

**Name:**

Huzaifa Waqar.

**Roll no:**

22011556-035.

**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 class

class ListNode {

public:

int data;

ListNode\* next;

// Constructor

ListNode(int value) {

data = value;

next = NULL;

}

};

// Function to insert at the head

void insertAtHead(ListNode\*& head, int value) {

ListNode\* newNode = new ListNode(value);

newNode->next = head;

head = newNode;

}

// Function to display or traverse the linked list

void displayList(ListNode\* head) {

ListNode\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << "NULL" << endl;

}

// Function to insert at the end

void insertAtEnd(ListNode\*& head, int value) {

ListNode\* newNode = new ListNode(value);

if (head == NULL) {

head = newNode;

return;

}

ListNode\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

// Function to insert at any nth position

void insertAtPosition(ListNode\*& head, int value, int position) {

if (position == 0) {

insertAtHead(head, value);

return;

}

ListNode\* newNode = new ListNode(value);

ListNode\* 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 searchNode(ListNode\* head, int target) {

ListNode\* current = head;

while (current != NULL) {

if (current->data == target)

return true;

current = current->next;

}

return false;

}

// Update at any nth position

void updateNodeValue(ListNode\* head, int value, int position) {

ListNode\* current = head;

int currentPosition = 0;

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

current = current->next;

currentPosition++;

}

if (current != NULL) {

current->data = value;

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

} else {

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

}

}

// Insert at any nth position

void insertNodeAtPosition(ListNode\*& head, int value, int position) {

ListNode\* newNode = new ListNode(value);

if (position == 0) {

newNode->next = head;

head = newNode;

} else {

ListNode\* 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(ListNode\*& head) {

if (head != NULL) {

ListNode\* 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(ListNode\*& 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;

}

ListNode\* current = head;

ListNode\* 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(ListNode\*& head, int position) {

if (head == NULL) {

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

return;

}

if (position == 0) {

ListNode\* temp = head;

head = head->next;

delete temp;

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

return;

}

ListNode\* current = head;

ListNode\* 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(ListNode\* head, int searchValue, int updateValue) {

if (head == NULL) {

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

return;

}

ListNode\* current = head;

bool found = false;

while (current != NULL) {

if (current->data == searchValue) {

current->data = updateValue;

found = true;

break;

}

current = current->next;

}

if (found) {

cout << "Successfully updated the node with value " << searchValue << " to " << updateValue << "." << endl;

} else {

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

}

}

// Main function

int main() {

ListNode\* head = NULL;

// Example usage of linked list operations

// Insert a node with value 1 at the end

insertAtEnd(head, 1);

displayList(head);

// Insert a node with value 2 at the end

insertAtEnd(head, 2);

displayList(head);

// Insert a node with value 3 at position 1

insertAtPosition(head, 3, 1);

displayList(head);

// Search for node with value 4 and display the result

cout << "Search for value 4: " << (searchNode(head, 4) ? "Found" : "Not Found") << endl;

// Update the node at position 2 with value 77

updateNodeValue(head, 77, 2);

displayList(head);

// Insert a node with value 22 at position 3

insertNodeAtPosition(head, 22, 3);

displayList(head);

// Delete node from the beginning

deleteFromBeginning(head);

displayList(head);

// Delete a node from the end

deleteFromEnd(head);

displayList(head);

// Delete a node from the 5th position

deleteFromNthPosition(head, 5);

displayList(head);

// Search a node with value 2 and update it with 3

searchAndUpdateValue(head, 2, 3);

displayList(head);

return 0;

}

