**Assignment no: 2**

**Subject:**

Data Structures and Algorithms (LAB)

**Submitted By:**

Aqsa Hussain

SP22-BSCS-083

(Section B)

**Submitted To:**

Mam Yasmeen Jana

**Submission Date:**

09-10-2023

**Program No: 1**

#include <iostream>

struct Node {

int data;

Node\* next;

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

};

void displayLinkedList(Node\* head) {

if (head == NULL) {

std::cout << "The linked list is empty." << std::endl;

return;

}

std::cout << "The linked list is:" << std::endl;

Node\* ptr = head;

while (ptr) {

std::cout << ptr->data << " ";

ptr = ptr->next;

}

std::cout << std::endl;

std::cout << "\*head address:\*\*" << &head << std::endl;

std::cout << "---------------------------------------" << std::endl;

ptr = head;

while (ptr) {

std::cout << "\nhead content: " << ptr << std::endl;

std::cout << "-------------------------------" << std::endl;

std::cout << "\*ptr address: \*\*" << ptr << std::endl;

std::cout << "----------------------------------" << std::endl;

std::cout << "ptr content: " << ptr << std::endl;

std::cout << "----------------------------------" << std::endl;

std::cout << "ptr->data: " << ptr->data << std::endl;

std::cout << "----------------------------------" << std::endl;

std::cout << "ptr: " << ptr << std::endl;

if (ptr->next) {

std::cout << "ptr->next: " << ptr->next << "." << std::endl;

std::cout << "ptr->data: " << ptr->next->data << std::endl;

std::cout << "---------------------------------" << std::endl;

} else {

std::cout << "ptr->next: 0" << std::endl;

std::cout << "---------------------------------" << std::endl;

}

ptr = ptr->next;

}

}

int main() {

Node\* head = new Node(1);

head->next = new Node(2);

head->next->next = new Node(20);

head->next->next->next = new Node(30);

displayLinkedList(head);

// Remember to free the memory allocated for the linked list nodes

while (head) {

Node\* temp = head;

head = head->next;

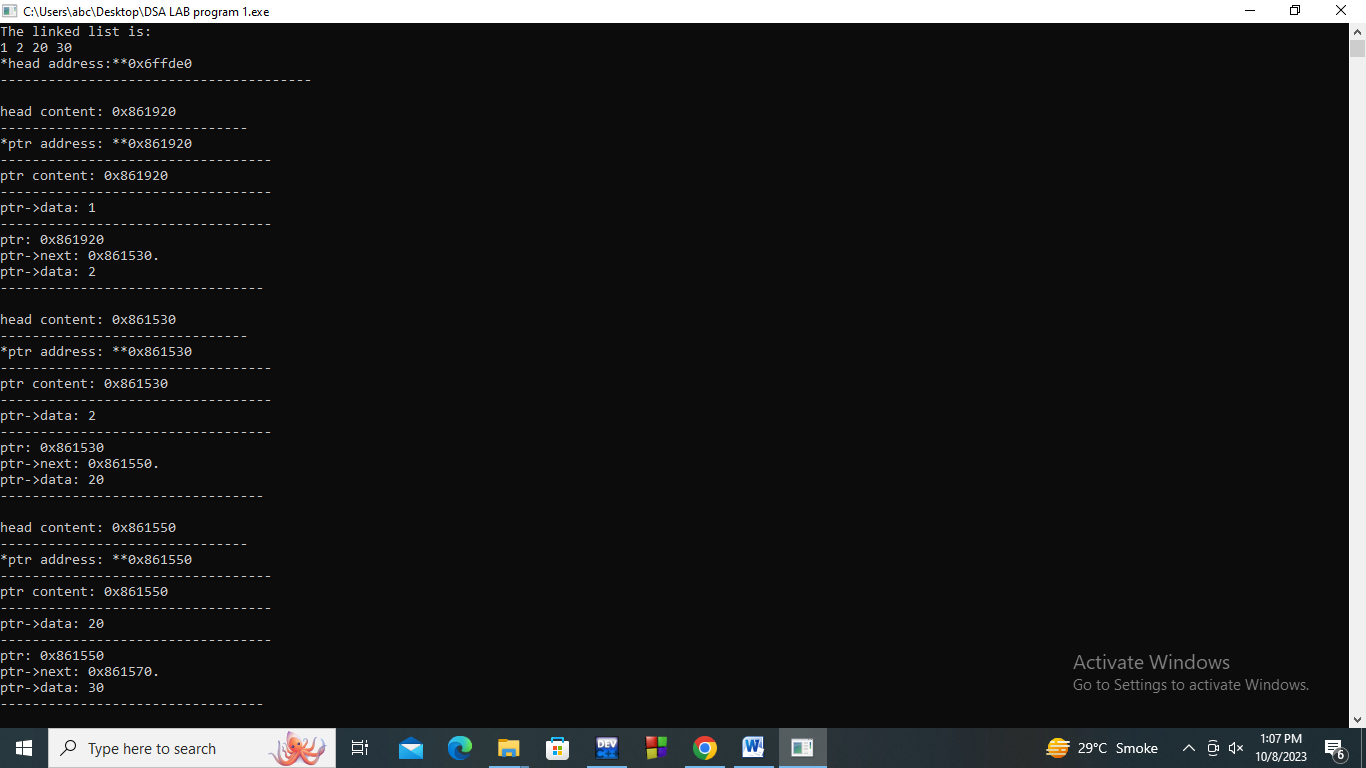
delete temp;

