|  |
| --- |
| **Computer Engineering Department - ITU** |
| **CE200L: Data Structures & Algorithms Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 06/10/2022** |
| **Teaching Assistant: Muhammad Sufyan Ashraf** | **Semester: Fall 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 7A. Queues with Linked Lists**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Roll number** | **Report**  **(out of 100)** | **Scaled to 10** | **Total**  **(out of 10)** |
| NIMRA MAQBOOL | BSCE21012 |  |  |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Objective**

The objective of this lab is to provide the knowledge of basic data structures and their implementations.

## **Equipment and Component**

|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **Conduct of Lab**

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

## **Theory and Background**

In computer science, a **linked list** is a linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence.

A **queue** is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order. We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end.

**Templates** are a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one.

**Lab Task**

**Task A**

**// Add function to insert the new element in queue**

void enqueue ()

{

}

**// Add function to remove the element at front from queue**

void dequeue ()

{

}

**// Add function to show the element at front**

void showFront ()

{

}

**// Add function to check if queue is empty**

void isEmpty ()

{

}

**// Add function to display queue elements**

void display ()

{

}

|  |
| --- |
| // Paste your code here  **FUNCTION.H:**  #include <iostream> using namespace std;  class Node { //made a class of node public:  int data; //declared data in public  Node \*nextPtr;   Node(int val) {  data = val; //made a constructor to set values  nextPtr = NULL;  }   void setNextPtr(Node \*n) {  nextPtr = n; //set the next ptr  }   Node \*getNextPtr() {  return nextPtr; //get the value of next ptr  } };  class linkList { //making 1 other class of linklist public:  Node \*tail;  Node \*head; //made some pointers  linkList() {  head = nullptr; //declared them to zero  tail = nullptr;  }   void enqueue(int value) {  Node \*temp = new Node(value); //declaring  temp->data = value;  temp->nextPtr = nullptr; //initializing the next ptr in the next of the new initialized node  if (head == nullptr) {  head = temp; //if the head is null then store the temp in head  } else {  Node \*temp1 = head; //else make a new node  while (temp1->nextPtr != nullptr) //iterate it till the node is not null  temp1 = temp1->nextPtr; //store the temp to next ptr address  temp1->nextPtr = temp; //store pointer to the last one  }  }  void dequeue(){  if(head== nullptr){ //is head is null the array is empty  cout<<"THE ARRAY IS EMPTY."<<endl;  }  else{  Node \*newTemp=head; //making a new node and giving it address of head  head=head->nextPtr; //giving head the address the head is pointing to  delete newTemp; // and deleting the temp  }  }   void showFront(){  Node \*temp = head;  if(temp != nullptr){  cout<<"THE VALUE AT FRONT IS = "<<temp->data<<endl;  }  else{  cout<<"THE STACK IS EMPTY."<<endl;  }  }  bool empty(){  if(head== nullptr){  cout<<"ARRAY IS EMPTY."<<endl; //checking is the head is null  return true;  }  else{  return false;  }  }  void display() {  Node \*temp = head; //declaring  while (temp != nullptr) {  cout << temp->data << "\t"; //displaying the data  temp = temp->nextPtr; //storing the next address  cout << " "; //displaying space  }  } }; class queue{ public:  linkList l;   void popElements(){  l.dequeue();  l.display();  }  void pushElement(){  int value;  cout<<"ENTER VALUE TO ENTER IN THE QUEUE = ";  cin>>value;  l.enqueue(value);  }  void showFrontElement(){  l.showFront();  }  void displayElement(){  l.display();  }  };  **Main.cpp:**  // // Created by Lenovo on 10/6/2022. // #include <iostream> #include "Functions.h"  using namespace std;  // int main() {  Node n(5); //making object  n.setNextPtr(0);  n.getNextPtr(); //calling  linkList l;  queue q;  int opt;  int opt1;  do{  cout<<"\nCHOOSE OPTIONS."<<endl;  cout<<"1.ENQUEUE ELEMENTS."<<endl;  cout<<"2.DEQUEUE ELEMENTS."<<endl;  cout<<"3.SHOW FRONT ELEMENT."<<endl;  cout<<"4.DISPLAY ELEMENTS OF STACK."<<endl;  cout<<"5.EXIT."<<endl;  cin>>opt;  if(opt==1){  q.pushElement();  do{  cout<<"YOU WANNA PUSH ELEMENT AGAIN?"<<"\nENTER 1 FOR YES AND 0 FOR NO."<<endl;  cin>>opt1;  if(opt1==1){  q.pushElement();  }  if(opt1==0){  cout<<"YOU CHOOSE NO."<<endl;  break;  }  if(opt1!=0 && opt1!=1){  cout<<"YOU HAVE ENTERED INVALID ARGUMENT."<<endl;  break;  }  }while(opt1>=0 && opt<=1);   }  if(opt==2){  q.popElements();   }  if(opt==3){  q.showFrontElement();  }  if(opt==4){  q.displayElement();  }  if(opt==5){  cout<<"YOU CHOOSE TO EXIT."<<endl;  exit(4);  }  }while(opt>=1 && opt<=5);  }  **Output:**    Text  Description automatically generated |

#### **Assessment Rubric for Lab**

**Method for assessment:**

Lab reports and instructor observation during lab sessions. Outcome assessed:

a. Ability to conduct experiments, as well as to analyze and interpret data (P) b. Ability to function on multi-disciplinary teams (A)

c. Ability to use the techniques, skills, and modern engineering tools necessary for engineering practice (P)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Performance metric** | **Task** | **CLO** | **Description** | **Max marks** | **Exceeds expectation** | **Meets expectation** | **Does not meet expectation** | **Obtained marks** |
| 1. Realization of experiment (a) | 1 | 1 | Functionality | 40 | Executes without errors excellent user prompts, good use of symbols, spacing in output. Through testing has been completed (35-40) | Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed (20-34) | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non-existent. No testing has been completed (0-19) |  |
| 2. Teamwork (b) | 1 | 3 | Group Performance | 5 | Actively engages and cooperates with other group member(s) in effective manner (4-5) | Cooperates with other group member(s) in a reasonable manner but conduct can be improved (2-3) | Distracts or discourages other group members from conducting the experiment (0-1) |  |
| 3. Conducting experiment (a, c) | 1 | 1 | On Spot Changes | 10 | Able to make changes (8-10) | Partially able to make changes (5-7) | Unable to make changes (0-4) |  |
| 1 | 1 | Viva | 10 | Answered all questions (8-10) | Few incorrect answers (5-7) | Unable to answer all questions (0-4) |  |
| 4. Laboratory safety and disciplinary rules (a) | 1 | 3 | Code commenting | 5 | Comments are added and does help the reader to understand the code (4-5) | Comments are added and does not help the reader to understand the code (2-3) | Comments are not added (0-1) |  |
| 5. Data collection (c) | 1 | 3 | Code Structure | 5 | Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap (4-5) | Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables (2-3) | Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy (0-1) |  |
| 6. Data analysis (a, c) | 1 | 4 | Algorithm | 20 | Solution is efficient, easy to understand, and maintain (15-20) | A logical solution that is easy to follow but it is not the most efficient (6-14) | A difficult and inefficient solution (0-5) |  |
| 7. Computer use (c) | 1 | 2 | Documentation & Github Submissions | 5 | Timely (4-5) | Late (2-3) | Not done (0-1) |  |
|  | Max Marks (total): | | | 100 | Obtained Marks (total): | | |  |

Lab Engineer Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_