**Department of IT & CS**

**Course Instructor: Rizwan Ali Lab Engr.: Usama Dated: 29/11/2023**

**Semester: Fall 2023 Section: SE(Green)**

**COMP-201L**

**Lab 06: Linked List**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **CLO1** | **CLO2** | **CLO3** |  |
| **Name** | **Reg. No.** | **Lab Tasks Marks**  **20** | **Report**  **Marks**  **5** | **Viva**  **Marks**  **5** | **Total**  **Marks**  **30** |
| **Abuzar Khan** | **B22F1053SE2023** |  |  |  |  |
|  |  |  |  |  |  |

**Lab Task 1**

**Implement a function to traverse and print the elements of a linked list.**

**Lab Task 2**

**Write a function to search for a specific element in the linked list and return its position.**

**Lab Task 3:**

**Write a function to reverse a linked list.**

**Lab Task 4:**

**Write a function to remove duplicate elements from an unsorted linked list.**

**-: TASKS: -**

**# 01:** Implement a function to traverse and print the elements of a linked list.

#include <iostream>

using namespace std;

// Define the structure for a node in the singly linked list

struct Node

{

    int data;

    Node\* next;

};

// Function to traverse and print the elements of a linked list

void traverseAndPrint(Node\* head)

{

    // Create a temporary pointer and set it to the head of the list

    Node\* temp = head;

    // Traverse the linked list

    while (temp != nullptr)

    {

        // Print the data of the current node

        cout << temp->data << " ";

        // Move to the next node in the list

        temp = temp->next;

    }

    // Print a newline at the end

    cout << endl;

}

int main()

{

    // Create an empty linked list

    Node\* head = nullptr;

    // Insert nodes at the head of the list

    head = new Node{10, nullptr};

    head->next = new Node{20, nullptr};

    head->next->next = new Node{30, nullptr};

    // Traverse and print the elements of the linked list

    traverseAndPrint(head);

    return 0;

}

**# 02:** Write a function to search for a specific element in the linked list and return its position.

#include <iostream>

using namespace std;

// Define the structure for a node in the singly linked list

struct Node

{

    int data;

    Node\* next;

};

// Function to search for a specific element in the linked list and return its position

int searchElementPosition(Node\* head, int target)

{

    // Create a temporary pointer and set it to the head of the list

    Node\* temp = head;

    // Initialize a variable to keep track of the position

    int position = 1;

    // Traverse the linked list

    while (temp != nullptr)

    {

        // Check if the data of the current node matches the target

        if (temp->data == target)

        {

            // Return the position if the target is found

            return position;

        }

        // Move to the next node in the list

        temp = temp->next;

        // Increment the position

        position++;

    }

    // Return -1 if the target is not found in the linked list

    return -1;

}

int main()

{

    // Create an empty linked list

    Node\* head = nullptr;

    // Insert nodes at the head of the list

    head = new Node{10, nullptr};

    head->next = new Node{20, nullptr};

    head->next->next = new Node{30, nullptr};

    // Search for a specific element (e.g., 20) and print its position

    int targetElement = 20;

    int position = searchElementPosition(head, targetElement);

    if (position != -1)

    {

        cout << "Element " << targetElement << " found at position " << position << endl;

    }

    else

    {

        cout << "Element " << targetElement << " not found in the linked list." << endl;

    }

    return 0;

}

**# 03:** Write a function to reverse a linked list.

#include <iostream>

using namespace std;

// Define the structure for a node in the singly linked list

struct Node

{

    int data;

    Node\* next;

};

// Function to reverse a linked list

Node\* reverseLinkedList(Node\* head)

{

    Node\* prev = nullptr;

    Node\* current = head;

    Node\* next = nullptr;

    while (current != nullptr)

    {

        next = current->next; // Save the next node

        current->next = prev;  // Reverse the link

        prev = current;        // Move one step forward

        current = next;        // Move one step forward

    }

    // The 'prev' pointer now points to the new head of the reversed list

    return prev;

}

// Function to traverse and print the elements of a linked list

void traverseAndPrint(Node\* head)

{

    Node\* temp = head;

    while (temp != nullptr)

    {

        cout << temp->data << " ";

        temp = temp->next;

    }

    cout << endl;

}

int main()

{

    // Create an empty linked list

    Node\* head = nullptr;

    // Insert nodes at the head of the list

    head = new Node{10, nullptr};

    head->next = new Node{20, nullptr};

    head->next->next = new Node{30, nullptr};

    // Print the original linked list

    cout << "Original linked list: ";

    traverseAndPrint(head);

    // Reverse the linked list

    head = reverseLinkedList(head);

    // Print the reversed linked list

    cout << "Reversed linked list: ";

    traverseAndPrint(head);

    return 0;

}

**# 04:** Write a function to remove duplicate elements from an unsorted linked list.

#include <iostream>

#include <unordered\_set>

using namespace std;

// Define the structure for a node in the singly linked list

struct Node

{

    int data;

    Node\* next;

};

// Function to remove duplicate elements from an unsorted linked list

void removeDuplicates(Node\* head)

{

    if (head == nullptr || head->next == nullptr)

    {

        // The list is empty or has only one node

        return;

    }

    unordered\_set<int> uniqueSet; // To store unique elements

    Node\* current = head;

    Node\* previous = nullptr;

    while (current != nullptr)

    {

        // Check if the current data is already in the set

        if (uniqueSet.find(current->data) != uniqueSet.end())

        {

            // Duplicate found, remove the current node

            previous->next = current->next;

            delete current;

            current = previous->next;

        }

        else

        {

            // Add the current data to the set

            uniqueSet.insert(current->data);

            previous = current;

            current = current->next;

        }

    }

}

// Function to traverse and print the elements of a linked list

void traverseAndPrint(Node\* head)

{

    Node\* temp = head;

    while (temp != nullptr)

    {

        cout << temp->data << " ";

        temp = temp->next;

    }

    cout << endl;

}

int main()

{

    // Create an unsorted linked list with duplicates

    Node\* head = new Node{10, nullptr};

    head->next = new Node{20, nullptr};

    head->next->next = new Node{10, nullptr};

    head->next->next->next = new Node{30, nullptr};

    head->next->next->next->next = new Node{20, nullptr};

    // Print the original linked list

    cout << "Original linked list: ";

    traverseAndPrint(head);

    // Remove duplicates

    removeDuplicates(head);

    // Print the linked list after removing duplicates

    cout << "Linked list after removing duplicates: ";

    traverseAndPrint(head);

    return 0;

}