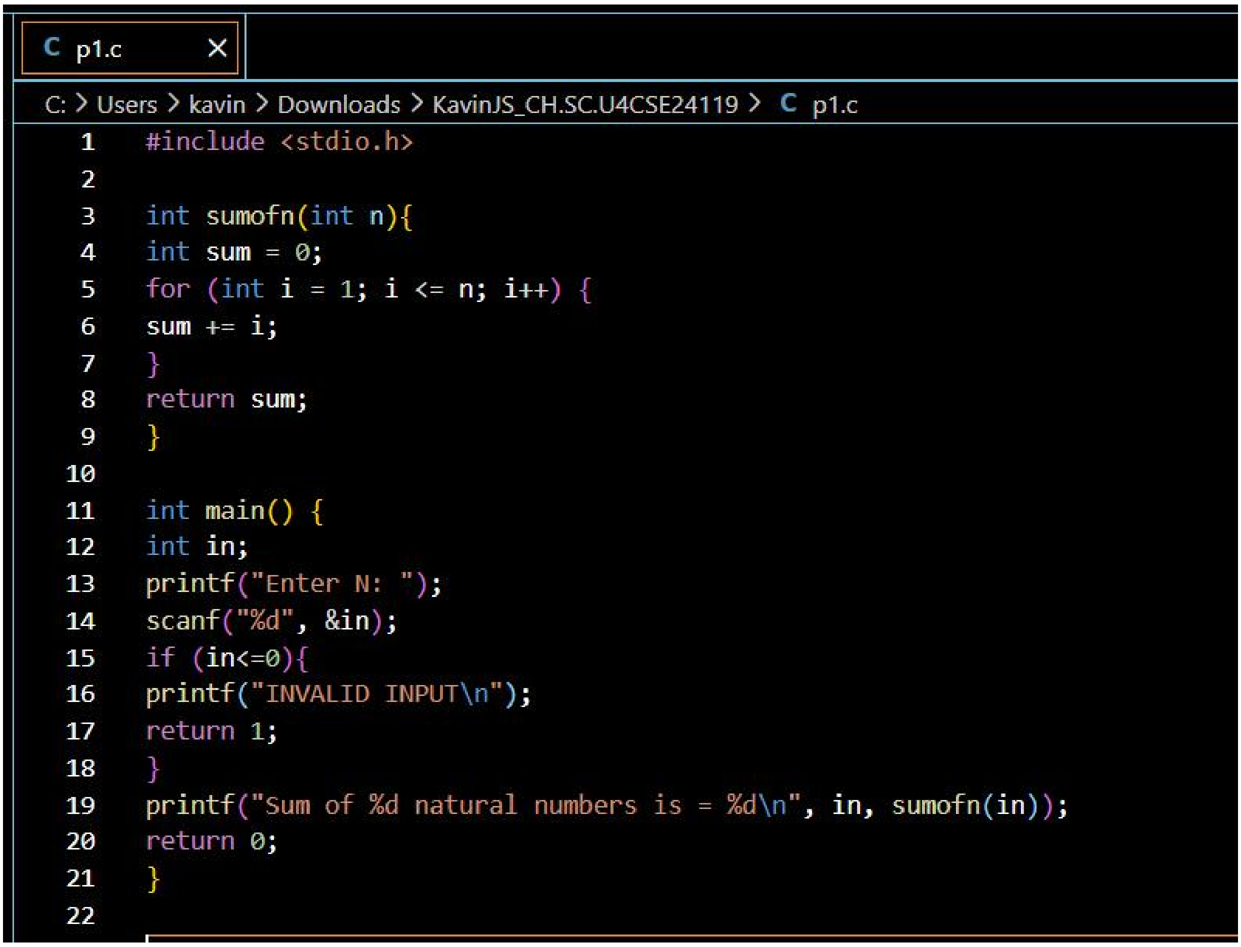
WEEK - 1

Kavin.J.S
CH.SC.U4CSE24119

Program 1:

