ALGORITHMS LABORATORY
[CS39001]
Individual Work

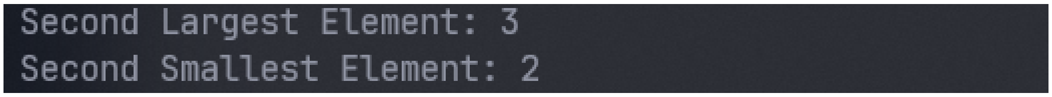
Lab. No.- 1 Date.14/7/2025

Roll Number:	2328159	**Branch/Section:**	CSSE-1
Name in Capital:	AYUSH RAJ		

Program No: 1.1

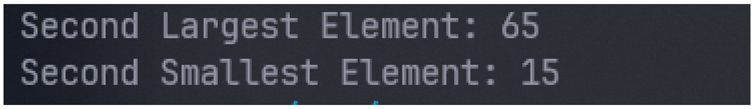
Program Title:
Write a program to find out the second smallest and second largest element stored in an array of n integers.

Input/Output Screenshots:
RUN-1:



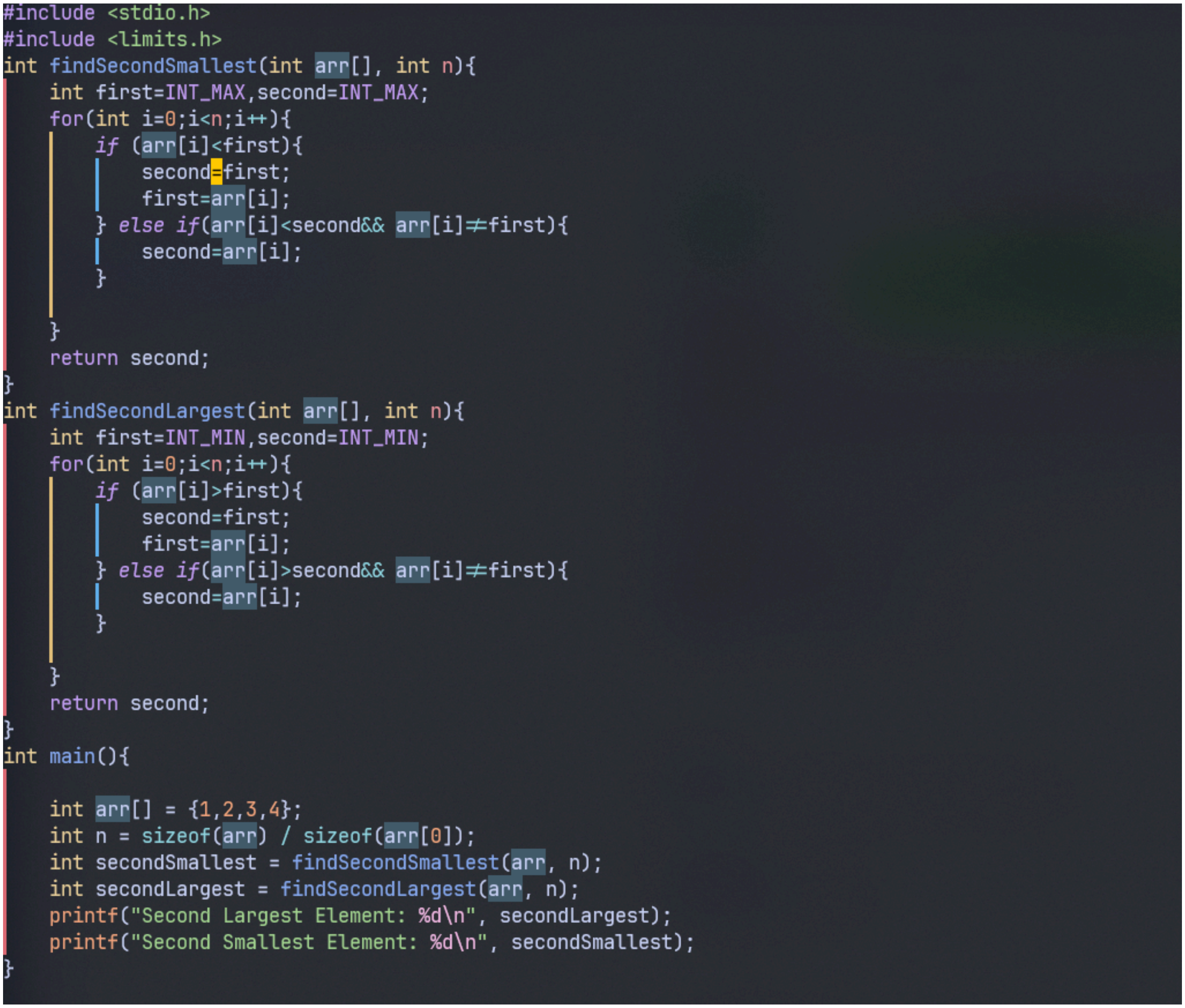
```
Second Largest Element: 3
Second Smallest Element: 2
```

