CS 202

Data Structures

Assignment 1

Linked Lists, Stacks, Queues

In this assignment, you are required to implement a linked list, stack, and queue. The Last part requires you to make use of these data structures to solve three problems related to the amusement park.

The course policy about plagiarism is as follows:

1. Students must not share the actual program code with other students.
2. Students must be prepared to explain any program code they submit.
3. Students cannot copy code from the Internet.
4. Students must indicate any assistance they received.
5. All submissions are subject to automated plagiarism detection.

Students are strongly advised that any act of plagiarism will be reported to the DisciplinaryCommittee.

Smart pointer:

Smart pointers are class objects that behave like built-in pointers but also manage objects that you create with new so that you don't have to worry about when and whether to delete them - the smart pointers automatically delete the managed object for you at the appropriate time.

To read an article explaining how to use smart pointers, click here.

In this assignment, you are not allowed to dynamically allocate memory. You need to make use of smart pointers to implement this assignment.

Templates:

Function templates are special functions that can operate with *generic types*. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.

In C++ this can be achieved using *template parameters*. A template parameter is a special kind of parameter that can be used to pass a type as argument: just like regular function parameters can be used to pass values to a function, template parameters allow to pass also types to a function.
These function templates can use these parameters as if they were any other regular type.

The format for declaring function templates with type parameters is:

template <class identifier> function_declaration;
template <typename identifier>function_declaration;

For example, to create a template function that returns the greater one of two objects we could use:

```
template<class myType>
myType GetMax(myType a, myType
b) { if (a > b) return a;
else return b;
}
```

PART 1:

DOUBLY LINKED LIST

In this part, you will be applying what you learned in class to implement different functionalities of a linked list efficiently. The basic layout of a linked list is given to you in theLinkedList.h file.

The template ListItem in LinkedList.h represents a node in a linked list. The class LinkedList implements the linked list which contains pointers to head and tail and other function declarations. You are also given a file, test1.cpp for checking your solution.However, you are only allowed to make changes in the LinkedList.cpp file.

NOTE:

When implementing functions, pay special attention to corner cases such as deleting from an empty list.

Member functions:

Write implementation for the following methods as described here.

- LinkedList():
    - Simple default constructor.
- **LinkedList(const LinkedList<T>& otherList):**
    - Simple copy constructor, given pointer to otherList this constructor copies all elements from otherList to the new list.
- **void InsertAtHead(T item):**
    - Inserts item at the start of the linked list.
- **void InsertAtTail(T item):**
    - Inserts item at the end of the linked list.
- **void InsertAfter(T toInsert, T afterWhat):**

    - Traverse the list to find afterWhat and insert the toInsert after it.
- **ListItem<T> *getHead():**
    - Returns the pointer to the head of the list.
- **ListItem<T> *getTail():**
    - Returns the pointer to the tail of the list.
- **ListItem<T> *searchFor (T item):**
    - Returns a pointer to the item if it is in the list, returns null otherwise.
- **void deleteElement(T item):**
    - Find the element item and delete it from the list.
- **void deleteHead():**
    - Delete the head of the list.
- **void deleteTail():**
    - Delete the tail of the list.
- **int length():**
    - Returns the number of nodes in the list.

PART 2:

STACKS AND QUEUES

In this part, you will use your implementation of a linked list to write code for different methods of stacks and queues. As Part 1 is a prerequisite for this part so you **must have completed Part 1 before you attempt this part.**

STACK:

Stack class contains LinkedList type objects. You can only access member functions of the LinkedList class for your implementation of stack (and queue).

Member functions:

Write implementation for the following methods as described here.

- Stack():
  - Simple base constructor.
- **Stack(const Stack<T>& otherStack):**
  - Copies all elements from otherStack to the new stack.
- **void push(T item):**
  - Pushes items at the top of the stack.
- **T top():**
  - Returns the top of the stack without deleting it.
- **T pop():**
  - Return and delete the top of the stack.
- **int length():**
  - Returns count of the number of elements in the stack.
- **bool isEmpty():**
  - Return true if there is no element in the stack, false otherwise.