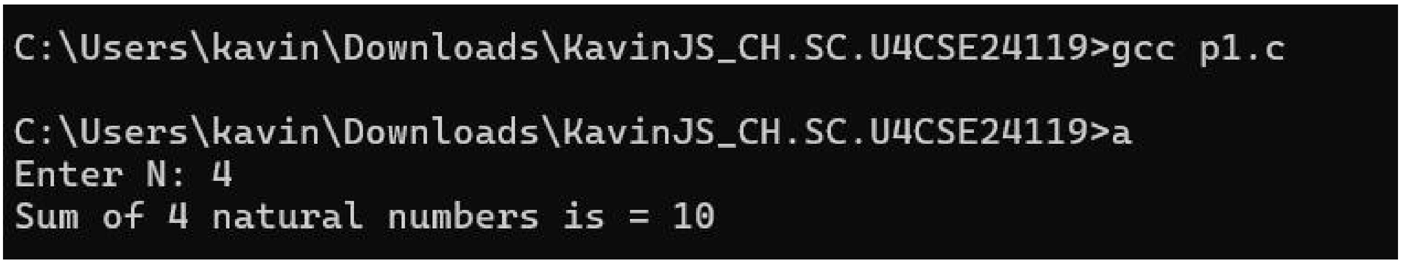
Code:



C p1.c ×
C: > Users > kavin > Downloads > KavinJS_CH.SC.U4CSE24119 > C p1.c

```c
#include <stdio.h>

int sumofn(int n){
int sum = 0;
for (int i = 1; i <= n; i++) {
sum += i;
}
return sum;
}

int main() {
int in;
printf("Enter N: ");
scanf("%d", &in);
if (in<=0){
printf("INVALID INPUT\n");
return 1;
}
printf("Sum of %d natural numbers is = %d\n", in, sumofn(in));
return 0;
}
```

Output:



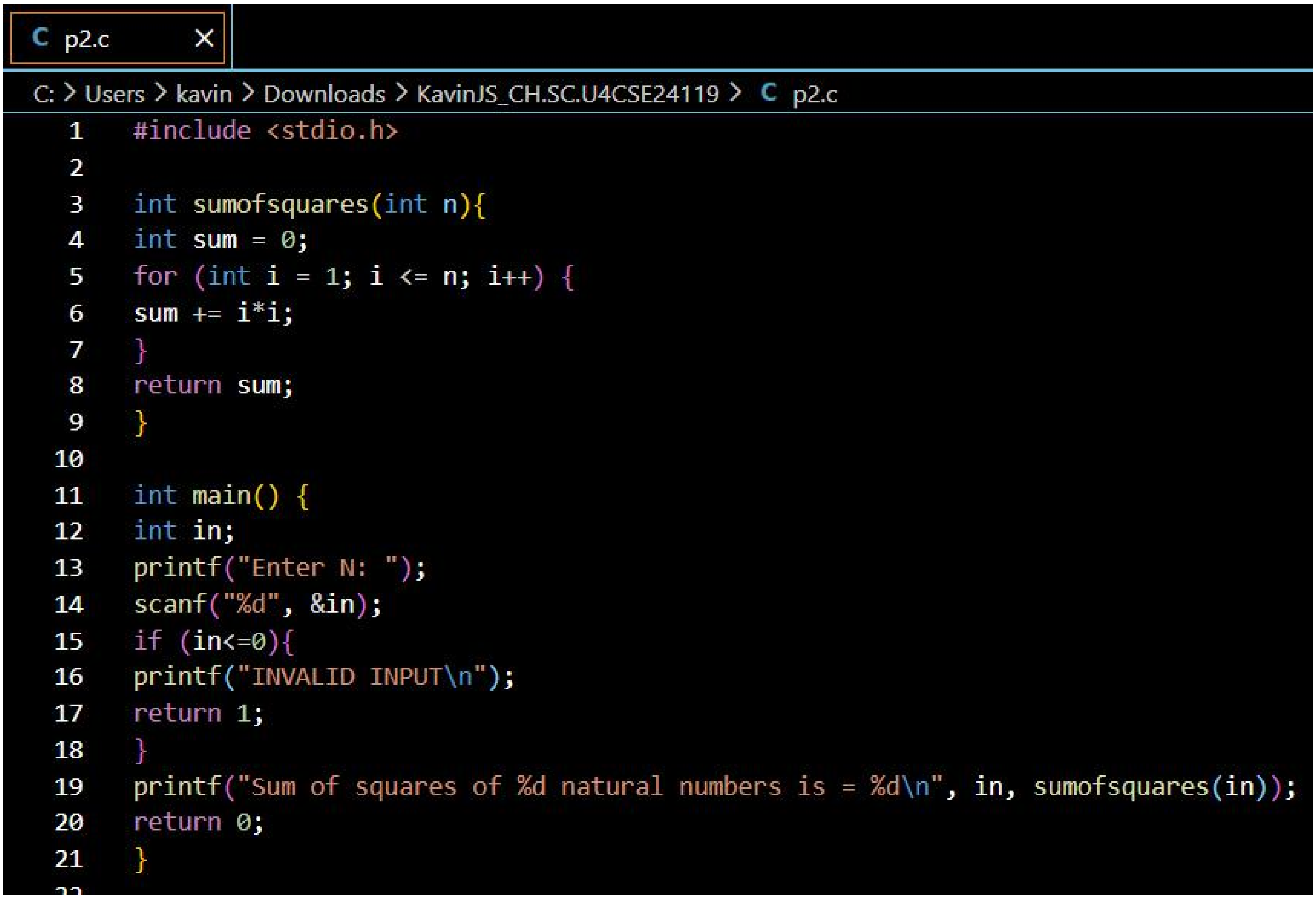
```
C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>gcc p1.c

C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>a
Enter N: 4
Sum of 4 natural numbers is = 10
```

