#include <iostream>

using namespace std;

// Node class to represent a node in the linked list

class Node {

public:

int data;

Node\* next;

};

// LinkedList class to implement the linked list operations

class LinkedList {

private:

Node\* head;

public:

// Constructor to initialize the head pointer

LinkedList() {

head = nullptr;

}

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

void insert(int value) {

Node\* newNode = new Node();

newNode->data = value;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

// Function to display the elements of the linked list

void display() {

if (head == nullptr) {

cout << "Linked list is empty." << endl;

} else {

Node\* temp = head;

while (temp != nullptr) {

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

temp = temp->next;

}

cout << endl;

}

}

// Function to search for a value in the linked list

void search(int value) {

if (head == nullptr) {

cout << "Linked list is empty." << endl;

} else {

Node\* temp = head;

int position = 1;

bool found = false;

while (temp != nullptr) {

if (temp->data == value) {

cout << "Value " << value << " found at position " << position << "." << endl;

found = true;

break;

}

temp = temp->next;

position++;

}

if (!found) {

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

}

}

}

// Function to update a value in the linked list

void update(int oldValue, int newValue) {

if (head == nullptr) {

cout << "Linked list is empty." << endl;

} else {

Node\* temp = head;

bool updated = false;

while (temp != nullptr) {

if (temp->data == oldValue) {

temp->data = newValue;

cout << "Value " << oldValue << " updated to " << newValue << "." << endl;

updated = true;

break;

}

temp = temp->next;

}

if (!updated) {

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

}

}

}

// Function to delete a node from the beginning of the linked list

void deleteFromBeginning() {

if (head == nullptr) {

cout << "Linked list is empty." << endl;

} else {

Node\* temp = head;

head = head->next;

delete temp;

cout << "Node deleted from the beginning of the linked list." << endl;

}

}

// Function to delete a node from the end of the linked list

void deleteFromEnd() {

if (head == nullptr) {

cout << "Linked list is empty." << endl;

} else if (head->next == nullptr) {

delete head;

head = nullptr;

cout << "Node deleted from the end of the linked list." << endl;

} else {

Node\* temp = head;

while (temp->next->next != nullptr) {

temp = temp->next;

}

delete temp->next;

temp->next = nullptr;

cout << "Node deleted from the end of the linked list." << endl;

}

}

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

void deleteFromPosition(int position) {

if (head == nullptr) {

cout << "Linked list is empty." << endl;

} else if (position == 1) {

Node\* temp = head;

head = head->next;

delete temp;

cout << "Node deleted from position " << position << " of the linked list." << endl;

} else {

Node\* temp = head;

int currentPosition = 1;

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

temp = temp->next;

currentPosition++;

}

if (temp == nullptr || temp->next == nullptr) {

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

} else {

Node\* nodeToDelete = temp->next;

temp->next = temp->next->next;

delete nodeToDelete;

cout << "Node deleted from position " << position << " of the linked list." << endl;

}

}

}

};

int main() {

LinkedList linkedList;

int choice, value, oldValue, newValue, position;

do {

cout << "Menu:" << endl;

cout << "1. Insert" << endl;

cout << "2. Display" << endl;

cout << "3. Search" << endl;

cout << "4. Update" << endl;

cout << "5. Delete from beginning" << endl;

cout << "6. Delete from end" << endl;

cout << "7. Delete from position" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter the value to insert: ";

cin >> value;

linkedList.insert(value);

break;

case 2:

linkedList.display();

break;

case 3:

cout << "Enter the value to search: ";

cin >> value;

linkedList.search(value);

break;

case 4:

cout << "Enter the value to update: ";

cin >> oldValue;

cout << "Enter the new value: ";

cin >> newValue;

linkedList.update(oldValue, newValue);

break;

case 5:

linkedList.deleteFromBeginning();

break;

case 6:

linkedList.deleteFromEnd();

break;

case 7:

cout << "Enter the position to delete: ";

cin >> position;

linkedList.deleteFromPosition(position);

break;

case 0:

cout << "Exiting..." << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

break;

}

cout << endl;

} while (choice != 0);

return 0;

}