RUN-2



```
Second Largest Element: 65
Second Smallest Element: 15
```

Source code



```c
#include <stdio.h>
#include <limits.h>
int findSecondSmallest(int arr[], int n){
    int first=INT_MAX,second=INT_MAX;
    for(int i=0;i<n;i++){
        if (arr[i]<first){
            second=first;
            first=arr[i];
        } else if(arr[i]<second&& arr[i]!=first){
            second=arr[i];
        }

    }
    return second;
}
int findSecondLargest(int arr[], int n){
    int first=INT_MIN,second=INT_MIN;
    for(int i=0;i<n;i++){
        if (arr[i]>first){
            second=first;
            first=arr[i];
        } else if(arr[i]>second&& arr[i]!=first){
            second=arr[i];
        }

    }
    return second;
}
int main(){

    int arr[] = {1,2,3,4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int secondSmallest = findSecondSmallest(arr, n);
    int secondLargest = findSecondLargest(arr, n);
    printf("Second Largest Element: %d\n", secondLargest);
    printf("Second Smallest Element: %d\n", secondSmallest);
}
```

Conclusion/Observation

The program correctly reads an array of integers from a file and finds the second smallest and second largest values using a single traversal. This method is efficient and avoids sorting, making it suitable for large datasets.

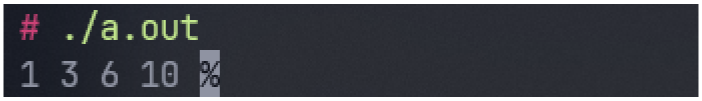
Program No: 1.2

Program Title:

Given an array arr[] of size N, find the prefix sum of the array. A
prefix sum array is another array prefixSum[] of the same size, such that the value of
prefixSum[i] is arr[0] + arr[1] + arr[2] . . . arr[i].

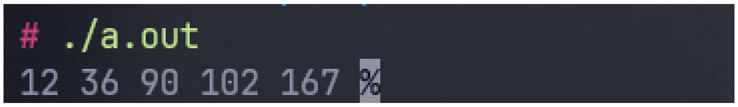
Input/Output Screenshots:

RUN-1:



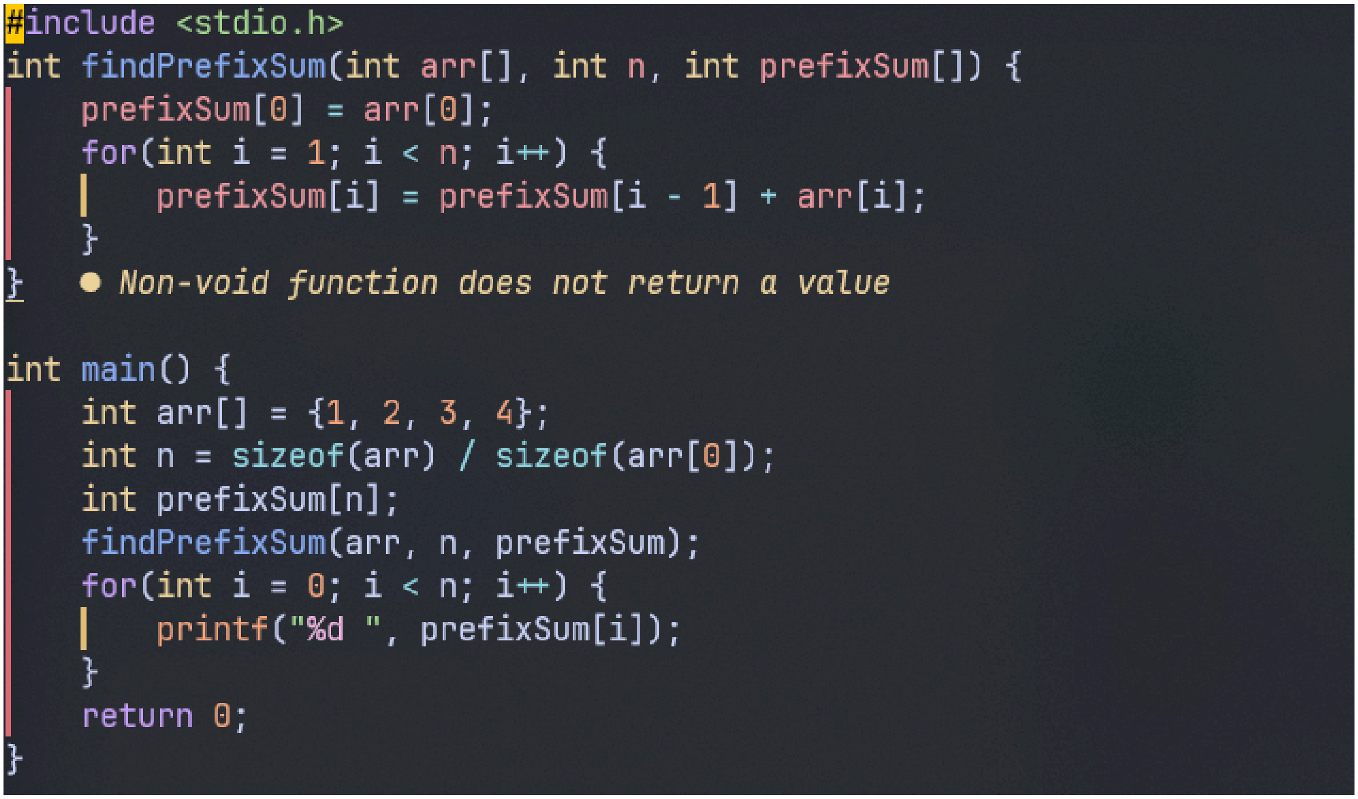
```
# ./a.out
1 3 6 10 %
```

RUN-2



```
# ./a.out
12 36 90 102 167 %
```

Source code



```
#include <stdio.h>
int findPrefixSum(int arr[], int n, int prefixSum[]) {
    prefixSum[0] = arr[0];
    for(int i = 1; i < n; i++) {
        prefixSum[i] = prefixSum[i - 1] + arr[i];
    }
}   ● Non-void function does not return a value

int main() {
    int arr[] = {1, 2, 3, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int prefixSum[n];
    findPrefixSum(arr, n, prefixSum);
    for(int i = 0; i < n; i++) {
        printf("%d ", prefixSum[i]);
    }
    return 0;
}
```

Conclusion/Observation

This program efficiently computes the prefix sum by maintaining a cumulative total as it iterates through the array. The result is useful in many scenarios like range queries, dynamic programming, and time-series analysis.

Program No: 1.3

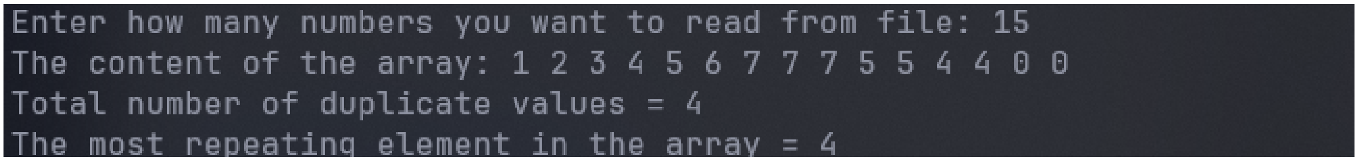
Program Title:

Write a program to read ‘n’ integers from a disc file that must contain
some duplicate values and store them into an array. Perform the following operations on the
array.

a) Find out the total number of duplicate elements.
b) Find out the most repeating element in the array.

