|  |
| --- |
| **0Computer Engineering Department - ITU** |
| **CE200L: Data Structures & Algorithms Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 29/11/2022** |
| **Teaching Assistant: Muhammad Sufyan Ashraf** | **Semester: Fall 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 14A. Basic Operations on Circular Linked List**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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**

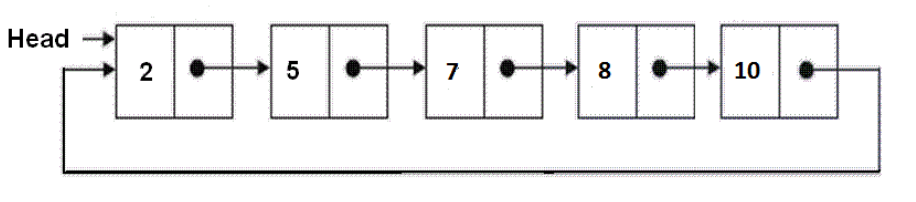
|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **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. **Circular linked list** is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.



**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**

As you have implemented the singly linked list before. Now implement the circular linked list. Implement the following functions:

**// Add function to insert the node at start in linked list**

void insertNodeAtStart (T)

{

}

**// Add function to insert the node at end in linked list**

void insertNodeAtEnd (T)

{

}

**// Add function to print linked list**

void printLinkedList ()

{

}

**// Add function to insert the node at any index in linked list**

void insertNodeAtAnyIndex (T,int)

{

}

**// Add function to delete the node from start in linked list**

void deleteNodeFromStart ()

{

}

**// Add function to delete the node from end in linked list**

void deleteNodeFromEnd ()

{

}

**// Add function to delete the node from any index in linked list**

void deleteNodeFromAnyIndex (int)

{

}

Make all above functions and handle all corner cases. Make a menu driven program.

