Assignment 3

Posted on: 2/7/14

Due on: 2/17/14 at 11:55 pm (SAKAI)

In this assignment, you will implement several methods based on the **derivedChainStack and derivedChainQueue class (as mentioned in the class lecture)**.
The files we will provide to you:

- **chainNode.h**
- **chain.h**
- **chain.cpp**
- **linearList.h**
- **stack.h**
- **queue.h**
- **derivedChainStack.h**
- **derivedChainStack.cpp**
- **derivedChainQueue.h**
- **derivedChainQueue.cpp**
- **mainProgramInterface.h**
- **mainProgramInterface.cpp**
- **main.cpp**
- **Myexception.h**
- **Myexception.cpp**
- **File1.txt**
- **File2.txt**
- **File3.txt**
- **Makefile**

You are asked to implement **FOUR NEW** methods in the **mainProgramInterface.cpp**. The detail of these methods are given below:

Part I:

Given an integer (stored in **long theNumber**), use Stack and Queue to check whether the number is **Palindrome or not**. A **palindromic number or numeral palindrome** is a number that remains the same

when its digits are reversed. **For example: 14641, 64146** are **Palindrome** but **2432** is **not Palindrome.** You should write your code inside the function **checkPalindrome** in **mainProgramInterface.cpp**. The function prototype is given below:

```
bool mainProgramInterface :: checkPalindrome(long theNumber)
```

You will check your method on 4 inputs stored in `File1.txt`.

Part II:

Suppose in a movie theater, there is a queue consists of n people. Let the movie ticket **price is $5**. Each of the n people, in the queue, has **either $5 bill or $10 bill** in their pocket. **Initially**, the cash register is **empty**. If a person with a $5 bill purchases a ticket, the number of $5 bills in the register gets increased by one. But, if a person with a $10 bill purchases a ticket, as the cashier needs to give $5 change, the number of $5 bills decreases by one. When a person offers a $10 bill and the cashier has no $5 bill to offer as change, the person can not purchase the ticket, hence neither that person nor the rest behind him in the queue will get tickets. So, we will call a queue a **'valid queue'** if everyone in the line will be able to purchase ticket, else it is an **'invalid queue'**. For example, the queue **5 10 5 10** is **valid**, but **5 10 10 5** is **invalid**.

In this problem, your goal is to write a method which takes an integer array (**consists of integers 5 and 10**) of **size n** as input (representing the queue for movie tickets) and returns **true** if it is a **'valid queue'** and **false** otherwise (**Use stack operations)**. You should write your code inside the function **validSequence** in **mainProgramInterface.cpp**. The function prototype is given below:

```
bool mainProgramInterface :: validSequence(int *theSequence, int n)
```

You will check your method on 4 inputs stored in `File2.txt`.

Part III:

Write a method **duplicateStack** that takes two stacks, **'source'** and **'dest'** as input. The **'dest'** stack is **initially empty**. This method **copies** the element of the 'source' stack in the 'dest' stack, so that in the end **both of them** will contain the **same elements in the same order**. Write your code inside the function **duplicateStack** in **mainProgramInterface.cpp**. The function prototype is given below:

```
void mainProgramInterface :: duplicateStack(derivedChainStack* source, derived-
ChainStack* dest)
```

This method copies the content of the derivedChainStack source to derivedChainStack dest. You will check your method on 3 inputs stored in **File3.txt**.

Part IV:

At a bank, the teller decides to go to lunch and a line forms with **m** people. **Number the customers 1, 2, ... , m**. When the teller returns, the teller tells the person at the head of the line to go to the back of the line, doing this **n** times. Then the teller serves the person at the head of the line. The teller repeats this process until the line is empty (assuming no one else enters the bank). Write a method **lastCustomer that has two parameters that specify the number of customers initially in line (m) and the number of customers sent to the back of the line each time (n)**. **The method should return the number of the customer that is served LAST**. You may assume that $m > 0$ and $1 \leq n \leq m$. The function prototype is given below:

```
int mainProgramInterface :: lastCutomer(int m, int n)
```

You will check your method on three inpt pairs, (12, 5), (10, 6) and (10, 3).

Creating and Extracting a tar file:

Linux/ Macintosh

- To create a tar file: **tar cvf (tar file name) (file 1) (file 2) (file 3)...**
- To extract the contents of a tar file: **tar xvf (tar file name)**

Windows

- You can use **ALZip** to create .tar file.

Things to Remember before Submission:

1. Check for the submission deadline (**both date and time**) and make sure you submit your **.tar file** before the deadline.
2. **LATE SUBMISSIONS ARE NOT ALLOWED.**
3. You should test your code on **thunder machine. TA will run your code on thunder machine and if it fails to compile there, you will be penalized.**
4. TA will only do **make** and then run the executable file, **e.g. ./main**.
5. You should submit **ONLY** a **.tar** file through **SAKAI** consists of **all the .h, .cpp and makefile.** The name of the **.tar** file should contain your name and UFID.
6. The output of your submission should be **exactly like (not "almost like")** the snapshot in Figure 1.
7. You can write any helper/ auxiliary method needed for the implementation of the four methods you are responsible for.

Figure 1: Snapshot of required input/ output