National University of Computer and Emerging Sciences



Lab Manual 03

**Section BCS 2A**

**2nd April, 2021**

Object Oriented Programming

|  |  |
| --- | --- |
| Course Instructor | Miss Abeeda Akram |
| Lab Instructor (s) National University of Computer and Emerging Sciences    Lab Manual 02  Object Oriented Programming  Department of Computer Science  FAST-NU, Lahore, Pakistan  1.1 Objectives  After performing this lab, students shall be able to:   Have an improved understanding of pointers.   Create and manipulate 1D dynamic array.   Allocation and de-allocation of 1D array.   Passing dynamic arrays into functions. | Ms. Abiha Aftab  Mr. Dilawar Shabbir |
| Section | BCS – 2A |
| Semester | Spring 2021 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

## Objectives

After performing this lab, students shall be able to:

* Have an improved understanding of 2D array pointers.
* Create and manipulate 2D dynamic array.
* Allocation and de-allocation of 2D array.
* Sorting in arrays

**TASK 1:**

Write a menu driven C++ program to do following operation on two-dimensional array A of

size m x n. You should use user-defined functions which accept 2-D array A, and its size m

and n as arguments. The options are:

• To input elements into matrix of size m x n

• To display elements of matrix of size m x n

• Sum of all elements of matrix of size m x n

• To display row-wise sum of matrix of size m x n

• To display column-wise sum of matrix of size m x n

• To create transpose of matrix B of size n x m

**TASK 2:**

Write a function int\*\* AddMatrix(int\*\* A, int\*\* B, const int& rows, const int& cols) that

creates a new matrix result of size rowsxcols, adds matrix A and B and saves the result in

matrix result and returns the result pointer. Test your function in main().

**Task 3:**

Write a program that will read scores into an array. The size of the array should be input by the user (dynamic array). It will call a function that will sort (using a bubble sort) the scores in ascending order. The values are then printed in this sorted order.

A bubble sort starts at the top of the list. Each element is compared to the next. If it is greater than the next element, then swap the two. Pass through the list as many times as necessary to sort it. Usually the number of passes required is equal to (**number of elements – 1**). The smallest value bubbles up to the top of the list while the largest value sinks to the bottom.

**8 6 11 3 15 5**

**swap**

**6 8 11 3 15 5**

**okay**

**6 8 11 3 15 5**

**swap**

**6 8 3 11 15 5**

**okay**

**6 8 3 11 15 5**

**swap**

**6 8 3 11 5 15**

**Pass 1:**

**Pass 2:**

**6 8 3 11 5 15**

**okay**

**6 8 3 11 5 15**

**swap**

**6 3 8 11 5 15**

**okay**

**6 3 8 11 5 15**

**swap**

**6 3 8 5 11 15**

**Pass4:**

**3 6 5 8 11 15**

**okay**

**3 6 5 8 11 15**

**swap**

**3 5 6 8 11 15**

**Pass3:**

**swap**

**6 3 8 5 11 15**

**okay**

**3 6 8 5 11 15**

**swap**

**3 6 8 5 11 15**

**3 6 5 8 11 15**

**Scan 5 will do no swapping,**

**so the algorithm terminates**

**after this pass.**

**Pass5:**