Space Complexity & Justification:

O(1), The program only uses a few integer variables: sum, i, and n (along with in in main). Since the number of variables does not increase with the input size, the space needed stays constant.

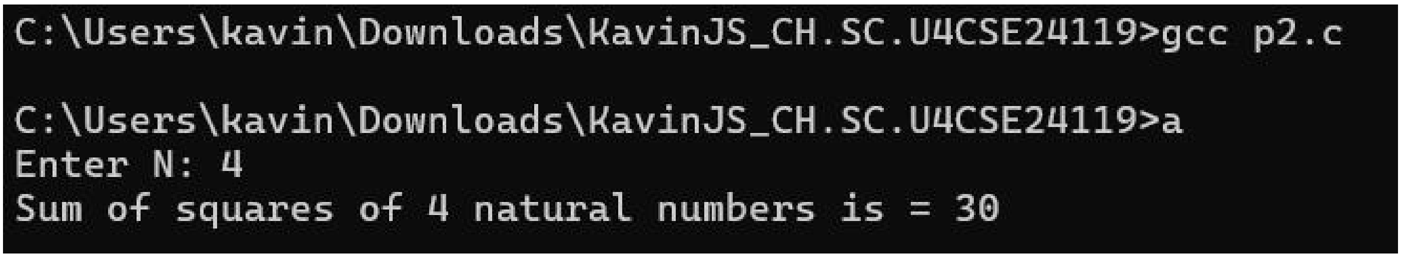
Program 2:

Code:


p2.c
C: > Users > kavin > Downloads > KavinJS_CH.SC.U4CSE24119 > p2.c

```c
#include <stdio.h>

int sumofsquares(int n){
int sum = 0;
for (int i = 1; i <= n; i++) {
sum += i*i;
}
return sum;
}

int main() {
int in;
printf("Enter N: ");
scanf("%d", &in);
if (in<=0){
printf("INVALID INPUT\n");
return 1;
}
printf("Sum of squares of %d natural numbers is = %d\n", in, sumofsquares(in));
return 0;
}
```

Output:



