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

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 20/10/2022** |
| **Teaching Assistant: Muhammad Sufyan Ashraf** | **Semester: Fall 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 8B. Basic Operations on Doubly 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. A **Doubly Linked List (DLL)** contains an extra pointer, typically called the previous pointer, together with the next pointer and data which are there in the singly 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 doubly 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:**  //doubly linklist  #include <iostream>  using namespace std; class node { public:  int data;  node \*nextPtr;  node \*previousPtr;   node(int d) {  data = d;  nextPtr = nullptr;  }   ~node() {  nextPtr = nullptr;  } };  class doubly { private:  node \*head;  node \*tail;  int count; public:  doubly() {  head = nullptr;  tail = nullptr;  count = 0;  }   ~doubly() {  head = nullptr;  tail = nullptr;  }   void append(int value){  node \*temp=new node(value);  temp->data=value;  if(head== nullptr){  temp->nextPtr= nullptr;  temp->previousPtr= nullptr;  head=temp;  }  else{  node \*newTemp=head;  while(newTemp->nextPtr!= nullptr){  newTemp= newTemp->nextPtr;  }  newTemp->nextPtr=temp;  temp->previousPtr= newTemp;  }  }  void prepend(int value){  node \*temp=new node(value);  temp->data=value;  if(head== nullptr){  temp->nextPtr= nullptr;  temp->previousPtr= nullptr;  head=temp;  }  else{  head->previousPtr=temp;  temp->nextPtr=head;  head=temp;  }  }  void addAtASpecificIndex(){  int value;  node \*temp=new node(value);   cout<<"enter value = ";  cin>>value;  int index;  cout<<"enter index = ";  cin>>index;  if(index<=0){  cout<<"ENTER POSITIVE VALUE."<<endl;  }  if(index==1){  temp->data=value;  if(head== nullptr){  temp->nextPtr= nullptr;  temp->previousPtr= nullptr;  head=temp;  }  else{  head->previousPtr=temp;  temp->nextPtr=head;  head=head->nextPtr;  head=temp;  }  }  else{  int i=1;  temp->data=value;  node \*newTemp=head;  while(i<=index-1){  newTemp=newTemp->nextPtr;  i++;  }  if(newTemp->nextPtr!= nullptr){  temp->nextPtr=newTemp->nextPtr;  temp->previousPtr= newTemp;  newTemp->nextPtr=temp;  if(temp->nextPtr!= nullptr){  temp->nextPtr->previousPtr=temp;  }  else{  cout<<"NO ITEM PRESENT BEHIND."<<endl;  }  }  }  }  void deleteFromLast(){  if(head== nullptr){  cout<<"LIST IS EMPTY."<<endl;  }  else{  node \*temp=head;  while(temp->nextPtr!= nullptr){  temp=temp->nextPtr;  }  temp->previousPtr->nextPtr= nullptr;  free(temp);  }  }  void deleteFromStart(){  if(head== nullptr){  cout<<"LIST IS EMPTY."<<endl;  }  else{  node \*temp=head;  head=head->nextPtr;  free(temp);  }  }  void deleteAtASpecificIndex(){  int index;  cout<<"enter index = ";  cin>>index;  if(index<=0){  cout<<"ENTER POSITIVE VALUE."<<endl;  }  if(index==1){  node \*temp=head;  if(head== nullptr){  cout<<"LIST IS EMPTY."<<endl;  }  else{  head=head->nextPtr;  free(temp);  }  }  else{  int i=1;  node \*newTemp=head;  while(i<=index-1){  newTemp=newTemp->nextPtr;  i++;  }  if(newTemp->nextPtr!= nullptr){  node \*toDelete=newTemp->nextPtr;  newTemp->nextPtr=newTemp->nextPtr->nextPtr;  if(newTemp->nextPtr->nextPtr!= nullptr){  newTemp->nextPtr->nextPtr->previousPtr= newTemp->nextPtr;  }  delete toDelete;  }  }  }  void display() {  node \*temp = head;  while (temp != nullptr) {  cout << "" << temp->data << "-> ";  temp = temp->nextPtr;  }  cout << "NULL" << endl;  }  };  **MAIN.CPP:**  // // Created by Lenovo on 11/28/2022. // #include <iostream> #include "Functions.h"  using namespace std;  int main() {  doubly d;  int opt;  do {  cout << "CHOOSE OPTIONS." << endl;  cout << "1.APPEND." << endl;  cout << "2.PREPEND." << endl;  cout << "3.ADD AT SPECIFIC INDEX." << endl;  cout << "4.ADD AT START." << endl;  cout << "5.ADD AT END." << endl;  cout << "6.DELETE AT SPECIFIC INDEX." << endl;  cout << "7.EXIT." << endl;  cin >> opt;  if (opt == 1) {  d.append(3);  d.append(4);  d.append(5);  d.append(6);  d.display();  }  if (opt == 2) {  int value;  cout<<"ENTER THE VALUE TO PREPEND."<<endl;  cin>>value;  d.prepend(value);  d.display();  }  if (opt == 3) {  d.addAtASpecificIndex();  d.display();  }  if (opt == 4) {   d.deleteFromStart();  d.display();  }  if (opt == 5) {  d.deleteFromLast();  d.display();  }  if (opt == 6) {   d.deleteAtASpecificIndex();  d.display();  }  if (opt == 7) {  cout << "YOU CHOOSE TO EXIT." << endl;  exit(4);  }  } while (opt >= 1 && opt <= 7); } |

#### **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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_