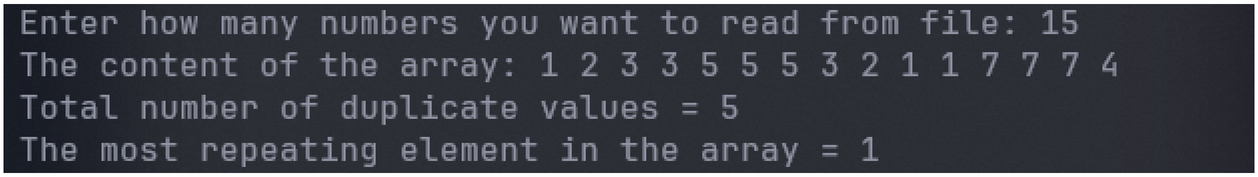
Input/Output Screenshots:

RUN-1:



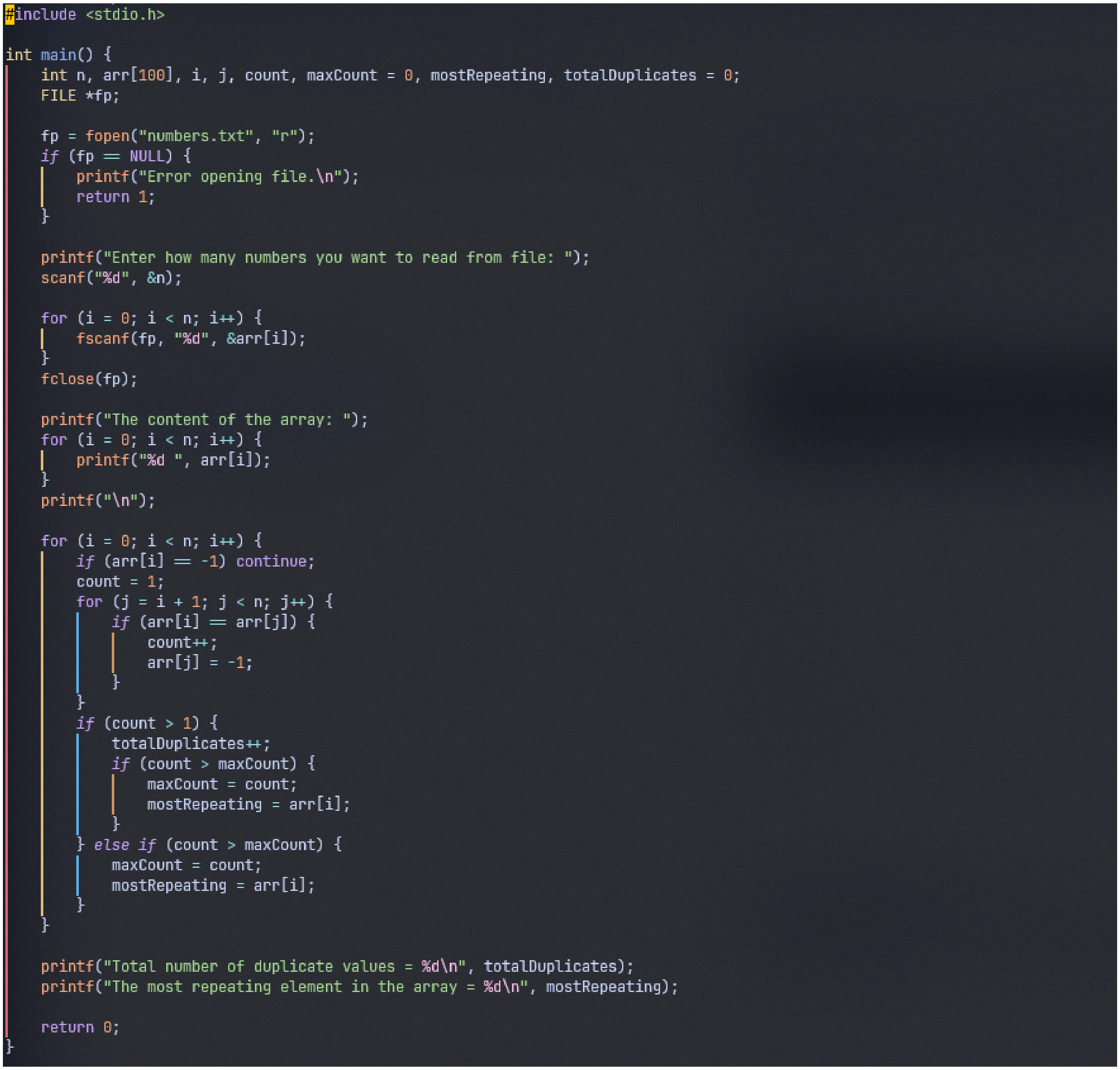
```
Enter how many numbers you want to read from file: 15
The content of the array: 1 2 3 4 5 6 7 7 7 5 5 4 4 0 0
Total number of duplicate values = 4
The most repeating element in the array = 4
```