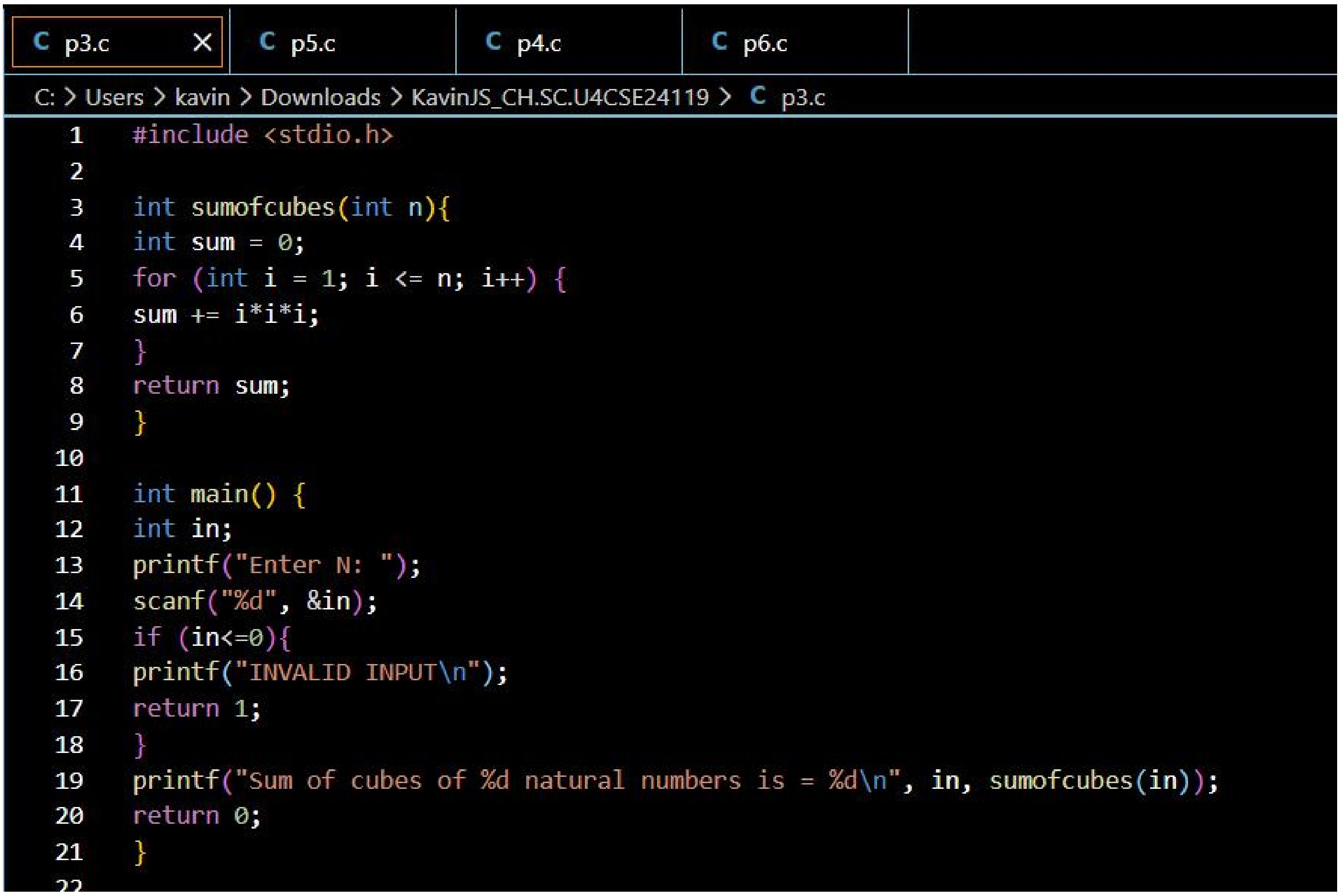
```
C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>gcc p2.c

C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>a
Enter N: 4
Sum of squares of 4 natural numbers is = 30
```

Space Complexity & Justification:

O(1), The program uses only a fixed set of integer variables: sum, i, n, and in. None of these grow with the input value, and no extra data structures are created. So the required space stays constant.

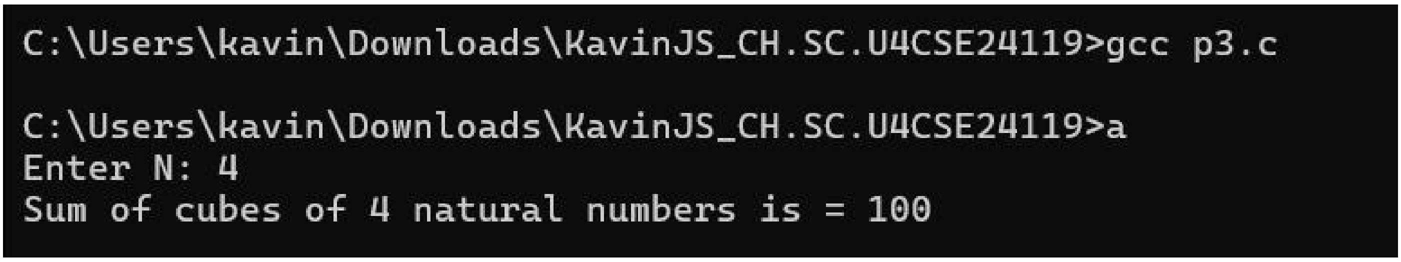
Program 3:

Code:


p3.c
p5.c
p4.c
p6.c
C: > Users > kavin > Downloads > KavinJS_CH.SC.U4CSE24119 > p3.c

```c
#include <stdio.h>

int sumofcubes(int n){
int sum = 0;
for (int i = 1; i <= n; i++) {
sum += i*i*i;
}
return sum;
}

int main() {
int in;
printf("Enter N: ");
scanf("%d", &in);
if (in<=0){
printf("INVALID INPUT\n");
return 1;
}
printf("Sum of cubes of %d natural numbers is = %d\n", in, sumofcubes(in));
return 0;
}
```

Output:



