**ASSGINMENT=5\\DSA**

Answer1:

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int value) {

data = value;

next = nullptr;

}

};

class SinglyLinkedList {

private:

Node\* head;

public:

SinglyLinkedList() {

head = nullptr;

}

// (a) Insert at beginning

void insertAtBeginning(int value) {

Node\* newNode = new Node(value);

newNode->next = head;

head = newNode;

}

// (b) Insert at end

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

head = newNode;

return;

}

Node\* temp = head;

while (temp->next != nullptr)

temp = temp->next;

temp->next = newNode;

}

// (c) Insert before or after a node

void insertBeforeAfter(int target, int value, bool insertBefore) {

Node\* newNode = new Node(value);

if (head == nullptr) {

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

return;

}

if (insertBefore) {

if (head->data == target) {

newNode->next = head;

head = newNode;

return;

}

Node\* prev = nullptr;

Node\* curr = head;

while (curr != nullptr && curr->data != target) {

prev = curr;

curr = curr->next;

}

if (curr == nullptr) {

cout << "Node with value " << target << " not found.\n";

return;

}

prev->next = newNode;

newNode->next = curr;

} else {

Node\* curr = head;

while (curr != nullptr && curr->data != target) {

curr = curr->next;

}

if (curr == nullptr) {

cout << "Node with value " << target << " not found.\n";

return;

}

newNode->next = curr->next;

curr->next = newNode;

}

}

// (d) Delete from beginning

void deleteFromBeginning() {

if (head == nullptr) {

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

return;

}

Node\* temp = head;

head = head->next;

delete temp;

}

// (e) Delete from end

void deleteFromEnd() {

if (head == nullptr) {

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

return;

}

if (head->next == nullptr) {

delete head;

head = nullptr;

return;

}

Node\* temp = head;

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

temp = temp->next;

delete temp->next;

temp->next = nullptr;

}

// (f) Delete specific node

void deleteNode(int value) {

if (head == nullptr) {

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

return;

}

if (head->data == value) {

Node\* temp = head;

head = head->next;

delete temp;

return;

}

Node\* prev = nullptr;

Node\* curr = head;

while (curr != nullptr && curr->data != value) {

prev = curr;

curr = curr->next;

}

if (curr == nullptr) {

cout << "Node with value " << value << " not found.\n";

return;

}

prev->next = curr->next;

delete curr;

}

// (g) Search for a node

void search(int value) {

Node\* temp = head;

int pos = 1;

while (temp != nullptr) {

if (temp->data == value) {

cout << "Node " << value << " found at position " << pos << ".\n";

return;

}

temp = temp->next;

pos++;

}

cout << "Node " << value << " not found.\n";

}

// (h) Display all nodes

void display() {

if (head == nullptr) {

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

return;

}

Node\* temp = head;

cout << "Linked List: ";

while (temp != nullptr) {

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

temp = temp->next;

}

cout << "NULL\n";

}

};

// Menu-driven interface

int main() {

SinglyLinkedList sll;

int choice, value, target;

string pos;

do {

cout << "\n=== Singly Linked List Menu ===\n";

cout << "1. Insert at Beginning\n";

cout << "2. Insert at End\n";

cout << "3. Insert Before/After a Node\n";

cout << "4. Delete from Beginning\n";

cout << "5. Delete from End\n";

cout << "6. Delete Specific Node\n";

cout << "7. Search for a Node\n";

cout << "8. Display List\n";

cout << "9. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to insert at beginning: ";

cin >> value;

sll.insertAtBeginning(value);

break;

case 2:

cout << "Enter value to insert at end: ";

cin >> value;

sll.insertAtEnd(value);

break;

case 3:

cout << "Enter target node value: ";

cin >> target;

cout << "Enter value to insert: ";

cin >> value;

cout << "Insert before or after the target? (before/after): ";

cin >> pos;

if (pos == "before")

sll.insertBeforeAfter(target, value, true);

else if (pos == "after")

sll.insertBeforeAfter(target, value, false);

else

cout << "Invalid option.\n";

break;

case 4:

sll.deleteFromBeginning();

break;

case 5:

sll.deleteFromEnd();

break;

case 6:

cout << "Enter node value to delete: ";

cin >> value;

sll.deleteNode(value);

break;

case 7:

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

cin >> value;

sll.search(value);

break;

case 8:

sll.display();

break;

case 9:

cout << "Exiting program.\n";

break;

default:

cout << "Invalid choice.\n";

}

} while (choice != 9);

return 0;

}

Answer2:

#include <iostream>

using namespace std;

// Node structure

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val;

next = nullptr;

}

};

class SinglyLinkedList {

private:

Node\* head;

public:

SinglyLinkedList() {

head = nullptr;

}

// Insert at end

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

return;

}

Node\* temp = head;

while (temp->next)

temp = temp->next;

temp->next = newNode;

}

// Display the linked list

void display() {

Node\* temp = head;

if (!temp) {

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

return;

}

cout << "Updated Linked List: ";

while (temp) {

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

temp = temp->next;

}

cout << "NULL\n";

}

// Count and delete all occurrences of key

int countAndDeleteOccurrences(int key) {

int count = 0;

// Handle deletion at head

while (head && head->data == key) {

Node\* toDelete = head;

head = head->next;

delete toDelete;

count++;

}

Node\* current = head;

while (current && current->next) {

if (current->next->data == key) {

Node\* toDelete = current->next;

current->next = current->next->next;

delete toDelete;

count++;

} else {

current = current->next;

}

}

return count;

}

};

int main() {

SinglyLinkedList list;

// Example list: 1 -> 2 -> 1 -> 2 -> 1 -> 3 -> 1

list.insertAtEnd(1);

list.insertAtEnd(2);

list.insertAtEnd(1);

list.insertAtEnd(2);

list.insertAtEnd(1);

list.insertAtEnd(3);

list.insertAtEnd(1);

int key = 1;

int count = list.countAndDeleteOccurrences(key);

cout << "Count: " << count << endl;

list.display();

return 0;

}

Answer3:

#include <iostream>

using namespace std;

// Node structure

class Node {

public:

int data;

Node\* next;

Node(int value) {

data = value;

next = nullptr;

}

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() {

head = nullptr;

}

// Insert at end

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

return;

}

Node\* temp = head;

while (temp->next)

temp = temp->next;

temp->next = newNode;

}

// Find and return middle element

void findMiddle() {

if (!head) {

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

return;

}

Node\* slow = head;

Node\* fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

}

cout << "Middle Element: " << slow->data << endl;

}

// Display list (optional for debugging)

void display() {

Node\* temp = head;

while (temp) {

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

temp = temp->next;

}

cout << "NULL\n";

}

};

int main() {

LinkedList list;

// Sample input: 1 -> 2 -> 3 -> 4 -> 5

list.insertAtEnd(1);

list.insertAtEnd(2);

list.insertAtEnd(3);

list.insertAtEnd(4);

list.insertAtEnd(5);

// Uncomment to see the list

// list.display();

list.findMiddle();

return 0;

}

Answer4:

#include <iostream>

using namespace std;

// Node definition

class Node {

public:

int data;

Node\* next;

Node(int value) {

data = value;

next = nullptr;

}

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() {

head = nullptr;

}

// Insert node at end

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

return;

}

Node\* temp = head;

while (temp->next)

temp = temp->next;

temp->next = newNode;

}

// Reverse the linked list

void reverse() {

Node\* prev = nullptr;

Node\* current = head;

Node\* next = nullptr;

while (current != nullptr) {

next = current->next; // Save next node

current->next = prev; // Reverse current's pointer

prev = current; // Move prev one step

current = next; // Move current one step

}

head = prev; // Update head to the new front

}

// Display the linked list

void display() {

Node\* temp = head;

while (temp != nullptr) {

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

temp = temp->next;

}

cout << "NULL\n";

}

};

int main() {

LinkedList list;

// Sample input: 1 -> 2 -> 3 -> 4 -> NULL

list.insertAtEnd(1);

list.insertAtEnd(2);

list.insertAtEnd(3);

list.insertAtEnd(4);

cout << "Original Linked List: ";

list.display();

list.reverse();

cout << "Reversed Linked List: ";

list.display();

return 0;

}