

**University of Petroleum**

**&**

**Energy Studies**

**SCHOOL OF COMPUTER SCIENCE**

**Name**: Akshat Agarwal

**Course**: BTech CSE

**SAP ID**: 500118953

**BATCH**: 1

**PRESENTED TO**: Dr. Rahul Kumar Singh

**Semester**: 3

**GitHub Link**

**For all the Experiments**

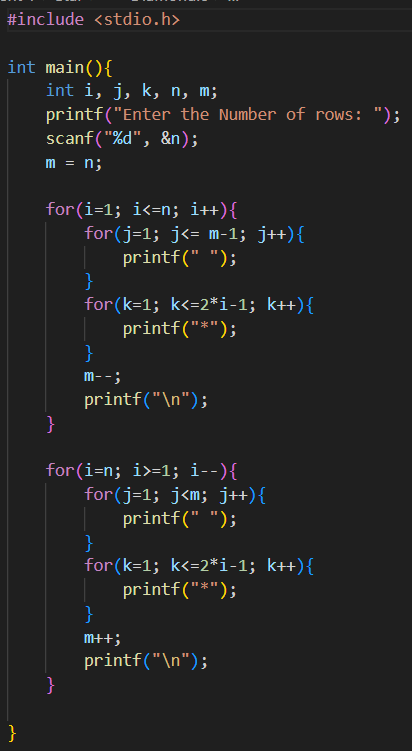
<https://github.com/Akshat29065/DAA_Lab.git>

**Experiment 1: Patterns**

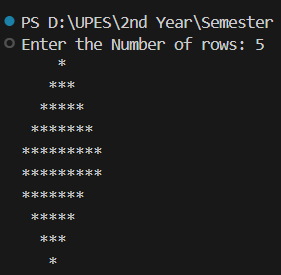
* **Star Patterns**

1. **Diamond:**

Input:

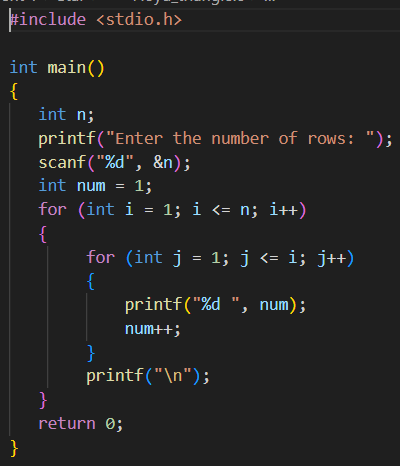


Output:

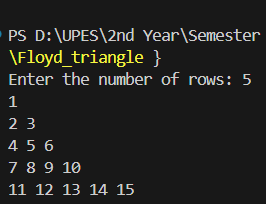


1. **Floyd’s Triangle:**

Input:

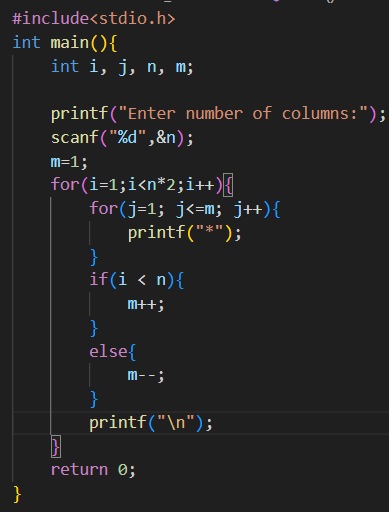


Output:

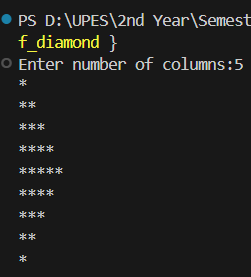


1. **Half Diamond:**

Input:

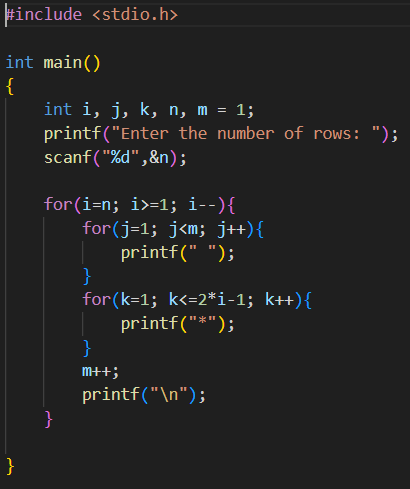


Output:

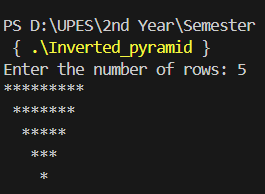


1. **Inverted Pyramid:**

Input:

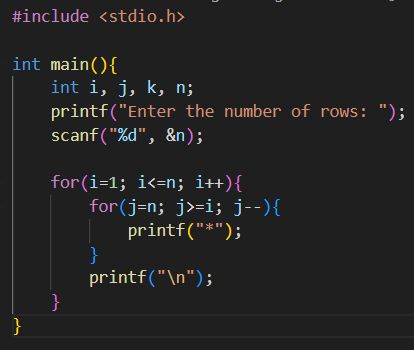


Output:

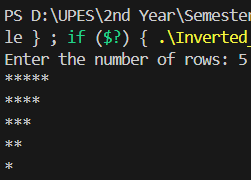


1. **Inverted Right Pyramid:**

Input:

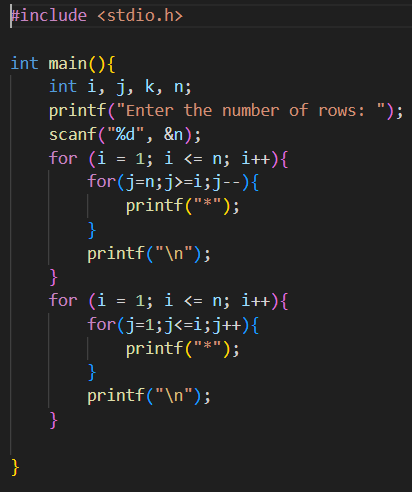


Output:

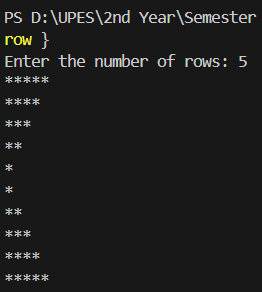


1. **Left Arrow:**

Input:

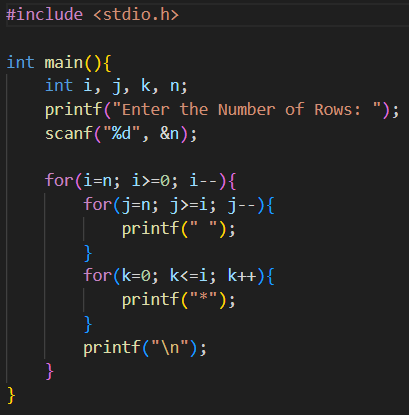


Output:

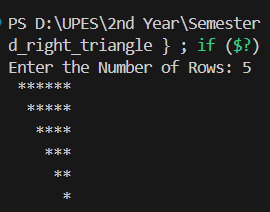


1. **Mirror Inverted Right Triangle:**

Input:

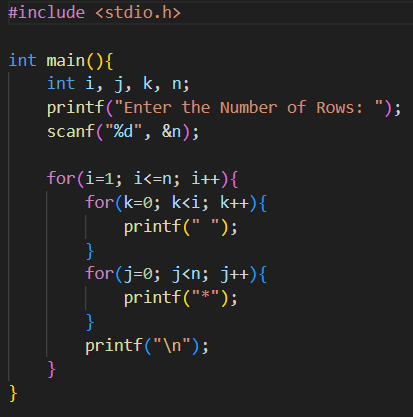


Output:

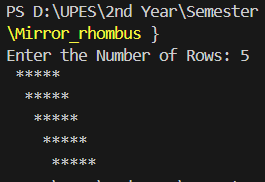


1. **Mirror Rhombus:**

Input:

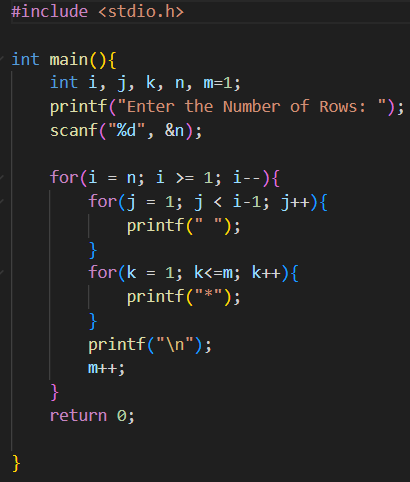


Output:

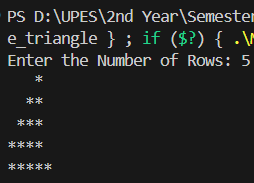


1. **Mirror Right Angle Triangle:**

Input:

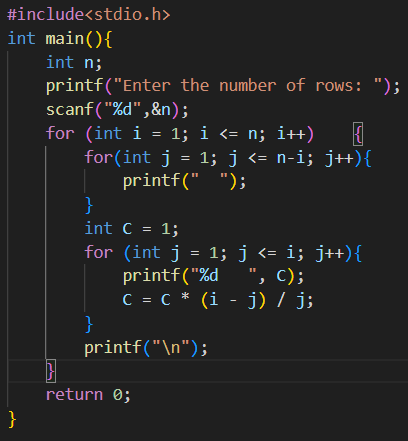


Output:

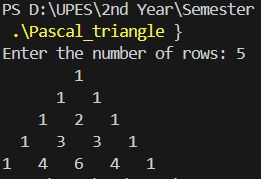


1. **Pascal’s Triangle:**

Input:

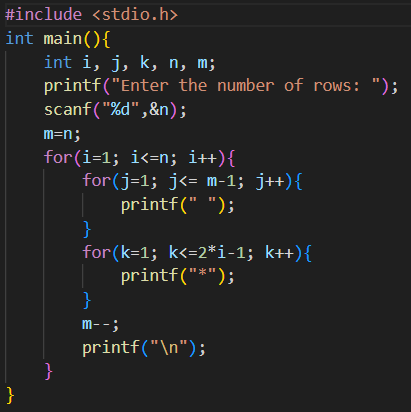


Output:

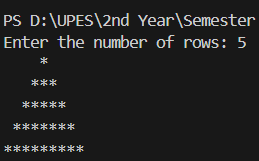


1. **Pyramid:**

Input:

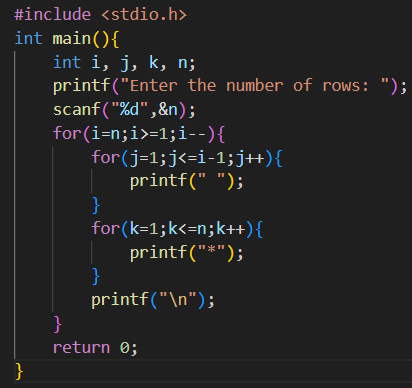


Output:

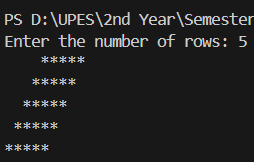


1. **Rhombus:**

Input:

