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

Sample test case:

|  |
| --- |
| Input: Enter elements for linked list (enter -1 to stop): 4 5 3 -1  Output: Linked List: 3 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 beginning of the linked list

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

Node\* newNode = new Node;

newNode->data = data;

newNode->next = head;

return newNode;

}

// 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 = insertAtBeginning(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 find the middle node.

Sample test case:

|  |
| --- |
| Input: Enter elements for linked list (enter -1 to stop): 4 6 8 9 12 -1  Output:  Linked List: 4 6 8 9 12  Middle Node: 8 |

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 find the middle node of the linked list

Node\* findMiddleNode(Node\* head) {

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

return head;

}

Node\* slow = head;

Node\* fast = head;

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

slow = slow->next;

fast = fast->next->next;

}

return slow;

}

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;

}

// Find the middle node of the linked list

Node\* middleNode = findMiddleNode(head);

if (middleNode != nullptr) {

cout << "\nMiddle Node: " << middleNode->data;

} else {

cout << "\nLinked List is empty.";

}

// Free memory by deleting nodes

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}

Q-3: Implement two sorted linked lists and merge them into a single sorted linked list.

Sample test case:

|  |
| --- |
| Input:  Enter elements for the first linked list (enter -1 to stop):  3 7 9 -1  Enter elements for the second linked list (enter -1 to stop):  1 10 8 -1  Output:  First Linked List: 3 7 9  Second Linked List: 1 10 8  Merged Linked List: 1 3 7 9 10 8 |
| 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 merge two sorted linked lists into a single sorted linked list

Node\* mergeSortedLinkedLists(Node\* list1, Node\* list2) {

if (list1 == nullptr)

return list2;

if (list2 == nullptr)

return list1;

Node\* mergedList = nullptr;

if (list1->data <= list2->data) {

mergedList = list1;

mergedList->next = mergeSortedLinkedLists(list1->next, list2);

} else {

mergedList = list2;

mergedList->next = mergeSortedLinkedLists(list1, list2->next);

}

return mergedList;

}

int main() {

Node\* head1 = nullptr;

Node\* head2 = nullptr;

int data;

// Create the first linked list with user input

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

while (true) {

cin >> data;

if (data == -1)

break;

head1 = insertAtEnd(head1, data);

}

// Create the second linked list with user input

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

while (true) {

cin >> data;

if (data == -1)

break;

head2 = insertAtEnd(head2, data);

}

// Display the first linked list

cout << "First Linked List: ";

displayLinkedList(head1);

// Display the second linked list

cout << "\nSecond Linked List: ";

displayLinkedList(head2);

// Merge the two sorted linked lists

Node\* mergedList = mergeSortedLinkedLists(head1, head2);

// Display the merged linked list

cout << "\nMerged Linked List: ";

displayLinkedList(mergedList);

// Free memory by deleting nodes

while (head1 != nullptr) {

Node\* temp = head1;

head1 = head1->next;

delete temp;

}

while (head2 != nullptr) {

Node\* temp = head2;

head2 = head2->next;

delete temp;

}

while (mergedList != nullptr) {

Node\* temp = mergedList;

mergedList = mergedList->next;

delete temp;

}

return 0;

}

Q-4: Implement a basic singly linked list with insert and display functions.

Sample test case:

|  |
| --- |
| Input:  list.insert(3);  list.insert(2);  list.insert(1);  list.display();  Output:  1 -> 2 -> 3 -> nullptr |

Solution:

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int value) : data(value), next(nullptr) {}

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() : head(nullptr) {}

void insert(int value) {

Node\* newNode = new Node(value);

newNode->next = head;

head = newNode; //insert at the head

}

void display() {

Node\* temp = head;

while (temp != nullptr) {

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

temp = temp->next;

}

cout << "nullptr" << endl;

}

~LinkedList() {

Node\* temp;

while (head != nullptr) {

temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

LinkedList list;

list.insert(3);

list.insert(2);

list.insert(1);

list.display();

return 0;

}

Q-5: Given a linked list of size N and a key. The task is to insert the key in the middle of the linked list.

Sample test case:

|  |
| --- |
| Input:  Enter elements for linked list (enter -1 to stop):  7 8 5 1 9 -1  Linked List before insertion: 7 8 5 1 9  Enter the key to be inserted in the middle: 99  Output: Linked List after insertion: 7 8 5 99 1 9 |

Solution:

#include <iostream>

using namespace std;

// Node structure for singly linked list

struct Node {

int data;

Node\* next;

};

// Function to insert a new node 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 middle node of the linked list

Node\* findMiddleNode(Node\* head) {

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

return head;

}

Node\* slow = head;

Node\* fast = head->next;

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

slow = slow->next;

fast = fast->next->next;

}

return slow;

}

// Function to insert the key in the middle of the linked list

Node\* insertInMiddle(Node\* head, int key) {

if (head == nullptr) {

return nullptr;

}

// Find the middle node

Node\* middle = findMiddleNode(head);

// Create a new node with the key

Node\* newNode = new Node;

newNode->data = key;

// Insert the new node after the middle node

newNode->next = middle->next;

middle->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, key;

// 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 inserting the key

cout << "Linked List before insertion: ";

displayLinkedList(head);

// Prompt the user to enter the key to be inserted

cout << "\nEnter the key to be inserted in the middle: ";

cin >> key;

// Insert the key in the middle of the linked list

head = insertInMiddle(head, key);

// Display the linked list after inserting the key

cout << "Linked List after insertion: ";

displayLinkedList(head);

// Free memory by deleting nodes

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}