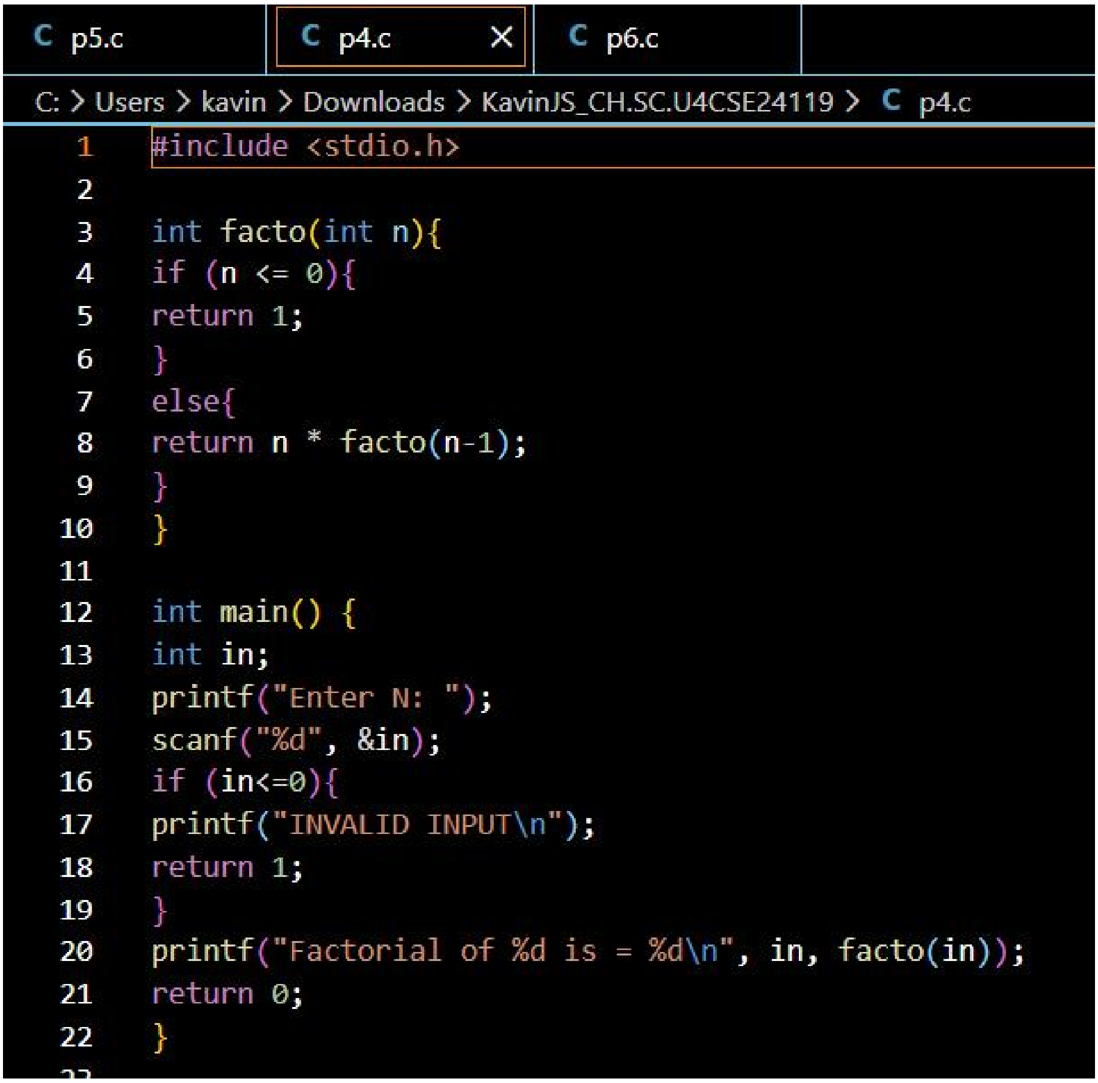
```
C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>gcc p3.c

C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>a
Enter N: 4
Sum of cubes of 4 natural numbers is = 100
```

Space Complexity & Justification:

O(1), The program uses only a fixed set of integer variables: sum, i, n, and in. None of these grow with the input value, and no extra data structures are created. So the required space stays constant.