|  |
| --- |
| // Paste your code here  **FUNCTION.H:**  #include <iostream>  using namespace std;  class node { public:  int data;  node \*nextPtr; //declaring   node(int d) {  data = d;  nextPtr = nullptr; //initializing  }   ~node() {  nextPtr = nullptr; //making a dectructor  } };  class linkList { private:  node \*head;  node \*tail; //declaring  int count; public:  linkList() {  head = nullptr;  tail = nullptr;  count = 0;  }   ~linkList() {  head = nullptr; //making a destructor  tail = nullptr;  }   void append(int value) {  node \*temp = new node(value); //making a pointer  temp->data = value;  if (head == nullptr) {  head = temp; //checking if head is null then store temp in head and make the next of temp as head  temp->nextPtr = head;  } else {  node \*newTemp = head; //made a pointer  while (newTemp->nextPtr != head) { //checking till the pointer next is not head means it is not again at the head  newTemp = newTemp->nextPtr; //storing addresses  }  newTemp->nextPtr = temp; //making the next of pointer as the pointer having value to be stored  temp->nextPtr = head; //and make the next of next pointer as head  }  count++; //incrementing  }   void prepend(int value) {  node \*temp = new node(value); //making a new pointer  temp->data = value; //storing data  if (head == nullptr) {  head = temp; //if head is null then store the temp in head  temp->nextPtr = head; //and make the next of temp as head  } else {  node \*newTemp = head; //making a new pointer  while (newTemp->nextPtr != head) { //iterating while the next of head is not head  newTemp = newTemp->nextPtr; //storing address  }  newTemp->nextPtr = temp; //storing the pointer in the next of new temp  temp->nextPtr = head; //making the next of temp as head  head = temp; //storing temp in head;  }  count++; //incrementing  }   void insertAtSpecificIndex(int value, int index) {  cout << "ENTER INDEX = ";  cin >> index; //taking index and value  cout << "ENTER THE VALUE TO STORE AT THE SPECIFIC INDEX = ";  cin >> value;  node \*newNode;  if(index<0) {  cout << "ENTER POSITIVE = ";  return;  }  newNode = new node(value);  newNode->data = value; //making a pointer and storing value  if (head == nullptr) {  head = newNode; //if head is null then store the new node in head  newNode->nextPtr = head;  } else {  node \*temp = head; //making a node as head  int i = 0;  while (i < index - 1) {  temp = temp->nextPtr; //iterating till the index -1 and storing the address  i++;  }  newNode->nextPtr = temp->nextPtr; //storing next in the new node  temp->nextPtr = newNode; //storing new node in next of temp  }  count++; //incrementing  }   void deleteNodeAtLastIndex() {  if (head == nullptr) {  cout << "NO ITEM IS PRESENT IN THE LIST." << endl; //if head is null means the list is empty  } else if (head->nextPtr == head) {  head = nullptr; //else check if the next of head is null or not if null then make head as null and free head  free(head);  } else {  node \*temp = head; //making a new pou=inter and initializing head  node \*newNode;  while (temp->nextPtr != head) { //iterating till head  newNode = temp; //storing temp in new temp  temp = temp->nextPtr; //storing the address of next  }  newNode->nextPtr = temp->nextPtr; //storing the next of temp in new node  free(temp); //and freeing temp  }  count--;//decrementing  }   void deleteNodeAtFirstIndex() {  if (head == nullptr) {  cout << "NO ITEM IS PRESENT IN THE LIST." << endl; //if head is null means the list is empty  } else if (head->nextPtr == head) {  head = nullptr; //else check if the next of head is null or not if null then make head as null and free head  free(head);  } else {  node \*temp = head; //making a new pou=inter and initializing head  while (temp->nextPtr != head) { //iterating till head  temp = temp->nextPtr; //storing the address of next  }  temp->nextPtr = head->nextPtr; //storing the next of head in new node  free(head); //freeing head  head = temp->nextPtr; //storing the next of temp in head  }  count--; //decrementing  }   void deleteNode(int index) {  cout << "ENTER INDEX = ";  cin >> index;  if (head == nullptr) {  cout << "NO ITEM PRESENT IN THE LIST." << endl; //if head is null means the list is empty  return;  }  if(index<=0){  cout<<"ENTER POSITIVE = ";  return;  }  node \*temp = head;  int i = 0;  if (temp != nullptr) { //checking if the temp is not null  while (i < index - 1) { //iterating till index-1  temp = temp->nextPtr; //storing the address  i++; //iterating  }  if (temp == nullptr || temp->nextPtr == nullptr) { //if temp is null or next of temp is null then return  return;  }  node \*newNode = temp->nextPtr->nextPtr; //otherwise,storing the next of next in new node  free(temp->nextPtr); //freeing the next of temp  temp->nextPtr = newNode; //making the temp of temp as new node  }  count--; //decrementing  }   void display() {  node \*temp;  if (head == nullptr) { //if head is null means the list is empty  cout << "List is empty";  } else {  temp = head;  do {  cout << temp->data << "-> "; //else displaying  temp = temp->nextPtr;  } while (temp != head); //till the temp becomes head  cout << "NULL." << endl;  }  }  };  **MAIN.CPP:**  // Created by Lenovo on 11/29/2022. // #include <iostream> #include "Function.h"  using namespace std;  int main() {  linkList a;  int opt;  do {  cout << "CHOOSE OPTIONS." << endl;  cout << "1.APPEND." << endl;  cout << "2.PREPEND." << endl;  cout << "3.ADD AT A SPECIFIC INDEX." << endl;  cout << "4.DELETE LAST NODE." << endl;  cout << "5.DELETE FIRST NODE." << endl;  cout << "6.DELETE AT A SPECIFIC INDEX." << endl;  cout << "7.PRINT." << endl;  cout << "8.EXIT." << endl;  cin >> opt;  if (opt == 1) {  a.append(2);  a.append(4);  a.append(2);  cout << "CIRCULAR LINK LIST AFTER APPENDING = ";  a.display();  }  if (opt == 2) {  a.prepend(9);  cout << "CIRCULAR LINK LIST AFTER PREPENDING = ";  a.display();  }  if (opt == 3) {  int value;  int index;  a.insertAtSpecificIndex(value, index);  cout << "CIRCULAR LINK LIST AFTER INSERTING AT GIVEN INDEX = ";  a.display();  }  if (opt == 4) {  a.deleteNodeAtLastIndex();  cout << "CIRCULAR LINK LIST AFTER DELETING LAST NODE = ";  a.display();  }  if (opt == 5) {  a.deleteNodeAtFirstIndex();  cout << "CIRCULAR LINK LIST AFTER DELETING FIRST NODE = ";  a.display();  }  if (opt == 6) {  int index;  a.deleteNode(index);  cout << "CIRCULAR LINK LIST AFTER DELETING FROM SPECIFIC INDEX = ";  a.display();  }  if (opt == 7) {  cout << "CIRCULAR LINKED LIST = ";  a.display();   }  if (opt == 8) {  cout << "YOU CHOOSE TO EXIT." << endl;  exit(4);  }  } while (opt >= 1 && opt <= 8); }  // Paste your output here    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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_