QUEUE:

Member functions:

Write implementation for the following methods as described here.

- **Queue():**
    - Simple base constructor.
- **Queue(const Queue<T>& otherQueue):**
    - Copy all elements of otherQueue into the new queue.
- **void enqueue(T item):**
    - Add items to the end of the queue.
- **T front():**
    - Returns an element at the front of the queue without deleting it.
- **T dequeue():**
    - Returns and deletes elements at the front of the queue.
- **int length():**
    - Returns the count of the number of elements in the queue.
- **bool isEmpty():**
    - Return true if there is no element in the queue, false otherwise.

PART 3:

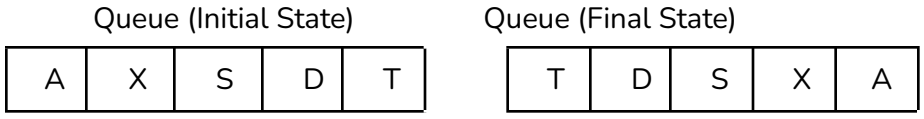
AMUSEMENT PARK

Suppose in the parallel universe, the earth is not struck by a pandemic and youdecide to take a break from the semester's workload to chill at an amusement park.

You register for a lucky draw and oh boy, you win a prize! Life is good.

FOR THE NEXT THREE PARTS, YOU CAN ONLY USE THE MEMBER FUNCTIONS DEFINED IN PART 1 AND PART 2 OF THIS ASSIGNMENT.

(a) (Reversing Queue)All the winners are asked to stand in a queue in the given positions. Unfortunately, the usher makes an error and you all are now standing in the reversed order of the names' announcement e.g the order of the names' announcement is [A, B, C] with person A to be called first and the queue is lined up as [C, B, A] with person C at the front of the queue. You take the task of helping the winners reshuffle themselves to correct their position in the queue. However, the place is very crowded and you have to ensure that the positions are reversed in an algorithmic manner without causing any hassle. Luckily, there's an empty lane next to your lane and you can use it in your algorithm but at the end, all the people should be standing in the reverse order in the same lane.

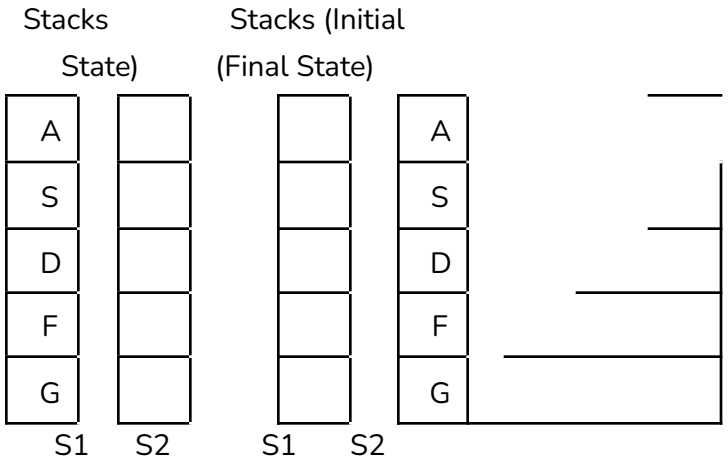


Write a code to solve the above problem with the followings conditions fulfilled:

- You cannot hard code this part i.e your code should work for any number of winners.
- Implement the two lanes as queues.

- You are not allowed to use any other data structure except for the given two queues.
- You are strictly not allowed to even use any variables.
- You are only allowed to use the member functions of Queue you defined in part 2.

(b) (Shifting Stack) Sadly, the event management has a rough day today. The gifts to be given out are stacked on the wrong table. They now have to stack the gifts on the designated table in the same order. However, there is very limited space and very limited labor available. The gifts are very heavy and can be lifted one at a time. Therefore, they approach you to give them a smart solution/algorithm since you managed to reverse the queue smartly.



