Q-1: Implement a singly linked list with operations to insert elements at the front, end, and a specific position in the list, and to delete elements from the list.

Sample test case:

|  |
| --- |
| Input:  insertEnd(5);  insertFront(3);  insertAtPosition(4, 2);  insertEnd(7);  deleteElement(4);  Output:  3 4 5 7  3 5 7 |

Solution:

#include <iostream>

using namespace std;

// Node structure for the linked list

struct Node {

int data;

Node\* next;

};

class SinglyLinkedList {

private:

Node\* head;

public:

SinglyLinkedList() {

head = nullptr;

}

// Function to insert an element at the front of the linked list

void insertFront(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = head;

head = newNode;

}

// Function to insert an element at the end of the linked list

void insertEnd(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

return;

}

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

// Function to insert an element at a specific position in the linked list

void insertAtPosition(int data, int position) {

Node\* newNode = new Node;

newNode->data = data;

if (position == 1) {

newNode->next = head;

head = newNode;

return;

}

Node\* current = head;

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

current = current->next;

}

if (current == nullptr) {

cout << "Invalid position.\n";

return;

}

newNode->next = current->next;

current->next = newNode;

}

// Function to delete an element from the linked list

void deleteElement(int data) {

Node\* temp = head;

Node\* prev = nullptr;

// If the element to be deleted is at the head

if (temp != nullptr && temp->data == data) {

head = temp->next;

delete temp;

return;

}

// Search for the element to be deleted

while (temp != nullptr && temp->data != data) {

prev = temp;

temp = temp->next;

}

// If the element is not found

if (temp == nullptr) {

cout << "Element not found.\n";

return;

}

// Unlink the node from the list and free memory

prev->next = temp->next;

delete temp;

}

// Function to print the elements of the linked list

void printList() {

Node\* current = head;

while (current != nullptr) {

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

current = current->next;

}

cout << endl;

}

};

int main() {

SinglyLinkedList list;

list.insertEnd(5);

list.insertFront(3);

list.insertAtPosition(4, 2);

list.insertEnd(7);

list.printList();

list.deleteElement(4);

list.printList();

return 0;

}

Q-2: Implement a linked list and reverse it.

Sample test case:

|  |
| --- |
| Input: Enter elements for linked list (enter -1 to stop):  4 5 7 8 -1  Linked List before reversal: 4 5 7 8  Output: Linked List after reversal: 8 7 5 4 |

Solution:

#include <iostream>

using namespace std;

// Node structure for linked list

struct Node {

int data;

Node\* next;

};

// Function to insert an element at the end of the linked list

Node\* insertAtEnd(Node\* head, int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = nullptr;

if (head == nullptr) {

return newNode;

}

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

return head;

}

// Function to display the linked list

void displayLinkedList(Node\* head) {

Node\* current = head;

while (current != nullptr) {

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

current = current->next;

}

}

// Function to reverse the linked list

Node\* reverseLinkedList(Node\* head) {

Node\* prev = nullptr;

Node\* current = head;

Node\* next = nullptr;

while (current != nullptr) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

return prev;

}

int main() {

Node\* head = nullptr;

int data;

// Create the linked list with user input

cout << "Enter elements for linked list (enter -1 to stop):\n";

while (true) {

cin >> data;

if (data == -1)

break;

head = insertAtEnd(head, data);

}

// Display the linked list before reversal

cout << "Linked List before reversal: ";

displayLinkedList(head);

// Reverse the linked list

head = reverseLinkedList(head);

// Display the reversed linked list

cout << "\nLinked List after reversal: ";

displayLinkedList(head);

// Free memory by deleting nodes

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}

Q-3: Implement a linked list and implement function to create and detect a loop.

