Algorithm for Array Rotation

Assignment 1

GROUP - 15 (IEC2021058, IEC2021059, IEC2021061, IEC2021062)

Department of Electronics and Communication Engineering

Indian Institute of Information Technology, Allahabad

Abstract— **Given an array ar[ ] of size 'x' which needs to be rotated either towards the left or towards the right for a certain number of times (based on user input). This task has been accomplished using an algorithm that involves the concepts of 'for' loops and functions in the C language.**

I. Introduction

To rotate an array ar[ ] of size 'x', either leftwards or rightwards for 'n' times, we will use the concept of loops, arrays, and functions to achieve the desired result. To begin with, we take the required inputs from the user. The further algorithm can be expressed by two approaches, the first one includes the usage of 'for' loops, and the second one is an amalgamation of functions and 'for' loops. Either approach would give us an array that has been rotated(left/right) as the output.

'For' loop is a conditional iterative statement used to check for certain conditions and then repeatedly execute a block of code as long as those conditions are satisfied. And functions are a set of programming statements enclosed by{}, which can be called multiple times to provide reusability and modularity to the program.

II. Algorithm design

- Following is the rotation algorithm that we have used -

Case 1 : For Right Rotation

1) Firstly, we execute 'for' loop from i=1 to i <= n, where, n is the number of rotations.
2) Then, we store the last element of the array in a different variable, say, t.

   int t = ar[x-1]

   where, x is the number of elements in the array.
3) After that, we execute 'for' loop in such a way that every element of the array shifts towards right. It will be observed that elements at ar[0] and ar[1] are the same.
4) To solve this problem, we assign the last element of the array, that we have earlier stored in 't' to ar[0] ,i.e.,

   ar[0] = t
5) Hence, right rotation is completed.

Case 2 : For Left Rotation

1) Again, we execute 'for' loop from i=1 to i <= n, where, n is the number of rotations.
2) Now this time, we store the first element of the array in a different variable, say, t.

   int t = ar[0]
3) After that, we execute 'for' loop such that every element of the array shifts towards the left. It will be observed that elements at ar[x-1] and ar[x-2] are the same, where, x is the number of elements in the array.
4) To solve this problem, we assign the first element of the array, that we have earlier stored in't' to ar[x-1].

   ar[x-1] = t
5) Hence, the left rotation is completed.

   Now, we can print the rotated array.

- The above algorithm can be implemented in our program using two methods-
  1. <u>Standard method</u>- In this method, we use the algorithm inside the 'main' function or the main body of the program embedded inside the 'if-else if' ladder. One drawback of implementing such a method is that we have to code the complete algorithm every time we want to run it in our program.
  2. <u>Using functions</u>- In this method, we write the algorithm inside the function which is declared outside the main function and whenever we need the algorithm to rotate an array, we can call that function inside the main function. This eliminates the above-mentioned drawback.

III. Algorithm and Illustration

Let's take an array ar =[1, 2, 3, 4, 5, 6].

A. Right rotation

We begin with storing the last element of the array in another variable, say, 't'. Then each and every element of the array is shifted towards its very next right index by executing 'for' loop in this way :

ar[5] = ar[4]

Now the array holds the following values:

ar =[1, 2, 3, 4, 5, 5]

The 'for' loop is executed again for the second element and the array holds the following values now:

ar[4] = ar[3]

ar =[1, 2, 3, 4, 4, 5]

The subsequent values processed by the compiler after executing 'for' loop repeatedly would be:

ar[3] = ar[2]

ar =[1, 2, 3, 3, 4, 5]

ar[2] = ar[1]

ar =[1, 2, 2, 3, 4, 5]

ar[1] = ar[0]

ar =[1, 1, 2, 3, 4, 5]

Since elements at ar[0] and ar[1] are the same, we assign the value of ar[5] (that we have initially assigned to another variable 't') to ar[0].

The final output after one rotation gets completed, would be:

ar =[6, 1, 2, 3, 4, 5]

This result is obtained because the right rotation of an array means shifting each element of the array towards the right without exhausting the bounds of the array.

