**ASSINMENT 1**

**SUBMITTED BY :**

Saif ul rehman

**SUBMITTED TO :**

MR Azib Mahmood

**ROLL NO :**

22011556\_074

**SUBJECT:**

Data structure and Algorithm

**UNIVERSITY OF GUJRAT**

**Q.\**

**LinkedList Assignment**

**•Complete this program**

**•Search**

**•update at any n point**

**•Insert at any n position**

**•Delete from beginning**

**•Delete from end**

**•Delete from any n position**

**•Search and update any point**

#include <iostream>

Using namespace std;

class Node {

public:

int data;

Node\* next;

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

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() : head(nullptr) {}

// Function to insert a new node at the end of the linked list

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

} else {

Node\* current = head;

while (current->next) {

current = current->next;

}

current->next = newNode;

}

}

// Function to insert a new node at a specific position in the linked list

void insertAtPosition(int value, int position) {

Node\* newNode = new Node(value);

if (position == 1 || !head) {

newNode->next = head;

head = newNode;

} else {

Node\* current = head;

for (int i = 1; i < position - 1 && current; ++i) {

current = current->next;

}

if (current) {

newNode->next = current->next;

current->next = newNode;

} else {

cout << "Invalid position!" << endl;

}

}

}

// Function to delete the first node from the linked list

void deleteFromBeginning() {

if (head) {

Node\* temp = head;

head = head->next;

delete temp;

} else {

cout << "List is empty!" << endl;

}

}

// Function to delete the last node from the linked list

void deleteFromEnd() {

if (head) {

if (!head->next) {

delete head;

head = nullptr;

} else {

Node\* current = head;

while (current->next->next) {

current = current->next;

}

delete current->next;

current->next = nullptr;

}

} else {

cout << "List is empty!" << endl;

}

}

// Function to delete a node from a specific position in the linked list

void deleteFromPosition(int position) {

if (head) {

if (position == 1) {

Node\* temp = head;

head = head->next;

delete temp;

} else {

Node\* current = head;

for (int i = 1; i < position - 1 && current; ++i) {

current = current->next;

}

if (current && current->next) {

Node\* temp = current->next;

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

delete temp;

} else {

cout << "Invalid position!" << endl;

}

}

} else {

cout << "List is empty!" << endl;

}

}

// Function to search for a node with a specific value

bool search(int value) {

Node\* current = head;

while (current) {

if (current->data == value) {

return true;

}

current = current->next;

}

return false;

}

// Function to update the value at a specific position

void updateAtPosition(int value, int position) {

Node\* current = head;

for (int i = 1; i < position && current; ++i) {

current = current->next;

}

if (current) {

current->data = value;

} else {

cout << "Invalid position!" << endl;

}

}

// Function to display the linked list

void display() {

Node\* current = head;

while (current) {

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

current = current->next;

}

cout << endl;

}

// Destructor to free allocated memory

~LinkedList() {

Node\* current = head;

while (current) {

Node\* next = current->next;

delete current;

current = next;

}

head = nullptr;

}

};

int main() {

LinkedList myList;

myList.insertAtEnd(10);

myList.insertAtEnd(20);

myList.insertAtEnd(30);

std::cout << "Original List: ";

myList.display();

// Search and update

int searchValue = 20;

if (myList.search(searchValue)) {

cout << "Value " << searchValue << " found in the list." << endl;

int newValue = 25;

myList.updateAtPosition(newValue, 2);

std::cout << "List after updating: ";

myList.display();

} else {

cout << "Value " << searchValue << " not found in the list." << endl;

}

// Insert at position

myList.insertAtPosition(15, 2);

cout << "List after inserting at position: ";

myList.display();

// Delete from beginning

myList.deleteFromBeginning();

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

myList.display();

// Delete from end

myList.deleteFromEnd();

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

myList.display();

// Delete from position

myList.deleteFromPosition(2);

cout << "List after deleting from position: ";

myList.display();

return 0;

}