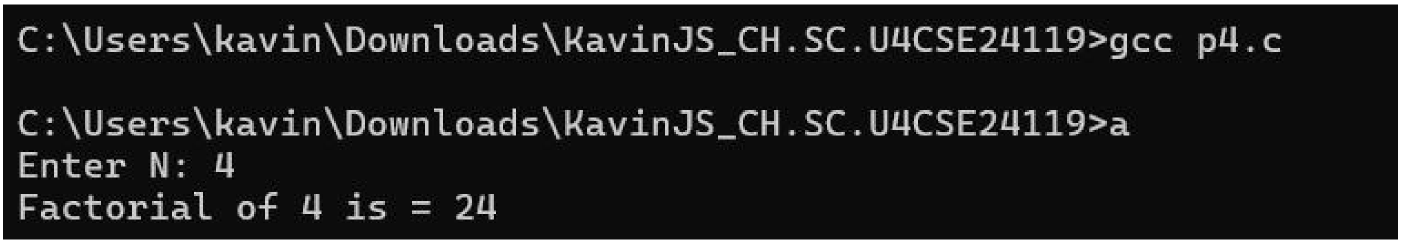
Program 4:

Code:


p5.c
p4.c
p6.c
C: > Users > kavin > Downloads > KavinJS_CH.SC.U4CSE24119 > p4.c

```c
#include <stdio.h>

int facto(int n){
if (n <= 0){
return 1;
}
else{
return n * facto(n-1);
}
}

int main() {
int in;
printf("Enter N: ");
scanf("%d", &in);
if (in<=0){
printf("INVALID INPUT\n");
return 1;
}
printf("Factorial of %d is = %d\n", in, facto(in));
return 0;
}
```

Output:



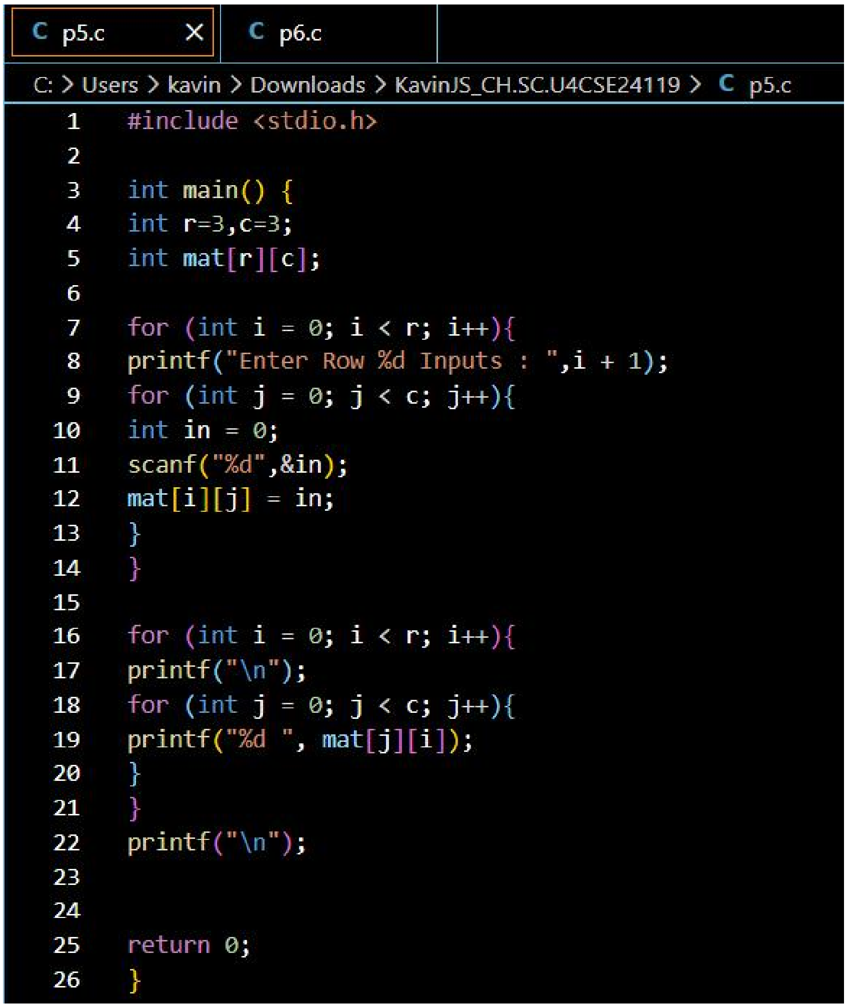
```
C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>gcc p4.c

C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>a
Enter N: 4
Factorial of 4 is = 24
```

Space Complexity & Justification:

O(n), The function facto uses recursion, so every call stores its own return address and local variable n on the call stack. For an input of n, the function makes n recursive calls before unwinding, so the stack depth grows linearly. Therefore, the space complexity is **O(n)**.