B. Left rotation

We begin with storing the first element of the array in another variable, say, 't'. Then each and every element of the array is shifted towards its very next right index by executing the 'for' loop in this way :

ar[0] = ar[1]

Now the array holds the following values:

ar =[2, 2, 3, 4, 5, 6]

The 'for' loop is executed again for the second element and the array holds the following values now:

ar[1] = ar[2]

ar =[2, 3, 3, 4, 5, 6]

The subsequent values processed by the compiler after executing 'for' loop repeatedly would be:

ar[2] = ar[3]

ar =[2, 3, 4, 4, 5, 6]

ar[3] = ar[4]

ar =[2, 3, 4, 5, 5, 6]

ar[4] = ar[5]

ar =[2, 3, 4, 5, 6, 6]

Since elements at ar[5] and ar[4] are the same, we assign the value of ar[0] (that we have initially assigned to another variable 't') to ar[5].

The final output after one rotation gets completed, would be:

ar =[2, 3, 4, 5, 6, 1]

This result is obtained because the left rotation of an array means shifting each element of the array towards the left without exhausting the bounds of the array.

In order to make multiple rotations happen, we would execute a 'for' loop from i= 1 to i<=n, where n is the desired number of rotations for the array.

IV. Algorithm Analysis

<u>Running time of the algorithm</u>:

Here, the running time of the algorithm for various sizes of the array and for different numbers of rotations has been recorded -

Size of array	No. of rotations			
	100	10000	1000000	100000000
7	0.003033	0.005014	0.015202	0.393884
9	0.005468	0.005567	0.012255	0.63772
11	0.00617	0.006818	0.023402	0.653512

Table-1: Time (in seconds) for right rotation of an array

V. Conclusion

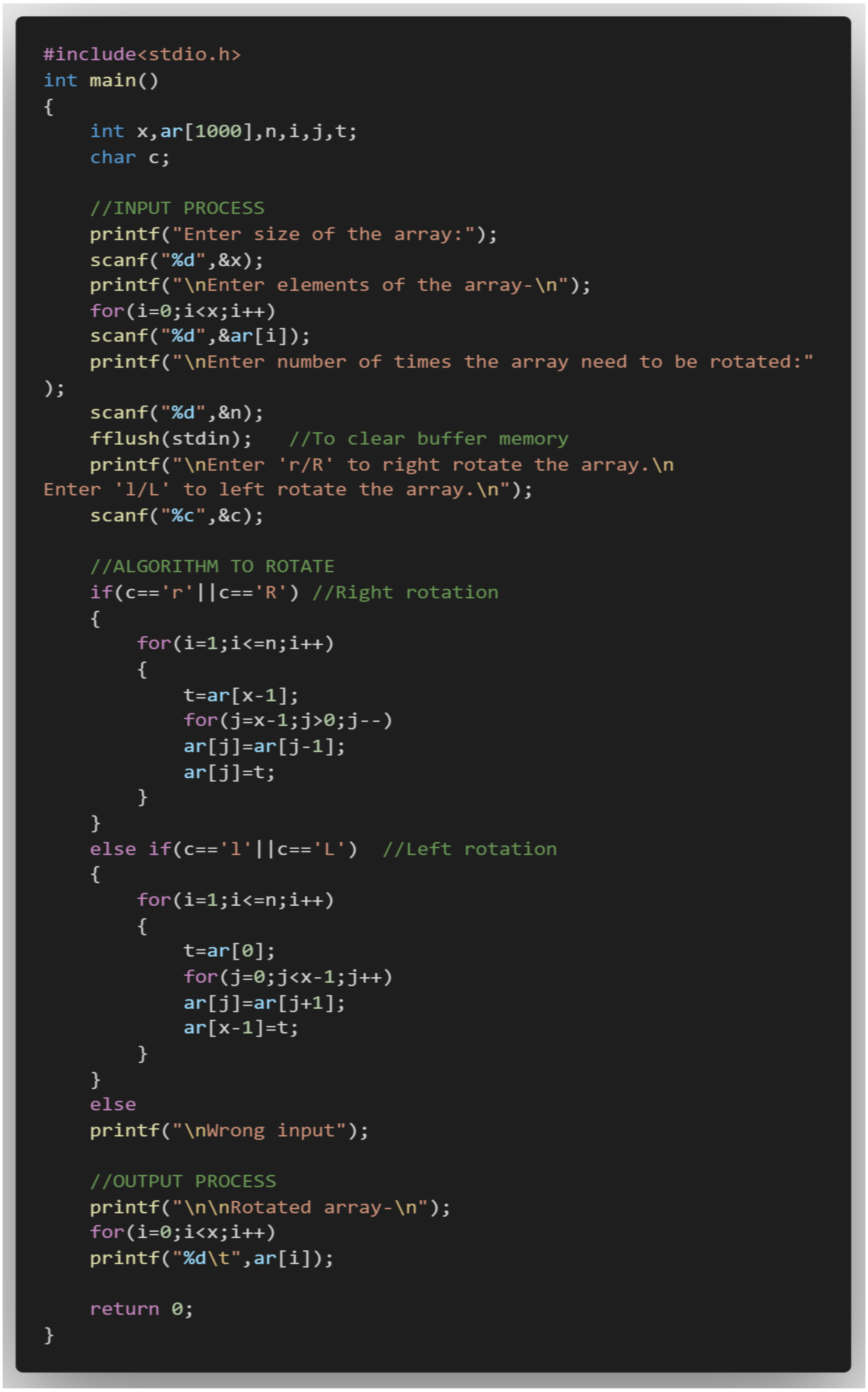
The Algorithm that we have explained executes perfectly, both for right as well as for left rotation as many times as the user wants to rotate the array. Also, if the user needs to rotate only a single array, then, the standard method is efficient. But, if we have to rotate many arrays, then, rotation using functions is more efficient because we don't have to write the complete algorithm again and again to rotate each and every array. We can simply call the rotation function.