Write a code to solve the above problem with the following conditions fulfilled:

- You cannot hard code this part i.e your code should work for any number of gifts.
- Implement the two tables as two stacks and a helper from the management as a variable.
- You are not allowed to use any other data structures except for the given two stacks and you cannot use more than one variable.
- You are only allowed to use member functions of Stack you defined in part 2.

(c) (Sorting Stack) The management started to feel that they needed to uplift the spirit of the crowd somehow after causing a lot of mismanagement. They decided to invite all the customers to a burger joint for FREE BURGERS!

How the burger joint operates:

The burger joint asks each of its customers to construct the burger of his/her choice -choose one topping from the following list:

1. tbun - top bun
2. ketchup
3. mayo
4. lettuce
5. onions
6. pickles
7. patty
8. mustard
9. chipotle
10. bbun - bottom bun

The list above also describes the order in which the toppings will be assembled by the staff of the burger joint (topping priority).

Your job is to assemble the burgers and hand them out to the customers.

How the assembly line operates:

You will be standing on the assembly line, here is what you have to do:

1. You will receive the customer ID of the customer whose order is to be assembled next and the toppings in an unsorted order.
2. You have to sort the toppings according to their priority (refer to the topping list above).
3. Hand out the order to the customer.

Here's how you have to implement the above three steps:

Step 1:

Load all of the toppings in the topping_priority from the text file named"topping_priority.txt" (the toppings are in sorted order in this file). Now load the customers and their respected unsorted toppings from the text file named "assembly.txt" in the queue named customers.

Step 2:

Sort the order of each customer in the customers queue. Here you have to make use of the assemble(Stack<string> &unsorted_toppings) function. But you have a couple of restrictions:

You have two stacks:

1. unsorted_toppings (input stack)
2. temp_stack (local)

You have one variable:

1. temp (local)

DO NOT CREATE ANYMORE VARIABLE OR DATA STRUCTURES IN THISFUNCTION. YOU CANNOT USE ANY PREDEFINED LIBRARY OR HELPERFUNCTION

Use the variable and data structures mentioned above to sort the unsorted_toppings stack. You need to sort it according to toppings_priority.

Step 3:

Use the generateOutput() function to output the orders of customers in a text file.Name the text file "takeaway.txt".

Running and Testing Code:

To run the code, you have to use g++ compiler. You have 4 test files at your disposal:

1. test1.cpp: Test for LinkedList.cpp.
2. test2.cpp: Test for stack.cpp and queue.cpp.
3. test3.cpp: Test for part (a) Reversing Queue and part (b) Shifting Stack.
4. test4.cpp: Test for part (c) Sorting Stack.

Write your implementations in the cpp files and then run the test 1 and test 2 using the following commands:

Compile: g++ test1.cpp -std=c++11

Run: ./a.out

To run test 3, you have to give the number of winners/ gifts as an argument. For example, the following commands will check your implementation for 100 winners/gifts:

Compile: g++ test3.cpp -std=c++11
Run: ./a.out 100

To run test 4, you have 3 assembly files at your disposal:

1. assembly10.txt: contains 10 scrambled burger toppings.
2. assembly50.txt: contains 50 scrambled burger toppings.
3. assembly100.txt: contains 100 scrambled burger toppings.

Suppose you want to run your test for 10 toppings, then copy the contents of “assembly10.txt” into “assembly.txt” and then run the test using the following commands:

Compile: g++ test4.cpp -std=c++11

Run: ./a.out 10

Similarly to run 50/100 toppings, copy contents of “assembly(50/100).txt” into“assembly.txt” and run the test using the following command:

Run: ./a.out (50/100)

Your code needs to run for all 10, 50, and 100 toppings correctly!

Submission Guide:

Zip the complete folder and use the following naming convention:

PA1_<rollnumber>.zip

Marks Distribution:

Part	Marks
Part 1: Linked Lists	**30**
Part 2: Stack and Queue	**20**
Part 3: ReversingQueue	**15**
Part 3: Shifting Stack	**15**
Part 3: Sorting Stack	**20**