Assignment 3: Implementation of Queue Data Structure

January 2024, CSE 106

September 18, 2024

Task 1: Introduction to Queue

A **queue** is a linear data structure that follows the *First In, First Out (FIFO)* principle. In other words, the first element added to the queue will be the first one removed. It is similar to a real-world queue, like waiting in line: the first person to get in line is the first one to be served.

The fundamental operations on a queue include:

- **Enqueue:** Adds an element at the end of the queue.
- **Dequeue:** Removes an element from the front of the queue
- **Peek:** Returns the front or peek element without removing it.
- **Length:** Returns the length of the queue.
- **IsEmpty:** Checks if the queue is empty.

How Queue Works

Here's a basic example of how a queue works:

Example

Assume we start with an empty queue and perform the following operations:

- **Enqueue 5:** The queue now contains [5].
- **Enqueue 10:** The queue now contains [5, 10].
- **Dequeue:** Removes the element at the front (5), leaving [10].
- **Enqueue 3:** The queue now contains [5, 3].
- **Length:** Returns (2) because the queue contains 2 elements.
- **Peek:** Returns 5, i.e., the element at the front, without removing it.
- **IsEmpty:** Returns false because the queue is not empty.
- **Clear:** Empties the queue.

Visual Representation

Operation	**Stack Contents**	**Returns**
Enqueue 5	[5]	
Enqueue 10	[5, 10]	
Dequeue	[10]	5
Enqueue 3	[5, 3]	
Length	[5, 3]	2
Peek	[5, 3]	5
IsEmpty	[5,3]	False
Clear	[]	
IsEmpty	[5,3]	True

Your Task

Your task is to implement queue data type for integers in C++. **You have to provide two implementations, one using an array list, and the other using a linked list.** Your implementation should provide the following functions:

1. **enqueue(x):** Enqueues the element `x` onto the queue.
2. **dequeue():** Removes and returns the front element of the queue.
3. **peek():** Returns the front element without removing it.
4. **length():** Returns the number of elements in the queue.
5. **isEmpty():** Returns `True` if the queue is empty; otherwise, returns `False`.
6. **clear():** Remove all elements from the queue.

Input Format

The input will be taken from a file, where each line contains an operation followed by a value (if applicable).

- **1 x**: Enqueue the element x onto the end of the queue.
- **2**: Dequeue the element at the front of the queue.
- **3**: Return the front element without removing it.
- **4**: Return the number of elements in the queue.
- **5**: Return `True` if the queue is empty; otherwise, return `False`.
- **6**: Clear all elements from the queue.

Sample Input

```
1 10
1 20
1 30
1 40
2
2
3
4
5
6
```

Example Output

```
Enqueueed 10 onto the queue.
Current queue: 10
-------------------
Enqueueed 20 onto the queue.
Current queue: 10 20
-------------------
Enqueueed 30 onto the queue.
Current queue: 10 20 30
-------------------
Enqueueed 40 onto the queue.
Current queue: 10 20 30 40
```

```
-------------------
Dequeueped value: 10
Current queue: 20 30 40
-------------------
Dequeueped value: 20
Current queue: 30 40
-------------------
Peek value: 30
Current queue: 30 40
-------------------
Current queue length: 2
Current queue: 10 20
-------------------
Is queue empty? No
Current queue: 10 20
-------------------
Cleared the queue.
Current queue:
-------------------
```

Hints

- You have been given two main functions: (i) **main_array_queue.cpp** for checking array-based implementations and (ii) **main_list_queue.cpp** for checking linked-list-based implementations. You do not have to edit a single line of code in those files. If you want to give your own input, just change in **input.txt**, and you will get the output with proper messages in **output.txt**. Initially, a sample input and output are given in those files.
- You have been given two header files: (i) **QueueArray.h** for array-based implementations and (ii) **QueueLinkedList.h** for linked-list-based implementations. You have to edit only in those files at the mentioned places "`//write your codes here`".
- Please use the best practices from the 1st Offline as to when to make the array capacity double or half.
- When doing linked-list-based implementation, first understand the theory behind it, e.g., how to add or remove a node from a linked list, etc.
- Carefully handle the pointers !!!

Task 2: Implementing Stack using Queue

In this task, you are required to implement a stack data structure using only queues. You already know that stack follows the LIFO principle, whereas queue follows FIFO principle. The challenge is to make the stack operations work using only queue operations. You have to implement the `push`, `pop`, and `top` operations of stack in this task. Your stack implemented with queues will function exactly the same as Offline-2. **You are not allowed to use any additional data structures like arrays or so for the task**.

Input Format

Here also, the input will be taken from a file, where each line contains an operation followed by a value (if applicable).

- **11 x**: Push the element x onto the stack.
- **12**: Pop the top element from the stack.
- **13**: Return the top element without removing it.

Sample Input

```
11 10
11 20
11 30
11 40
12
12
13
```

Example Output

```
Pushed 10 onto the stack.
Current stack: 10
-------------------
Pushed 20 onto the stack.
Current stack: 10 20
-------------------
Pushed 30 onto the stack.
Current stack: 10 20 30
-------------------
Pushed 40 onto the stack.
Current stack: 10 20 30 40
-------------------
Popped value: 40
Current stack: 10 20 30
-------------------
Popped value: 30
Current stack: 10 20
-------------------
Top value: 20
Current stack: 10 20
-------------------
```

Hints

- You have been given a main function **main_queue_based_stack.cpp** for checking queue-based implementations. You do not have to edit a single line of code in this file. If you want to give your own input, just change in **input.txt**, and you will get the output with proper messages in **output.txt**. Initially, a sample input and output are given in those files.
- You have been given a header file for task-2: (i) **StackUsingQueue.h** for the queue-based implementations. You have to edit only in this file at the mentioned places "`//write your codes here`".

Submission Guidelines

- You are not allowed to use vector, list, or any of the STL functionalities. You have to use the knowledge of 1st and 2nd Offline.
- Create a directory with your 7-digit student ID as its name.
- Put the source files only into the directory created in step 1.
- Zip the directory (compress in .zip format; .rar, .7z or any other format is not acceptable)
- Upload the .zip file on Moodle. For example, if your student ID is 2205xxx, create a directory named 2205xxx. Put only your source files (.cpp, .h, etc.) into 2205xxx. Compress 2205xxx

into 2205xxx.zip and upload the 2205xxx.zip on Moodle. Failure to follow the above-mentioned submission guidelines may result in upto 10% penalty.

- **Please DO NOT COPY solutions from anywhere (your friends, seniors, the internet, etc.). Any form of plagiarism (irrespective of source or destination) will be penalized severely.**

Submission Deadline

September 29, 2024, 11:55 PM

Tentative Marks Distribution

ArrayList-based Implementation	**20%**
LinkedList-based Implementation	**40%**
Stack Implementation using Queues	**40%**

Version History

The document may be updated periodically to address any ambiguities, clarify instructions, or incorporate feedback from students to ensure the assignment is fully understandable and free of confusion. All changes in the document will be marked as red text with underline, along with a version change. Please keep yourself updated on the changes in the document.
Version 1.0