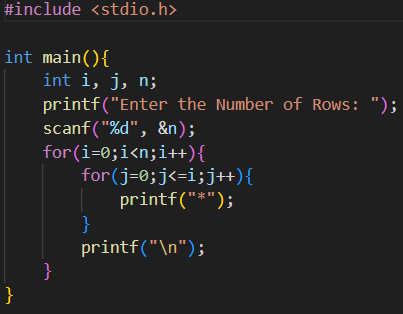


Output:

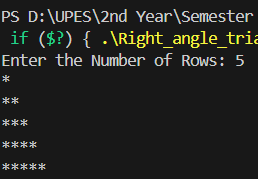


1. **Right Angle Triangle:**

Input:

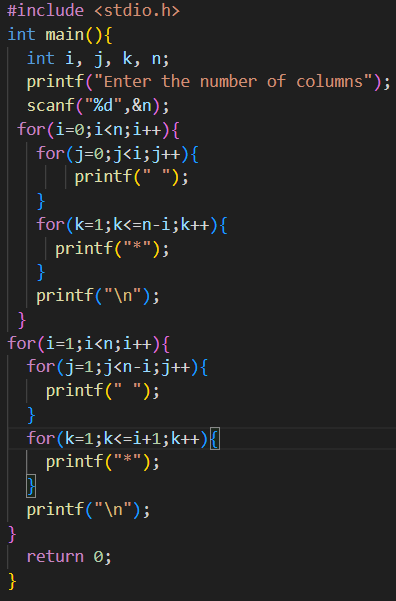


Output:

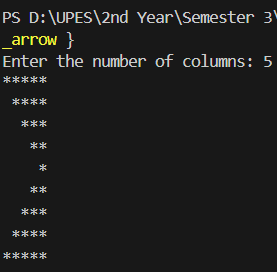


1. **Right Arrow:**

Input:

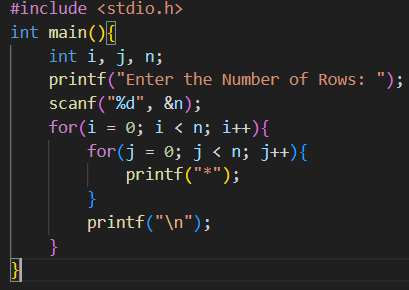


Output:

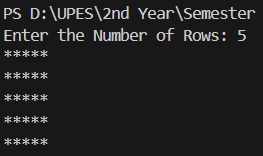


1. **Square:**

Input:



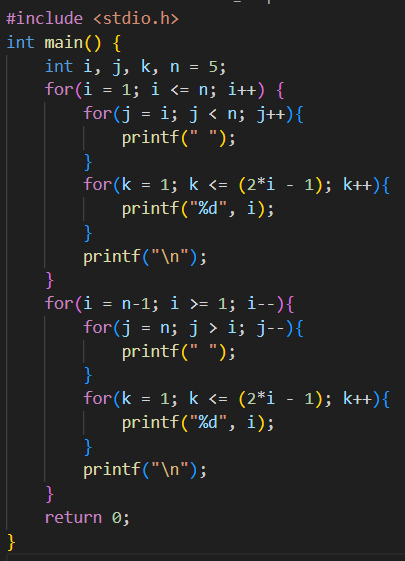
Output:



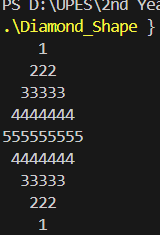
* **Number**

1. **Diamond Shape:**

Input:

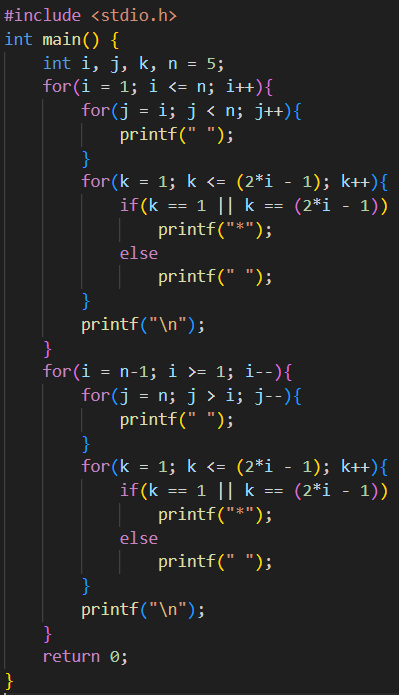


Output:



1. **Hollow Triangle:**

Input:

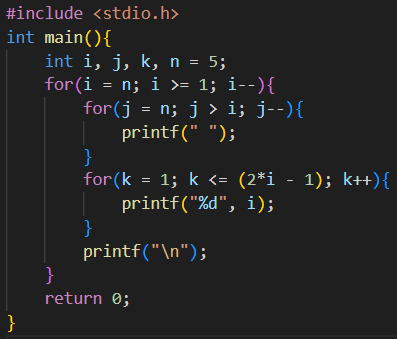


Output:

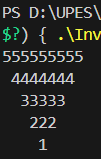


1. **Inverted Pyramid:**

Input:

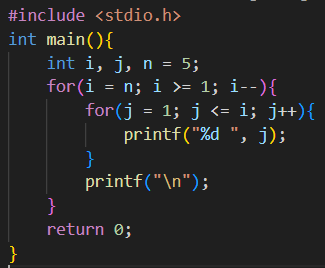


Output:

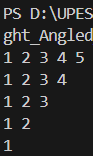


1. **Inverted Right Angle Pyramid:**

Input:

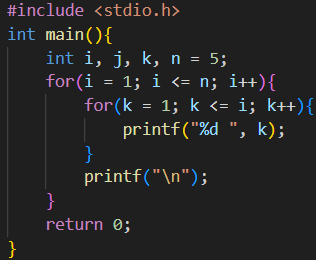


Output:

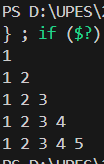


1. **Left Aligned Pyramid:**

Input:

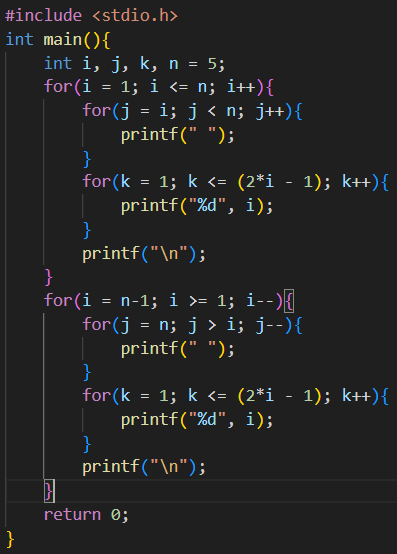


Output:

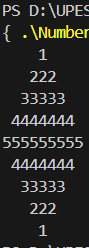


1. **Number Diamond:**

Input:

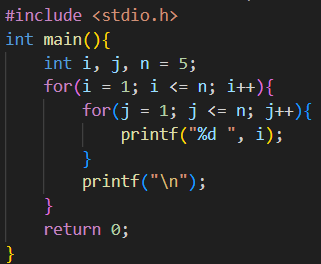


Output:

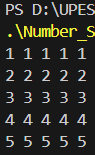


1. **Number Square:**

Input:

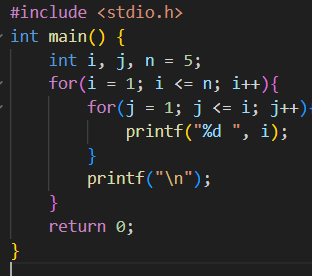


Output:

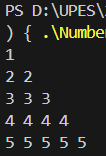


1. **Number Triangle:**

Input:

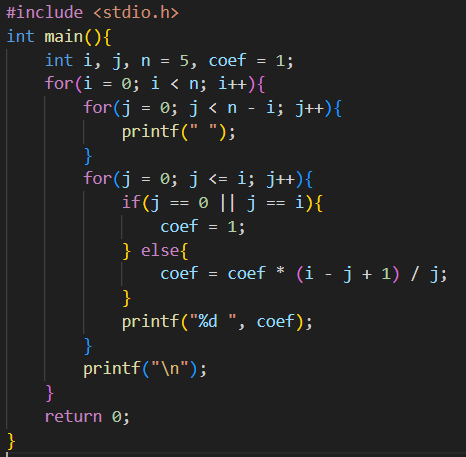


Output:

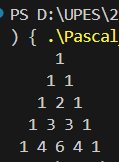


1. **Pascal’s Triangle:**

Input:

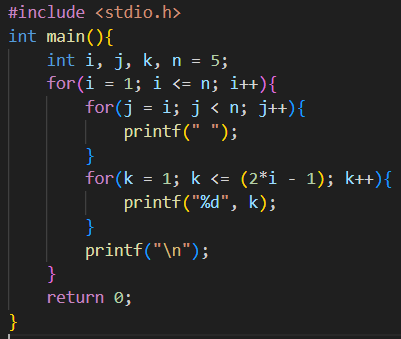
’

Output:

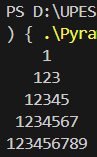


1. **Pyramid Triangle:**

Input:

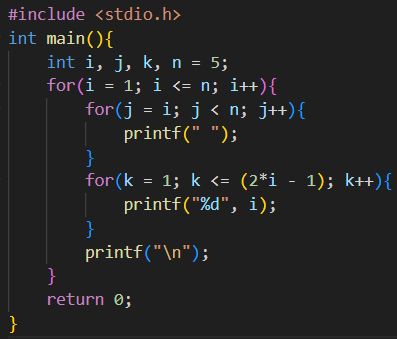


Output:

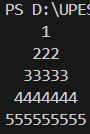


1. **Pyramid:**

Input:

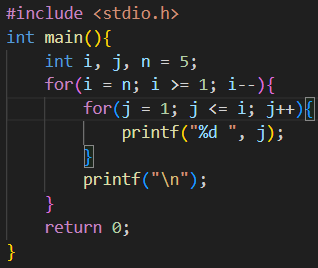


Output:

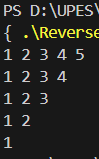


1. **Reverse Number:**

Input:

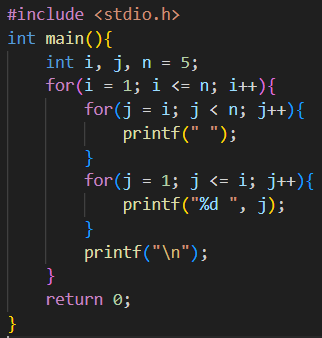


Output:

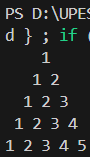


1. **Right Aligned Pyramid:**

Input:

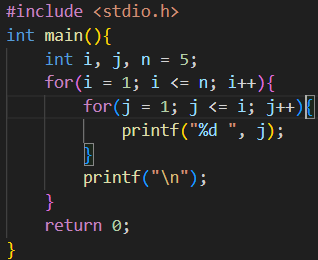


Output:

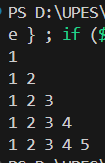


1. **Right Angled Pyramid:**

Input:



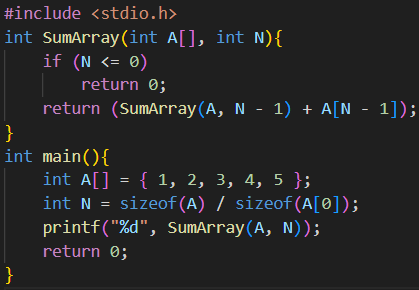
Output:



**Experiment 2: Problems**

1. **Find sum of all array elements using recursion.**

Input:

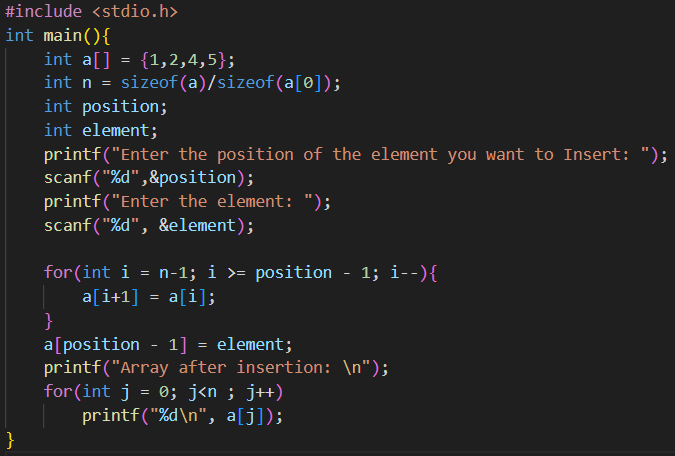


Output:

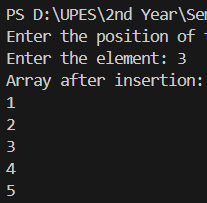


1. **Create an array ‘a1’ with ‘n’ elements. Insert an element in ith position of ‘a1’ and also delete an element from jth position of ‘a1’.**

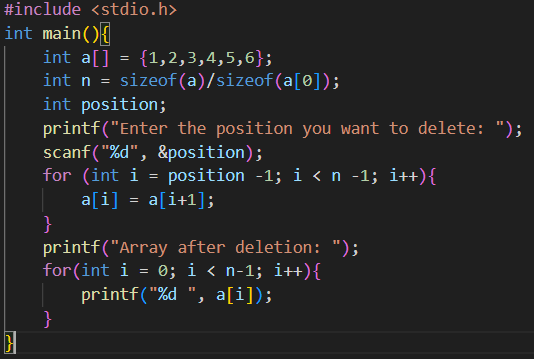
Inserting Input:



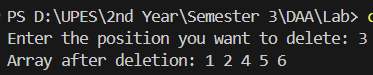
Output:



Deleting Input:

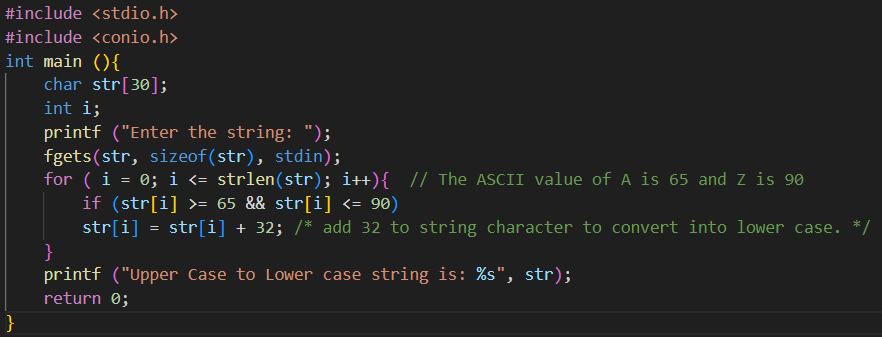


Output:

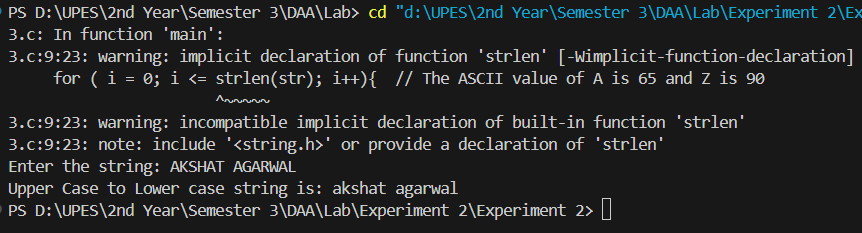


1. **Convert uppercase string to lowercase using for loop.**

Input:

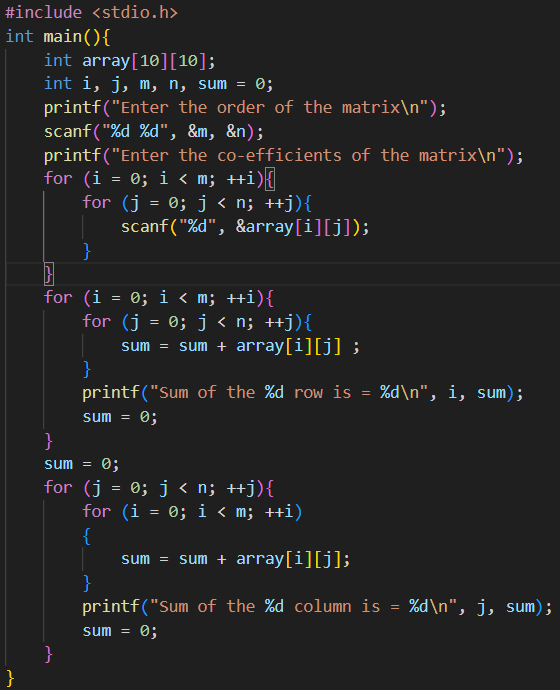


Output:

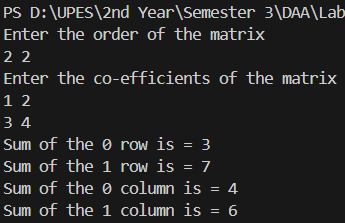


1. **Find the sum of rows and columns of matrix of given order (row x column).**

Input:

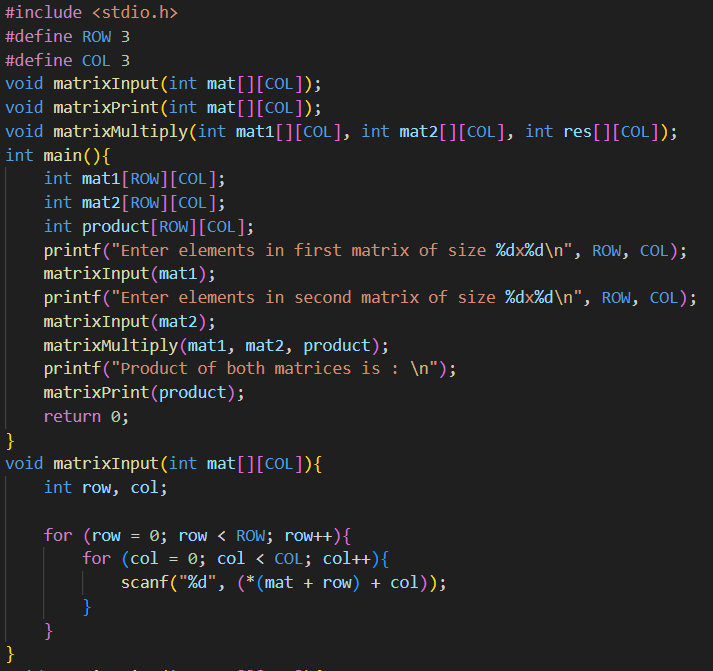


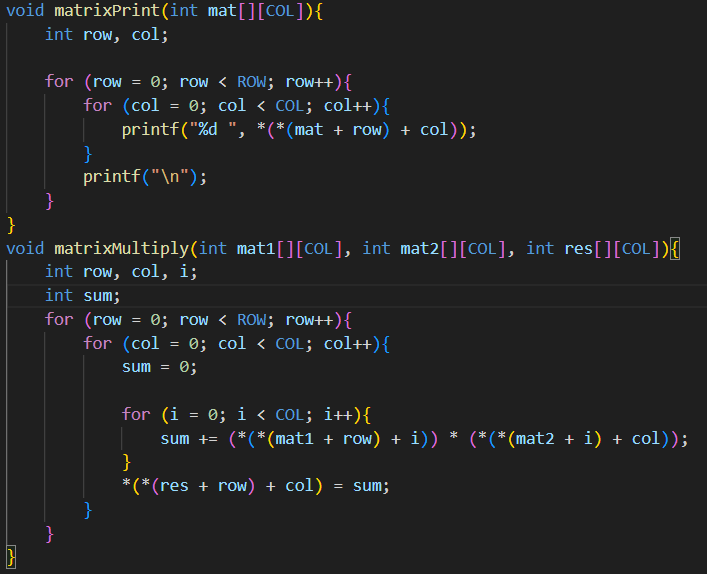
Output:



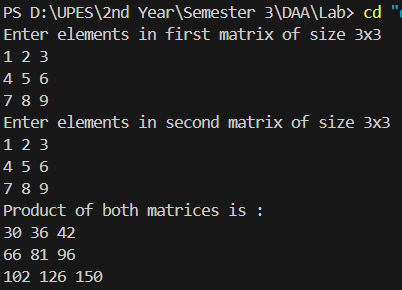
1. **Find the product of two matrices using pointers.**

Input:



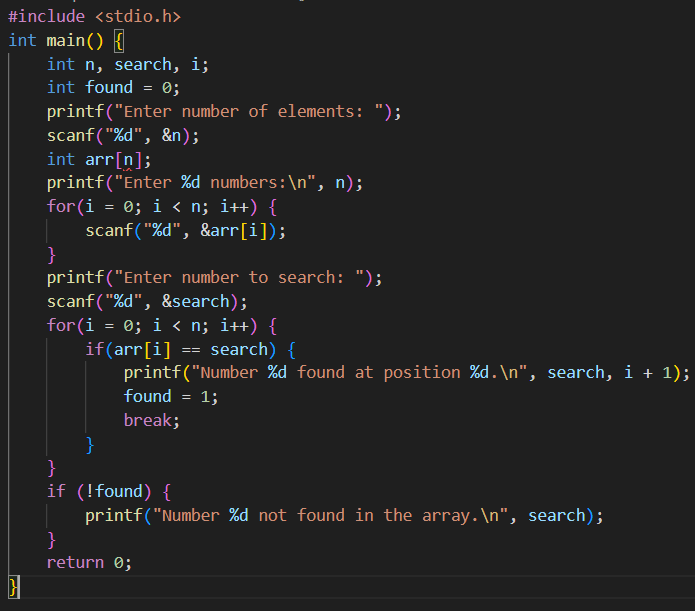


Output:

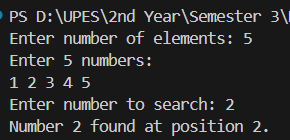
****

1. **Store ‘n’ numbers (integers or real) in an array. Conduct a linear search for a given number and report success or failure in the form of a suitable message.**

Input:

