Data Structures and Algorithms — Lab 5

Topic

Understanding the Concept of Queue (First In, First Out) and Circular Queue through Basic Applications

Objectives

- Understand the purpose of Queue and Circular Queues and their First In, First Out (FIFO) approach.
- Explore the use of enqueue and dequeue operations in Queue and Circular Queue.
- Practice converting and evaluating expressions using Queues and Stacks for expression manipulation tasks.
- Familiarize with menu-driven programs to apply Queue concepts in simulations.

Outcomes

1. Develop a comprehensive understanding of creating and using Queue and Circular Queue structures to manage ordered data efficiently.
2. Gain experience implementing enqueue and dequeue operations and understanding their role in managing data in a FIFO manner.
3. Enhance problem-solving abilities by designing functions that meet specific operational requirements, such as customer management and expression evaluation.
4. Gain expertise in expression evaluation by converting infix expressions to prefix expressions and calculating their results using Queue and Stack data structures.

Content

The Queue class has the following attributes:

1. `T *arr` - This is a pointer to the array that holds the elements in the queue.
2. `int maxSize` - This is the maximum size of the queue.
3. `int currentSize` - This is the current size of the queue.

There are several pure virtual functions inside the Queue class:

1. `void enqueue(T data)` - This function is used to add data to the back of the queue.
2. `T dequeue()` - This function is used to remove and return the data at the front of the queue.
3. `T peek()` - This function is used to get the data at the front of the queue without removing it.
4. `bool isFull()` - This function is used to check if the queue is full.
5. `bool isEmpty()` - This function is used to check if the queue is empty.

The Queue class also has a constructor and a destructor:

1. `Queue(int maxQueueSize)` - This constructor takes the maximum size of the queue as a parameter and initializes the queue.
2. `~Queue()` – The destructor makes sure that there is no memory leakage.

To use the `Queue<T>` class, you need to create your own class, `MyQueue<T>`, which inherits from the `Queue<T>` class. `MyQueue<T>` should be a template class that takes a type `T` as a parameter.

Task Instructions

Students are required to complete the following tasks during lab time. Create a private repository on your GitHub accounts. The name of the repository should be **Lab-5-DSA**. All files should be uploaded to this repository, including the header and cpp files. Please note that the name of class-related files should be the same as the class name. If there is only 1 task, we can name the main file Task_1.cpp or Task.cpp. We recommend you work in .h files for classes only since we must work on templates.

Task 1

Use Non-Circular Queue (Linear Queue) to solve the problem.

Write a C++ program that simulates a quiz competition. You would require two queues where you would store student data. The attributes of the student class in this context could be:

1. `name`: A string to store the name of the student.
2. `score`: An integer to store the score obtained by the student in the quiz competition.
3. `timeTaken`: An integer to store the time taken by the student to complete the quiz.

At the start of the competition, all the students are standing in the queue for their turn. Each student has a name entered. The score and time taken are zero.

When the quiz starts, one student should be dequeued at a time from the waiting queue, and their score and time taken should be generated and enqueued in a new queue (For simulation, your program should randomly generate the score and time taken for each student using the `rand()` function.

After all the students have completed the quiz, the program should display the contents of the new queue, which should contain the names of the students along with their scores and time taken.

The program should provide a menu-based interface with the following options:

1. Add a student to the waiting queue
2. Start the quiz and generate scores
3. Display the scores, name and time taken by the students
4. Exit

Explanation of the menu:

1. When the user selects option 1, the program should prompt the user to enter the name of the student and enqueue them in the waiting queue.
2. When the user selects option 2, the program should dequeue one student at a time from the waiting queue, generate their score and time taken, and enqueue them in the new queue. You will use this option to dequeue one student at a time. Once all students have completed the quiz (when queue 1 becomes empty), the program should display a message stating that the quiz has ended.
3. When the user selects option 3, the program should display the contents of the new queue, which should contain the names of the students along with their scores and time taken.
4. The program should exit when the user selects option 4.

Task 2

Using Circular Queue, write a C++ program that simulates a queue of people waiting in line at a coffee shop. ***Note: If you are not able to implement the code using a Circular Queue, use a Non-Circular queue.*** The program should provide a menu-based interface with the following options:

1. Add a customer to the queue
2. Remove the first customer from the queue
3. Display the number of customers waiting in line
4. View the first customer in the queue
5. View the last customer in the queue
6. Exit

Explanation of the menu:

1. When the user selects option 1, the program should prompt the user to enter the name of the customer and enqueue them in the queue.
2. When the user selects option 2, the program should remove the first customer from the queue and display their name.
3. When the user selects option 3, the program should display the number of customers waiting in line.
4. When the user selects option 4, the program should display the name of the first customer in the queue.
5. When the user selects option 5, the program should display the name of the last customer in the queue.
6. The program should exit when the user selects option 6.

Task 3

Write down the code to convert an infix expression into a prefix expression.

Task 4

Input an infix expression as an input and find the result of that expression.
Consider the example below:

- Input: **`6 3 + 2 *`**
- Output: **`18`**