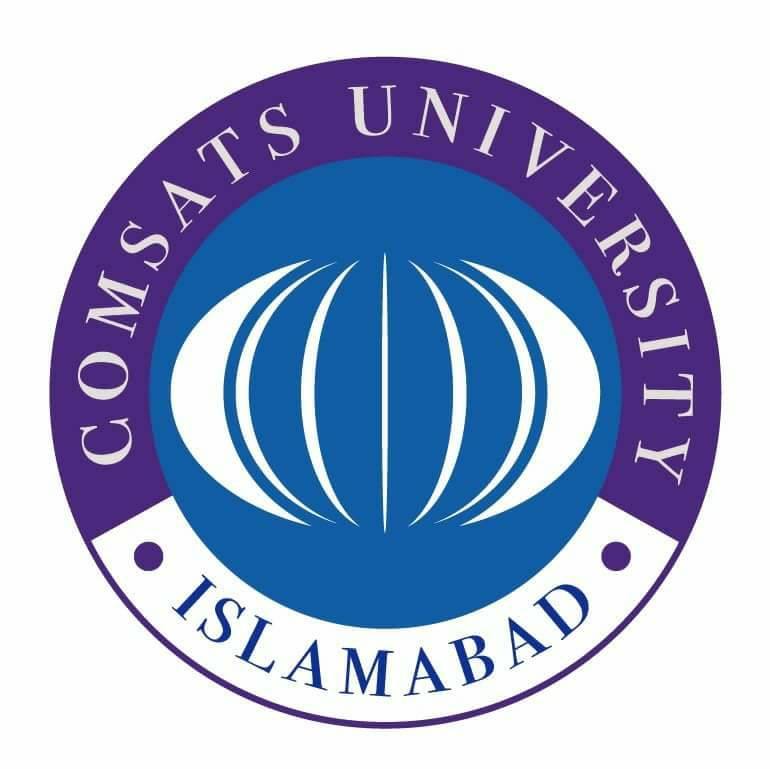
***Comsats University Islamabad (Vehari campus)***



***DSA Lab (Assignment 2)***

***Submitted by:***

*Saman Atif*

***Roll no:***

*SP22-BCS-070(Section B)*

***Department*:**

*Computer Science*

***Subject*:**

*Data structure and algorithm*

***Submitted to:***

*Mam Yasmeen Jana*

**ACTIVITY 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) {

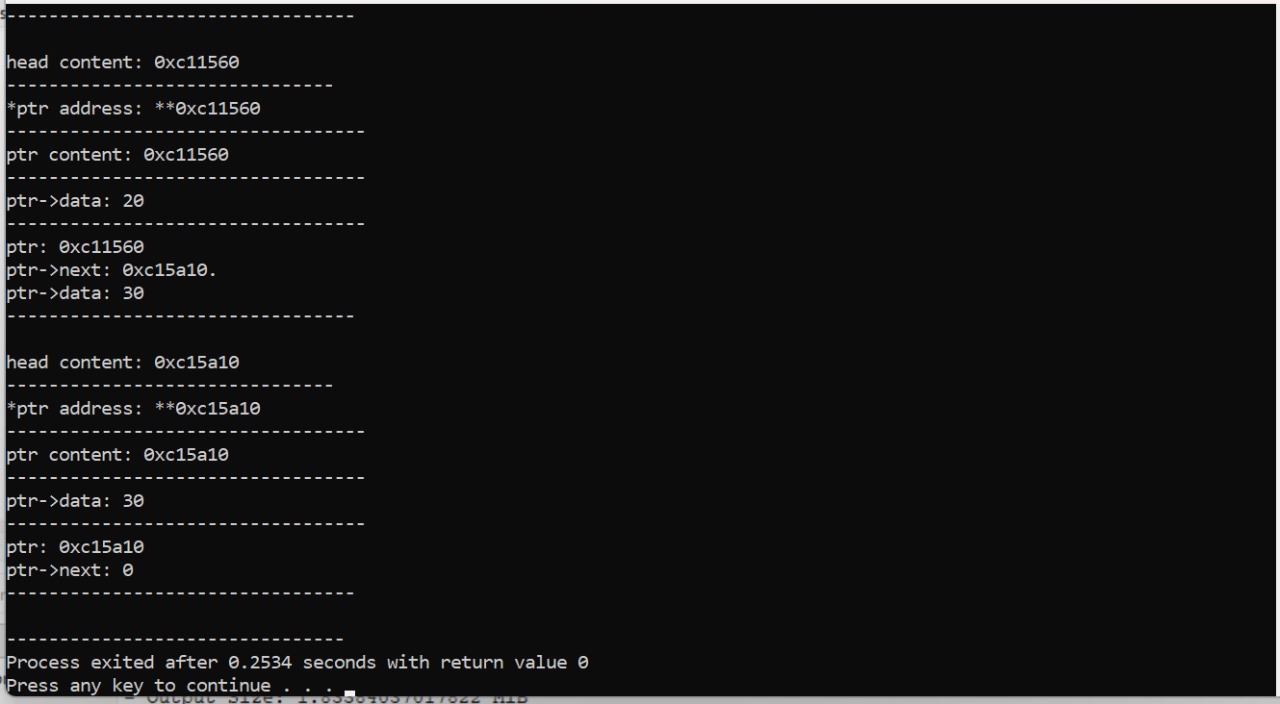
Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}