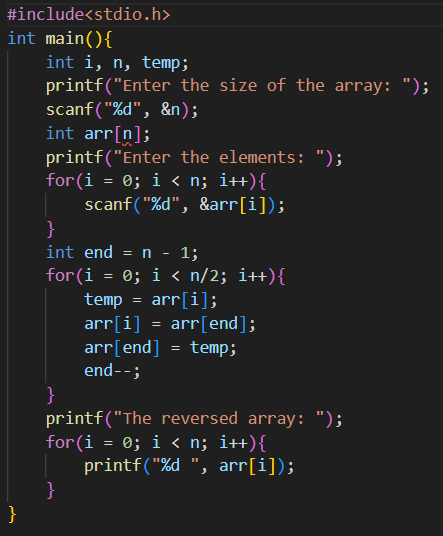


Output:

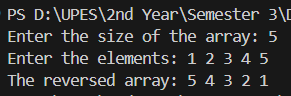


1. **Write a program to reverse an array.**

Input:

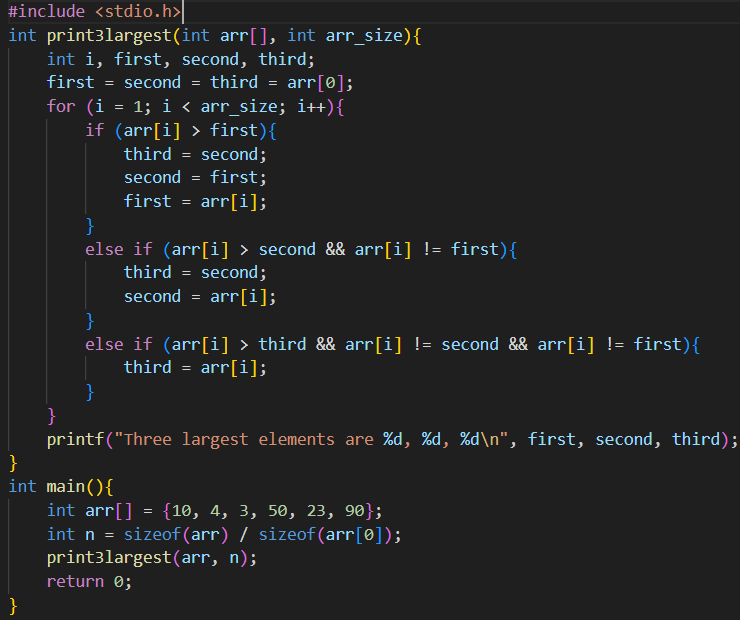


Output:

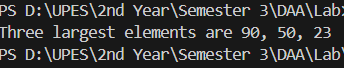


1. **Find the largest three distinct elements in an array: Input: arr[] = {10, 4, 3, 50, 23, 90} Output: 90, 50, 23**

Input:

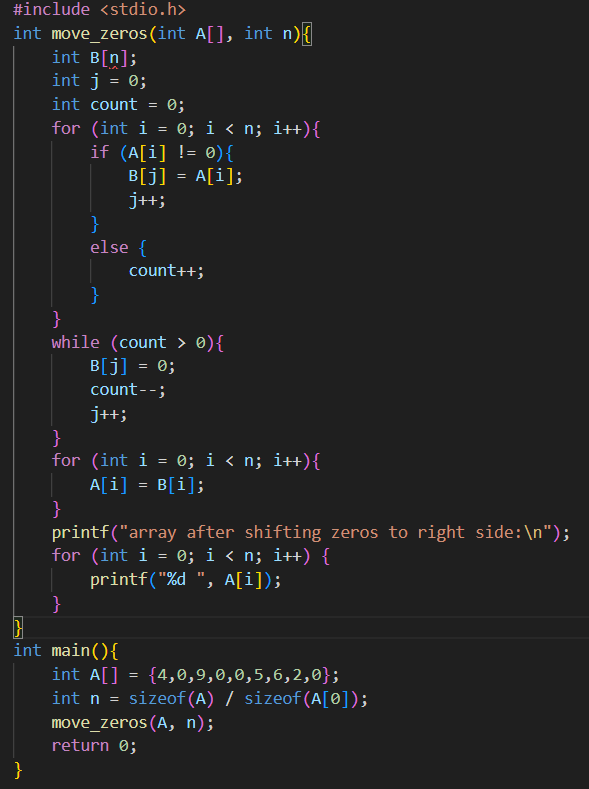


Output:

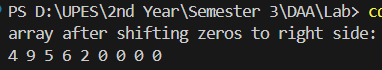


1. **Move all zeroes to end of array**

Input:

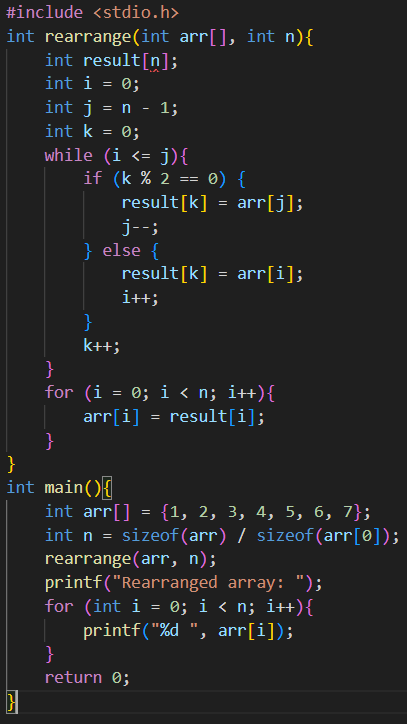


Output:

****

1. **Rearrange an array in maximum minimum form using Two Pointer Technique. Input: arr[] = {1, 2, 3, 4, 5, 6, 7} Output: arr[] = {7, 1, 6, 2, 5, 3, 4}.**

Input:

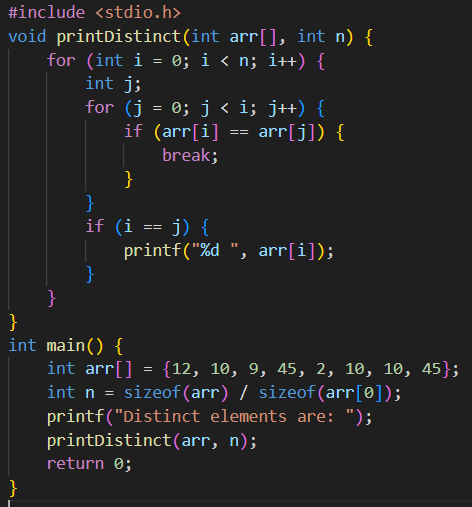


Output:



1. **Print all Distinct ( Unique ) Elements in given Array: Input: arr[] = {12, 10, 9, 45, 2, 10, 10, 45} Output: 12, 10, 9, 2**

Input:

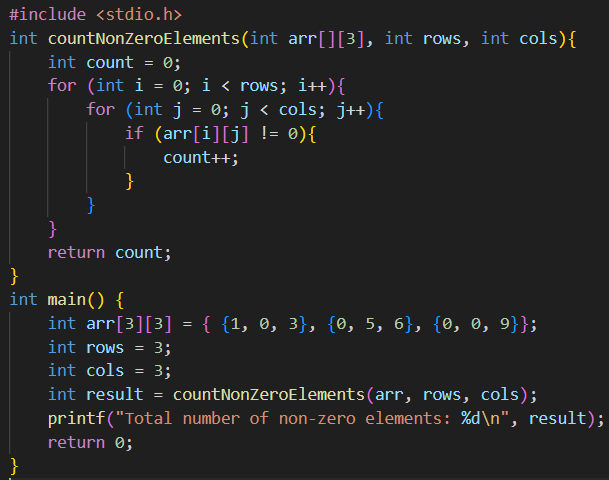


Output:



1. **Write a program to count the total number of nonzero elements in a two-dimensional array.**

Input:

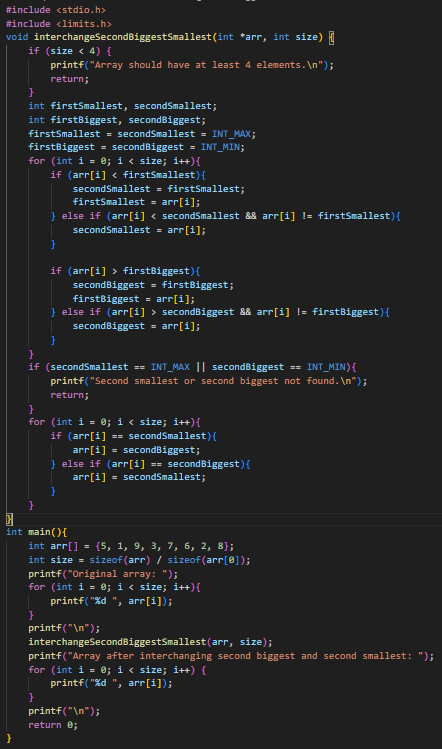


Output:

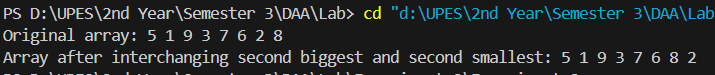


1. **Write a program using pointers to interchange the second biggest and the second smallest number in the array.**

Input:



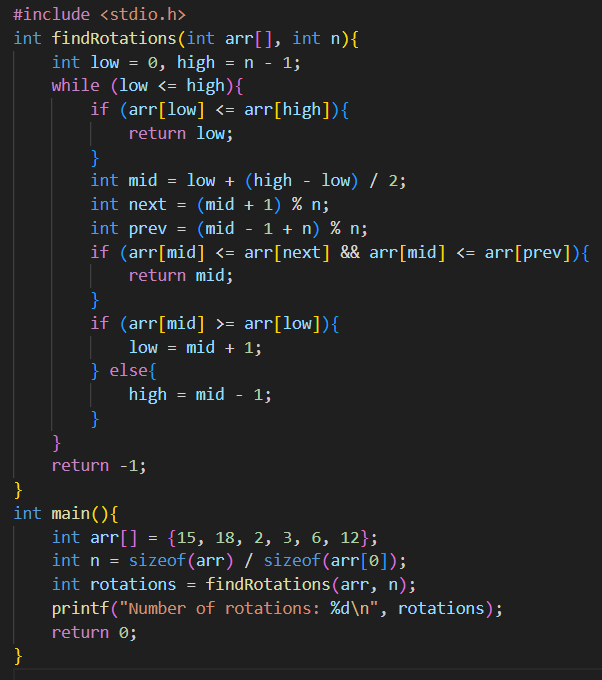
Output:



**Experiment 3: Problems in Binary Search**

1. **Find the number of rotations in a circularly sorted array**

Input:

