|  |  |
| --- | --- |
| Data Structures & Algorithms Diploma in CSF, IT  Year 2/3 (2020/21) Semester 4/6 | Week 5 |
| 2-3 Hours |
| **Practical 5 – Queues** | |

**Objectives**

At the end of this practical, the students should be able to:

* Implement a Queue ADT
* Apply use of queues (together with stacks) in a simple application

|  |
| --- |
| **IMPORTANT**   * Upload all your work to DSA network folder **\\ictspace.ict.np.edu.sg\dsa** * You may use the Stack ADT from your previous practical * Download the code samples from MEL before you begin the practical. |

***Determining Palindromes***

1. The specification of the Stack ADT (similar to Practical 4 with exception of displayInOrderOfInsertion) implemented using Pointers is given below.

|  |
| --- |
| *// Stack.h - Specification of Stack ADT (Pointer-based)*  #pragma once  #include<string>  #include<iostream>  using namespace std;  typedef char ItemType;  class Stack  {  private:  struct Node  {  ItemType item; // item  Node\* next; // pointer pointing to next item  };  Node \*topNode; // point to the first item  public:  // constructor  Stack();  ~Stack();  *// push item on top of the stack*  bool push(ItemType item);  *// pop item from top of stack*  bool pop();  *// retrieve and pop item from top of stack*  bool pop(ItemType &item);  *// retrieve item from top of stack*  void getTop(ItemType &item);  *// check if the stack is empty*  bool isEmpty();  }; |

2. Implement the operations of the Stack ADT

*Note : You should implement (and test) one operation at a time.*

3. The specification of the Queue ADT implemented using Pointers is given below.

|  |
| --- |
| *// Queue.h - Specification of Queue ADT (Pointer-based)*  #pragma once  #include<string>  #include<iostream>  using namespace std;  typedef char ItemType;  class Queue  {  private:  struct Node  {  ItemType item; // item  Node \*next; // pointer pointing to next item  };  Node \*frontNode; // point to the first item  Node \*backNode; // point to the first item  public:  // constructor  Queue();  ~Queue();  *// enqueue (add) item at the back of queue*  bool enqueue(ItemType item);  *// dequeue (remove) item from front of queue*  bool dequeue();  *// dequeue (remove) and retrieve item from front of queue*  bool dequeue(ItemType &item);  *// retrieve (get) item from front of queue*  void getFront(ItemType &item);  *// check if the queue is empty*  bool isEmpty();  *// display items in queue from front to back*  void displayItems();  }; |

4. Implement the operations of the Queue ADT

*Note : You should implement (and test) one operation at a time.*

5. Write a sample program, QueueDemo.cpp, to do the following:

1. Create an empty queue, q
2. Enqueue ‘a’ to q.
3. Enqueue ‘b’ to q.
4. Get front of q and display
5. Display all the items in q
6. Dequeue q.
7. Display all the items in q

6. Now improve QueueDemo.cpp to do the following:

1. Read user input
2. Analyse user input ***to determine if it is a palindrome***

7. Use a queue to simulate the flow of customers through a check-out line in a store. In order to create this simulation, you must model both the passage of time and the flow of customers through the line.

You can model time using a loop in which each pass corresponds to a set of time interval – 1 minute, for example. You can model the flow of customers using a queue in which each data item corresponds to a customer in the line.

In order to complete the simulation, you need to know the rate at which customers join the line, as well as the rate at which they are served and leave the line. Assume the check-out line has the following properties:

* One customer is served and leaves the line every minute (assuming there is at least one customer waiting to be served during that minute).
* Between zero and two customers join the line every minute, where there is a 50% chance that no customers arrive, a 25% chance that one customer arrives, and a 25% chance that two customers arrive.

You may simulate the flow of customers through the line during a time period n minutes long using the following algorithm:

Initialize the queue to empty.

While the simulation is not done

Increment simulated time by one minute

If the queue is not empty,

then remove the customer at the front of the queue.

Compute a random number k between 0 and 3.

If k is 1, then add one customer to the line.

If k is 2, then add two customers to the line.

Otherwise (if k is 0 or 3), do not add any customers to the line.

Update queue statistics.

Display the average time that a customer is served.

Notes:

1. Before adding the customer to the line, you may ask for his name. The Customer.h and Customer.cpp files are given below.
2. You may use the rand() function to generate pseudo-random numbers. It is available through the <random> package.
3. To get a different simulation each time the program is run, initialize the random number generator using the srand() function.

Research online on how to use the rand() and srand() functions.

A sample simulation is shown on page 6.

#pragma once

// Customer.h - Definition of Customer class

#include<string>

#include<iostream>

using namespace std;

class Customer

{

private:

string name;

int queueNum; // the i-th minute Customer joins queue.

public:

Customer();

Customer(string n, int q);

void setName(string n);

string getName();

void setQueueNum(int q);

int getQueueNum();

};

// Customer.cpp - Implementation of Customer class

#include "Customer.h"

Customer::Customer() {}

Customer::Customer(string n, int qNum)

{

name = n;

queueNum = qNum;

}

void Customer::setName(string n) { name = n; }

string Customer::getName() { return name; }

void Customer::setQueueNum(int q) { queueNum = q; }

int Customer::getQueueNum() { return queueNum; }

Sample simulation:

