

**Name:**

Malaika Kashaf.

**Roll no:**

22011556-051.

**Section:**

A.

**Course Title:**

Data structures and algorithms.

**Course Code:**

IT-209.

**Submitted To:**

Sir.Azib Mehmood.

**Department:**

Information Technology.

**Statement:**

**Write a program of link list which performs the following tasks:**

**1.Function to insert at the beginning**

**2.Function to display or traverse the list**

**3.Function to insert at end**

**4.Function to insert at nth position**

**5.Search function**

**6.Update at any nth position**

**7.Insert at any nth position**

**8.Delete from beginning**

**9.Delete from end**

**10.Delete from any nth position**

**11. Search and update at any point**

**Program :**

#include<iostream>

using namespace std;

// Node structure

class LinkedListNode {

public:

int value;

LinkedListNode\* next;

// Constructor

LinkedListNode(int val) {

value = val;

next = NULL;

}

};

// Function to insert at the head

void insertAtHead(LinkedListNode\*& head, int val) {

LinkedListNode\* newNode = new LinkedListNode(val);

newNode->next = head;

head = newNode;

}

// Function to display or traverse the linked list

void displayLinkedList(LinkedListNode\* head) {

LinkedListNode\* current = head;

while (current != NULL) {

cout << current->value << "->";

current = current->next;

}

cout << "NULL" << endl;

}

// Function to insert at the end

void insertAtEnd(LinkedListNode\*& head, int val) {

LinkedListNode\* newNode = new LinkedListNode(val);

if (head == NULL) {

head = newNode;

return;

}

LinkedListNode\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

// Function to insert at any nth position

void insertAtPosition(LinkedListNode\*& head, int val, int position) {

if (position == 0) {

insertAtHead(head, val);

return;

}

LinkedListNode\* newNode = new LinkedListNode(val);

LinkedListNode\* temp = head;

int currentPosition = 0;

while (currentPosition != position - 1 && temp != NULL) {

temp = temp->next;

currentPosition++;

}

if (temp != NULL) {

newNode->next = temp->next;

temp->next = newNode;

}

}

// Search function

bool searchLinkedList(LinkedListNode\* head, int target) {

LinkedListNode\* current = head;

while (current != NULL) {

if (current->value == target)

return true;

current = current->next;

}

return false;

}

// Update at any nth position

void updateNodeValue(LinkedListNode\* head, int val, int position) {

LinkedListNode\* current = head;

int currentPosition = 0;

while (current != NULL && currentPosition != position) {

current = current->next;

currentPosition++;

}

if (current != NULL) {

current->value = val;

cout << "Node at position " << position << " updated successfully!" << endl;

} else {

cout << "Invalid position to update." << endl;

}

}

// Insert at any nth position

void insertNodeAtPosition(LinkedListNode\*& head, int val, int position) {

LinkedListNode\* newNode = new LinkedListNode(val);

if (position == 0) {

newNode->next = head;

head = newNode;

} else {

LinkedListNode\* current = head;

int currentPosition = 0;

while (current != NULL && currentPosition != position - 1) {

current = current->next;

currentPosition++;

}

if (current != NULL) {

newNode->next = current->next;

current->next = newNode;

}

}

}

// Delete from the beginning

void deleteFromBeginning(LinkedListNode\*& head) {

if (head != NULL) {

LinkedListNode\* temp = head;

head = head->next;

delete temp;

cout << "Node deleted from the beginning successfully!" << endl;

} else {

cout << "Linked list is empty. Unable to delete." << endl;

}

}

// Delete from the end

void deleteFromEnd(LinkedListNode\*& head) {

if (head == NULL) {

cout << "Linked list is empty. Unable to delete." << endl;

return;

}

if (head->next == NULL) {

delete head;

head = NULL;

cout << "Node deleted from the end successfully!" << endl;

return;

}

LinkedListNode\* current = head;

LinkedListNode\* prev = NULL;

while (current->next != NULL) {

prev = current;

current = current->next;

}

if (prev != NULL) {

prev->next = NULL;

delete current;

}

}

// Delete from any nth position

void deleteFromNthPosition(LinkedListNode\*& head, int position) {

if (head == NULL) {

cout << "Linked list is empty. Unable to delete." << endl;

return;

}

if (position == 0) {

LinkedListNode\* temp = head;

head = head->next;

delete temp;

cout << "Node deleted from position " << position << " successfully!" << endl;

return;

}

LinkedListNode\* current = head;

LinkedListNode\* prev = NULL;

int count = 0;

while (current != NULL && count < position) {

prev = current;

current = current->next;

count++;

}

if (current != NULL) {

prev->next = current->next;

delete current;

cout << "Node deleted from position " << position << " successfully!" << endl;

} else {

cout << "Position " << position << " exceeds the length of the linked list." << endl;

}

}

// Search and update at any point

void searchAndUpdateValue(LinkedListNode\* head, int searchVal, int updateVal) {

if (head == NULL) {

cout << "Linked list is empty. Unable to search and update." << endl;

return;

}

LinkedListNode\* current = head;

bool found = false;

while (current != NULL) {

if (current->value == searchVal) {

current->value = updateVal;

found = true;

break;

}

current = current->next;

}

if (found) {

cout << "Successfully updated the node with value " << searchVal << " to " << updateVal << "." << endl;

} else {

cout << "Node with value " << searchVal << " not found in the linked list." << endl;

}

}

// Main function

int main() {

LinkedListNode\* head = NULL;

// Example usage of linked list operations

// Insert a node with value 10 at the end

insertAtEnd(head, 10);

displayLinkedList(head);

// Insert a node with value 20 at the end

insertAtEnd(head, 20);

displayLinkedList(head);

// Insert a node with value 30 at position 1

insertAtPosition(head, 30, 1);

displayLinkedList(head);

// Search for node with value 40 and display the result

cout << "Search for value 40: " << (searchLinkedList(head, 40) ? "Found" : "Not Found") << endl;

// Update the node at position 2 with value 77

updateNodeValue(head, 77, 2);

displayLinkedList(head);

// Insert a node with value 22 at position 3

insertNodeAtPosition(head, 22, 3);

displayLinkedList(head);

// Delete node from the beginning

deleteFromBeginning(head);

displayLinkedList(head);

// Delete a node from the end

deleteFromEnd(head);

displayLinkedList(head);

// Delete a node from the 5th position

deleteFromNthPosition(head, 5);

displayLinkedList(head);

// Search a node with value 20 and update it with 33

searchAndUpdateValue(head, 20, 33);

displayLinkedList(head);

return 0;

}

