|  |
| --- |
| **Software Engineering Department - ITU** |
| **SE200L: Data Structures & Algorithms Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 21/08/2024** |
| **Teaching Assistant: Zainab Bashir & Ryan Naveed** | **Semester: Fall 2024** |
| **Lab Engineer: Sadia Ijaz** | **Batch: BSSE2023B** |

# **Lab 1. Implementing Arrays by Utilization of Pointers and Dynamic Memory Allocation**

|  |  |  |
| --- | --- | --- |
| **Name** | **Roll number** | **Report**  **(out of 35)** |
|  |  |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Objective**

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

## **Equipment and Component**

|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **Conduct of Lab**

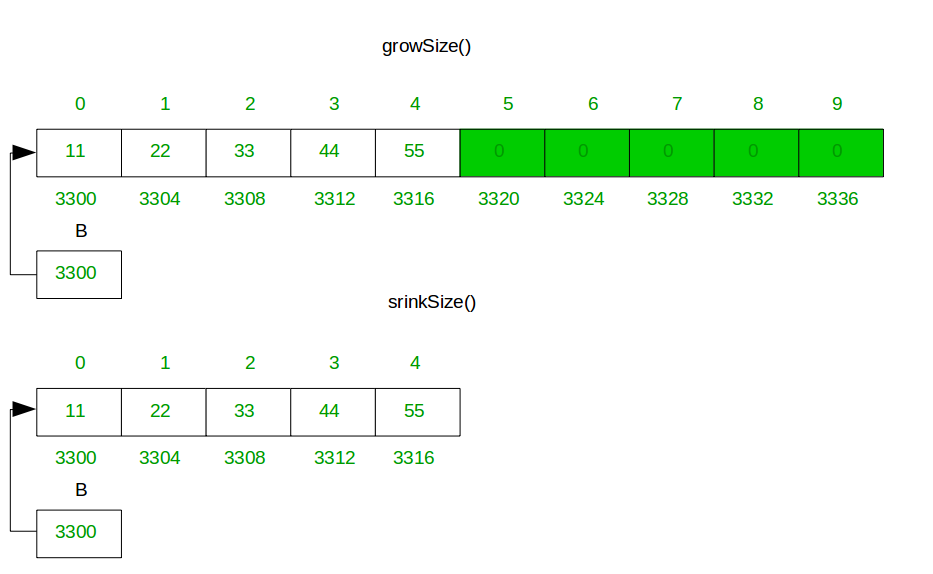
1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

## **Theory and Background**

The variable that is used to hold the memory address of another variable is called a **pointer** variable or simply a pointer. The data type of the variable (whose address a pointer is to hold) and the pointer variable must be the same. A pointer variable is declared by placing an asterisk (\*) after the data type or before the variable name in the data type statement. E.g., if the pointer variable “p” is to hold the memory address of an integer variable it is declared as:

**int \*p;**

A **dynamic array** is quite similar to a regular array, but its size is modifiable during program runtime. Dynamic Array elements occupy a contiguous block of memory. Once an array has been created, its size cannot be changed. However, a dynamic array is different. A dynamic array can expand its size even after it has been filled. During the creation of an array, it is allocated a predetermined amount of memory. This is not the case with a dynamic array as it grows its memory size by a certain factor when there is a need.



**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

**Lab Tasks**

**Task 1**

Design a class named **MyArray** to keep a list of similar data in the form of an array, the data type of the array will be taken from templates.

**Data Members:**

* **data**: A dynamically allocated array to store the data.
* **size**: An integer to track the number of elements in the list.
* **capacity**: An integer to track the maximum number of elements the array can hold before resizing.

**Member Functions:**

**// Add a constructor that Initializes an empty array with a default capacity**

MyArray () {  
}

**// Add a destructor that frees the dynamically allocated memory used by the array.**

~MyArray () {  
}

**// Add function to add an element at the last index of the array**

void append (T) {  
}

**// Add function to add an element at the start index of the array**

void prepend (T) {  
}

**// Add function to contract an array by deleting a value from the last index**

void deleteFromEnd () {  
}

**// Add function to contract an array by deleting a value from the first index**

void deleteFromStart () {  
}

**// Add function to display the elements of the array**

void display () {  
}

**// Also add functions to get size, capacity, and array**

int get\_size () {  
}

int get\_capacity () {  
}

T\* get\_array () {  
}

|  |
| --- |
|  |

**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):** | | | | |  |