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

**Submission Deadline: 9 Oct 2023 Subject: Data Structures and Algorithms-Lab Instructor: Yasmeen Jana Reg. No: SP22-BCS-112**

**Activity 1:**

**Code:**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

void displayLinkedList(Node\* head) {

cout << "The linked list is: ";

Node\* ptr = head;

while (ptr != NULL) {

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

ptr = ptr->next;

}

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

cout << "--------------------------" << endl;

cout << "head content: " << head<< endl;

cout << "--------------------------" << endl;

cout << "\*\*\*ptr address:\*\*\* @" << &head << endl;

cout << "--------------------------" << endl;

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

cout << "----------------------" << endl;

ptr = head;

while (ptr != NULL) {

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

cout << "----------------------" << endl;

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

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

ptr = ptr->next;

}

}

int main() {

Node\* head = new Node();

Node\* second = new Node();

Node\* third = new Node();

Node\* fourth = new Node();

head->data = 1;

head->next = second;

second->data = 2;

second->next = third;

third->data = 20;

third->next = fourth;

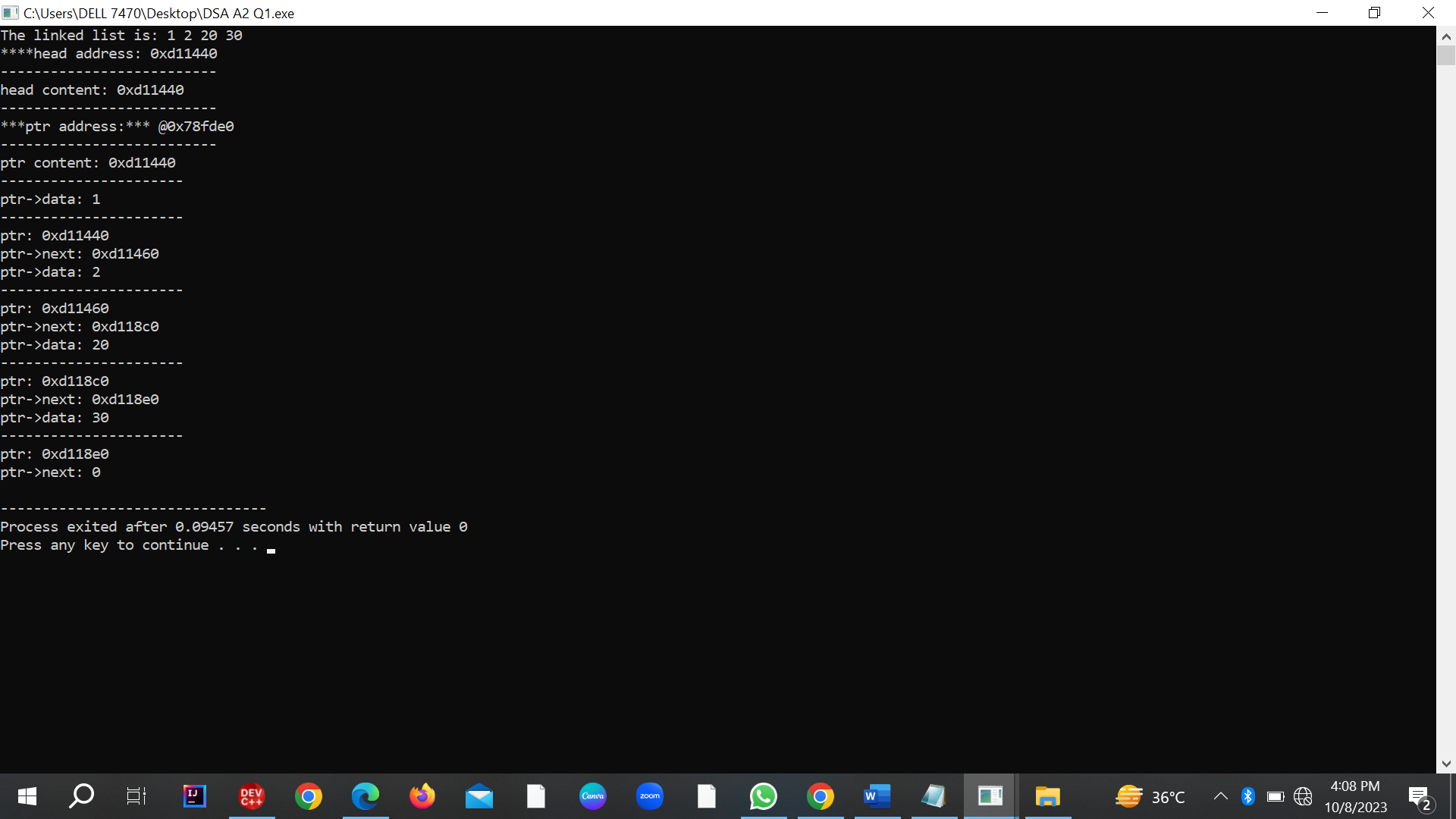
fourth->data = 30;

fourth->next = NULL;

displayLinkedList(head);

return 0;