VI. References

1. https://www.tutorialspoint.com/c-program-for-program-for-array-rotation
2. https://www.javatpoint.com/c-program-to-right-rotate-the-elements-of-an-array
3. https://www.geeksforgeeks.org/array-rotation/
4. https://www.codechef.com/ide

Appendix

Code (without using functions) for implementation of this paper is given below:

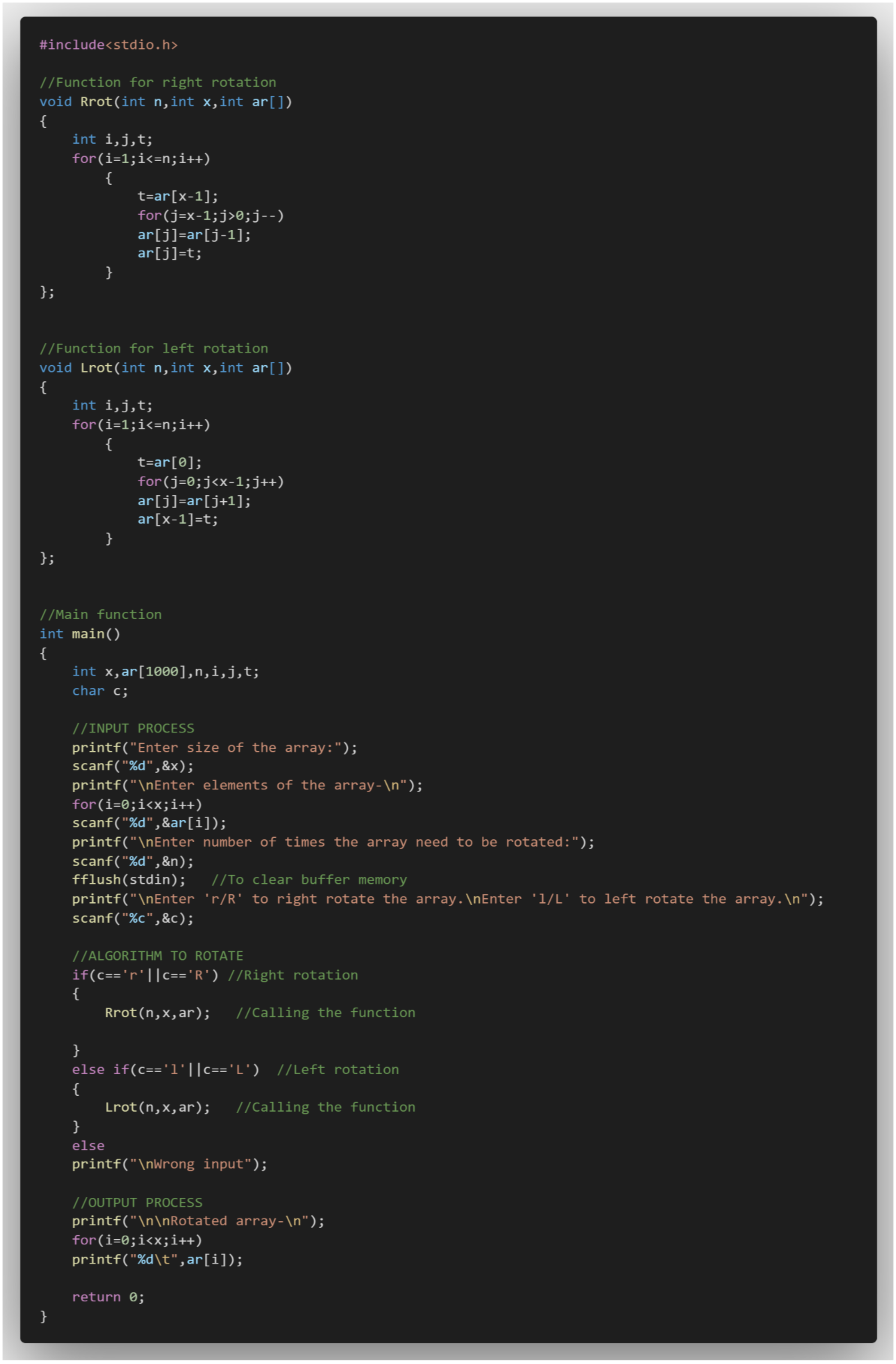


```c
#include<stdio.h>
int main()
{
    int x,ar[1000],n,i,j,t;
    char c;

    //INPUT PROCESS
    printf("Enter size of the array:");
    scanf("%d",&x);
    printf("\nEnter elements of the array-\n");
    for(i=0;i<x;i++)
    scanf("%d",&ar[i]);
    printf("\nEnter number of times the array need to be rotated:"
);
    scanf("%d",&n);
    fflush(stdin);   //To clear buffer memory
    printf("\nEnter 'r/R' to right rotate the array.\n
Enter 'l/L' to left rotate the array.\n");
    scanf("%c",&c);

    //ALGORITHM TO ROTATE
    if(c=='r'||c=='R') //Right rotation
    {
        for(i=1;i<=n;i++)
        {
            t=ar[x-1];
            for(j=x-1;j>0;j--)
            ar[j]=ar[j-1];
            ar[j]=t;
        }
    }
    else if(c=='l'||c=='L')  //Left rotation
    {
        for(i=1;i<=n;i++)
        {
            t=ar[0];
            for(j=0;j<x-1;j++)
            ar[j]=ar[j+1];
            ar[x-1]=t;
        }
    }
    else
    printf("\nWrong input");

    //OUTPUT PROCESS
    printf("\n\nRotated array-\n");
    for(i=0;i<x;i++)
    printf("%d\t",ar[i]);

    return 0;
}
```