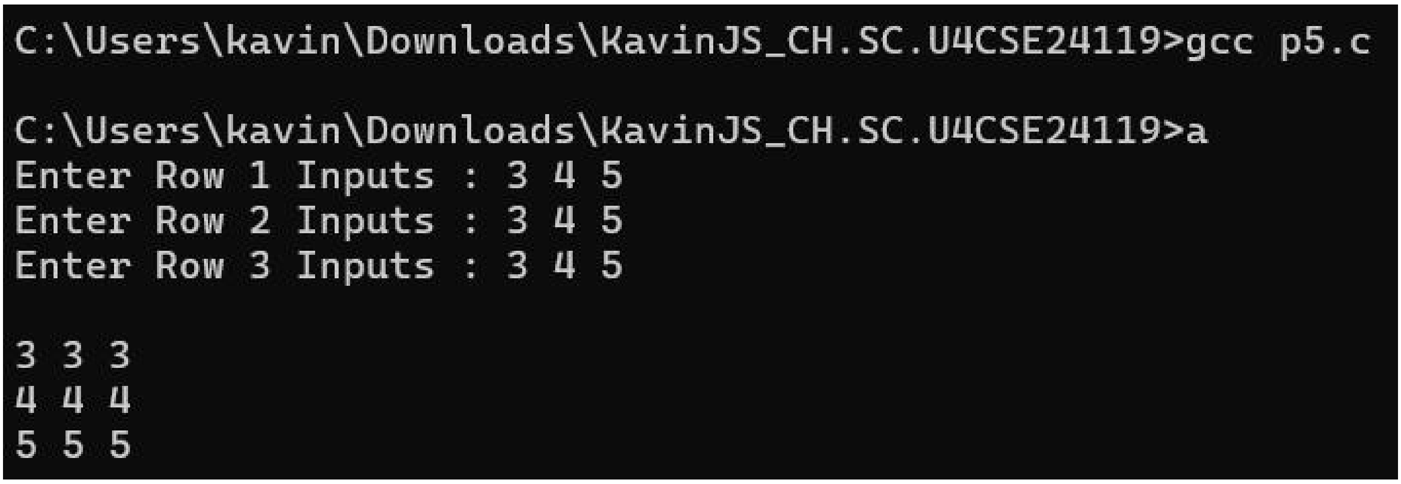
Program 5:

Code:


p5.c
p6.c
C: > Users > kavin > Downloads > KavinJS_CH.SC.U4CSE24119 > p5.c

```c
#include <stdio.h>

int main() {
int r=3,c=3;
int mat[r][c];

for (int i = 0; i < r; i++){
printf("Enter Row %d Inputs : ",i + 1);
for (int j = 0; j < c; j++){
int in = 0;
scanf("%d",&in);
mat[i][j] = in;
}
}

for (int i = 0; i < r; i++){
printf("\n");
for (int j = 0; j < c; j++){
printf("%d ", mat[j][i]);
}
}
printf("\n");


return 0;
}
```

Output:



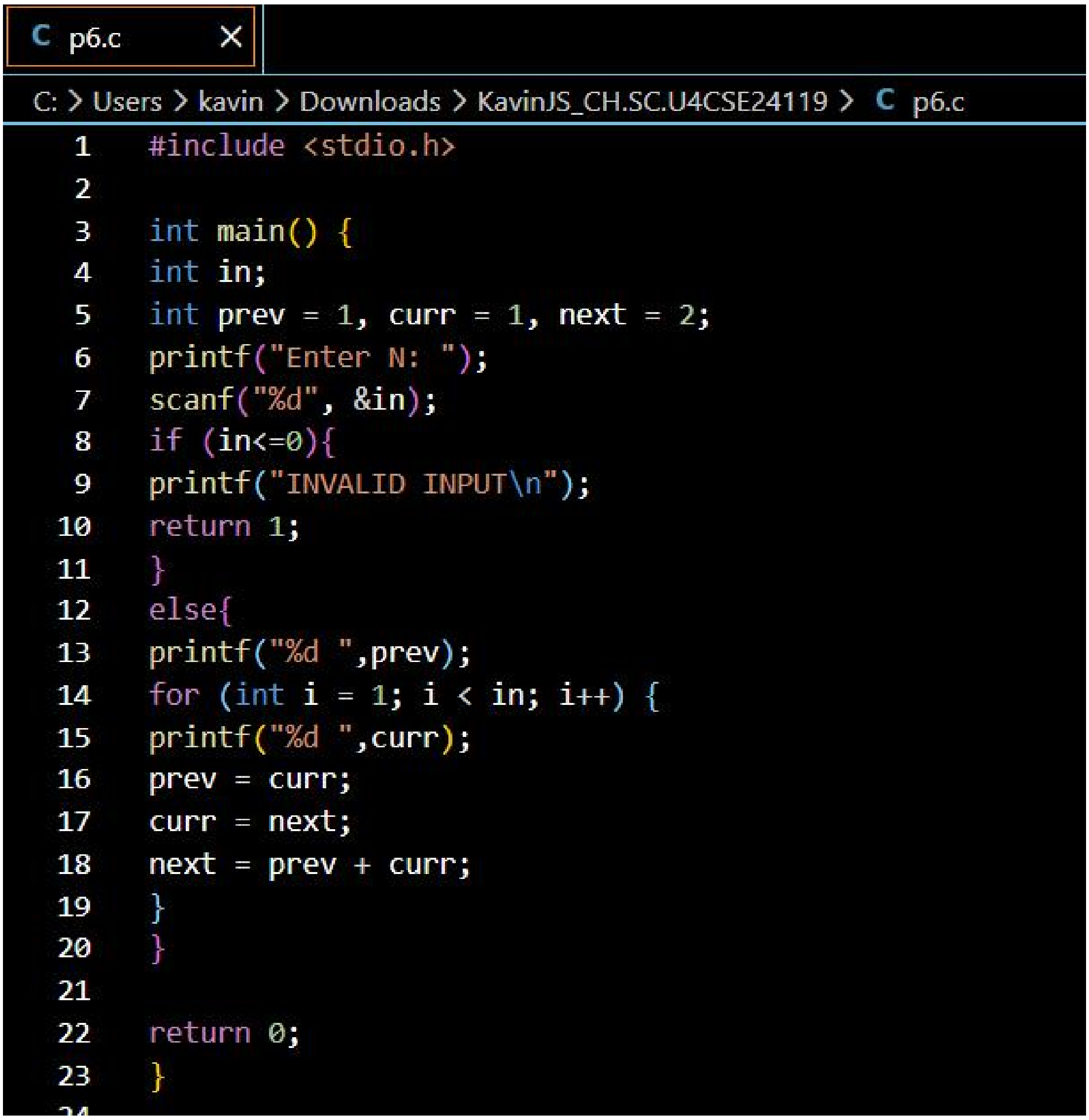
```
C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>gcc p5.c

C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>a
Enter Row 1 Inputs : 3 4 5
Enter Row 2 Inputs : 3 4 5
Enter Row 3 Inputs : 3 4 5

3 3 3
4 4 4
5 5 5
```

Space Complexity & Justification:

O(1), The program allocates a fixed 3×3 integer matrix mat[3][3] and uses a constant set of variables (r, c, i, j, in). The size of the matrix does not depend on user input, so the total memory used stays constant.

Program 6:

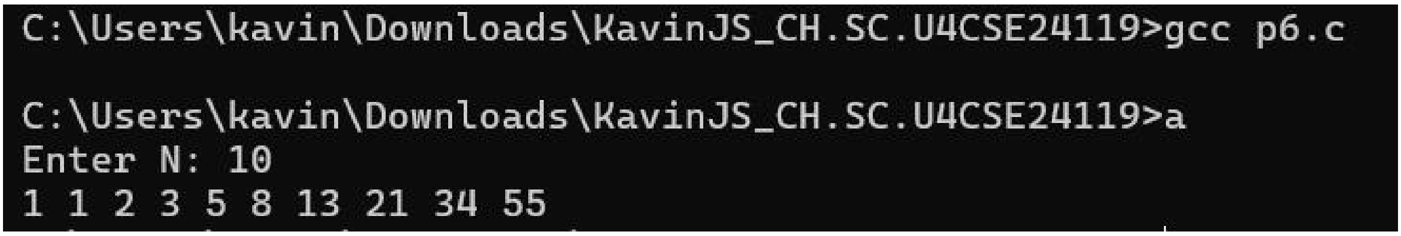
Code:



p6.c
C: > Users > kavin > Downloads > KavinJS_CH.SC.U4CSE24119 > p6.c

```c
#include <stdio.h>

int main() {
int in;
int prev = 1, curr = 1, next = 2;
printf("Enter N: ");
scanf("%d", &in);
if (in<=0){
printf("INVALID INPUT\n");
return 1;
}
else{
printf("%d ",prev);
for (int i = 1; i < in; i++) {
printf("%d ",curr);
prev = curr;
curr = next;
next = prev + curr;
}
}

return 0;
}
```

Output:



```
C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>gcc p6.c

C:\Users\kavin\Downloads\KavinJS_CH.SC.U4CSE24119>a
Enter N: 10
1 1 2 3 5 8 13 21 34 55
```

Space Complexity & Justification:

O(1), The program uses only a fixed set of integer variables: in, prev, curr, next, and the loop variable i. This number never increases with the value of *N*, so the memory usage stays constant.