RUN-2



```
Enter how many numbers you want to read from file: 15
The content of the array: 1 2 3 3 5 5 5 3 2 1 1 7 7 7 4
Total number of duplicate values = 5
The most repeating element in the array = 1
```

Source code



```c
#include <stdio.h>

int main() {
    int n, arr[100], i, j, count, maxCount = 0, mostRepeating, totalDuplicates = 0;
    FILE *fp;

    fp = fopen("numbers.txt", "r");
    if (fp == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    printf("Enter how many numbers you want to read from file: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        fscanf(fp, "%d", &arr[i]);
    }
    fclose(fp);

    printf("The content of the array: ");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    for (i = 0; i < n; i++) {
        if (arr[i] == -1) continue;
        count = 1;
        for (j = i + 1; j < n; j++) {
            if (arr[i] == arr[j]) {
                count++;
                arr[j] = -1;
            }
        }
        if (count > 1) {
            totalDuplicates++;
            if (count > maxCount) {
                maxCount = count;
                mostRepeating = arr[i];
            }
        } else if (count > maxCount) {
            maxCount = count;
            mostRepeating = arr[i];
        }
    }

    printf("Total number of duplicate values = %d\n", totalDuplicates);
    printf("The most repeating element in the array = %d\n", mostRepeating);

    return 0;
}
```

Conclusion/Observation

The program reads an array with duplicates from a file, counts how many values are repeated, and identifies the most frequent element. It demonstrates the use of nested loops and counters to handle array analysis tasks.

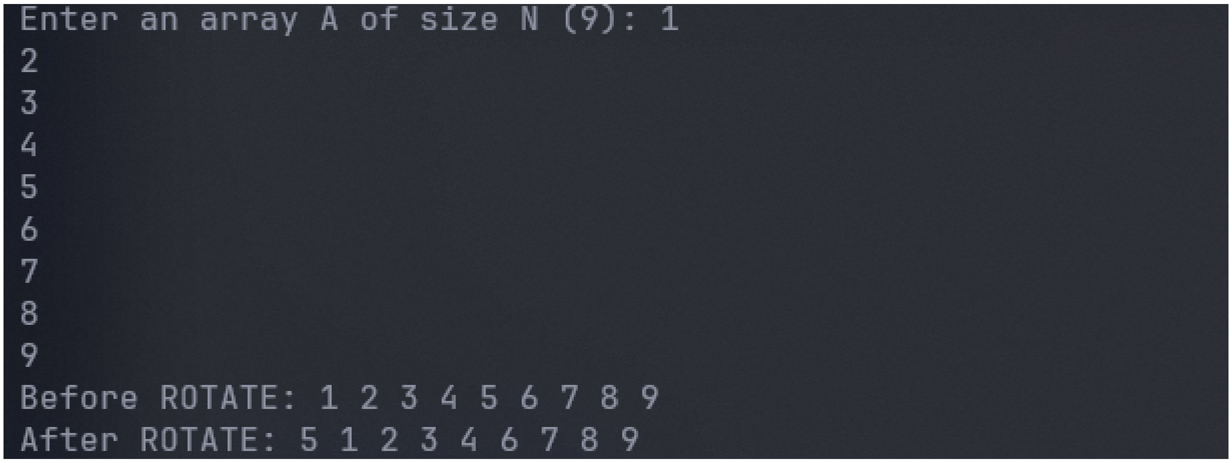
Program No: 1.4

Program Title:

