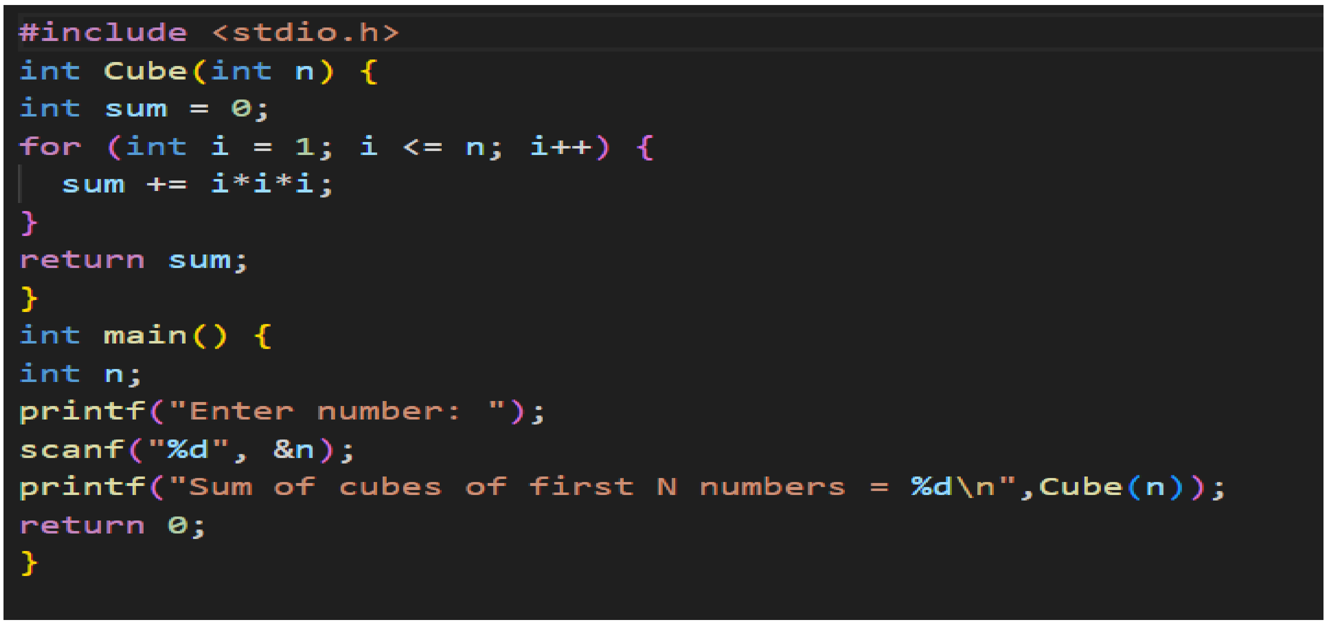
```
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> gcc Nsquares.c -o Nsquares
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> ./Nsquares
Enter n: 5
Sum of squares = 55
```

Space complexity & Justification:

Space Complexity: O(1)
The only variables used by the function Squares(n) are i, sum, and the parameter n.
No more memory is created as n increases because the same variables are used for each iteration, even if the loop runs from 1 to n.
There is only one instance of the function in memory at any given time because there is no recursion; call stack growth does not occur.
No list, array, or dynamic memory whose size is dependent on n exists.
Whether n is 10, 100, or 10,000, the program uses the same amount of memory.

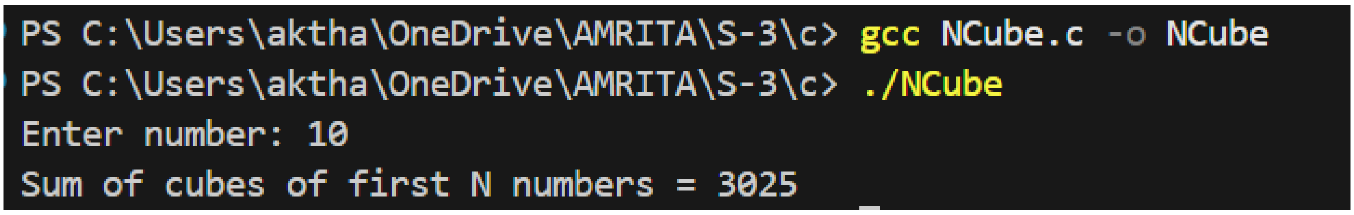
3) Write a code to find the Sum of Cubes of first n natural numbers and its space complexity with justification.

