

***Instructions:***

- This lab is based on the topics: Functions, Recursion.
- You should save each program with the file name as specified against each problem as <Lab#>\_<Assignment#>\_<Roll#>.c. For example, **06\_01\_23CS10006.c** to save Program to 1<sup>st</sup> assignment in Lab 6 with Roll Number 23CS10006
- You should upload each program to the Moodle system. Also, copy+paste your programs to the text window on the test page.
- There will be no evaluation and hence grade, if you don't submit your .c files to the Moodle server. Use **emacs** editor and **gcc command** in terminal to run the following programs.
- Document your programs meaningfully using appropriately named variable and sufficient amount of comments. Documentation and proper code indentation carry marks.
- Do not use advanced concepts like structures anywhere in your code.
- **The top two lines of your programs must contain the following information:**
  - //Roll No.: <Type in your roll no.>
  - //Name: <Type in your name>

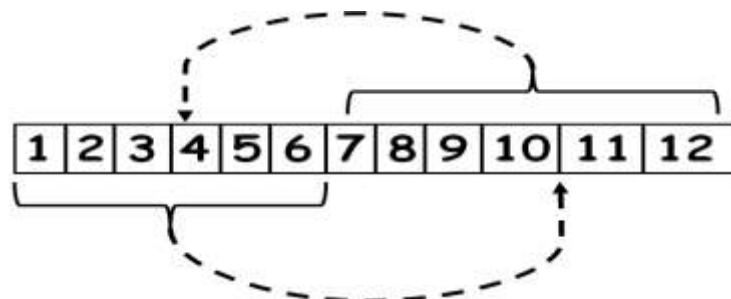
1. Write a C program that would help perform various types of operations on an integer array. First declare a global integer array of size **100** named **arr** and an integer variable named **size** that indicates the present size (number of valid entries) of the array. For example, if **size** is **50**, only **arr[0]** to **arr[49]** contain valid values and **arr[50]** to **arr[99]** do not contain valid values and are to be ignored. Your program should have the following functions:

- a. **main:** Perform the following in an infinite loop: a) display the following menu choice, b) read the user choice and c) call the appropriate function based on the user choice. It should exit when the user enters either **0** or any number greater than or equal to **9**.


**Menu choices:**

- i. 1 ---- Fill
- ii. 2 ---- Exchange
- iii. 3 ---- Rotate right
- iv. 4 ---- Rotate left
- v. 5 ---- Segregate
- vi. 6 ---- Search
- vii. 7 ---- Fold
- viii. 8 ---- Analyze
- ix. 0, 9 or any higher number ---- Exit


- b. **fill:** This function should prompt the user to enter the size of the array to be used and store the entered value in the global variable **size**. It should only accept value of **size** in the range **[1,100]**, if an entered value is outside this range, it should keep on prompting until, the user enters a valid value within the range. It should then fill the array with random integral numbers in the range **[100,999]** generated by appropriately calling the **rand()** library function. It should finally display the array contents nicely formatted.
- c. **exchange:** This function should first display the elements of the original array. It should then exchange the lower and upper halves of the array (illustrated in the following diagram) and display the resultant array. If the number of elements in the array is odd, the middle element remains undisturbed. Do not use an extra array for implementing the exchange operation, **otherwise 50% penalty will apply**.



- d. **rotate right:** It should first display the elements of the original array. It should then right rotate the array one position (the last element should become the first element) and then display the resultant array. An example is given in the following.

5 7 21 33 15 57 17 98  
 Rotate Right  
 98 5 7 21 33 15 57 17

- e. **rotate left:** It should first display the elements of the original array. It should then left rotate the array one position (the first element should become the last element) and then display the resultant array. An example is given in the following.

5 7 21 33 15 57 17 98  
 Rotate Left  
 7 21 33 15 57 17 98 5

- f. **segregate:** It should first display the elements of the original array. Then, it should segregate the odd and even numbers in the array without disturbing their relative position among themselves. For example, if the original array contents is [8,7,13,17,20,5,40] then the segregate function should change it as [8,20,40,7,13,17,5].
- g. **search:** It should first display the elements of the original array. It should then ask the user to enter an integer value to be searched. It should then perform a linear search for the element and display the array index at which the element is located. If the element is not present in the array, it should display "Not found."
- h. **Fold:** This function should fold the array in the middle and add the corresponding values. If the array is [1,2,3,4,5] after folding it should be [6,6,3]. That is, it should fold it in middle and add the corresponding values as shown:

1 2 3 4 5 →  
 5 4 3  
 1 2  
 → 6 6 3

If the array is [1,2,3,4,5,6], after folding it should be [7,7,7]. That is, it should fold it in middle and add the corresponding values as shown:

1 2 3 4 5 6 →  
 6 5 4  
 1 2 3  
 → 7 7 7

- i. **analyze:** This function should first display the elements of the original array. It should then find and display all increasing sequence of consecutive numbers and also display the largest increasing sequence. It may so happen that there are no increasing sequences of numbers. In this case, it should display an appropriate message. If there are multiple largest sequences, displaying any one of these should be fine. If there are no sequences of increasing numbers, it should display "No increasing sequence of numbers present". [Hint: To determine the largest increasing sequence, you need to define 2 integer variables pos, and isize to keep track of the position of the array at which the largest sequence occurs and also the sequence size and update these as required. Further, you can use another pair of variables: mpos and msize to keep track of the position of the maximum increasing sequence and its size.]

**Example:** if the array elements are [1,5,3,2,7,15,23,12,15,21], then it should display: The increasing sequences are: (1,5), (2,7,15,23), (12,15,21) and the largest increasing sequence is (2,7,15,23).

2. In the main function, declare an integer array of size **30**. Fill it with random integers in the range **[20,30]**. Display the array elements properly formatted. Next call recursive functions **sum()**, **max()**, and **reverse ()**. Display the results returned by the called functions.

**sum:** This **recursive** function should take an integer array and its size (int) as its arguments. It should add all the elements of the array and return the value.

**max:** This **recursive** function should take an integer array and its size (int) as its arguments. It should return the largest element stored in the array.

**reverse:** This **recursive** function should take an integer array, its first position index (int), and its last position index (int) as its arguments. It should reverse the array.

**If any of the three functions is non-recursive, 50% penalty will apply.**