Name: Aryan Juneja roll no. :- 1024030920 Batch :- 2C64

Q1

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

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

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() : head(nullptr) {}

void insertAtBeginning(int data) {

Node\* newNode = new Node(data);

newNode->next = head;

head = newNode;

cout << "Node " << data << " inserted at the beginning." << endl;

}

void insertAtEnd(int data) {

Node\* newNode = new Node(data);

if (head == nullptr) {

head = newNode;

cout << "Node " << data << " inserted at the end." << endl;

return;

}

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

cout << "Node " << data << " inserted at the end." << endl;

}

void insertAfter(int target, int data) {

Node\* current = head;

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

current = current->next;

}

if (current == nullptr) {

cout << "Target node " << target << " not found." << endl;

} else {

Node\* newNode = new Node(data);

newNode->next = current->next;

current->next = newNode;

cout << "Node " << data << " inserted after " << target << "." << endl;

}

}

void insertBefore(int target, int data) {

if (head == nullptr) {

cout << "List is empty. Cannot insert before a node." << endl;

return;

}

if (head->data == target) {

insertAtBeginning(data);

return;

}

Node\* current = head;

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

current = current->next;

}

if (current->next == nullptr) {

cout << "Target node " << target << " not found." << endl;

} else {

Node\* newNode = new Node(data);

newNode->next = current->next;

current->next = newNode;

cout << "Node " << data << " inserted before " << target << "." << endl;

}

}

void deleteFromBeginning() {

if (head == nullptr) {

cout << "List is empty. Cannot delete." << endl;

return;

}

Node\* temp = head;

head = head->next;

cout << "Node " << temp->data << " deleted from the beginning." << endl;

delete temp;

}

void deleteFromEnd() {

if (head == nullptr) {

cout << "List is empty. Cannot delete." << endl;

return;

}

if (head->next == nullptr) {

cout << "Node " << head->data << " deleted from the end." << endl;

delete head;

head = nullptr;

return;

}

Node\* current = head;

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

current = current->next;

}

Node\* temp = current->next;

current->next = nullptr;

cout << "Node " << temp->data << " deleted from the end." << endl;

delete temp;

}

void deleteNode(int target) {

if (head == nullptr) {

cout << "List is empty. Cannot delete." << endl;

return;

}

if (head->data == target) {

Node\* temp = head;

head = head->next;

cout << "Node " << temp->data << " deleted." << endl;

delete temp;

return;

}

Node\* current = head;

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

current = current->next;

}

if (current->next == nullptr) {

cout << "Node " << target << " not found." << endl;

} else {

Node\* temp = current->next;

current->next = temp->next;

cout << "Node " << temp->data << " deleted." << endl;

delete temp;

}

}

void searchNode(int target) {

Node\* current = head;

int position = 1;

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

current = current->next;

position++;

}

if (current == nullptr) {

cout << "Node " << target << " not found in the list." << endl;

} else {

cout << "Node " << target << " found at position " << position << "." << endl;

}

}

void display() {

if (head == nullptr) {

cout << "List is empty." << endl;

return;

}

Node\* current = head;

cout << "Current List: ";

while (current != nullptr) {

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

current = current->next;

}

cout << "nullptr" << endl;

}

~LinkedList() {

Node\* current = head;

while (current != nullptr) {

Node\* next = current->next;

delete current;

current = next;

}

head = nullptr;

}

};

int main() {

LinkedList list;

int choice, data, target;

do {

cout << "\n--- Singly Linked List Menu ---" << endl;

cout << "1. Insert at the beginning" << endl;

cout << "2. Insert at the end" << endl;

cout << "3. Insert after a specific node" << endl;

cout << "4. Insert before a specific node" << endl;

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

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

cout << "7. Delete a specific node" << endl;

cout << "8. Search for a node" << endl;

cout << "9. Display all nodes" << endl;

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

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter data to insert: ";

cin >> data;

list.insertAtBeginning(data);

break;

case 2:

cout << "Enter data to insert: ";

cin >> data;

list.insertAtEnd(data);

break;

case 3:

cout << "Enter target node data: ";

cin >> target;

cout << "Enter new node data: ";

cin >> data;

list.insertAfter(target, data);

break;

case 4:

cout << "Enter target node data: ";

cin >> target;

cout << "Enter new node data: ";

cin >> data;

list.insertBefore(target, data);

break;

case 5:

list.deleteFromBeginning();

break;

case 6:

list.deleteFromEnd();

break;

case 7:

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

cin >> target;

list.deleteNode(target);

break;

case 8:

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

cin >> target;

list.searchNode(target);

break;

case 9:

list.display();

break;

case 0:

