Data Structures and Algorithms Lab Journal - Lab 4

Name: MUHAMMAD HAMMAD ULLAH

Enrollment #: 01-134231-050

Class/Section: 3-D

Objective

This lab session is aimed at introducing students to different variants of linked lists.

Task 1 :

Give answers to the following.

|  |  |
| --- | --- |
| 1. | In some cases, a dummy header node is introduced in linked lists so that every node has a predecessor and generic insertions and deletions can be implemented.    An empty list, in such cases is given by:    Provide the implementation of the constructor of a linked list with a dummy header node.  #include <iostream>  using namespace std;  struct Node {  int data;  Node\* next;  };  void insertAtStart(Node\*& head, int value) {  head = new Node{ value, head };  }  void insertAtEnd(Node\*& head, int value) {  Node\* newNode = new Node{ value, NULL };  if (!head) { head = newNode; return; }  Node\* current = head;  while (current->next) current = current->next;  current->next = newNode;  }  void insertAtPosition(Node\*& head, int value, int k) {  if (k == 0) return insertAtStart(head, value);  Node\* current = head;  for (int i = 0; current && i < k - 1; ++i) current = current->next;  if (!current) return;  current->next = new Node{ value, current->next };  }  void printList(Node\* head) {  while (head) {  cout << head->data << " -> ";  head = head->next;  }  cout << "NULL" << endl;  }  int main() {  Node\* head = new Node{ 1, new Node{2, new Node{5, new Node{7, new Node{4, NULL}}}} };  cout << "Original list: ";  printList(head);  insertAtStart(head, 8);  cout << "After inserting 8 at the start: ";  printList(head);  insertAtEnd(head, 9);  cout << "After inserting 9 at the end: ";  printList(head);  int value = 8, k = 4;  insertAtPosition(head, value, k);  cout << "After inserting " << value << " at position " << k + 1 << ": ";  printList(head);  return 0;  } |

|  |  |
| --- | --- |
| 2. | Consider the following two circular linked lists with pointers on their last nodes.      Write C++ statements to merge the two lists into one single list with contents POWERPOINT.  #include <iostream>  using namespace std;  struct Node {  char data;  Node\* next;  Node(char value) {  data = value;  next = nullptr;  }  };  class CircularLinkedList {  private:  Node\* last;  public:  CircularLinkedList() {  last = nullptr;  }  void insertEnd(char value) {  Node\* newNode = new Node(value);  if (last == nullptr) {  last = newNode;  newNode->next = newNode;  }  else {  newNode->next = last->next;  last->next = newNode;  last = newNode;  }  }  void merge(CircularLinkedList& otherList) {  if (otherList.last == nullptr) {  return;  }  if (this->last == nullptr) {  this->last = otherList.last;  }  else {  Node\* temp = this->last->next;  this->last->next = otherList.last->next;  otherList.last->next = temp;  this->last = otherList.last;  }  }  void traverse() {  if (last == nullptr) {  cout << "List is empty." << endl;  return;  }  Node\* current = last->next;  do {  cout << current->data << " ";  current = current->next;  } while (current != last->next);  cout << endl;  }  };  int main() {  CircularLinkedList l1, l2;  l1.insertEnd('P');  l1.insertEnd('O');  l1.insertEnd('W');  l1.insertEnd('E');  l1.insertEnd('R');  l2.insertEnd('P');  l2.insertEnd('O');  l2.insertEnd('I');  l2.insertEnd('N');  l2.insertEnd('T');  l1.merge(l2);  l1.traverse();  return 0;  } |

Task 2 :

Implement the following exercises.

Exercise 1

Implement the (class) Linked List with Dummy head node to create a list of integers. You need to provide the implementation of the member functions as described in the following.

class HList

{

private:

Node \* head; public:

HList();

// Checks if the list is empty or not

bool emptyList();

// Inserts a new node with value ‘newV’ after node with value ‘oldV’in the list

void insert\_after(int oldV, int newV);

// 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);

// Deletes a node of value ‘val’ from the list void delete\_Node(int val);

// Deletes a node from the beginning of the list void delete\_begin();

// Deletes a node from the end of the list void delete\_end();

// Displays the values stored in the list void traverse();

};

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

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

};

class HList {

private:

Node\* head;

public:

HList() {

head = new Node(0);

}

bool emptyList() {

return head->next == NULL;

}

void insert\_after(int oldV, int newV) {

Node\* current = head->next;

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

current = current->next;

}

if (current != NULL) {

Node\* newNode = new Node(newV);

newNode->next = current->next;

current->next = newNode;

}

else {

cout << "Node with value " << oldV << " not found.\n";

}

}

void insert\_begin(int value) {

Node\* newNode = new Node(value);

newNode->next = head->next;

head->next = newNode;

}

void insert\_end(int value) {

Node\* newNode = new Node(value);

Node\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

void delete\_Node(int val) {

Node\* current = head;

while (current->next != NULL && current->next->data != val) {

current = current->next;

}

if (current->next != NULL) {

Node\* temp = current->next;

current->next = temp->next;

delete temp;

}

else {

cout << "Node with value " << val << " not found.\n";

}

}

void delete\_begin() {

if (head->next != NULL) {

Node\* temp = head->next;

head->next = temp->next;

delete temp;

}

else {

cout << "List is empty, nothing to delete.\n";

