INFORMATION TECHNOLOGY UNIVERSITY

Software Engineering Department - ITU

SE200L: Data Structures & Algorithms Lab

Course Instructor: Usama Bin Shakeel	**Dated: 28/08/2024**
Teaching Assistant: Zainab Bashir & Ryan Naveed	**Semester: Fall 2024**
Lab Engineer: Sadia Ijaz	**Batch: BSSE2023B**

Lab 2. Implementing a 2D Arrays by Utilization of Pointers and Dynamic Memory Allocation

Name	Roll number	Report (out of 35)

Checked on: _____

Signature: _____

1.1 Objective

The objective of this lab is to practice problems related to pointers and dynamic memory allocation.

1.2 Equipment and Component

Component Description	Value	Quantity
Computer	Available in lab	1

1.3 Conduct of Lab

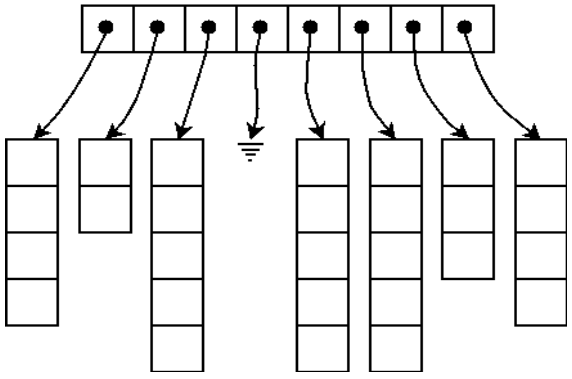
1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

1.4 Theory and Background

A pointer is a variable that stores the memory address of another variable. The type of data a pointer can point to must match the data type of the variable it references. Arrays can be created dynamically, allowing their size to be modified during runtime. Dynamic arrays allocate a contiguous block of memory for their elements, which can be resized during the program's execution.

A **2D dynamic array** is an array of arrays, where each row can be dynamically resized, offering flexibility for handling data that doesn't fit into fixed-size structures. These arrays can grow or shrink in both dimensions (rows and columns) based on the program's needs. For example, a 2D dynamic array can expand its size by allocating additional memory when new elements are added.

A **2D pointer** is a pointer that points to a pointer, which in turn points to another pointer, forming a multi-level indirection. In the context of a 2D array, a 2D pointer can be used to create and manage a dynamic array of pointers. This allows you to create a structure where each row of the array is dynamically allocated, and the number of columns in each row can vary independently.



Templates are a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one.

INFORMATION TECHNOLOGY UNIVERSITY

Lab Tasks

Task 1

In the previous lab, you implemented a dynamic 1D array using templates. In this lab, you will extend this concept to manage 2D arrays dynamically. Modify and extend the class **MyArray** to handle 2D array with dynamic memory allocation. The data type of the array will be taken from templates. The class should handle basic operations such as appending, prepending, expanding rows, displaying elements, and sorting rows using bubble sort.

Class Name: MyArray

Data Members:

- **int row_size**: A fixed size of 5 for the number of rows, which cannot be changed.
- **int* col_size**: An array to store the number of columns for each row. Initially set to 0 for all rows. Column size and capacity are the same here.
- **T** array**: A 2D dynamic pointer that will point to the array data.

Member Functions:

1. **Constructor**: Initializes an empty 2D array with row size set to 5 and column sizes set to 0 for each row.
   ```
   MyArray() {
   }
   ```

2. **Destructor**: Frees the dynamically allocated memory.
   ```
   ~MyArray() {
   }
   ```

3. **Append Function**: Adds an element at the end of the specified row.
   ```
   void append(T value, int row_num) {
   }
   ```

4. **Prepend Function**: Adds an element at the start of the specified row.
   ```
   void prepend(T value, int row_num) {
   }
   ```

5. **Expand Function**: Expands the specified row by one element. it will be called inside append and prepend function.
   ```
   void expand(int row_num) {
   }
   ```

6. **Delete From Last Function**: Deletes the last element of the given row and shrinks its size.
   ```
   Void delete_from_last (int row_num){
   }
   ```

7. **Delete From Start Function**: Deletes the first element of the given row and shrinks its size.
   ```
   void delete_from_start (int row_num){
   }
   ```

8. **Display Function**: Displays all rows, leaving a space (endl) if a row is empty.
   ```
   void display() {
   }
   ```

9. **Bubble Sort Function**: Sorts the specified row using the bubble sort algorithm.
   ```
   void bubbleSort(int row_num) {
   }
   ```

10. **Get Row Size Function**: Returns the fixed row size.
    ```
    int getRowSize() {
    }
    ```

11. **Get Column Size Function**: Returns the column size for a specific row.
    ```
    int* getColSize() {
    }
    ```

12. **Get Array Function**: Returns a pointer to the 2D array.
    ```
    T** getArray() {
    }
    ```

Assessment Rubric for Lab

Performance metric	CLO	Able to complete the task over 80% (4-5)	Able to complete the task 50-80% (2-3)	Able to complete the task below 50% (0-1)	Marks
1. Realization of experiment	1	Executes without errors excellent user prompts, good use of symbols, spacing in output. The testing has been completed.	Executes without errors, user prompts are understandable,minimum use of symbols or spacing in output. Some testing has been completed.	Does not execute due to syntax errors, runtime errors, user prompts are misleading or non- existent. No testing has been completed.	
2. Conducting experiment	1	Able to make changes and answer all questions.	Partially able to make changes and few incorrect answers.	Unable to make changes and answer all questions.	
3. Computer use	2	Document submission timely.	Document submission late.	Document submission not done.	
4. Teamwork	3	Actively engages and cooperates with other group member(s) in an effective manner.	Cooperates with other group member(s) in a reasonable manner but conduct can be improved.	Distracts or discourages other group members from conducting the experiment	
5. Laboratory safety and disciplinary rules	3	Code comments are added and do help the reader to understand the code.	Code comments are added and do not help the reader to understand the code.	Code comments are not added.	
6. Data collection	3	Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap.	Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables.	Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy.	
7. Data analysis	4	Solution is efficient, easy to understand, and maintain.	A logical solution that is easy to follow but it is not the most efficient.	A difficult and inefficient solution.	
Total (out of 35):					