CSCI 2270 Midterm 1 - Part 1: Coding Problems

Read these directions carefully.

You will be solving two of the three programming problems detailed below. *Problem 1* is **mandatory**, but you only have to do **either** *Problem 2* **or** *Problem 3*. For each problem you must submit:

1. A C++ program that solves the given problem. It is required that your code files are well commented. `Note: you will not receive more than 25% of the possible points on any problem if your submitted code does not compile.`
2. If feel like your code does not fully meet the specifications, then include a readme text file with your explanations for potential partial credit. This will help your TA understand your thought process. Mention anything you have done to write, test, and debug your code. Incomplete code can still receive points if you show that you have identified the errors and tried to debug them.

- Use the starter code provided with the exam. `NOT THE EARLY RELEASE STARTER CODE.`
- Your submission should be valid C++ 11 code.
- Run all the test cases given in the starter code. You are encouraged to add your own test cases, to ensure full functionallity of your code.
- We will be running this code on other computers, so make sure to avoid **any** undefined behavior such as uninitialized variables.

Problem 1 (Mandatory)

Task:

Given a linked list, remove the n-th node from the end of the list, where n is an integer greater than 0. The function should be a void function.

Requirements:

Implement a `removeNthFromEnd` function: *(You may not change anything in hpp file.)*

```
void LinkedList::removeNthFromEnd(int n);
```

Examples:

Example 1:

- If the original linked list (*original_LL*) is-

```
10 -> 20 -> 30 -> 40 -> NULL
```

Example function call for n = 2:

`original_LL.removeNthFromEnd(2);`

the linked list should become:

```
10 -> 20 -> 40 -> NULL
```

Because the 2nd node from the end is 30 so we should remove 30.

Example 2:

- If the original linked list(*original_LL*) is-

```
    10 -> 20 -> 40  -> NULL
```

Example function call for n = 1:

`original_LL.removeNthFromEnd(1);`

the linked list should become:

```
    10 -> 20 -> NULL
```

Because the 1st node from the end is 40 so we should remove 40.

Hint:

1. **Test your function** to make sure that it works in every case, especially the edge cases like n = 1 or n = the length of linked list.
2. One possible way to do this:

   Use two seperate pointers(e.g. *prev* and *pres*);

   Move *pres* n nodes after *prev* ;

   Then move them together until the fast pointer(*pres*) reaches the end;

   Now the slow pointer(*pres*) will point to the nth node from the end;

   Delete the required node and return the head.

Problem 2

Task:

In this assignment you will create a Queue with **stack(s)**. Stack class implementation is complete. **You do not have to do any implementation for stack class**. Most of the functions for the queue are also implemented in the `.cpp` file. **You need to implement the dequeue function. You will use methods of stack class to implement the dequeue function in the queue class.**

Requirements:

The dequeue function will have following signature

```
int Queue:: dequeue()
```

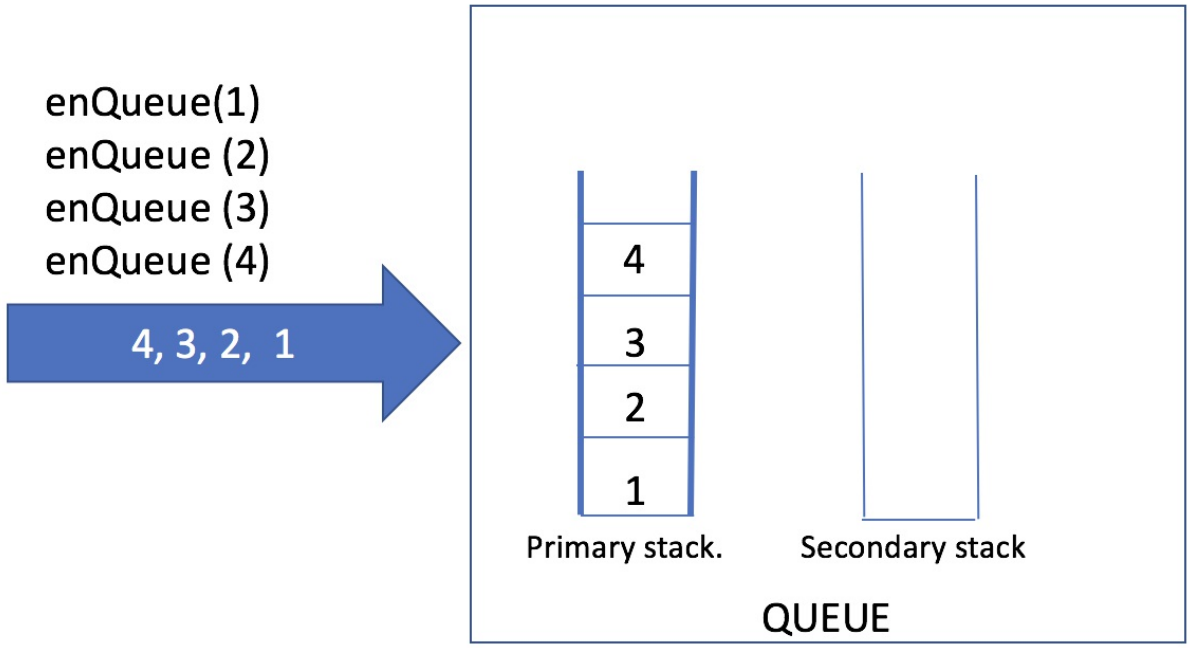
The function will dequeue element at the front of the queue and **will return that value**. If the queue is empty dequeue should print `Queue is empty. Can not dequeue` and return `-999`.