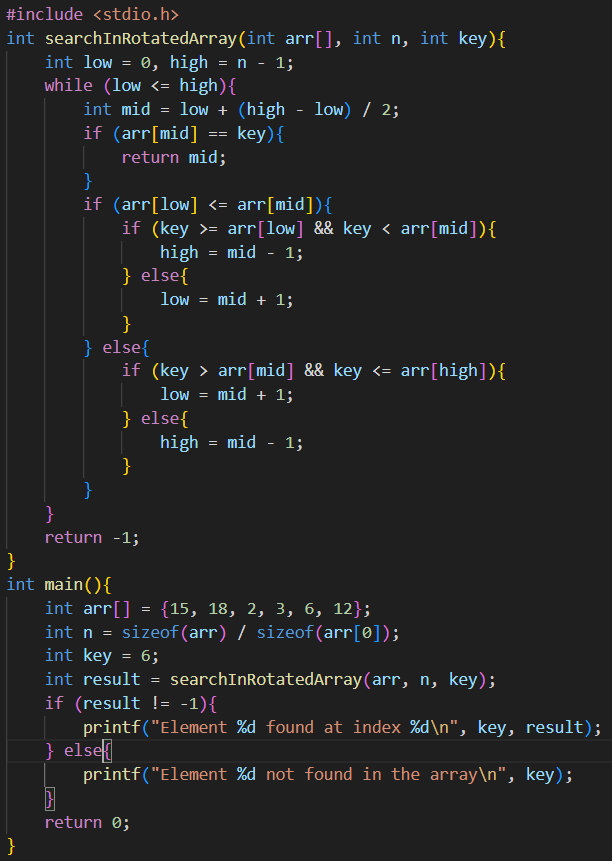


Output:



1. **Search an element in a circularly sorted array**

Input:

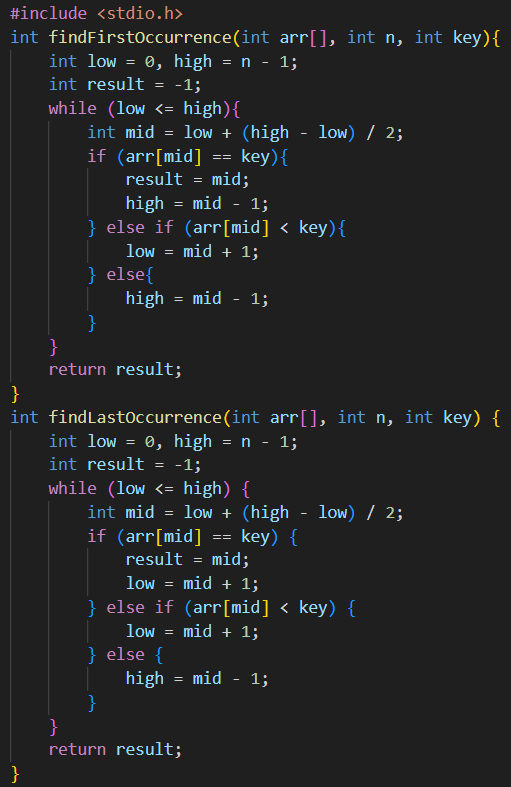
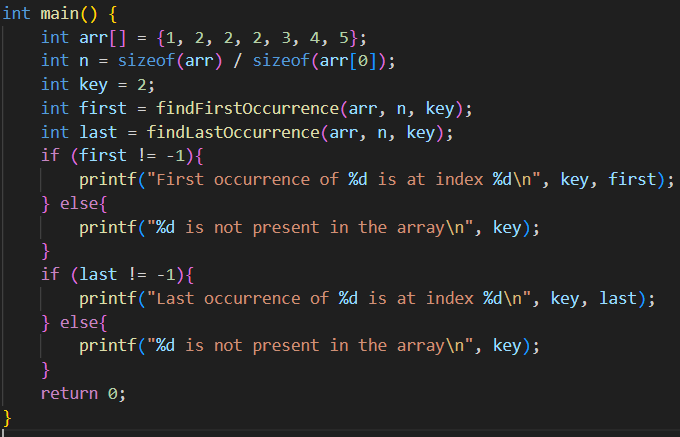


Output:

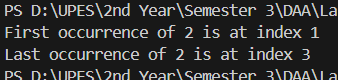


1. **Find the first or last occurrence of a given number in a sorted array**

Input:

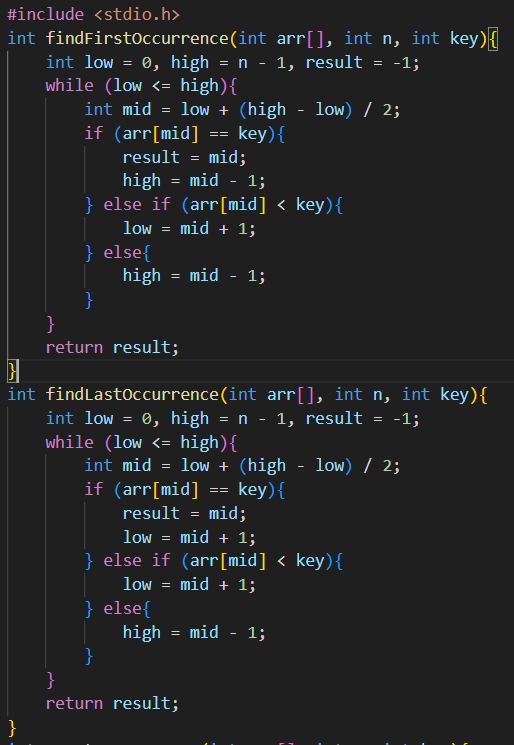
 

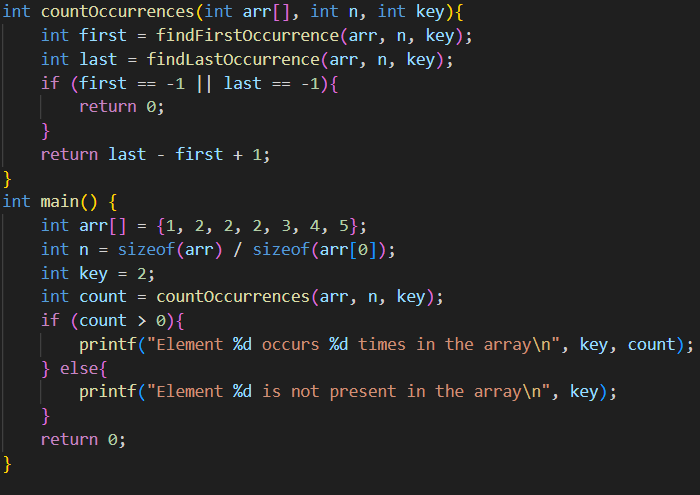
Output:



1. **Count occurrences of a number in a sorted array with duplicates**

Input:



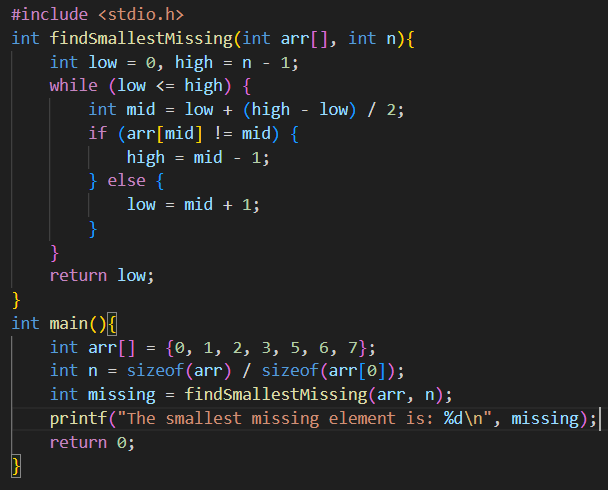


Output:



1. **Find the smallest missing element from a sorted array**

Input:

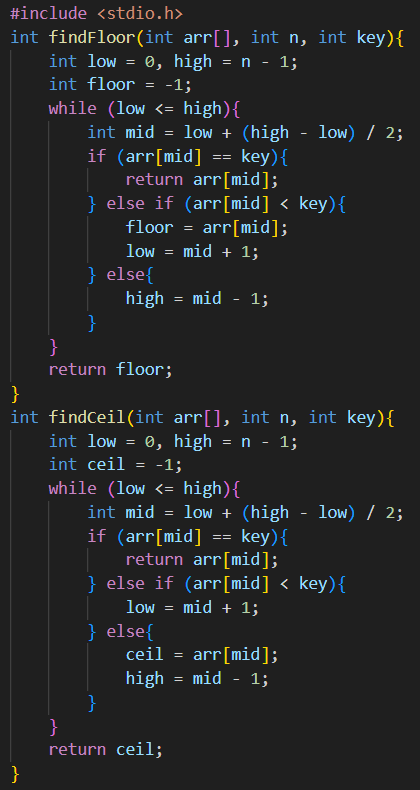


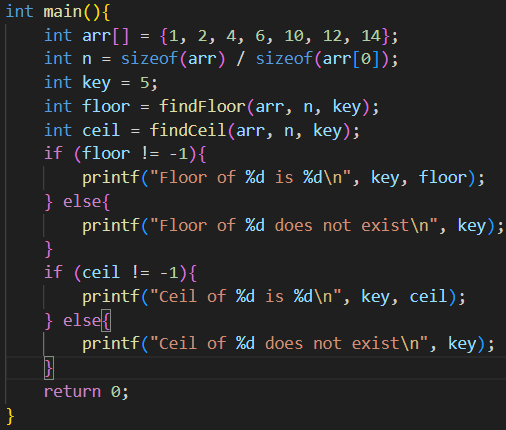
Output:



1. **Find floor and ceil of a number in a sorted integer array**

Input:



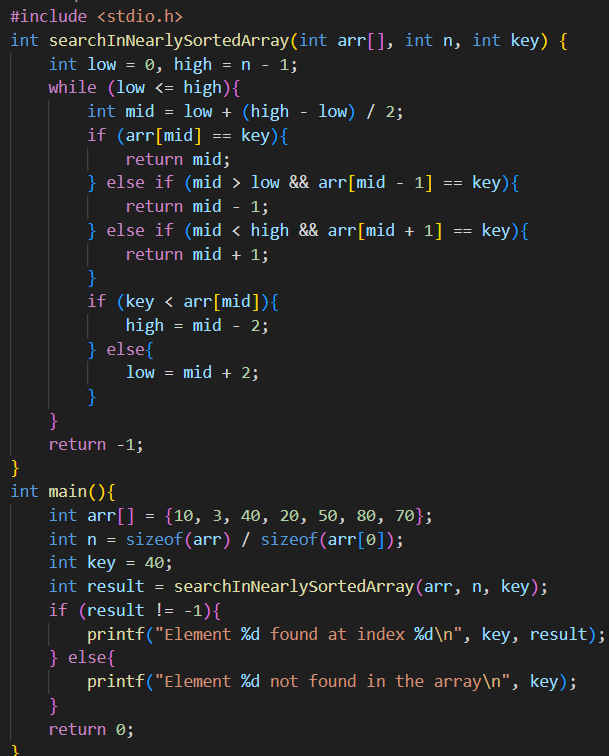


Output:



1. **Search in a nearly sorted array in logarithmic time**

Input:

