Assignment #2

CS 2001 – Data Structures (CS)
Fall 2024

General Guidelines

1. Peer plagiarism and the late submissions are strictly not allowed
2. Total Marks: 100

Submission Guidelines

3. Your assignment submission will be on CLASSROOM within the givendeadline.
4. Analytical and mathematical questions can be handwritten, while code questions should preferably be computer-typed
5. **Deadline on google classroom**

You are not allowed to use C++ STL Containers for any purpose. Adopt good structured, modular, and object-oriented programming practices.

Task1:

STACK ADT

Implement the following class. class

Stack

{

Node* Top;

public: Stack();

bool puch(char);

char pop(); bool

isEmpty(); void

display();

bool isPalindrome(String); //This will check if the string is palindrome or not

}

Note: You're not allowed to use any built-in libraries.

Task 2 :

We want to design a calculator application Pre-Post Calculator that evaluates an input Infix expression through Prefix and Postfix notations using the Stack data structure (implemented with linked nodes). An Infix expression may contain positive integers, the five arithmetic operators (+, -, *, /, %), and parenthesis where each token is separated by a space symbol; for example: 24 * (490 + 6 / 12). Use the String class functions as necessary to parse the Infix expression.

Refer to your lecture notes for conversion and evaluation algorithms. Provide a menu-driven interface to the user which allows the user to decide how the Infix expression will be evaluated (Prefix notation or Postfix notation). Your application should display all the necessary elements (i.e., converted expression and stack) after every update made during expression conversion and evaluation.

Task 3: Shelfshuffle using STACK ADT

You and your friends are tasked with organizing a collection of vintage vinyl records. You have three empty shelves available, and each shelf can accommodate records of various sizes. Your objective isto move all the records from the first shelf to the third shelf using the fewest moves possible, following specific rules.

Rules of the Game:

1. You can only move one stack of records at a time.
2. A larger record cannot be placed on top of a smaller record.
3. You can only move the top stack of records from a shelf.

You decide to turn this task into a fun game to challenge yourselves and make the organizing process more enjoyable. Your goal is to optimize your moves while adhering to the rules of the game. Each move brings you closer to successfully organizing the records and completing the task.

For better understanding you can play a similar game on the link

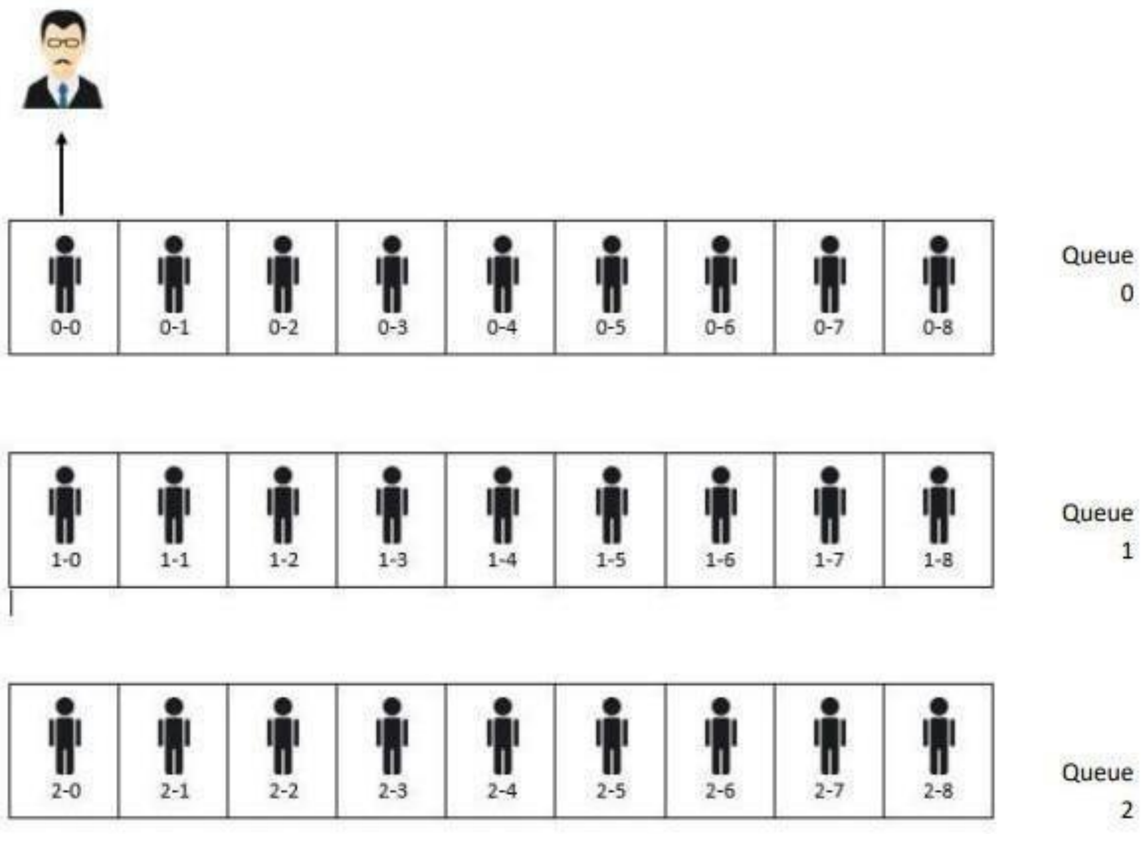
https://www.mathsisfun.com/games/towerofhanoi.html