Code:



```c
#include <stdio.h>
int Cube(int n) {
int sum = 0;
for (int i = 1; i <= n; i++) {
    sum += i*i*i;
}
return sum;
}
int main() {
int n;
printf("Enter number: ");
scanf("%d", &n);
printf("Sum of cubes of first N numbers = %d\n",Cube(n));
return 0;
}
```

OUTPUT:



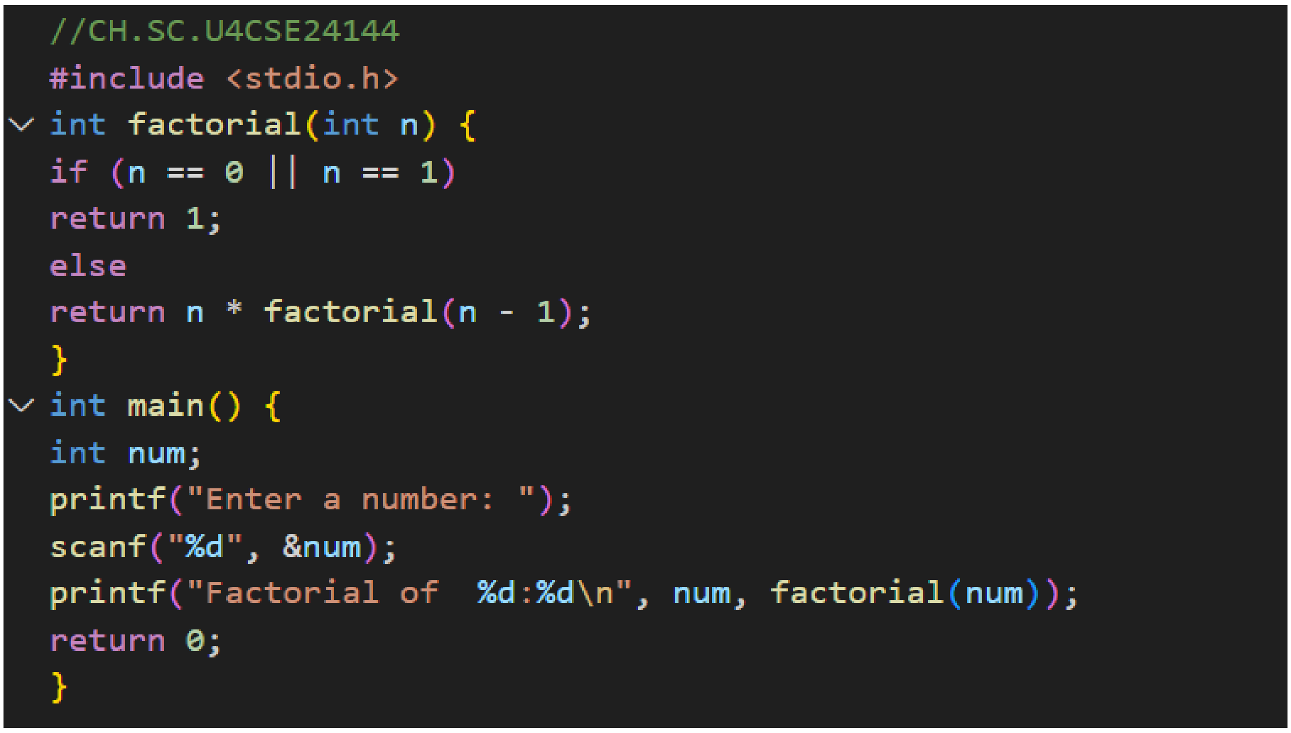
```
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> gcc NCube.c -o NCube
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> ./NCube
Enter number: 10
Sum of cubes of first N numbers = 3025
```

Space complexity & Justification:

Space Complexity: O(1)
Only three variables are used by the function Cube(n): the input n, sum, and i.
The loop reuses the same variables each time, even though it iterates from 1 to n without allocating additional memory.
There is only one function frame in memory since there is no recursion; stack growth does not occur.
No lists, arrays, or dynamic memory whose size varies with n are created by the program.
The space complexity is O(1) since the amount of memory used does not rise as the input does.