}

return 0;

}

**Output:**



**Program No: 2**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val;

next = NULL;

}

};

class DoublyNode {

public:

int data;

DoublyNode\* prev;

DoublyNode\* next;

DoublyNode(int val) {

data = val;

prev = NULL;

next = NULL;

}

};

class CircularNode {

public:

int data;

CircularNode\* next;

CircularNode(int val) {

data = val;

next = NULL;

}

};

class SingleLinkedList {

public:

Node\* head;

SingleLinkedList() {

head = NULL;

}

void insertAtBeginning(int val) {

Node\* newNode = new Node(val);

newNode->next = head;

head = newNode;

}

void insertAtEnd(int val) {

Node\* newNode = new Node(val);

if (head == NULL) {

head = newNode;

return;

}

Node\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

void insertAfterData(int val, int target) {

Node\* newNode = new Node(val);

Node\* current = head;

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

current = current->next;

}

if (current == NULL) {

cout << "Data not found in the list." << endl;

return;

}

newNode->next = current->next;

current->next = newNode;

}

void deleteNode(int val) {

Node\* current = head;

Node\* prev = NULL;

while (current != NULL && current->data != val) {

prev = current;

current = current->next;

}

if (current == NULL) {

cout << "Data not found in the list." << endl;

return;

}

if (prev == NULL) {

head = current->next;

} else {

prev->next = current->next;

}

delete current;

}

void reverse() {

Node\* prev = NULL;

Node\* current = head;

Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

}

void display(){

Node\* current = head;

while (current != NULL) {

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

current = current->next;

}

cout << endl;

}

bool seek(int val) {

Node\* current = head;

while (current != NULL) {

if (current->data == val) {

return true;

}

current = current->next;

}

return false;

}

};

class DoublyLinkedList {

public:

DoublyNode\* head;

DoublyLinkedList() {

head = NULL;

}

void insertAtBeginning(int val) {

DoublyNode\* newNode = new DoublyNode(val);

newNode->next = head;

newNode->prev = NULL;

if (head != NULL) {

head->prev = newNode;

}

head = newNode;

}

void insertAtEnd(int val) {

DoublyNode\* newNode = new DoublyNode(val);

if (head == NULL) {

head = newNode;

return;

}

DoublyNode\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

}

void insertAfterData(int val, int target) {

DoublyNode\* newNode = new DoublyNode(val);

DoublyNode\* current = head;

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

current = current->next;

}

if (current == NULL) {

cout << "data not found in the list." << endl;

return;

}

newNode->next = current->next;

newNode->prev = current;

if (current->next != NULL) {

current->next->prev = newNode;

}

current->next = newNode;

}

void deleteNode(int val) {

DoublyNode\* current = head;

while (current != NULL && current->data != val) {

current = current->next;

}

if (current == NULL) {

cout << "Data not found in the list." << endl;

return;

}

if (current->prev != NULL) {

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

} else {

head = current->next;

}

if (current->next != NULL) {

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

}

delete current;

}

void reverse() {

DoublyNode\* current = head;

while (current != NULL) {

swap(current->next, current->prev);

head = current;

current = current->prev;

}

}

void display(){

DoublyNode\* current = head;

while (current != NULL) {

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

current = current->next;

}

cout << endl;

}

bool seek(int val) {

DoublyNode\* current = head;

while (current != NULL) {

if (current->data == val) {

return true;

}

current = current->next;

}

return false;

}

};

class CircularLinkedList {

public:

CircularNode\* head;

CircularLinkedList() {

head = NULL;

}

void insertAtBeginning(int val) {

CircularNode\* newNode = new CircularNode(val);

if (head == NULL) {

newNode->next = newNode;

} else {

CircularNode\* current = head;

while (current->next != head) {

current = current->next;

}

current->next = newNode;

newNode->next = head;

}

head = newNode;

}

void insertAtEnd(int val) {

CircularNode\* newNode = new CircularNode(val);

if (head == NULL) {

newNode->next = newNode;

head = newNode;