Note: you can use numbers to representset of records. The state of each shelf should be displayed after each iteration.

Task 4:

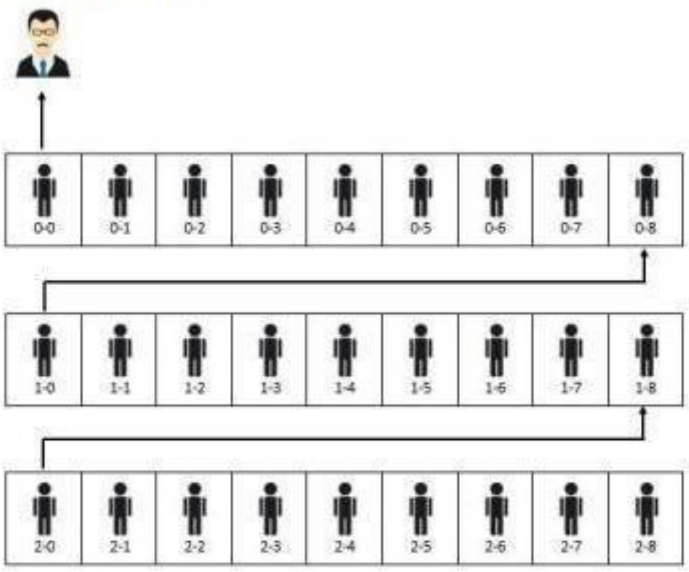
Bank queue management

In a world without digital bookings when people had to buy movie tickets. Imagine it's 1990s and a blockbuster movie has just been released. To avoid longer queues and to keep track of queues outside the cinema, they have decided to divide the queue into sub-queues but due to limited resources the cinemas can only afford one ticket collector who is standing at the front of the last sub-queue. The ticket collector takes 2 seconds to process a person. The queues look like this:



The queues work in the following way

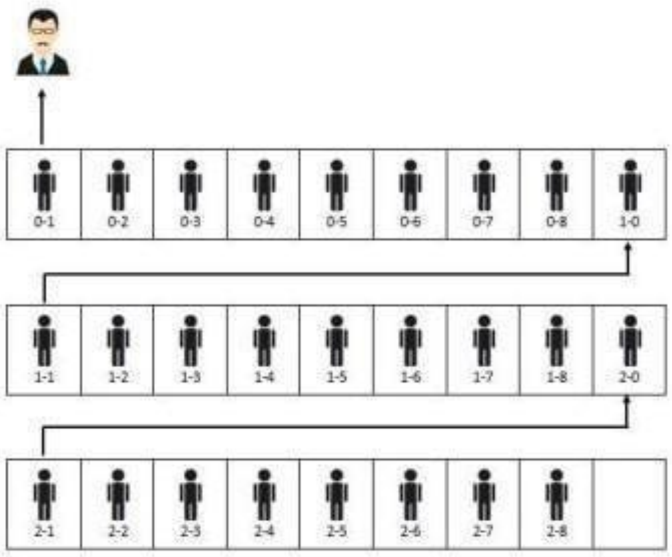
The queues work in the following way:



i.e: After processing "Person 0" of "Queue x", the queue moves forward like this:

1. "Person 0" of "Queue x-1" leaves his/her queue and enters in "Queue x".
2. "Person 0" of "Queue x-2" leaves his/her queue and enters in "Queue x-1".
3. "Person 0" of "Queue x-3" leaves his/her queue and enters in "Queue x-2".
4. This keeps happening all the way to queue 0 and at the end, "Person 0" of "Queue 0" leaves his/her queue and enters in "Queue 1".