}  
  
**Output**



**Activity 2:**

**Code**

#include <iostream>

// Define a simple Node structure for the linked list

struct Node {

int data;

Node\* next;

Node\* prev; // For doubly linked list

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

};

// Class for the linked list operations

class LinkedList {

private:

Node\* head; // Pointer to the head of the list

Node\* tail; // Pointer to the tail of the list (for doubly linked list)

bool isCircular;

public:

LinkedList(bool circular = false) : head(nullptr), tail(nullptr), isCircular(circular) {}

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

void insertAtBeginning(int value) {

Node\* newNode = new Node(value);

if (isCircular) {

if (head == nullptr) {

newNode->next = newNode;

} else {

newNode->next = head;

Node\* lastNode = head;

while (lastNode->next != head) {

lastNode = lastNode->next;

}

lastNode->next = newNode;

}

head = newNode;

} else {

newNode->next = head;

head = newNode;

}

std::cout << "Inserted successfully at the beginning." << std::endl;

}

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

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (isCircular) {

if (head == nullptr) {

newNode->next = newNode;

head = newNode;

} else {

newNode->next = head;

Node\* lastNode = head;

while (lastNode->next != head) {

lastNode = lastNode->next;

}

lastNode->next = newNode;

}

} else {

if (head == nullptr) {

head = newNode;

tail = newNode;

} else {

tail->next = newNode;

tail = newNode;

}

}

std::cout << "Inserted successfully at the end." << std::endl;

}

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

void insertAfterValue(int value, int target) {

Node\* newNode = new Node(value);

Node\* current = head;

while (current != nullptr) {

if (current->data == target) {

newNode->next = current->next;

current->next = newNode;

std::cout << "Inserted successfully after " << target << "." << std::endl;

return;

}

current = current->next;

}

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

}

// Function to display the linked list

void display() {

Node\* current = head;

std::cout << "The items present in the list are: ";

if (current == nullptr) {

std::cout << "Empty";

} else {

if (isCircular) {

do {

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

current = current->next;

} while (current != head);

} else {

while (current != nullptr) {

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

current = current->next;

}

}

}

std::cout << std::endl;

}

// Function to reverse the linked list

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;

std::cout << "List reversed." << std::endl;

}

// Function to seek a specific value in the linked list

void seekValue(int value) {

Node\* current = head;

int position = 0;

while (current != nullptr) {

if (current->data == value) {

std::cout << "Value " << value << " found at position " << position << "." << std::endl;

return;

}

current = current->next;

position++;

}

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

}

// Function to delete the entire linked list

void deleteList() {

Node\* current = head;

while (current != nullptr) {

Node\* next = current->next;

delete current;

current = next;

}

head = nullptr;

std::cout << "List deleted." << std::endl;

}

~LinkedList() {

deleteList();

}

};

int main() {

int choice;

bool isCircular = false;

LinkedList list(isCircular);

do {

std::cout << "Operations on List.." << std::endl;

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

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

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

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

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

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

std::cout << "Enter your choice: ";

std::cin >> choice;

switch (choice) {

case 1:

int insertChoice;

std::cout << "1. Insertion at the beginning" << std::endl;

std::cout << "2. Insertion at the end" << std::endl;

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

std::cout << "Enter your choice: ";

std::cin >> insertChoice;

int insertValue;

std::cout << "Enter the value to insert: ";

std::cin >> insertValue;

switch (insertChoice) {

case 1:

list.insertAtBeginning(insertValue);

break;

case 2:

list.insertAtEnd(insertValue);

break;

case 3:

int insertTarget;

std::cout << "Enter the target value: ";

std::cin >> insertTarget;

list.insertAfterValue(insertValue, insertTarget);

break;

default:

std::cout << "Invalid choice!" << std::endl;

break;

}

break;

case 2:

// Implement deletion options here (e.g., delete by value or position)

// You can add these functions to the LinkedList class

break;

case 3:

list.display();

break;

case 4:

list.reverse();

break;

case 5:

int seekValue;

std::cout << "Enter the value to seek: ";

std::cin >> seekValue;

list.seekValue(seekValue);

break;

case 6:

std::cout << "Exiting the program..." << std::endl;

// Clean up the linked list memory

list.deleteList();

exit(0);

default:

std::cout << "Invalid choice!" << std::endl;

break;

}

std::cout << "Press any key to continue...";

std::cin.ignore();

std::cin.get();

} while (choice != 6);

return 0;

}

**Output**