**ACTIVITY 2:**

#include <iostream>

using namespace std;

// Define a Node structure

struct Node {

int data;

Node\* next;

Node\* prev; // Used for doubly linked list

};

// Define a LinkedList class

class LinkedList {

private:

Node\* head;

Node\* tail; // Used for doubly linked list

bool isDoubly;

public:

LinkedList(bool doubly) {

head = NULL;

tail = NULL;

isDoubly = doubly;

}

// Function to insert a node at the beginning of the list

void insertAtBeginning(int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->next = head;

if (isDoubly) {

newNode->prev = NULL;

if (head != NULL) {

head->prev = newNode;

}

}

head = newNode;

if (!isDoubly && tail == NULL) {

tail = head;

}

}

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

void insertAtEnd(int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->next = NULL;

if (isDoubly) {

newNode->prev = tail;

if (tail != NULL) {

tail->next = newNode;

}

tail = newNode;

} else {

if (tail == NULL) {

head = newNode;

tail = newNode;

} else {

tail->next = newNode;

tail = newNode;

}

}

}

// Function to insert a node after a specific data node

void insertAfter(int target, int value) {

Node\* current = head;

while (current != NULL) {

if (current->data == target) {

Node\* newNode = new Node;

newNode->data = value;

newNode->next = current->next;

if (isDoubly) {

newNode->prev = current;

if (current->next != NULL) {

current->next->prev = newNode;

}

}

current->next = newNode;

break;

}

current = current->next;

}

}

// Function to delete a node with a specific data value

void deleteNode(int value) {

Node\* current = head;

Node\* prev = NULL;

while (current != NULL) {

if (current->data == value) {

if (prev != NULL) {

prev->next = current->next;

} else {

head = current->next;

}

if (isDoubly && current->next != NULL) {

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

}

delete current;

break;

}

prev = current;

current = current->next;

}

}

// Function to display the linked list

void display() {

Node\* current = head;

while (current != NULL) {

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

current = current->next;

}

cout << endl;

}

// Function to reverse the linked list

void reverse() {

if (isDoubly) {

Node\* temp = NULL;

Node\* current = head;

while (current != NULL) {

temp = current->prev;

current->prev = current->next;

current->next = temp;

current = current->prev;

}

if (temp != NULL) {

head = temp->prev;

}

} else {

Node\* prev = NULL;

Node\* current = head;

Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

}

}

// Function to seek a specific data node

bool seek(int value) {

Node\* current = head;

while (current != NULL) {

if (current->data == value) {

return true;

}

current = current->next;

}

return false;

}

};

// Function to display the linked list after an operation

void displayLinkedList(LinkedList& list) {

cout << "Linked List: ";

list.display();

}

int main() {

int choice;

bool isDoubly;

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

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

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

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

cin >> choice;

switch (choice) {

case 1:

isDoubly = false;

break;

case 2:

isDoubly = true;

break;

case 3:

// Handle circular linked list

cout << "Circular linked list is not implemented in this example." << endl;

return 0;

default:

cout << "Invalid choice. Exiting..." << endl;

return 0;

}

LinkedList list(isDoubly);

while (true) {

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;

cin >> choice;

switch (choice) {

case 1:

int insertChoice;

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

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

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

cin >> insertChoice;

switch (insertChoice) {

case 1:

int value;

cout << "Enter value to insert: ";

cin >> value;

list.insertAtBeginning(value);

displayLinkedList(list);

break;

case 2:

cout << "Enter value to insert: ";

cin >> value;

list.insertAtEnd(value);

displayLinkedList(list);

break;

case 3:

int targetValue;

cout << "Enter the data value after which to insert: ";

cin >> targetValue;

cout << "Enter value to insert: ";

cin >> value;

list.insertAfter(targetValue, value);

displayLinkedList(list);

break;

default:

cout << "Invalid choice." << endl;

}

break;

case 2:

int deleteValue;

cout << "Enter value to delete: ";

cin >> deleteValue;

list.deleteNode(deleteValue);

displayLinkedList(list);

break;

case 3:

displayLinkedList(list);

break;

case 4:

list.reverse();

cout << "Linked List reversed." << endl;

displayLinkedList(list);

break;

case 5:

int seekValue;

cout << "Enter value to seek: ";

cin >> seekValue;

if (list.seek(seekValue)) {

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

} else {

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

}

break;

case 6:

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

return 0;

default:

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

}

}

return 0;

}