After the queue is moved one-step forward, here's how it looks:



By the end, the ticket collector processes all the people in the queues so that everyone can enjoy the movie.

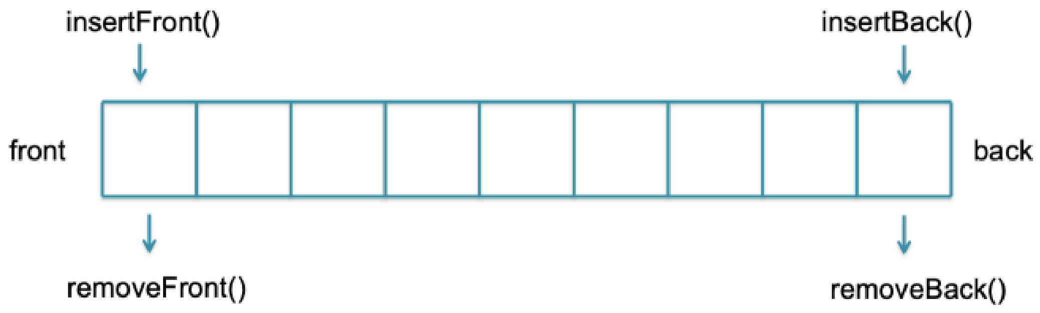
You are going to simulate the above explained process using Queues ADT in C++. To implement this program, you will also be creating a Template Queue Class all by yourself to keep it generic and to create queues of any-types. The flow of the program will be like this:

User inputs an integer and N queues (<int>) are created. After creating N queues, you will enqueue some number of persons. For your ease, you can use same number of persons in every queue (minimum number of people in a queue must be 10). The ticket collector starts to process persons in “Queue n” one by one until all the persons are processed OR all the queues are empty.

Note: You’re not allowed to any built-in libraries like vectors or queues.

Task 5:

A double-ended queue or deque (pronounced as “deck”) is a variant of QueueADT which allows insertions and deletions at both ends using the operations: insertFront, insertBack, removeFront, removeBack, as shown in the figure below. If you perform insertions/deletions at one end, it will act as a stack while if you exclusively perform insertions at one end (i.e., back) and deletions at the other end (i.e., front) it will act as a queue. In a simple queue, we increment front and rear indexes upon deletion and insertion, respectively. The same approach will be followed in deque if a user performs insertion/deletion on usual ends. However, as a user inserts a value at front or delete value from rear, the corresponding index should move backward. Provide an array-based C++ implementation for deque that allows insertion and deletion operations in O(1) time. The class template is given at the end for your reference.



Suppose we performed some insertions and deletions at both ends of a deque (of size 10). The dequeue after the following operations is shown below (grey shaded row shows the array index). Front and rear indexes are set to -1 initially.

deque.insertBack(3); front = 0, rear = 0

3									
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

deque.insertBack(14); front = 0, rear = 1

3	14								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

deque.insertFront (90); front = 9, rear = 1

3	14								90
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

deque.removeFront ( ); front = 0, rear = 1

3	14								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

deque.removeBack ( ); front = 0, rear = 0

3									
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

deque.removeFront ( ); front = -1, rear = -1

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

```
// Deque class template
class deque {
  int *dequeArr;
  int maxSize, rear, front;
public:
   deque ( );
   bool isEmpty( );
   bool isFull( );
   void insertFront (int value);
   void insertBack (int value);
   int removeFront ( );
   int removeBack ( );
   ~ deque ( );
};
```

Task 6:

You are tasked with determining if the sequence of characters stored in a queue forms a palindrome. A palindrome is a sequence that reads the same backward as forward, such as "madam" or "racecar". To solve this problem, you must use both a queue and a stack to check the sequence. You will use the queue to process the sequence in its original order and a stack to process it in reverse order. By comparing the elements from the queue and the stack, you can determine if the sequence is a palindrome.

Input:

- A queue of characters, which could be a string or a list of characters.

Output:

- Return True if the sequence in the queue is a palindrome.
- Return False if the sequence is not a palindrome.

Example:

1. Input: ['r', 'a', 'c', 'e', 'c', 'a', 'r']

Output: True

Explanation: "racecar" is the same when read forward and backward.

2. Input: ['a', 'b', 'c', 'd']

Output: False

Explanation: "abcd" is not the same when read forward and backward.

Best of Luck ♦

Keep Exploring