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

break;

default:

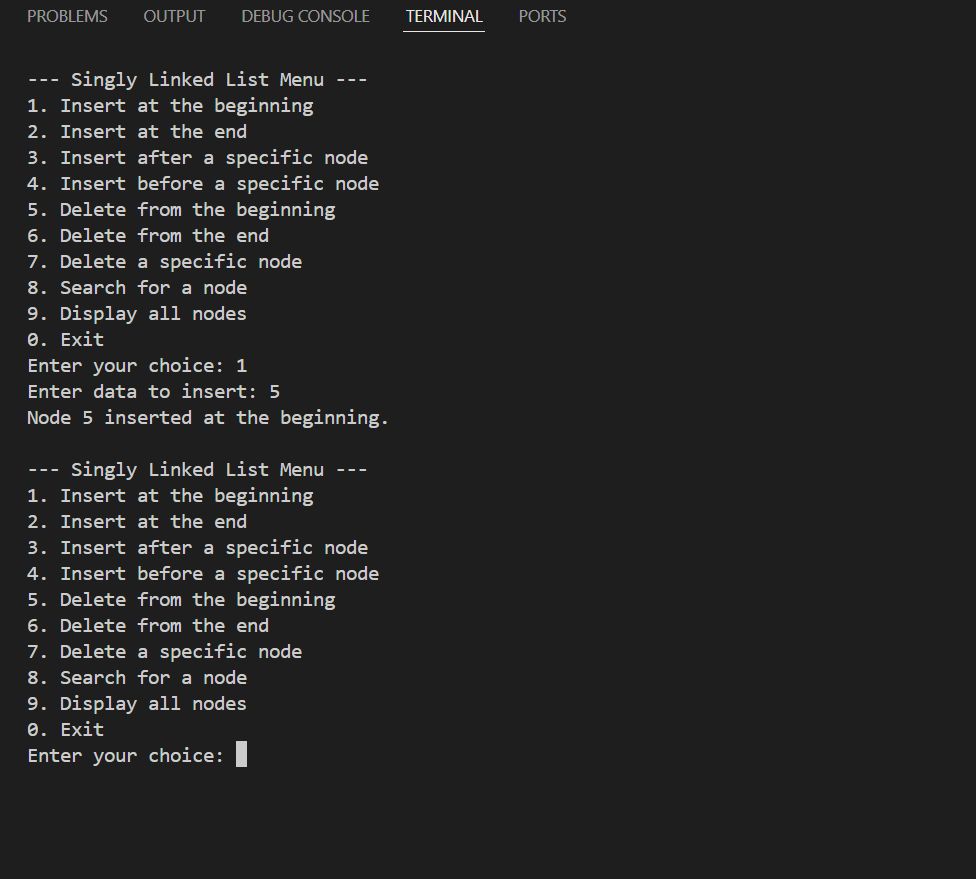
cout << "Invalid choice. Please try again." << endl;

}

} while (choice != 0);

return 0;

}



Q2

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

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

};

void printList(Node\* head) {

Node\* temp = head;

while (temp != nullptr) {

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

temp = temp->next;

}

cout << "nullptr" << endl;

}

void insertEnd(Node\*& head, int data) {

Node\* newNode = new Node(data);

if (head == nullptr) {

head = newNode;

return;

}

Node\* last = head;

while (last->next != nullptr) {

last = last->next;

}

last->next = newNode;

}

void countAndDelete(Node\*& head, int key) {

int count = 0;

Node\* current = head;

Node\* prev = nullptr;

while (current != nullptr) {

if (current->data == key) {

count++;

if (prev == nullptr) {

Node\* temp = current;

current = current->next;

head = current;

delete temp;

} else {

// If the key is in the middle or at the end

Node\* temp = current;

prev->next = current->next;

current = current->next;

delete temp;

}

} else {

prev = current;

current = current->next;

}

}

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

}

int main() {

Node\* head = nullptr;

// Build the linked list: 1->2->1->2->1->3->1

insertEnd(head, 1);

insertEnd(head, 2);

insertEnd(head, 1);

insertEnd(head, 2);

insertEnd(head, 1);

insertEnd(head, 3);

insertEnd(head, 1);

int key = 1;

cout << "Original Linked List: ";

printList(head);