}

}

void delete\_end() {

if (head->next == NULL) {

cout << "List is empty, nothing to delete.\n";

return;

}

Node\* current = head;

while (current->next->next != NULL) {

current = current->next;

}

Node\* temp = current->next;

current->next = NULL;

delete temp;

}

void traverse() {

Node\* current = head->next;

if (current == NULL) {

cout << "List is empty.\n";

return;

}

while (current != NULL) {

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

current = current->next;

}

cout << endl;

}

~HList() {

Node\* current = head;

while (current != NULL) {

Node\* next = current->next;

delete current;

current = next;

}

}

};

int main() {

HList l;

l.insert\_begin(10);

l.insert\_begin(20);

l.insert\_begin(30);

l.insert\_end(40);

cout << "List after inserting: ";

l.traverse();

l.insert\_after(20, 25);

cout << "List after inserting 25 after 20: ";

l.traverse();

l.delete\_begin();

cout << "List after deleting from the beginning: ";

l.traverse();

l.delete\_end();

cout << "List after deleting from the end: ";

l.traverse();

l.delete\_Node(25);

cout << "List after deleting node with value 25: ";

l.traverse();

return 0;

}

Exercise 2

Implement the (class) Circular Linked List to create a list of integers. You need to provide the implementation of the member functions as described in the following.

class CList

{

private:

Node \*head; public:

CList();

// Checks if the list is empty or not bool emptyList();

// Inserts a new node with value ‘value’ at position ‘pos’ in the list (restrict user from entering pos=1)

void insert\_at(int pos, 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);

// Deletes a node from position ‘pos’ of the list (restrict user from entering first node)

void deleteNode(int pos);

// Deletes a node from the beginning of the list void delete\_begin();

// Deletes a node from the end of the list void delete\_end();

// Displays the values stored in the list void traverse();

};

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

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

};

class CList {

private:

Node\* head;

public:

CList() {

head = NULL;

}

bool emptyList() {

return head == NULL;

}

void insert\_at(int pos, int value) {

if (pos == 1) {

cout << "Cannot insert at position 1. Use insert\_begin instead.\n";

return;

}

Node\* newNode = new Node(value);

if (emptyList()) {

cout << "List is empty. Use insert\_begin for the first element.\n";

return;

}

Node\* current = head;

int currentPosition = 2;

while (current->next != head && currentPosition < pos) {

current = current->next;

currentPosition++;

}

newNode->next = current->next;

current->next = newNode;

}

void insert\_begin(int value) {

Node\* newNode = new Node(value);

if (emptyList()) {

head = newNode;

newNode->next = head;

}

else {

Node\* current = head;

while (current->next != head) {

current = current->next;

}

newNode->next = head;

current->next = newNode;

head = newNode;

}

}

void insert\_end(int value) {

Node\* newNode = new Node(value);

if (emptyList()) {

head = newNode;

newNode->next = head;

}

else {

Node\* current = head;

while (current->next != head) {

current = current->next;

}

current->next = newNode;

newNode->next = head;

}

}

void deleteNode(int pos) {

if (pos == 1) {

cout << "Cannot delete the first node using this function\n";

return;

}

if (emptyList()) {

cout << "List is empty\n";

return;

}

Node\* current = head;

int currentPos = 2;

while (current->next != head && currentPos < pos) {

current = current->next;

currentPos++;

}

if (current->next != head) {

Node\* temp = current->next;

current->next = temp->next;

delete temp;

}

else {

cout << "Position " << pos << " is out of bounds.\n";

}

}

void delete\_begin() {

if (emptyList()) {

cout << "List is empty\n";

return;

}

Node\* current = head;

if (current->next == head) {

delete head;

head = NULL;

}

else {

while (current->next != head) {

current = current->next;

}

Node\* temp = head;

head = head->next;

current->next = head;

delete temp;

}

}

void delete\_end() {

if (emptyList()) {

cout << "List is empty, nothing to delete.\n";

return;

}

Node\* current = head;

if (current->next == head) {

delete head;

head = NULL;

}

else {

while (current->next->next != head) {

current = current->next;

}

Node\* temp = current->next;

current->next = head;

delete temp;

}

}

void traverse() {

if (emptyList()) {

cout << "List is empty.\n";

return;

}

Node\* current = head;

do {

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

current = current->next;

} while (current != head);

cout << endl;

}

~CList() {

if (!emptyList()) {

Node\* current = head;

Node\* temp;

while (current->next != head) {

temp = current->next;

delete current;

current = temp;

}

delete current;

}

}

};

int main() {

CList l;

l.insert\_begin(10);

l.insert\_begin(20);

l.insert\_begin(30);

l.insert\_end(40);

l.insert\_end(50);

cout << "List after insertions: ";

l.traverse();

l.insert\_at(3, 25);

cout << "List after inserting 25 at 3RD position : ";

l.traverse();

l.delete\_begin();

cout << "List after deleting from the beginning: ";

l.traverse();

l.delete\_end();

cout << "List after deleting from the end: ";

l.traverse();

l.deleteNode(3);

cout << "List after deleting at 3RD position : ";

l.traverse();

return 0;

}s

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.

|  |  |  |
| --- | --- | --- |
| **S No.** | **Exercise** | **Checked By:** |
| 1. | Exercise 1 |  |
| 2. | Exercise 2 |  |

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