Q-1: Implement a linked list and insert elements at the end.

Sample test case:

|  |
| --- |
| Input: Enter elements for linked list (enter -1 to stop): 5 7 9 -1  Output: Linked List: 5 7 9 |

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;

}

}

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);

// Free memory by deleting nodes

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}

Q-2: Implement a linked list and swap two nodes.

Sample test case:

|  |
| --- |
| Input: Enter elements for linked list (enter -1 to stop): 6 8 9 12 17 -1  Linked List before swapping nodes: 6 8 9 12 17  Enter the positions (0-indexed) of the nodes to be swapped: 1 3  Output: Linked List after swapping nodes: 6 12 9 8 17 |

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 find the previous node of a given node in the linked list

Node\* findPreviousNode(Node\* head, Node\* targetNode) {

Node\* current = head;

Node\* prev = nullptr;

while (current != nullptr) {

if (current == targetNode) {

return prev;

}

prev = current;

current = current->next;

}

return nullptr; // Target node not found

}

// Function to swap two nodes in the linked list

Node\* swapNodes(Node\* head, Node\* node1, Node\* node2) {

if (head == nullptr || node1 == nullptr || node2 == nullptr) {

return head;

}

// Find the previous nodes of node1 and node2

Node\* prevNode1 = findPreviousNode(head, node1);

Node\* prevNode2 = findPreviousNode(head, node2);

// Check if any of the nodes are not present in the linked list

if (prevNode1 == nullptr || prevNode2 == nullptr) {

return head;

}

// Update the next pointers to swap the nodes

prevNode1->next = node2;

prevNode2->next = node1;

Node\* temp = node1->next;

node1->next = node2->next;

node2->next = temp;

// Update the head if either node1 or node2 was the head of the linked list

if (head == node1) {

head = node2;

} else if (head == node2) {

head = node1;

}

return head;

}

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 swapping nodes

cout << "Linked List before swapping nodes: ";

displayLinkedList(head);

// Prompt the user to enter the positions of the nodes to be swapped (0-indexed)

int position1, position2;

cout << "\nEnter the positions (0-indexed) of the nodes to be swapped: ";

cin >> position1 >> position2;

// Find the nodes to be swapped based on the positions entered

Node\* node1 = head;

Node\* node2 = head;

for (int i = 0; i < position1 && node1 != nullptr; i++) {

node1 = node1->next;

}

for (int i = 0; i < position2 && node2 != nullptr; i++) {

node2 = node2->next;

}

// Swap the nodes

head = swapNodes(head, node1, node2);

// Display the linked list after swapping nodes

cout << "Linked List after swapping nodes: ";

displayLinkedList(head);

// Free memory by deleting nodes

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}

Q-3: Implement a circular linked list and split it into two halves.

Sample test case:

|  |
| --- |
| Input:  Enter elements for circular linked list (enter -1 to stop): 5 8 9 12 15 19 4 -1  Circular Linked List: 5 8 9 12 15 19 4  Output:  First Half of Circular Linked List: 5 15 19 4  Second Half of Circular Linked List: 15 19 4 5 |
| Solution: |

#include <iostream>

using namespace std;

// Node structure for circular linked list

struct Node {

int data;

Node\* next;

};

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

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

Node\* newNode = new Node;

newNode->data = data;

if (head == nullptr) {

newNode->next = newNode; // Only one node, points to itself

return newNode;

}

Node\* tail = head;

while (tail->next != head) {

tail = tail->next;

}

tail->next = newNode;

newNode->next = head;

return head;

}

// Function to display the circular linked list

void displayCircularLinkedList(Node\* head) {

if (head == nullptr) {

return;

}

Node\* current = head;

do {

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

current = current->next;

} while (current != head);

}

// Function to split a circular linked list into two halves

void splitCircularLinkedList(Node\* head, Node\*\* head1, Node\*\* head2) {

if (head == nullptr) {

\*head1 = \*head2 = nullptr;

return;

}

Node\* slow = head;

Node\* fast = head->next;

// Move fast pointer twice as fast as slow pointer

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

slow = slow->next;

fast = fast->next->next;

}

// If there are even number of nodes, move fast one more step

if (fast->next == head) {

fast = fast->next;

}

// Split the circular linked list into two halves

\*head1 = head;

\*head2 = slow->next;

slow->next = head;

fast->next = \*head2;

}

int main() {

Node\* head = nullptr;

int data;

// Create the circular linked list with user input

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

while (true) {

cin >> data;

if (data == -1)

break;

head = insertAtEnd(head, data);

}

// Display the circular linked list

cout << "Circular Linked List: ";

displayCircularLinkedList(head);

// Split the circular linked list into two halves

Node\* head1 = nullptr;

Node\* head2 = nullptr;

splitCircularLinkedList(head, &head1, &head2);

// Display the first half of the circular linked list

cout << "\nFirst Half of Circular Linked List: ";

displayCircularLinkedList(head1);

// Display the second half of the circular linked list

cout << "\nSecond Half of Circular Linked List: ";

displayCircularLinkedList(head2);

// Free memory by deleting nodes (note: this is not necessary for circular linked list implementation)

return 0;

}

Q-4: Given a singly linked list. Find the number of nodes present in the linked list.

Sample test case:

|  |
| --- |
| Input: Enter elements for linked list (enter -1 to stop): 9 8 5 12 4 7 5 1 -1  Linked List: 9 8 5 12 4 7 5 1  Output: Length of the Linked List: 8 |

Solution:

#include <iostream>

using namespace std;

// Node structure for singly 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 find the length of the linked list

int findLength(Node\* head) {

int length = 0;

Node\* current = head;

while (current != nullptr) {

length++;

current = current->next;

}

return length;

}

// Function to display the linked list

void displayLinkedList(Node\* head) {

Node\* current = head;

while (current != nullptr) {

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

current = current->next;

}

}

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);

// Find the length of the linked list

int length = findLength(head);

// Display the length of the linked list

cout << "\nLength of the Linked List: " << length << endl;

// Free memory by deleting nodes

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}