**Data Structures and Algorithms**

**Lab Journal - Lab 5**

Name: **MUHAMMAD HAMMAD ULLAH**

Enrollment #: **01-134231-050**

Class/Section: **3-D**

**Objective**

This lab session is aimed at introducing students to doubly linked lists.

**Task 1 :**

Give answers to the following.

|  |  |
| --- | --- |
| 1. | Redraw the following list after the given instructions are executed:  **ptr->next->prev = ptr->prev;**  **ptr->prev->next = ptr->next;**  **delete ptr;** |
| 2. | Consider the following doubly linked list : |

|  |  |
| --- | --- |
|  | Write C++ statements to:   1. Print the value 26 using the pointer ‘last’:   cout << last->data << endl;       1. Print the value 17 using the pointer ‘first:     Node\* current = first;  while (current != nullptr) {  if (current->data == 17) {  cout << current->data << endl;  break;  }  current = current->next; }       1. Print the address of the node containing value 9 using the pointer ‘last’:     Node\* current = last;  while (current != nullptr) {  if (current->data == 9) {  cout << current << endl;  break;  }  current = current->prev;  } |
| 3. | Given the following linked list, state what does each of the following statements refer to?      1.The data in the first node (5)  2.Null (as there is no node after the last)  3.NULL  4.The data in the third node (15)  5.The data in the second to last node (20) |

**Task 2 :**

Implement the following exercises.

|  |  |  |
| --- | --- | --- |
| **S No.** | **Exercise** | **Checked By:** |

**Exercise 1**

|  |
| --- |
| Implement the class Doubly Linked List to create a list of integers. You need to provide the implementation of the member functions as described in the following. **class DList**  **{ private:**  **Node \*head; public:**  **DList();**  **// Checks if the list is empty or not**  **bool emptyList();**  **// Inserts a new node with value ‘newV’ after the node containing value ‘oldV’. If a node with value ‘oldV’ does not exist, does not insert the new node. void insert\_after(int oldV, int newV);**  **// Deletes the node containing the specified value void deleteNode(int value);**  **// Inserts a new node at the start of the list void insert\_begin(int value);**  **// Inserts a new node at the end of the list void insert\_end(int value);**  **// Displays the values stored in the list starting from head void traverse();**    **};** |

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node\* prev;

Node(int value) {

data = value;

next = nullptr;

prev = nullptr;

}

};

class DList {

public:

Node\* head;

DList() {

head = nullptr;

}

bool emptyList() {

return head == nullptr;

}

void insertBegin(int value) {

Node\* newNode = new Node(value);

newNode->next = head;

if (head != nullptr) {

head->prev = newNode;

}

head = newNode;

}

void insertAfter(int oldV, int newV) {

Node\* current = head;

while (current != nullptr && current->data != oldV) {

current = current->next;

}

if (current != nullptr) {

Node\* newNode = new Node(newV);

newNode->prev = current;

newNode->next = current->next;

if (current->next != nullptr) {

current->next->prev = newNode;

}

current->next = newNode;

}

}

void deleteNode(int value) {

Node\* current = head;

while (current != nullptr && current->data != value) {

current = current->next;

}

if (current != nullptr) {

if (current == head) {

head = current->next;

}

else {

current->prev->next = current->next;

}

if (current->next != nullptr) {

current->next->prev = current->prev;

}

delete current;

}

}

void insertEnd(int value) {

if (emptyList()) {

insertBegin(value);

}

else {

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

Node\* newNode = new Node(value);

newNode->prev = current;

current->next = newNode;

}

}

void traverse() {

Node\* current = head;

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

};

int main() {

DList L1;

L1.insertEnd(10);

L1.insertBegin(5);

L1.insertAfter(10, 15);

L1.insertEnd(20);

cout << "Elements in the list: ";

L1.traverse();

L1.deleteNode(15);

cout << "Elements after deleting 15: ";

L1.traverse();

return 0;

}

**Exercise 2**

|  |
| --- |
| **Write C++ functions to :**   * **Reverse a doubly linked list.** * **Display the contents of alternate nodes of doubly linked list.** |

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* prev;

Node\* next;

Node(int value) : data(value), prev(nullptr), next(nullptr) {}

};

class DoublyLinkedList {

public:

Node\* head;

DoublyLinkedList() : head(nullptr) {}

void insert(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

return;

}

Node\* temp = head;

while (temp->next) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

void display() {

Node\* temp = head;

while (temp) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

void reverse() {

Node\* current = head;

Node\* temp = nullptr;

while (current) {

temp = current->prev;

current->prev = current->next;

current->next = temp;

current = current->prev;

}

if (temp) {

head = temp->prev;

}

}

};

int main() {

DoublyLinkedList d1;

d1.insert(1);

d1.insert(2);

d1.insert(3);

d1.insert(4);

d1.insert(5);

d1.insert(6);

d1.insert(7);

cout << "Original List: ";

d1.display();

d1.reverse();

cout << "Reversed List: ";

d1.display();

return 0;

}

**-------------------------------------------------------------------------------------------------------------------------------**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* prev;

Node\* next;

Node(int value) : data(value), prev(nullptr), next(nullptr) {}

};

class Dlist {

public:

Node\* head;

Dlist() : head(nullptr) {}

void insert(int value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

head = newNode;

}

else {

Node\* temp = head;

while (temp->next) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

}

void displayAlternateNodes() {

Node\* temp = head;

while (temp) {

cout << temp->data << " ";

if (temp->next) {

temp = temp->next->next;

}

else {

break;

}

}

cout << endl;

}

void display() {

Node\* temp = head;

while (temp) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

};

int main() {

Dlist d1;

d1.insert(1);

d1.insert(2);

d1.insert(3);

d1.insert(4);

d1.insert(5);

d1.insert(6);

cout << "Original List: ";

d1.display();

cout << "Alternate Nodes: ";

d1.displayAlternateNodes();

return 0;

}

**Implement the given exercises and get them checked by your instructor. If you are unable to complete the tasks in the lab session, deposit this journal alongwith your programs (printed) as per the submission date given.**

|  |  |  |
| --- | --- | --- |
| 1. | Exercise 1 |  |
| 2. | Exercise 2 |  |

+++++++++++++++++++++++++