|  |  |
| --- | --- |
| download | COMSATS University Islamabad, Vehari Campus Department of Computer Science |

**Class: BCS-SP22 Submission Deadline: 9 Oct 2023**

**Subject: Data Structures and Algorithms-Lab Instructor: Yasmeen Jana Max Marks: 20**

**NAME: MUBASHIR NADEEM**

**Reg. No: BCS-SP22-087**

**Email:** [**yasmeenjana@cuivehari.edu.pk**](mailto:yasmeenjana@cuivehari.edu.pk)

**You can ask queries related to Lab Activities on the above email.**

**Activity 1:**

**PROGRAM CODE**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val;

next = nullptr;

}

};

class SinglyLinkedList {

public:

Node\* head;

SinglyLinkedList() {

head = nullptr;

}

// Function to insert a new node at the end of the linked list

void Insert\_Node(int val) {

Node\* newNode = new Node(val);

if (head == nullptr) {

head = newNode;

} else {

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

}

// Function to display the data, address, and next address of each node in the linked list

void display() {

Node\* current = head;

while (current != nullptr) {

cout << "Data in the Node: " << current->data << endl;

cout << "Node Address: " << current << endl;

cout << "Next Node Address: " << current->next << endl;

cout << endl; // Add a blank line for separation

current = current->next;

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

}

}

};

int main() {

SinglyLinkedList myList;

myList.Insert\_Node(23);

myList.Insert\_Node(87);

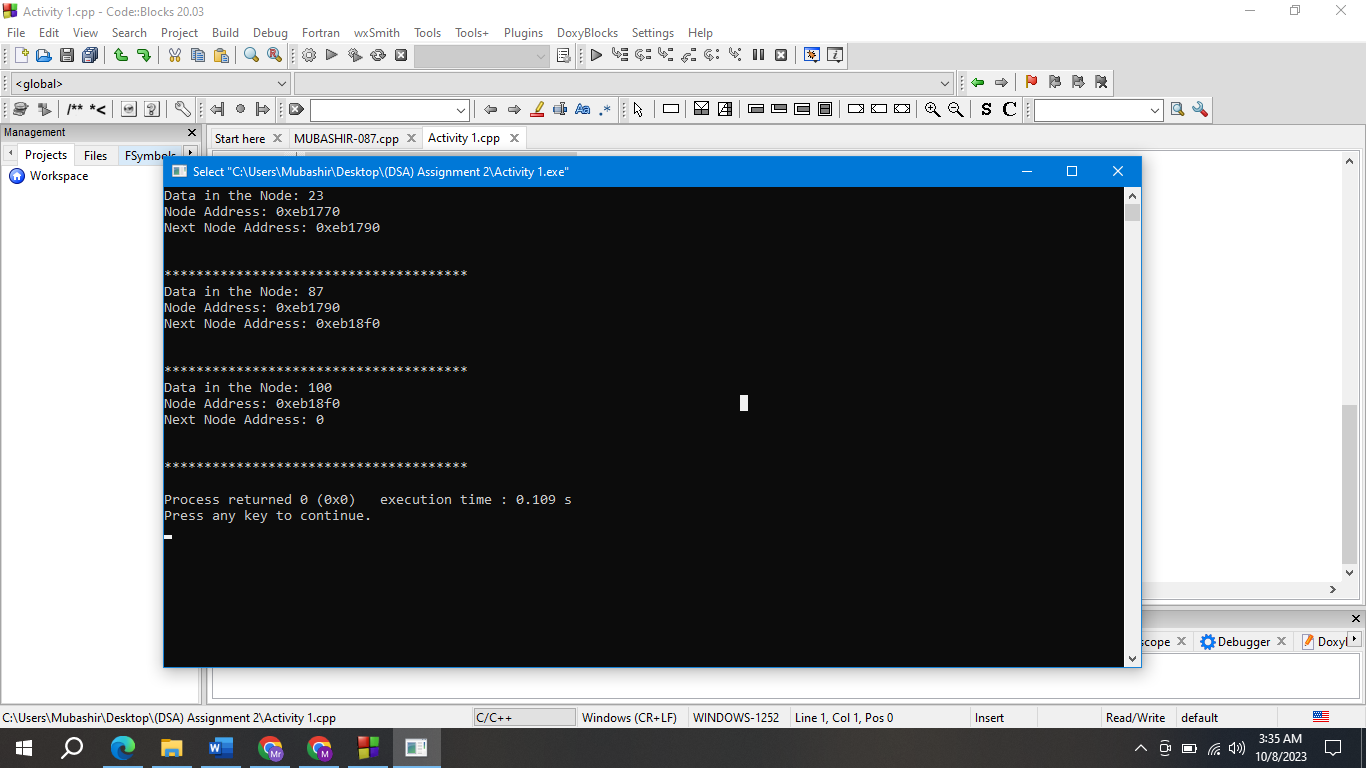
myList.Insert\_Node(100);

myList.display();

return 0;