You are supposed to implement queue using stack. So you can use push, pop, peek functions of the member stack(s) of your queue.

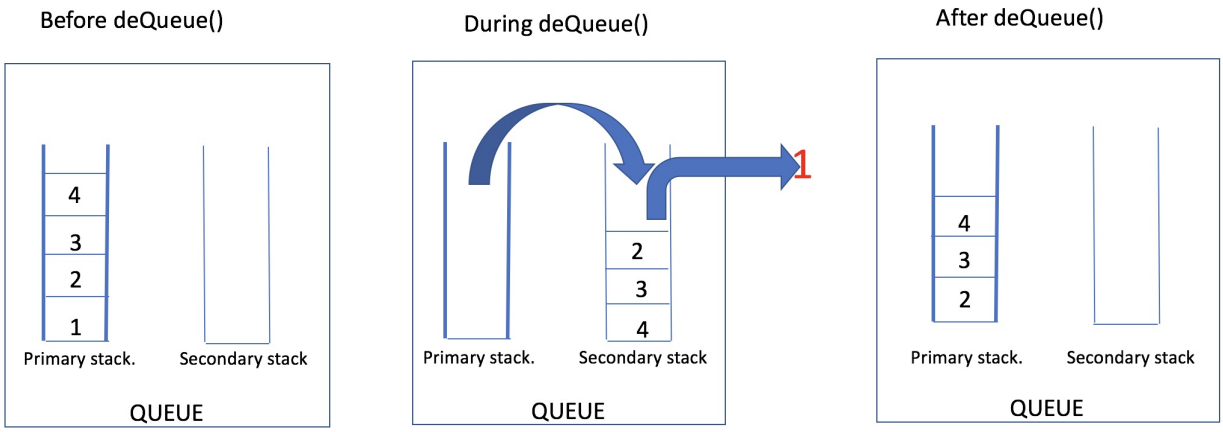
- **Do not access element by array indices inside the dequeue function**.
- **Do not use any other data structure than stack to implement this queue**.

EXAMPLE (Hint)

Let's assume we will enqueue < 1, 2, 3, 4 > in the order mentioned. 1 will be enqueued first. Then 2 will enqueued and so on. Following image shows the state of the member stacks after enqueue. Note 4 is towards the top of the stack and 1 is at the bottom.



However, when we will dequeue we need to return 1. Recall that queue is First in First out. So if we dequeue we should get 1 first. Next dequeue should return 2. Third dequeue should return 3 and so on. FOllowing image shows the how can you acheive such result using the secondary stack.



2. **Test your function** to make sure that it passes all the test cases.

Problem 3

Task:

In this assignment you will create a function that will split an array into two arrays, one with the even numbers and the other with odd numbers.

Requirements:

The function prototype is given as-

```
void split(int* &arr, int* &even_arr, int* &odd_arr, int size, int &even_size,
  int &odd_size)
```

where

- `arr` : pointer to the original array
- `even_arr` : pointer to the even array
- `odd_arr` : pointer to the odd array
- `size` : size of the original array
- `even_size` : size of the even array
- `odd_size` : size of the odd array

Function will split the original array into even array and odd array. After calling the function, `arr` will split into `even_arr` pointing to an array with even numbers and `odd_arr` pointing to an array with odd numbers. Size of even array will be `even_size` and size of odd array will be `odd_size` . **Counting of `even_size` and `odd_size` should be performed inside the function `split` (NOT in main function)**.

The main function will take size as a command line argument. Sample code for creating original array is given in the starter code.

Apart from completing the above function, you need to do the following in the main -

- Call the split function and then print out the elements of the even and odd arrays.
- If the original array only has even numbers, print "no odd items in the array" for the empty odd array. Do the vice versa if the original array has only odd numbers.

- After splitting the arrays, **do not forget to free up the memory space as required**.
- When printing, each element should be separated by a `,`. There should not be any `,` after the last element.

  Examples:

```
./a.out 5

original array: 58, 45, 35, 18, 99
even array: 58, 18
odd array: 45, 35, 99
```