INFORMATION TECHNOLOGY UNIVERSITY

# Department of Computer and Software Engineering – ITU

## SE200T: Data Structures & Algorithms

| | |
|---|---|
| **Course Instructor: Usama Bin Shakeel** | **Dated: 28th Aug 2024** |
| **Teaching Assistant: Zainab, Sadia & Ryan** | **Semester: Fall 2024** |
| **Session: 2024-2028** | **Batch: BSSE2023B** |

# Assignment 6. Stack and Queue Implementation and Manipulation using Linked List

| Name | Roll number | Obtained Marks/35 |
|---|---|---|
| | | |

Checked on: _____

Signature: _____

**Submission:**

- Email instructor or TA if there are any questions. You cannot look at others' solutions or use others' solutions, however, you can discuss it with each other. Plagiarism will be dealt with according to the course policy.
- Submission after due time will not be accepted.

**In this assignment you have to do following tasks:**

**Task 1:** Ensure that you have installed all three softwares in your personal computer (Github, Cygwin & CLion). Now, accept the assignment posted in the classroom (e.g Google, LMS etc) and after accepting, clone the repository to your computer. Make sure you have logged into the github app with your account.

**Task 2:** Open Cygwin app, Move to your code directory with following command "cd <path_of_folder>", <path_of_folder> can be automatically populated by dragging the folder and dropping it to the cygwin window.
Run the code through Cygwin, use command "make run", to get the output of the code

**Task 3:** Solve the given problems, write code using **CLion** or any other IDE.
**Task 4:** Keep your code in the respective git cloned folder.
**Task 5:** Commit and Push the changes through the Github App
**Task 5**: Write the code in separate files **(as instructed)**. Ensure that file names are in lowercase (e,g **main.cpp**).

**Task 6:** Run '**make run**' to run C++ code
**Task 7:** Run '**make test**' to test the C++ code
Write code in functions, after completing each part, verify through running code using **"make run"** on Cygwin. Make sure to test the code using **"make test".**

# Assignment Tasks

## Task 1

**Implement the Node Class**

The **Node** class represents a single node in the linked list. It will store an integer value and a pointer to the next node.

**Data Members:**

- int data: The data stored in the node.
- Node* next: Pointer to the next node in the stack.

**Member Functions to Implement:**

1. **Constructor**: Initializes the node with a data value and sets the next pointer to nullptr.
2. **Destructor**: Cleans up resources (if any).
3. **void setNext(Node* nextNode):** Sets the next pointer to the provided node.
4. **Node* getNext():** Returns the next node pointer.
5. **void setData(int dataValue):** Sets the data of the node.
6. **int getData():** Returns the data of the node.

## Task 2

**Implement the Stack Class**

The **Stack** class manages the stack operations and maintains the linked list of Node objects.

**Data Members:**

- **Node* top:** Pointer to the top node of the stack.
- int count: The number of elements in the stack.

**Member Functions to Implement:**

1. **Constructor**: Initializes the stack with top set to nullptr and count to 0.
2. **Destructor**: Ensures all nodes are properly deleted when the stack is destroyed.
3. **bool isEmpty():** Returns true if the stack is empty; otherwise, false.

4. **void push(int data):** Adds a new node with the given data to the top of the stack.

5. **void pop():** Removes and returns the data from the top node of the stack. If the stack is empty, handle the error appropriately (e.g., throw an exception or return a sentinel value).

6. **int peek():** Returns the data from the top node without removing it. If the stack is empty, handle the error appropriately.
7. **int size():** Returns the number of elements in the stack.
8. **void clear():** Removes all elements from the stack.
9. **void printStack():** Prints all the elements in the stack from top to bottom.

# Task 3

**Implement the Queue Class**

The **Queue** class manages the queue operations and maintains the linked list of Node objects.

**Data Members:**

- **Node* top:** Pointer to the top node of the queue.
- int count: The number of elements in the queue.

**Member Functions to Implement:**

10. **Constructor**: Initializes the stack with top set to nullptr and count to 0.
11. **Destructor**: Ensures all nodes are properly deleted when the queue is destroyed.
12. **bool isEmpty():** Returns true if the queue is empty; otherwise, false.
13. **void enqueue(int data):** Adds a new node with the given data to the top of the stack.

14. **void dequeue():** Removes and returns the data from the top node of the queue.
15. **int size():** Returns the number of elements in the queue.
16. **void printQueue():** Prints all the elements in the queue

*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 Assignment

| 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 | 3 | 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 | 2 | 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 | 4 | Document submission timely. | Document submission late. | Document submission not done. | |
| 4. Teamwork | 4 | 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 | 2 | 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 | 2 | 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 | 3 | 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):** | | | | | |