} else {

CircularNode\* current = head;

while (current->next != head) {

current = current->next;

}

current->next = newNode;

newNode->next = head;

}

}

void insertAfterData(int val, int target) {

CircularNode\* newNode = new CircularNode(val);

if (head == NULL) {

cout << "List is empty. Cannot insert after data." << endl;

return;

}

CircularNode\* current = head;

while (current->data != target) {

current = current->next;

if (current == head) {

cout << "Target data not found in the list." << endl;

return;

}

}

newNode->next = current->next;

current->next = newNode;

}

void deleteNode(int val) {

if (head == NULL) {

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

return;

}

CircularNode\* current = head;

CircularNode\* prev = NULL;

do {

if (current->data == val) {

if (prev == NULL) {

CircularNode\* temp = head;

while (temp->next != head) {

temp = temp->next;

}

head = head->next;

temp->next = head;

delete current;

return;

} else {

prev->next = current->next;

delete current;

return;

}

}

prev = current;

current = current->next;

} while (current != head);

cout << "Data not found in the list." << endl;

}

void display(){

if (head == NULL) {

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

return;

}

CircularNode\* current = head;

do {

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

current = current->next;

} while (current != head);

}

bool seek(int val) {

if (head == NULL) {

return false;

}

CircularNode\* current = head;

do {

if (current->data == val) {

return true;

}

current = current->next;

} while (current != head);

return false;

}

};

int main() {

int choice;

int listType;

SingleLinkedList sll;

DoublyLinkedList dll;

CircularLinkedList cll;

do {

cout << "Which linked list you want:" << endl;

cout << "1: Single" << endl;

cout << "2: Double" << endl;

cout << "3: Circular" << endl;

cout << "Enter your choice : ";

cin >> listType;

if (listType < 1 || listType > 3) {

cout << "Invalid choice. Please enter a valid option (1/2/3)." << endl;

continue;

}

cout << "Which operation you want to perform:" << endl;

cout << "1: Insertion" << endl;

cout << "2: Deletion" << endl;

cout << "3: Display" << endl;

cout << "4: Reverse" << endl;

cout << "5: Seek" << endl;

cout << "6: Exit" << endl;

cout << "Enter your choice : ";

cin >> choice;

switch (choice) {

case 1: {

int insertChoice;

cout << "1: Insertion at beginning" << endl;

cout << "2: Insertion at end" << endl;

cout << "3: Insertion at a specific data node" << endl;

cout << "Enter your choice (1/2/3): ";

cin >> insertChoice;

int data;

int targetData;

switch (insertChoice) {

case 1:

cout << "Enter data to insert: ";

cin >> data;

if (listType == 1) {

sll.insertAtBeginning(data);

} else if (listType == 2) {

dll.insertAtBeginning(data);

} else {

cll.insertAtBeginning(data);

}

break;

case 2:

cout << "Enter data to insert: ";

cin >> data;

if (listType == 1) {

sll.insertAtEnd(data);

} else if (listType == 2) {

dll.insertAtEnd(data);

} else {

cll.insertAtEnd(data);

}

break;

case 3:

cout << "Enter data to insert: ";

cin >> data;

cout << "Enter target data: ";

cin >> targetData;

if (listType == 1) {

sll.insertAfterData(data, targetData);

} else if (listType == 2) {

dll.insertAfterData(data, targetData);

} else {

cll.insertAfterData(data, targetData);

}

break;

default:

cout << "Invalid choice. Please enter a valid option (1/2/3)." << endl;

break;

}

break;

}

case 2: {

int deleteData;

cout << "Enter data to delete: ";

cin >> deleteData;

if (listType == 1) {

sll.deleteNode(deleteData);

} else if (listType == 2) {

dll.deleteNode(deleteData);

} else {

cll.deleteNode(deleteData);

}

break;

}

case 3:

cout << "Linked List:" << endl;

if (listType == 1) {

sll.display();

} else if (listType == 2) {

dll.display();

} else {

cll.display();

}

break;

case 4:

if (listType == 1) {

sll.reverse();

} else if (listType == 2) {

dll.reverse();

} else {

cout << "Reverse operation not supported for Circular Linked List." << endl;

}

break;

case 5: {

int seekData;

cout << "Enter data to seek: ";

cin >> seekData;

bool found = false;

if (listType == 1) {

found = sll.seek(seekData);

} else if (listType == 2) {

found = dll.seek(seekData);

} else {

found = cll.seek(seekData);

}

if (found) {

cout << "Data found in the list." << endl;

} else {

cout << "Data not found in the list." << endl;

}

break;

}

case 6:

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

break;

default:

cout << "Invalid choice. Please enter a valid option (1/2/3/4/5/6)." << endl;

break;

}

} while (choice != 6);

return 0;

}

**Output:**

