INFORMATION TECHNOLOGY UNIVERSITY

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

Course Instructor: Usama Bin Shakeel	**Dated: 25/09/2024**
Teaching Assistant: Zainab Bashir & Ryan Naveed	**Semester: Fall 2024**
Lab Engineer: Sadia Ijaz	**Batch: BSSE2023B**

Lab 6. Stack and Queue Implementation using dynamic arrays

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

Checked on: ______________________________

Signature: __________________________________

1.1 Objective

The objective of this lab is to practice problems related to Stack and Queue implementation.

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

Stack

A **stack** is a linear data structure that follows the **Last-In-First-Out (LIFO)** principle. This means the last element added (pushed) onto the stack is the first one to be removed (popped) from it. Think of a stack of plates; you add plates to the top and also remove from the top.

Implementing Stack Using Dynamic Arrays

- **Dynamic Array**: An array that can grow or shrink in size at runtime.
- **Top Pointer/Index**: Keeps track of the last inserted element.
- **Resizing**: If the stack becomes full, we can resize the array by doubling its capacity.

Queue

A **queue** is a linear data structure that follows the **First-In-First-Out (FIFO)** principle. This means the first element added to the queue is the first one to be removed. Think of a line of people waiting; the first person in line is the first to be served.

Implementing Queue Using Dynamic Arrays

- **Dynamic Array**: Stores the elements of the queue.
- **Front and Rear Indices**: `front` points to the first element, `rear` points to the position where the next element will be inserted.
- **Circular Buffer**: To efficiently utilize the array space, we wrap around the indices when they reach the end of the array.
- **Resizing**: If the queue becomes full, we can resize the array by doubling its capacity and realigning the elements.

Lab Tasks

Task 1:

Implement the **Stack** Class Using Dynamic Arrays

Data Members:

- **int* array**: Pointer to the dynamic array storing the stack elements.
- **int size**: The current size of the stack.
- **int capacity:** The current capacity of stack.

Member Functions to Implement:

1. **Constructor**: Initializes the stack with an initial capacity 10, allocates the dynamic array with the initial capacity, sets size to 0.
2. **Destructor**: Deallocates the dynamic array to prevent memory leaks.
3. **bool isEmpty()**: Returns true if the stack is empty; otherwise, false.
4. **void push(int data)**: If the array is full, resizes it by doubling its capacity. Updates top and size accordingly.
5. **void pop()**: Removes the top element of the stack and resize it. If the stack is empty, handle the error appropriately.
6. **int getSize()**: Returns the number of elements in the stack.
7. **void clear()**: Removes all elements from the stack.
8. **void printStack():** Prints all the elements in the stack from top to bottom.

Task 2:

Implement the **Queue** Class Using Dynamic Arrays

Data Members:

- **int* array:** Pointer to the dynamic array storing the queue elements.
- **int size:** The current size of the queue.
- **int capacity:** The current capacity of the queue.

Member Functions to Implement:

1. **Constructor**: Initializes the stack with an initial capacity 10, allocates the dynamic array with the initial capacity, sets size to 0 and front and rear to 0.
2. **Destructor**: Deallocates the dynamic array to prevent memory leaks.
3. **bool isEmpty():** Returns true if the queue is empty; otherwise, false.
4. **void enqueue(int data):** Adds an element to the rear of the queue. If the array is full, resize it by doubling its capacity.

5. **void dequeue():** Removes the front element of the queue. If the queue is empty, handle the error appropriately.
6. **int getSize():** Returns the number of elements in the queue.
7. **void printQueue():** Prints all the elements in the queue from front to rear.

Please read the following instructions carefully:

1. ***Do Not Modify test.cpp:*** *You are strictly prohibited from making any changes to the test.cpp file. This file is designed to test your implementation and any modifications will lead to the assignment being graded as zero.*
2. ***Class Definitions:*** *All class definitions and implementations must be provided solely within the files functions.h and functions.cpp. You are not allowed to create any additional files for your class definitions or implementations.*

Any deviation from these rules, including creating additional files or modifying the test.cpp file, will result in your assignment receiving a grade of zero.

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):					