Write a function to ROTATE_RIGHT (p1, p2) right an array for first
p2 elements by 1 position using EXCHANGE (p, q) function that swaps/exchanges the
numbers p & q. Parameter p1 be the starting address of the array and p2 be the number of
elements to be rotated.

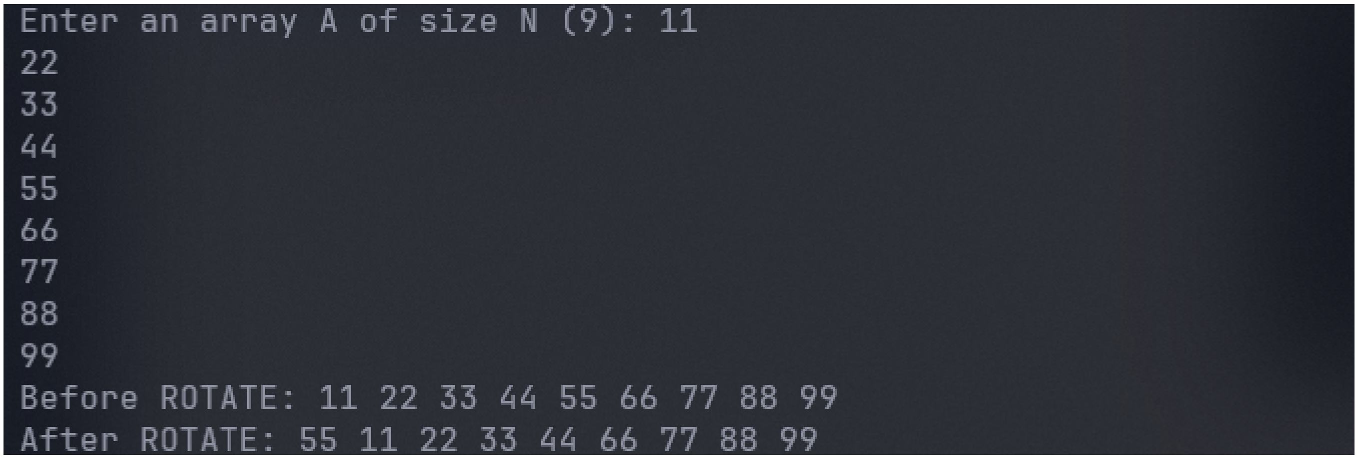
Input/Output Screenshots:

RUN-1:



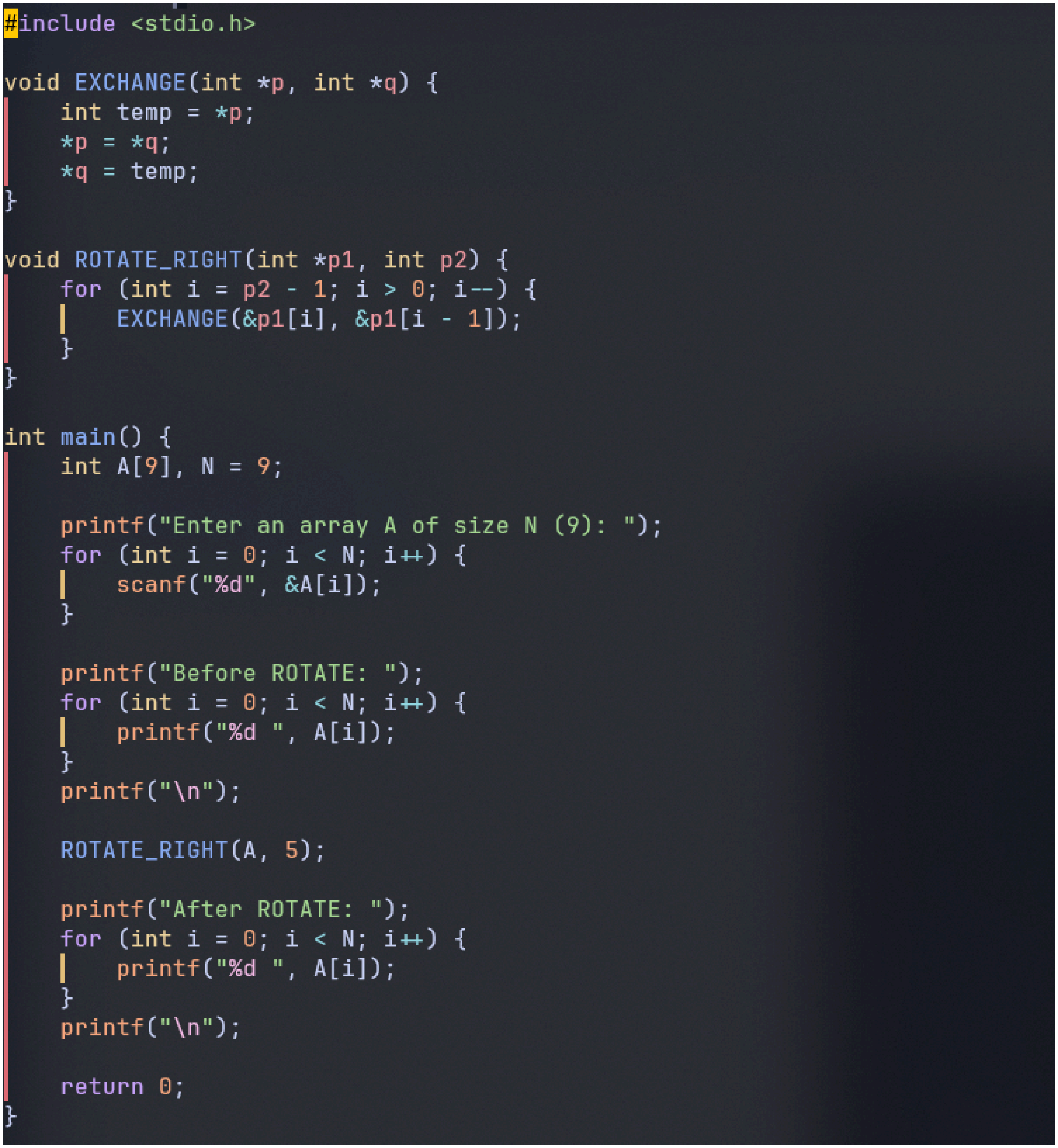
```
Enter an array A of size N (9): 1
2
3
4
5
6
7
8
9
Before ROTATE: 1 2 3 4 5 6 7 8 9
After ROTATE: 5 1 2 3 4 6 7 8 9
```

RUN-2



```
Enter an array A of size N (9): 11
22
33
44
55
66
77
88
99
Before ROTATE: 11 22 33 44 55 66 77 88 99
After ROTATE: 55 11 22 33 44 66 77 88 99
```

Source code



```
#include <stdio.h>

void EXCHANGE(int *p, int *q) {
    int temp = *p;
    *p = *q;
    *q = temp;
}

void ROTATE_RIGHT(int *p1, int p2) {
    for (int i = p2 - 1; i > 0; i--) {
        EXCHANGE(&p1[i], &p1[i - 1]);
    }
}

int main() {
    int A[9], N = 9;

    printf("Enter an array A of size N (9): ");
    for (int i = 0; i < N; i++) {
        scanf("%d", &A[i]);
    }

    printf("Before ROTATE: ");
    for (int i = 0; i < N; i++) {
        printf("%d ", A[i]);
    }
    printf("\n");

    ROTATE_RIGHT(A, 5);

    printf("After ROTATE: ");
    for (int i = 0; i < N; i++) {
        printf("%d ", A[i]);
    }
    printf("\n");

    return 0;
}
```

Conclusion/Observation

The program rotates the first few elements of an array to the right by one position using a custom exchange (swap) function. It effectively shows how pointer manipulation and modular functions can