}

**PROGRAM OUTPUT**

****

**Activity 2:**

**PROGRAM CODE**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val;

next = nullptr;

}

};

class DoublyNode {

public:

int data;

DoublyNode\* prev;

DoublyNode\* next;

DoublyNode(int val) {

data = val;

prev = nullptr;

next = nullptr;

}

};

class CircularNode {

public:

int data;

CircularNode\* next;

CircularNode(int val) {

data = val;

next = nullptr;

}

};

class SingleLinkedList {

public:

Node\* head;

SingleLinkedList() {

head = nullptr;

}

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 == nullptr) {

head = newNode;

return;

}

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

void insertAfterData(int val, int target) {

Node\* newNode = new Node(val);

Node\* current = head;

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

current = current->next;

}

if (current == nullptr) {

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

return;

}

newNode->next = current->next;

current->next = newNode;

}

void deleteNode(int val) {

Node\* current = head;

Node\* prev = nullptr;

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

prev = current;

current = current->next;

}

if (current == nullptr) {

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

return;

}

if (prev == nullptr) {

head = current->next;

} else {

prev->next = current->next;

}

delete current;

}

void reverse() {

Node\* prev = nullptr;

Node\* current = head;

Node\* next = nullptr;

while (current != nullptr) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

}

void display() {

Node\* current = head;

while (current != nullptr) {

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

cout << "Address: " << current << endl;

cout << "Next Address: " << current->next << endl << endl;

current = current->next;

}

}

bool seek(int val) {

Node\* current = head;

while (current != nullptr) {

if (current->data == val) {

return true;

}

current = current->next;

}

return false;

}

};

class DoublyLinkedList {

public:

DoublyNode\* head;

DoublyLinkedList() {

head = nullptr;

}

void insertAtBeginning(int val) {

DoublyNode\* newNode = new DoublyNode(val);

newNode->next = head;

newNode->prev = nullptr;

if (head != nullptr) {

head->prev = newNode;

}

head = newNode;

}

void insertAtEnd(int val) {

DoublyNode\* newNode = new DoublyNode(val);

if (head == nullptr) {

head = newNode;

return;

}

DoublyNode\* current = head;

while (current->next != nullptr) {

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

current = current->next;

}

if (current == nullptr) {

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

return;

}

newNode->next = current->next;

newNode->prev = current;

if (current->next != nullptr) {

current->next->prev = newNode;

}

current->next = newNode;

}

void deleteNode(int val) {

DoublyNode\* current = head;

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

current = current->next;

}

if (current == nullptr) {

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

return;

}

if (current->prev != nullptr) {

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

} else {

head = current->next;

}

if (current->next != nullptr) {

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

}

delete current;

}

void reverse() {

DoublyNode\* current = head;

while (current != nullptr) {

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

head = current;

current = current->prev;

}

}

void display() {

DoublyNode\* current = head;

while (current != nullptr) {

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

cout << "Address: " << current << endl;

cout << "Next Address: " << current->next << endl;

cout << "Prev Address: " << current->prev << endl << endl;

current = current->next;

}

}

bool seek(int val) {

DoublyNode\* current = head;

while (current != nullptr) {

if (current->data == val) {

return true;

}

current = current->next;

}

return false;

}

};

class CircularLinkedList {

public:

CircularNode\* head;

CircularLinkedList() {

head = nullptr;

}

void insertAtBeginning(int val) {

CircularNode\* newNode = new CircularNode(val);

if (head == nullptr) {

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 == nullptr) {

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 == nullptr) {

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 == nullptr) {

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

return;

}

CircularNode\* current = head;

CircularNode\* prev = nullptr;

do {

if (current->data == val) {

if (prev == nullptr) {

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 == nullptr) {

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

return;

}

CircularNode\* current = head;

do {

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

cout << "Address: " << current << endl;

cout << "Next Address: " << current->next << endl << endl;

current = current->next;

} while (current != head);

}

bool seek(int val) {

if (head == nullptr) {

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 (1/2/3): ";

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 (1/2/3/4/5/6): ";

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 Contents:" << 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;

**PROGRAM OUTPUT**