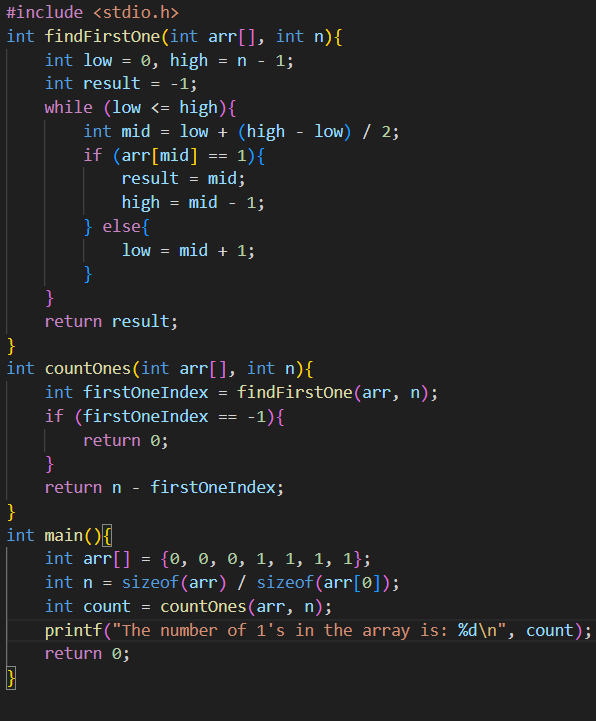


Output:



1. **Find the number of 1's in a sorted binary array**

Input:

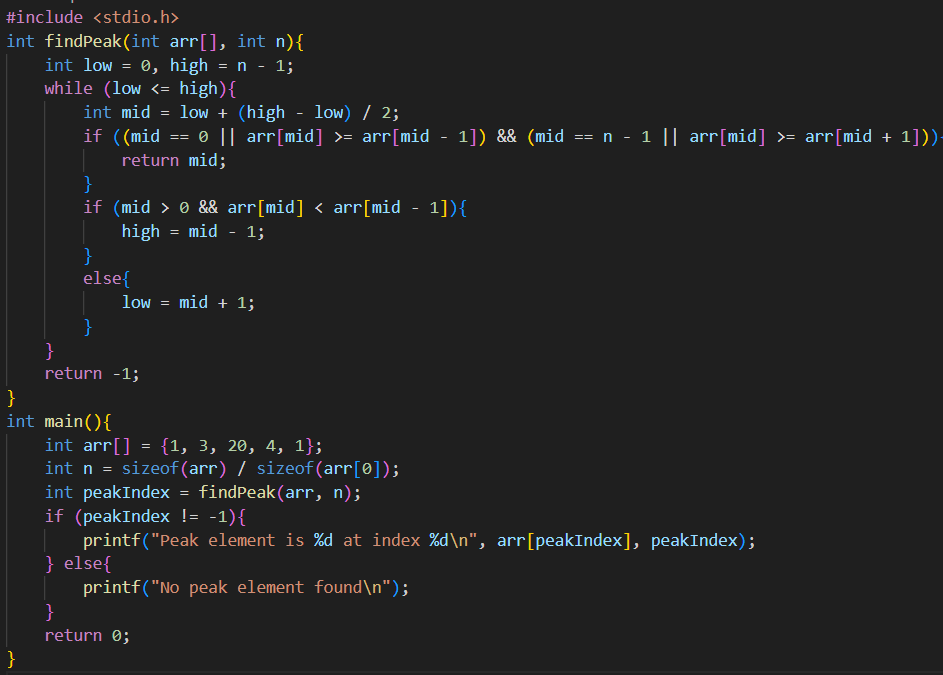


Output:

****

1. **Find the peak element in an array**

Input:

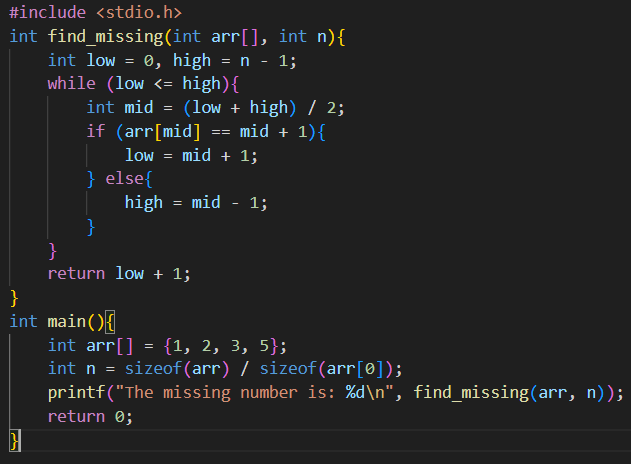


Output:



1. **Find the missing term in a sequence in logarithmic time**

Input:

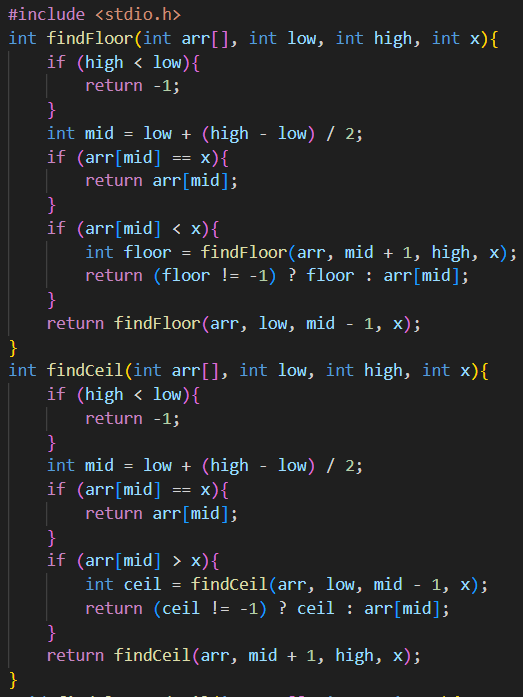


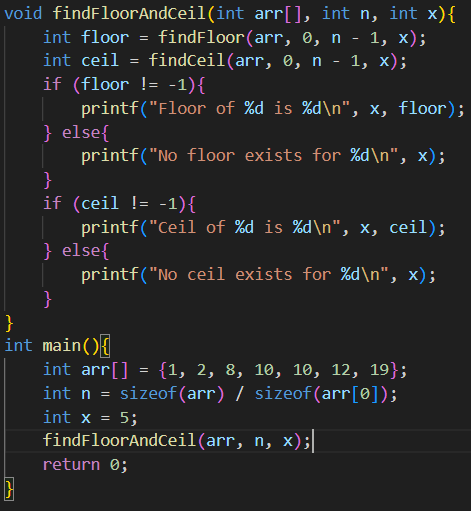
Output:



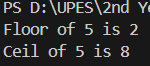
1. **Find floor and ceil of a number in a sorted array (Recursive solution)**

Input:



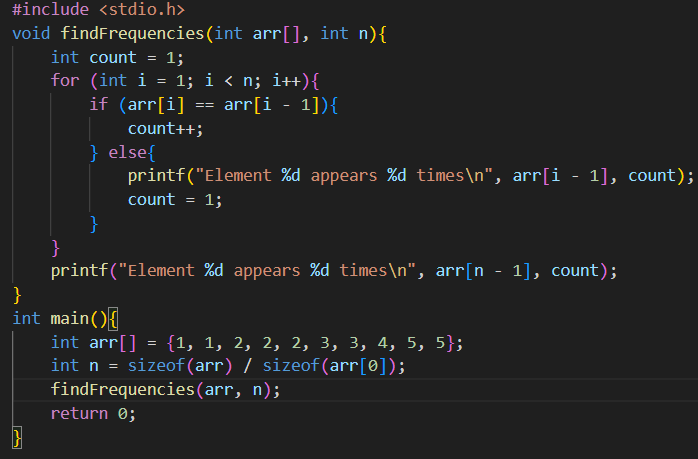


Output:

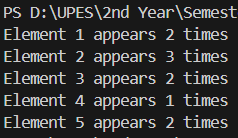


1. **Find the frequency of each element in a sorted array containing duplicates**

Input:

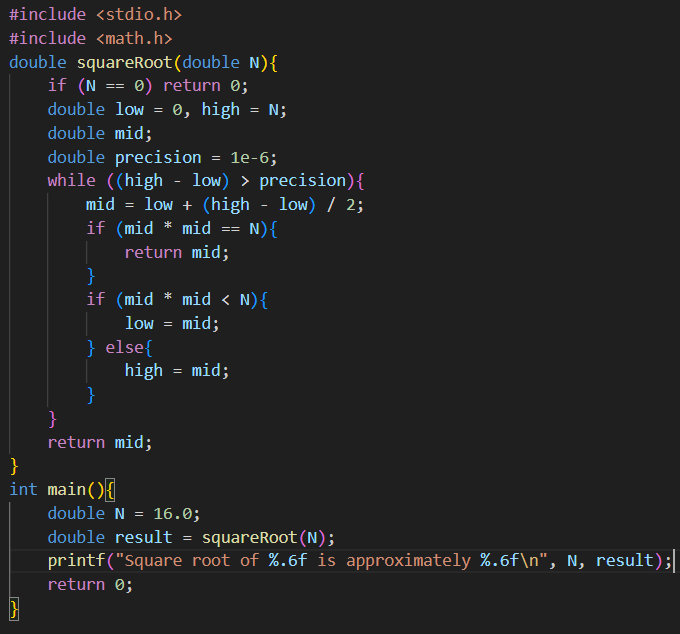


Output:



1. **Find the square root of a number using a binary search**

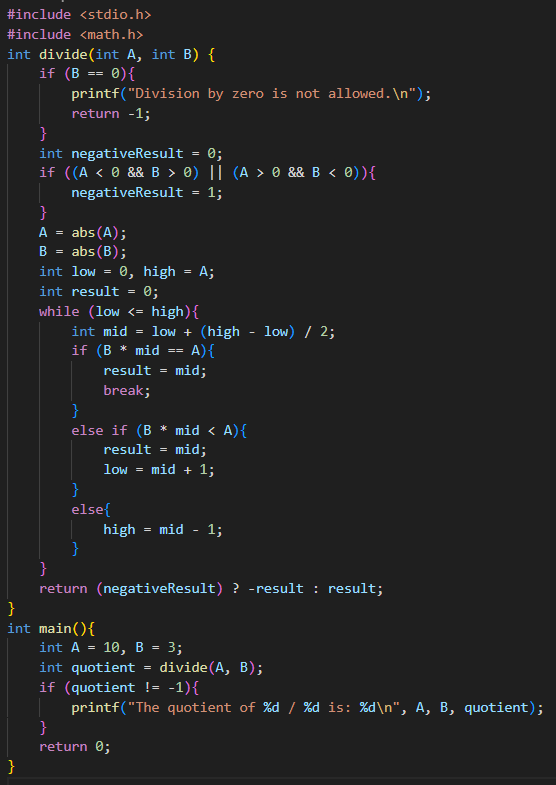
Input:



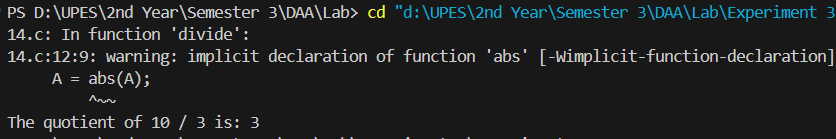
Output:  