Listing 1. Code for this paper

Appendix

Code (using functions) for implementation of this paper is given below:



```c
#include<stdio.h>

//Function for right rotation
void Rrot(int n,int x,int ar[])
{
    int i,j,t;
    for(i=1;i<=n;i++)
        {
            t=ar[x-1];
            for(j=x-1;j>0;j--)
            ar[j]=ar[j-1];
            ar[j]=t;
        }
};


//Function for left rotation
void Lrot(int n,int x,int ar[])
{
    int i,j,t;
    for(i=1;i<=n;i++)
        {
            t=ar[0];
            for(j=0;j<x-1;j++)
            ar[j]=ar[j+1];
            ar[x-1]=t;
        }
};


//Main function
int main()
{
    int x,ar[1000],n,i,j,t;
    char c;

    //INPUT PROCESS
    printf("Enter size of the array:");
    scanf("%d",&x);
    printf("\nEnter elements of the array-\n");
    for(i=0;i<x;i++)
    scanf("%d",&ar[i]);
    printf("\nEnter number of times the array need to be rotated:");
    scanf("%d",&n);
    fflush(stdin);   //To clear buffer memory
    printf("\nEnter 'r/R' to right rotate the array.\nEnter 'l/L' to left rotate the array.\n");
    scanf("%c",&c);

    //ALGORITHM TO ROTATE
    if(c=='r'||c=='R') //Right rotation
    {
        Rrot(n,x,ar);   //Calling the function

    }
    else if(c=='l'||c=='L')  //Left rotation
    {
        Lrot(n,x,ar);   //Calling the function
    }
    else
    printf("\nWrong input");

    //OUTPUT PROCESS
    printf("\n\nRotated array-\n");
    for(i=0;i<x;i++)
    printf("%d\t",ar[i]);

    return 0;
}
```

Listing 2. Code for this paper (using functions)