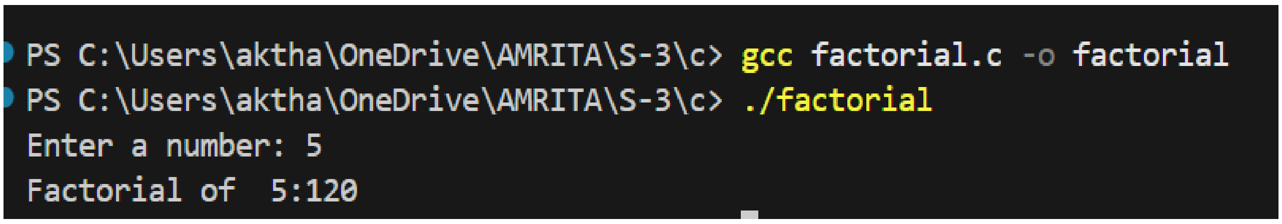
4) Finding the factorial of a number using recursive function and its space complexity with justification.

Code:



```c
//CH.SC.U4CSE24144
#include <stdio.h>
int factorial(int n) {
if (n == 0 || n == 1)
return 1;
else
return n * factorial(n - 1);
}
int main() {
int num;
printf("Enter a number: ");
scanf("%d", &num);
printf("Factorial of  %d:%d\n", num, factorial(num));
return 0;
}
```

OUTPUT:



```
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> gcc factorial.c -o factorial
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> ./factorial
Enter a number: 5
Factorial of  5:120
```

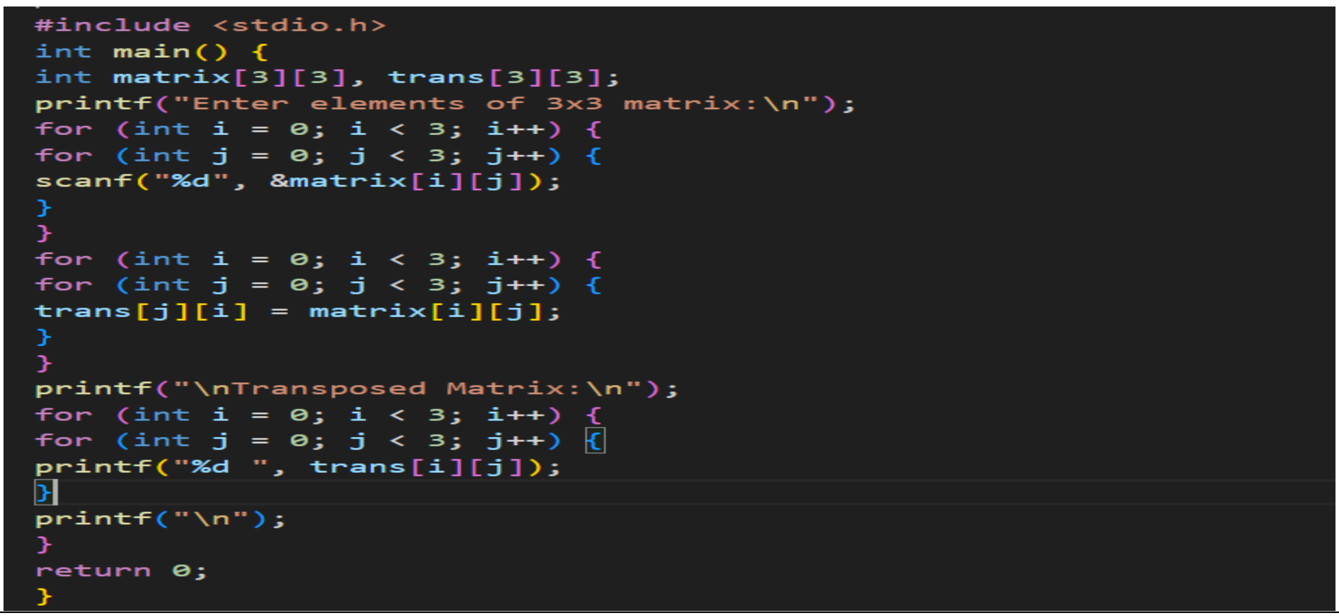
Space complexity & Justification:

The space complexity is O(n) because the function keeps calling itself with smaller values until it reaches 1, and each call needs a small amount of memory to hold its own information. None of the calls finish until the very last call completes, so all the earlier calls remain active at the same time. As the input n becomes larger, the number of active calls also becomes larger in the same way, which makes the total memory used grow directly with n.

If n doubles, the memory used also doubles; if n becomes 10 times bigger, the memory used becomes 10 times bigger, leading to O(n) space complexity.

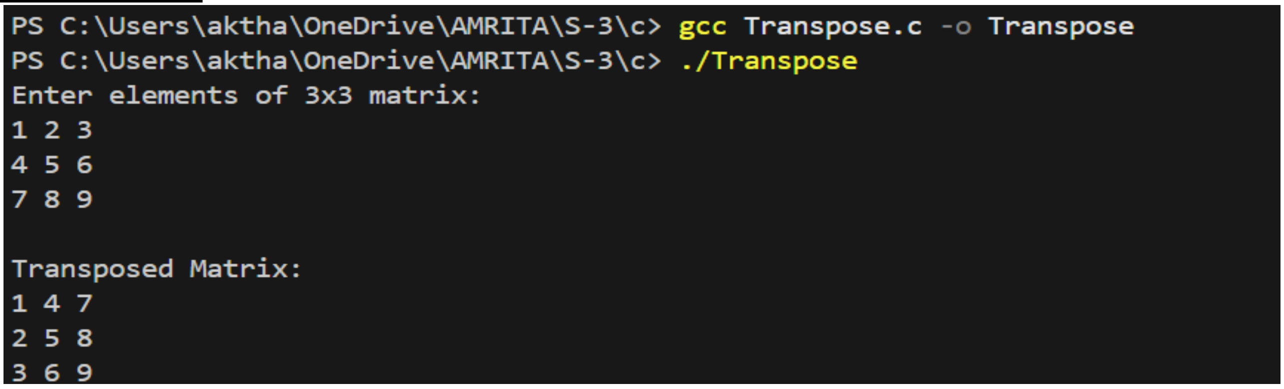
5) Write a program to Transpose a 3X3 Matrix and find its space complexity with justification.

Code:



```
#include <stdio.h>
int main() {
int matrix[3][3], trans[3][3];
printf("Enter elements of 3x3 matrix:\n");
for (int i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++) {
scanf("%d", &matrix[i][j]);
}
}
for (int i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++) {
trans[j][i] = matrix[i][j];
}
}
printf("\nTransposed Matrix:\n");
for (int i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++) {
printf("%d ", trans[i][j]);
}
printf("\n");
}
return 0;
}
```

OUTPUT:



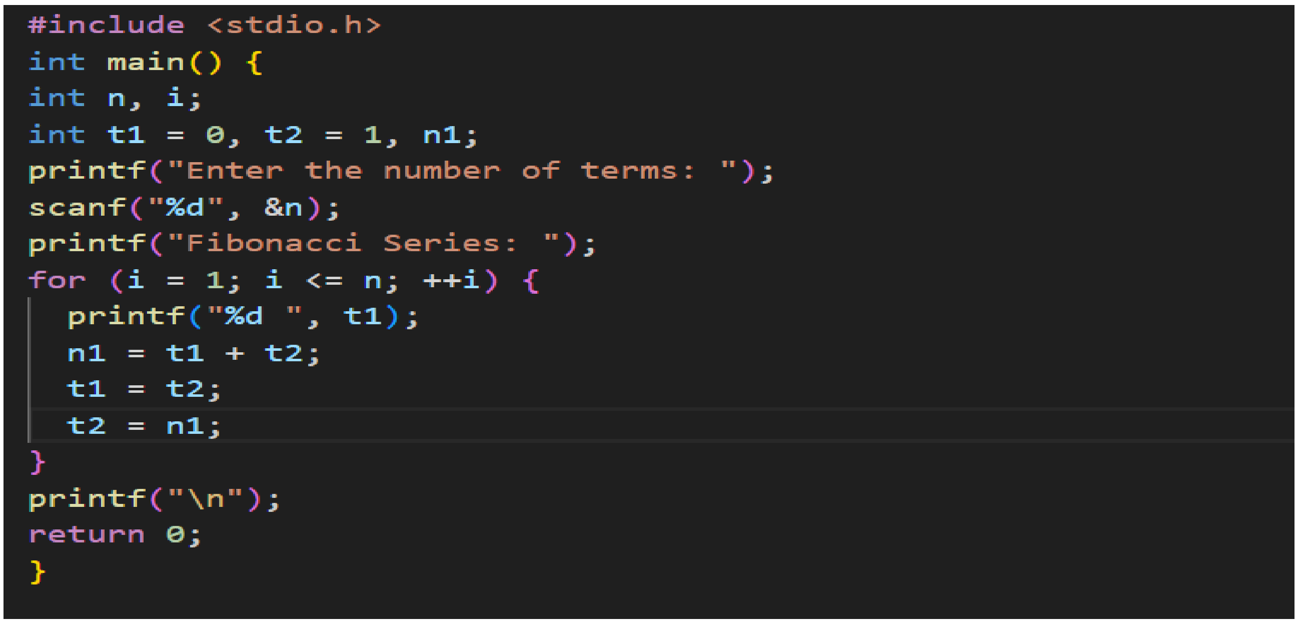
```
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> gcc Transpose.c -o Transpose
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> ./Transpose
Enter elements of 3x3 matrix:
1 2 3
4 5 6
7 8 9

Transposed Matrix:
1 4 7
2 5 8
3 6 9
```

Space complexity & Justification:

Space Complexity O(1)
Two fixed-size 3x3 arrays (matrix and trans) are used in the program, and they always take up the same amount of memory.
Memory usage is constant because user input has no effect on the size of these arrays.
There is no additional growing space because the loop variables (i, j) are reused.
Memory does not grow during execution because there is neither dynamic memory allocation nor recursion.
The space complexity is O(1) because the total amount of memory used is constant regardless of the input values.

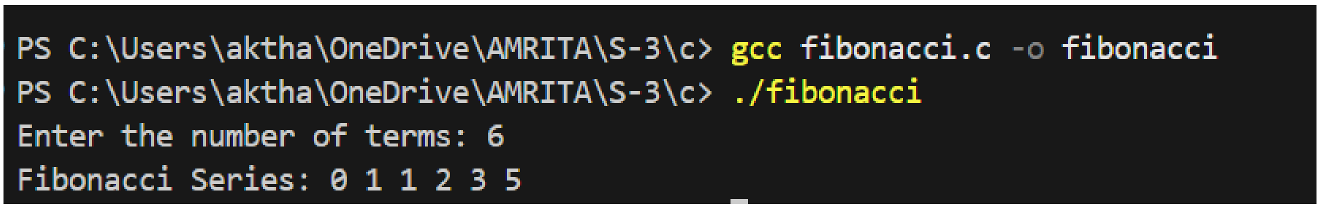
6) Finding the Fibonacci series and its space complexity with justification.

Code:



```c
#include <stdio.h>
int main() {
int n, i;
int t1 = 0, t2 = 1, n1;
printf("Enter the number of terms: ");
scanf("%d", &n);
printf("Fibonacci Series: ");
for (i = 1; i <= n; ++i) {
  printf("%d ", t1);
  n1 = t1 + t2;
  t1 = t2;
  t2 = n1;
}
printf("\n");
return 0;
}
```

OUTPUT:



```
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> gcc fibonacci.c -o fibonacci
PS C:\Users\aktha\OneDrive\AMRITA\S-3\c> ./fibonacci
Enter the number of terms: 6
Fibonacci Series: 0 1 1 2 3 5
```

Justification:

Space Complexity: O(1)
Only a set number of variables (t1, t2, n1, i, and n) are used by the program; this count remains constant regardless of the value of n.
Despite running n times, the loop only modifies the same variables each time; no new memory is created.
There is no use of dynamic memory, arrays, or lists that expand according to n.
Multiple function calls are not stored on the stack because the program does not use recursion.
The space complexity is O(1) since the amount of memory used is constant regardless of the size of the input.