1. **Division of two numbers using binary search algorithm**

Input:

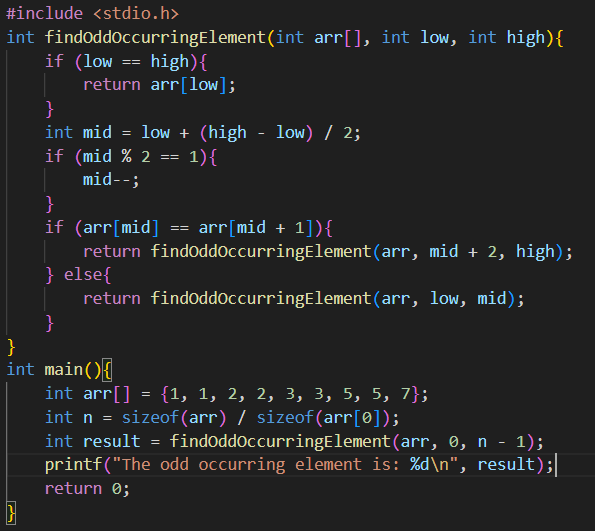


Output:



1. **Find the odd occurring element in an array in logarithmic time**

Input:

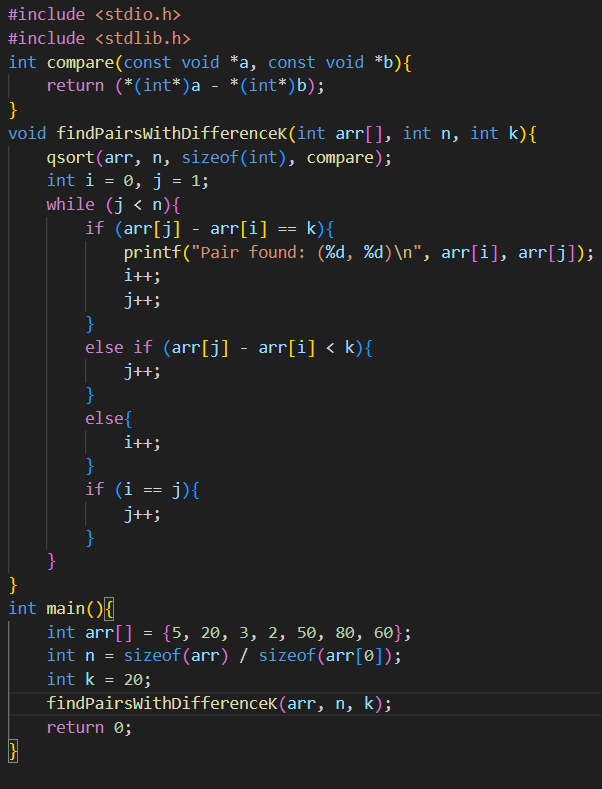


Output:



1. **Find pairs with difference k in an array | Constant Space Solution**

Input:

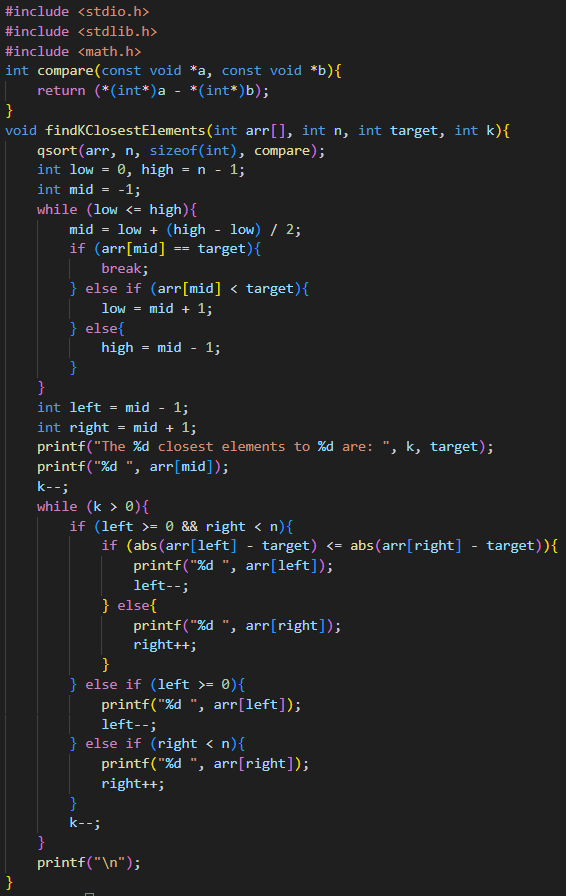


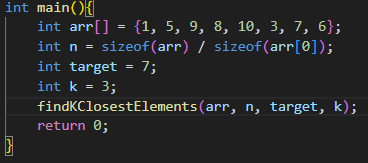
Output:



1. **Find k closest elements to a given value in an array**

Input:





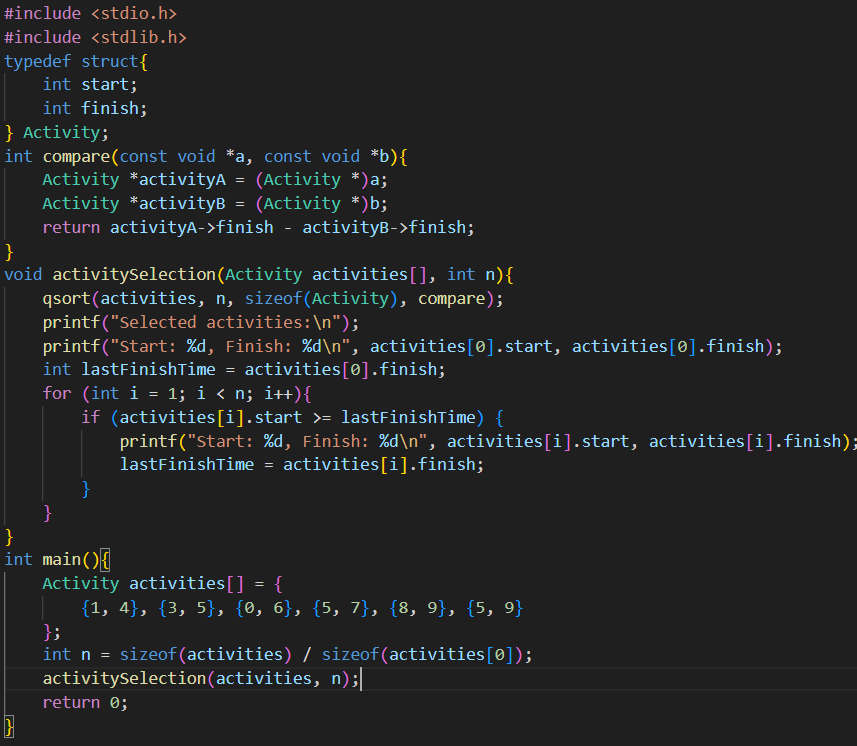
Output:



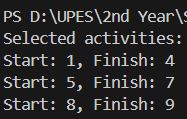
**Experiment 4: Patterns**

1. **Implement the activity selection problem to get a clear understanding of greedy approach.**

Input:

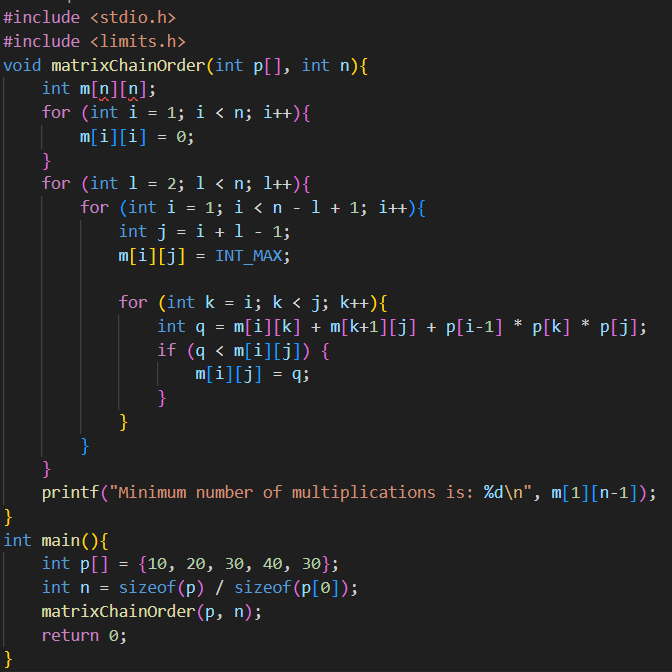


Output:



1. **Get a detailed insight of dynamic programming approach by the implementation of Matrix Chain Multiplication problem and see the impact of parenthesis positioning on time requirements for matrix multiplication.**

Input:



Output:



1. **Compare the performance of Dijkstra and Bellman ford algorithm for the single source shortest path problem.**

**Comparison of Dijkstra and Bellman-Ford Algorithms:**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Dijkstra** | **Bellman-Ford** |
| **Graph Type** | Works only for graphs with non-negative weights. | Works for graphs with negative weights. |
| **Time Complexity** | O(V+ElogV) with a priority queue. | O(V⋅E) |
| **Optimality** | Always finds the shortest path if conditions are met. | Handles graphs with negative weight edges correctly. |
| **Ease of Implementation** | Slightly more complex due to priority queue. | Simpler to implement iteratively. |
| **Use Case** | Faster for dense graphs with positive weights. | Essential when negative weights are present. |
| **Edge Relaxation** | Relaxes each edge once during iteration. | Relaxes edges V−1 times. |

1. **Through 0/1 Knapsack problem, analyze the greedy and dynamic programming approach for the same dataset.**

**Greedy Approach**

Input:

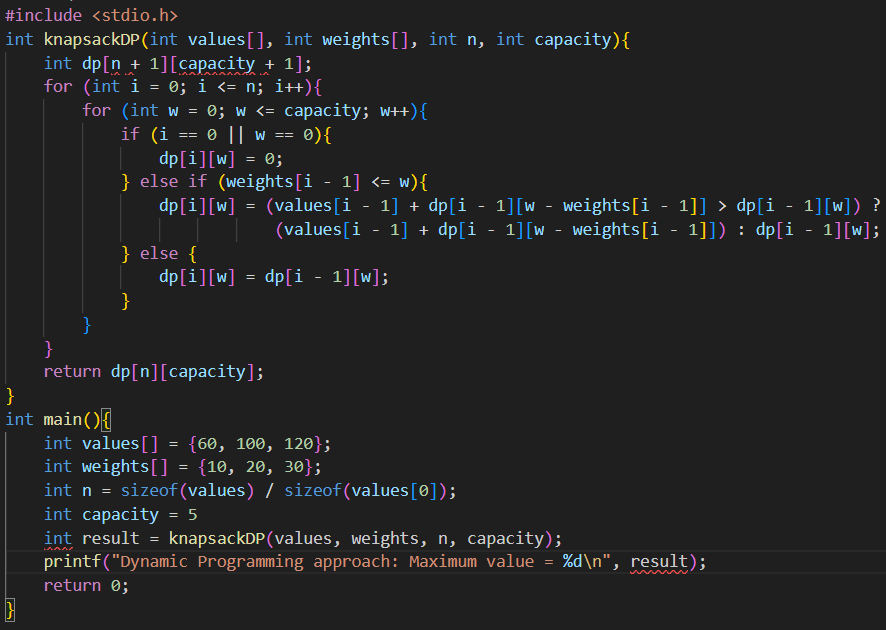


Output:



**Dynamic Programming**

Input:

****

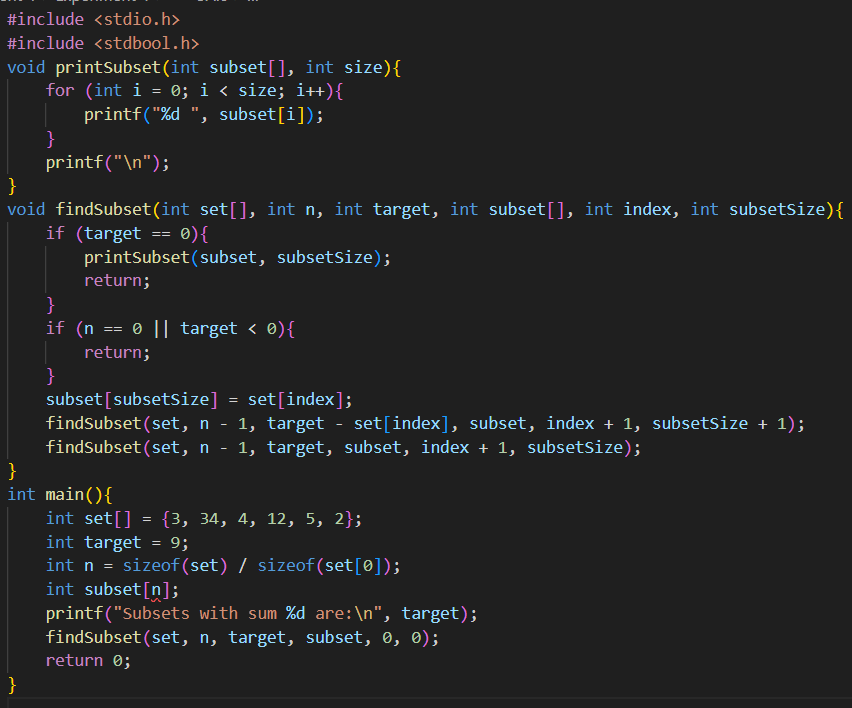
Output:



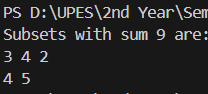
1. **Implement the sum of subset and N Queen problem.**

**Sum of Subset**

Input:

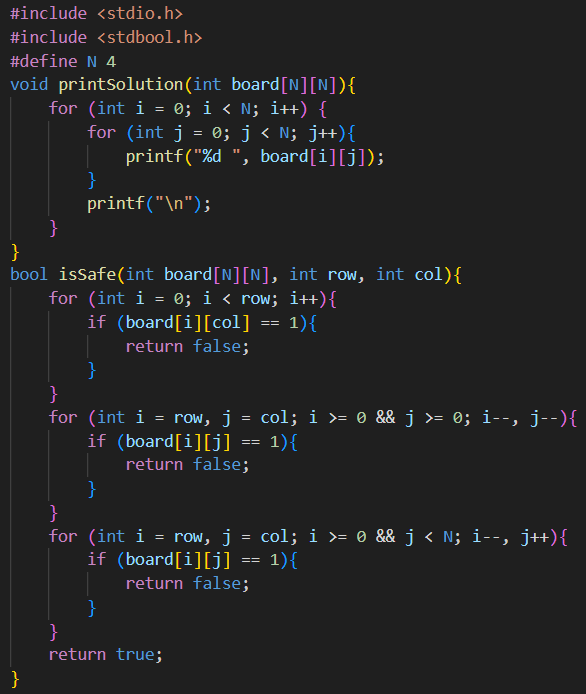


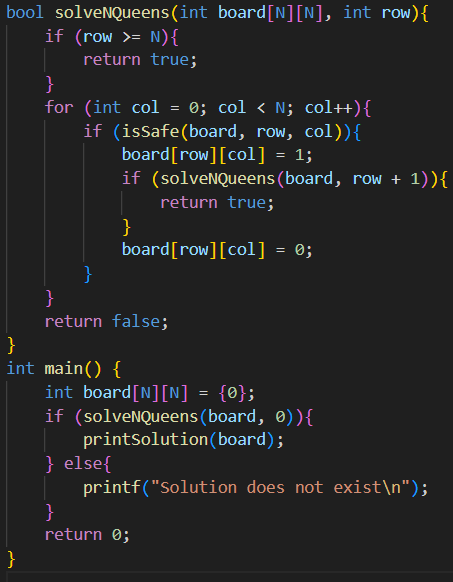
Output:



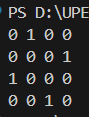
**N Queen problem**

Input:





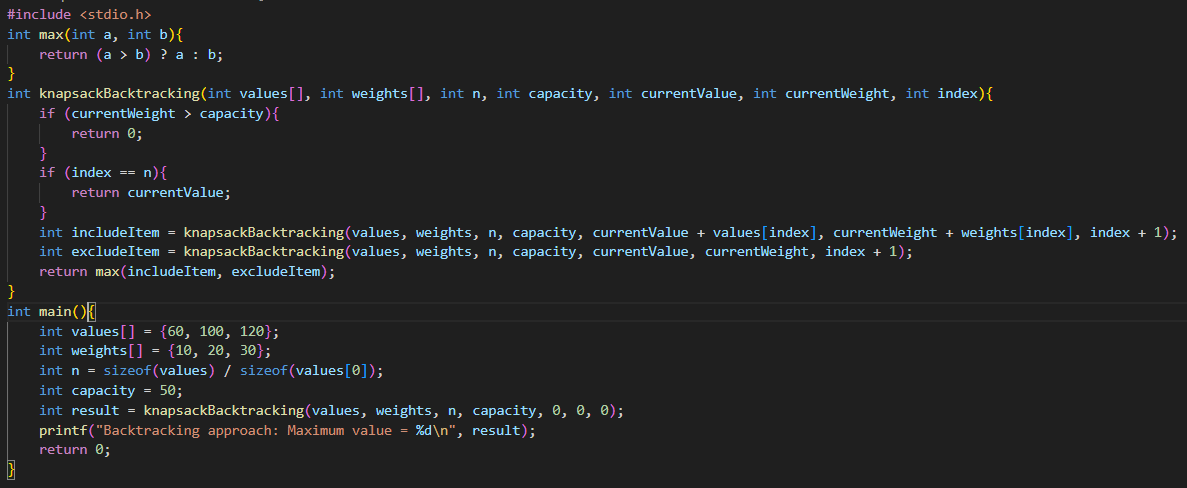
Output:



1. **Compare the Backtracking and Branch & Bound Approach by the implementation of 0/1 Knapsack problem. Also compare the performance with dynamic programming approach.**

**Backtracking**

Input:

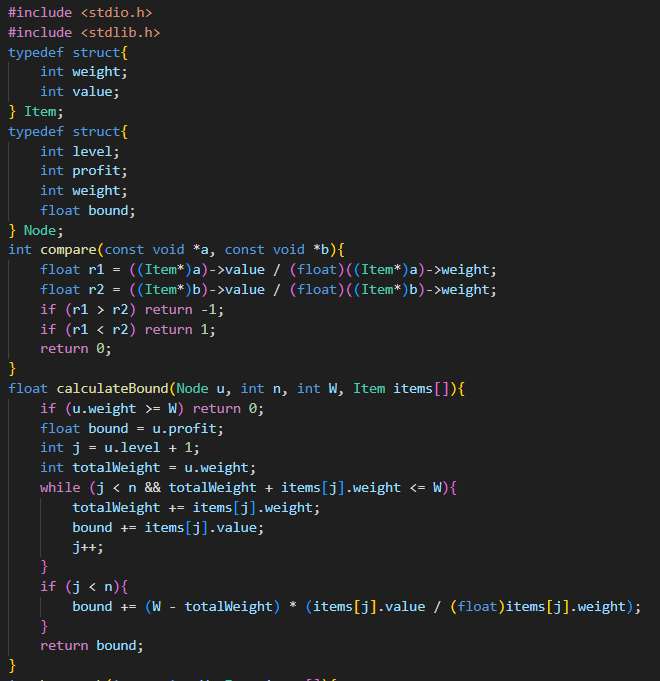


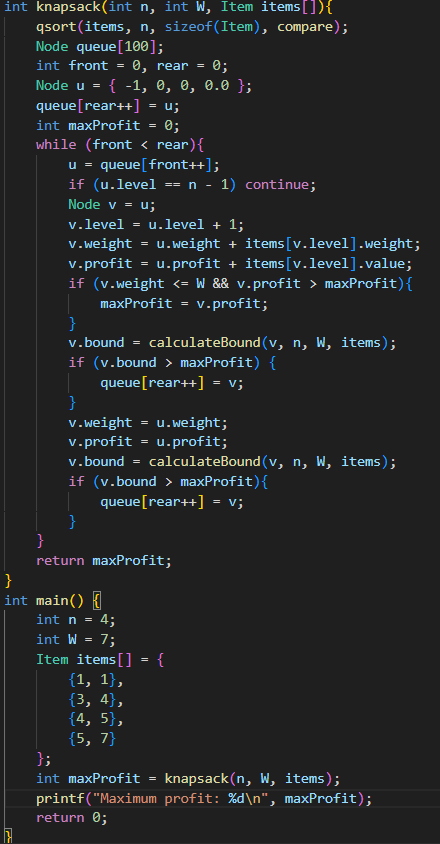
Output:



**Branch & Bound**

Input:



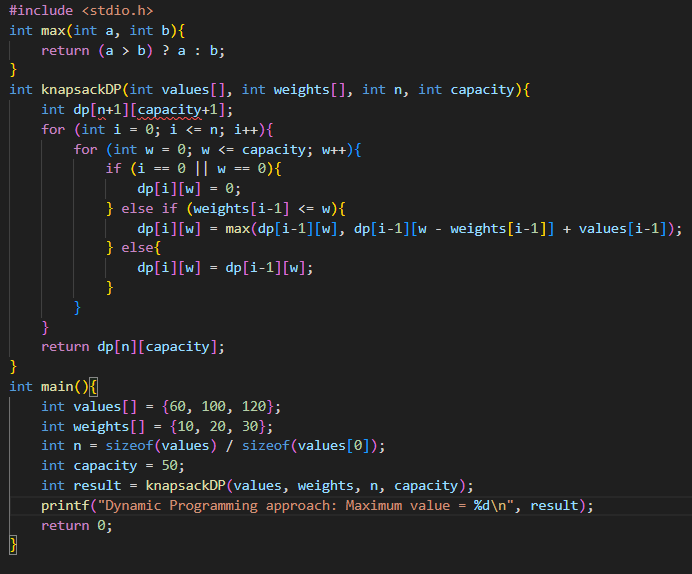


Output:



**Dynamic Programming**

Input:



Output:

