AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956

SCHOOL OF COMPUTING

Design & Analysis of Algorithm

CSE211

Date: 27/11/2025

Week-1

CH.SC.U4CSE24136

P.Geetesh

1. Write a program to find sum of first n natural numbers using user defined function.

Code:

```c
//CH.SC.U4CSE24136
#include <stdio.h>
int sum_n(int n) {
    int s = 0;
    for(int i = 1; i <= n; i++) {
        s += i;
    }
    return s;
}
int main() {
    int n;
    printf("Enter n :");
    scanf("%d", &n);
    printf("%d\n", sum_n(n));
    printf("CH.SC.U4CSE24136\n");
    return 0;
}
```

Output:

```
root@Ubuntu:/home/geetesh# gcc sum_n.c -o sum.n
root@Ubuntu:/home/geetesh# ./sum_n
Enter n :5
15
CH.SC.U4CSE24136
```

Space Complexity and it's justification:

- Space Complexity: **O(1)** because only a fixed number of variables are used.
- In main(), there is only **one int variable** (n).
- In sum_n(), only **two int variables** (s and i) are used.
- No arrays or dynamic memory → space stays constant.

2. Write a program to find sum of squares of first n natural numbers

Code:

```c
//CH.SC.U4CSE24136
#include <stdio.h>
int sum_sq(int n) {
    int s = 0;
    for(int i = 1; i <= n; i++) {
        s += i*i;
    }
    return s;
}
int main() {
    int n;
    printf("Enter n :");
    scanf("%d", &n);
    printf("%d\n", sum_sq(n));
    printf("CH.SC.U4CSE24136\n");
    return 0;
}
```

Output:

```
root@Ubuntu:/home/geetesh# gcc sum_sq.c -o sum_sq
root@Ubuntu:/home/geetesh# ./sum_sq
Enter n :4
30
CH.SC.U4CSE24136
```

Space Complexity and it's justification:

- Space Complexity: O(1)
- main() uses only one int (n).
- sum_sq() uses two ints (s, i).
- No arrays → constant memory.

3. Write a program to find sum of cubes of first n natural numbers

Code:

```c
//CH.SC.U4CSE24136
#include <stdio.h>
int sum_cub(int n) {
    int s = 0;
    for(int i = 1; i <= n; i++) {
        s += i*i*i;
    }
    return s;
}
int main() {
    int n;
    printf("Enter n :");
    scanf("%d", &n);
    printf("%d\n", sum_cub(n));
    printf("CH.SC.U4CSE24136\n");
    return 0;
}
```

Output:

```
root@Ubuntu:/home/geetesh# gcc sum_cub.c -o sum_cub
root@Ubuntu:/home/geetesh# ./sum_cub
Enter n :3
36
CH.SC.U4CSE24136
```

Space Complexity and it's justification:

- Space Complexity: O(1)
- Only one int in main().
- Only two or three ints inside the function.
- Memory stays constant.

4. Write a program to find factorial of a given integer using recursion.

Code:

```c
//CH.SC.U4CSE24136
#include <stdio.h>
int factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}
int main() {
    int n;
    printf("Enter n :");
    scanf("%d", &n);
    printf("%d\n", factorial(n));
    printf("CH.SC.U4CSE24136\n");
    return 0;
}
```

Output:

```
root@Ubuntu:/home/geetesh# gcc fact.c -o fact
root@Ubuntu:/home/geetesh# ./fact
Enter n :6
720
CH.SC.U4CSE24136
```

Space Complexity and it's justification:

- ✓ Space Complexity: O(n)
- ✓ Each recursive call adds one frame to the stack.
- ✓ Maximum depth is n.
- ✓ So space grows linearly with input.

5. Write a program to transpose a 3x3 matrix.

Code:

```c
//CH.SC.U4CSE24136
#include <stdio.h>
int main() {
    int a[3][3], t[3][3], i, j;
    printf("Enter the elements in the matrix :\n");
    for (i = 0; i < 3; i++){
        for (j = 0; j < 3; j++){
            scanf("%d", &a[i][j]);
            }
           }
    for (i = 0; i < 3; i++){
        for (j = 0; j < 3; j++){
            t[j][i] = a[i][j];
            }
           }
    printf("Matrix after transpose:\n");
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++){
            printf("%d ", t[i][j]);
            }
        printf("\n");
    }
    printf("CH.SC.U4CSE24136\n");
    return 0;
}
```

Output:

```
root@Ubuntu:/home/geetesh# gcc transp_matr.c -o transp_matr
root@Ubuntu:/home/geetesh# ./transp_matr
Enter the elements in the matrix :
1 2 3
4 5 6
7 8 9
Matrix after transpose:
1 4 7
2 5 8
3 6 9
CH.SC.U4CSE24136
```

Space Complexity and it's justification:

- Space Complexity: O(1) (because matrix size is fixed 3×3)
- Uses two fixed 3×3 arrays (a and t).
- Total memory stays constant (not dependent on input).
- No dynamic allocation.

6. Write a program to find fibonacci series

Code:

```c
//CH.SC.U4CSE24136
#include <stdio.h>
int main() {
    int n, a = 0, b = 1, c, i;
    printf("Enter n : ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("%d ", a);
        c = a + b;
        a = b;
        b = c;
    }
    printf("\nCH.SC.U4CSE24136\n");
    return 0;
}
```

Output:

```
root@Ubuntu:/home/geetesh# gcc fib.c -o fib
root@Ubuntu:/home/geetesh# ./fib
Enter n : 8
0 1 1 2 3 5 8 13
CH.SC.U4CSE24136
```

Space Complexity and it's justification:

- Space Complexity: O(1)
- Uses only a few ints (a, b, c, i, n).
- No arrays or recursion.
- Memory fixed ⇒ constant space.