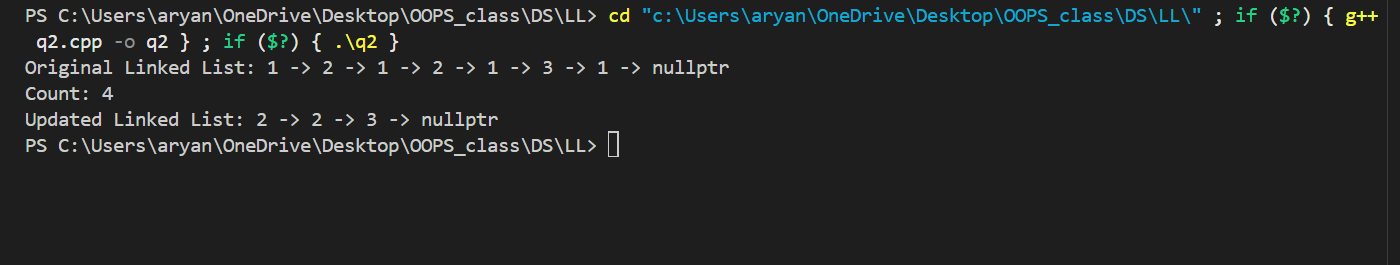
countAndDelete(head, key);

cout << "Updated Linked List: ";

printList(head);

return 0;

}



Q3

#include <iostream>

using namespace std;

struct ListNode {

int val;

ListNode \*next;

ListNode(int x) : val(x), next(NULL) {}

};

int findMiddle(ListNode\* head) {

if (head == NULL) {

return -1;

}

ListNode\* slow = head;

ListNode\* fast = head;

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

slow = slow->next;

fast = fast->next->next;

}

return slow->val;

}

int main() {

ListNode\* head = new ListNode(1);

head->next = new ListNode(2);

head->next->next = new ListNode(3);

head->next->next->next = new ListNode(4);

head->next->next->next->next = new ListNode(5);

cout << "Input: 1->2->3->4->5" << endl;

cout << "Output: " << findMiddle(head) << endl;

ListNode\* current = head;

while (current != NULL) {

ListNode\* temp = current;

current = current->next;

delete temp;

}

ListNode\* head\_even = new ListNode(1);

head\_even->next = new ListNode(2);

head\_even->next->next = new ListNode(3);

head\_even->next->next->next = new ListNode(4);

head\_even->next->next->next->next = new ListNode(5);

head\_even->next->next->next->next->next = new ListNode(6);

cout << "\nInput: 1->2->3->4->5->6" << endl;

cout << "Output: " << findMiddle(head\_even) << endl;

current = head\_even;

while (current != NULL) {

ListNode\* temp = current;

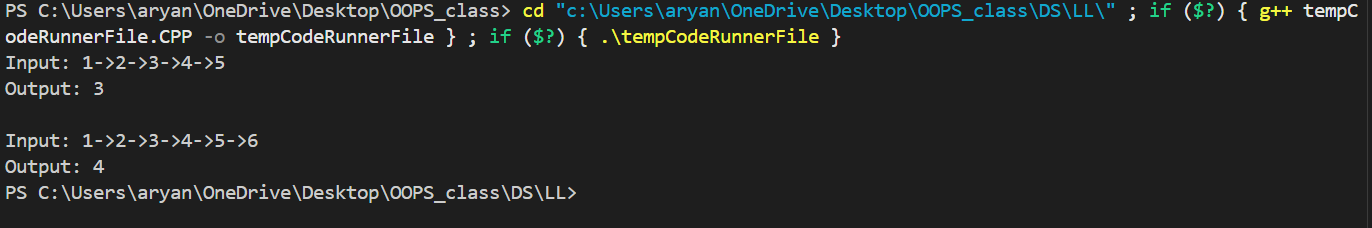
current = current->next;

delete temp;

}

return 0;

}



Q4

#include <iostream>

using namespace std;

struct ListNode {

int val;

ListNode \*next;

ListNode(int x) : val(x), next(NULL) {}

};

ListNode\* reverseList(ListNode\* head) {

ListNode\* prev = NULL;

ListNode\* current = head;

ListNode\* next\_node = NULL;

while (current != NULL) {

next\_node = current->next;

current->next = prev;

prev = current;

current = next\_node;

}

return prev;

}

void printList(ListNode\* node) {

while (node != NULL) {

cout << node->val << "->";

node = node->next;

}

cout << "NULL" << endl;

}

int main() {

ListNode\* head = new ListNode(1);

head->next = new ListNode(2);

head->next->next = new ListNode(3);

head->next->next->next = new ListNode(4);

cout << "Input: ";

printList(head);

ListNode\* reversed\_head = reverseList(head);

cout << "Output: ";

printList(reversed\_head);

ListNode\* current = reversed\_head;

while (current != NULL) {

ListNode\* temp = current;

current = current->next;

delete temp;

}

return 0;

}