Sample test case:

|  |
| --- |
| Input:  Enter elements for linked list (enter -1 to stop):  4 5 7 9 -1  Linked List: 4 5 7 9  Before creation of loop  Linked List does not contain a loop.  Enter the position of the node where loop starts (0-indexed, enter -1 for no loop): 2  After creation of loop  Output: Linked List contains a loop. |
| Solution: |

#include <iostream>

using namespace std;

// Node structure for linked list

struct Node {

int data;

Node\* next;

};

// Function to insert an element at the end of the linked list

Node\* insertAtEnd(Node\* head, int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = nullptr;

if (head == nullptr) {

return newNode;

}

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

return head;

}

// Function to create a loop in the linked list (for testing)

void createLoop(Node\* head, int pos) {

if (head == nullptr) {

return;

}

Node\* current = head;

Node\* loopStart = nullptr;

int i = 0;

while (current->next != nullptr) {

if (i == pos) {

loopStart = current;

}

current = current->next;

i++;

}

current->next = loopStart;

}

// Function to detect loop in the linked list using Floyd's cycle detection algorithm

bool detectLoop(Node\* head) {

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

return false;

}

Node\* slow = head;

Node\* fast = head;

while (fast != nullptr && fast->next != nullptr) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) {

return true;

}

}

return false;

}

int main() {

Node\* head = nullptr;

int data;

// Create the linked list with user input

cout << "Enter elements for linked list (enter -1 to stop):\n";

while (true) {

cin >> data;

if (data == -1)

break;

head = insertAtEnd(head, data);

}

// Display the linked list

cout << "Linked List: ";

Node\* current = head;

while (current != nullptr) {

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

current = current->next;

}

cout<<"\nBefore creation of loop \n";

// Detect loop in the linked list

if (detectLoop(head)) {

cout << "Linked List contains a loop.\n ";

} else {

cout << "Linked List does not contain a loop.\n";

}

// Create a loop in the linked list (for testing)

int pos;

cout << "\nEnter the position of the node where loop starts (0-indexed, enter -1 for no loop): ";

cin >> pos;

createLoop(head, pos);

cout<<"After creation of loop \n";

// Detect loop in the linked list

if (detectLoop(head)) {

cout << "Linked List contains a loop.";

} else {

cout << "Linked List does not contain a loop.";

}

// Free memory by deleting nodes

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}

Q-4: Implement a linked list and check if it is a palindrome.

Sample test case:

|  |
| --- |
| Input: Enter elements for linked list (enter -1 to stop): 1 2 1 -1  Output:  Linked List: 1 2 1  Linked List is a palindrome. |

Solution:

#include <iostream>

#include <stack>

using namespace std;

// Node structure for linked list

struct Node {

int data;

Node\* next;

};

// Function to insert an element at the end of the linked list

Node\* insertAtEnd(Node\* head, int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = nullptr;

if (head == nullptr) {

return newNode;

}

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

return head;

}

// Function to display the linked list

void displayLinkedList(Node\* head) {

Node\* current = head;

while (current != nullptr) {

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

current = current->next;

}

}

// Function to check if the linked list is palindrome using stack

bool isPalindrome(Node\* head) {

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

return true;

}

Node\* slow = head;

Node\* fast = head;

// Find the middle of the linked list

while (fast != nullptr && fast->next != nullptr) {

slow = slow->next;

fast = fast->next->next;

}

// Push the first half of the linked list onto the stack

stack<int> halfList;

Node\* current = head;

while (current != slow) {

halfList.push(current->data);

current = current->next;

}

// If the length of the linked list is odd, skip the middle element

if (fast != nullptr) {

slow = slow->next;

}

// Check for palindrome by comparing second half with the elements popped from the stack

while (slow != nullptr) {

if (slow->data != halfList.top()) {

return false;

}

halfList.pop();

slow = slow->next;

}

return true;

}

int main() {

Node\* head = nullptr;

int data;

// Create the linked list with user input

cout << "Enter elements for linked list (enter -1 to stop):\n";

while (true) {

cin >> data;

if (data == -1)

break;

head = insertAtEnd(head, data);

}

// Display the linked list

cout << "Linked List: ";

displayLinkedList(head);

// Check if the linked list is palindrome

if (isPalindrome(head)) {

cout << "\nLinked List is a palindrome.";

} else {

cout << "\nLinked List is not a palindrome.";

}

// Free memory by deleting nodes

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}