**Data Structures (Lab)**

**Assignment- 4 (Linked List-1)**

1. **WAP to implement Linked List.**
2. **Create a linked list.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    printLinkedList(head); // print linked list

    return 0;

}

1. **Insert an element at the start of the linked list.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* insertBeforeFirst(struct node\* head, int data) {

    struct node\* temp = new node{data, nullptr};

    temp->next = head;

    return temp;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    head = insertBeforeFirst(head, 5); // insert before first

    printLinkedList(head); // print linked list

    return 0;

}

1. **Insert an element at the end of the linked list.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* insertAfterLast(struct node\* head, int data) {

    struct node\* temp = head;

    while (temp->next != nullptr) {

        temp = temp->next;

    }

    struct node\* newNode = new node{data, nullptr};

    temp->next = newNode;

    return head;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    head = insertAfterLast(head, 5); // insert after last

    printLinkedList(head); // print linked list

    return 0;

}

1. **Insert an element before an existing element whose information is x in a linked list.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* insertBefore(struct node\* head, int before, int data) {

    if (head == nullptr) {

        return nullptr;

    }

    if (head->data == before) {

        struct node\* newNode = new node{data, head};

        return newNode;

    }

    struct node\* temp1 = head->next;

    struct node\* temp2 = head;

    while (temp1 != nullptr) {

        if (temp1->data == before) {

            struct node\* newNode = new node{data, temp1};

            temp2->next = newNode;

            return head;

        }

        temp2 = temp1;

        temp1 = temp1->next;

    }

    return head;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    head = insertBefore(head, 5, 50); // insert before

    printLinkedList(head); // print linked list

    return 0;

}

1. **Insert an element after an existing element whose information is x in a linked list.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* insertAfter(struct node\* head, int after, int data) {

    struct node\* temp = head;

    while (temp->next != nullptr) {

        if (temp->data == after) {

            struct node\* newNode = new node{data, nullptr};

            newNode->next = temp->next;

            temp->next = newNode;

            return head;

            break;

        }

        temp = temp->next;

    }

    return head;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    head = insertAfter(head, 5, 50); // insert after

    printLinkedList(head); // print linked list

    return 0;

}

1. **Delete the first element of the linked list.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* deleteFirst(struct node\* head) {

    if (head == nullptr) {

        return head;

    }

    struct node\* temp = head;

    head = head->next;

    delete temp;

    return head;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    head = deleteFirst(head); // delete first

    printLinkedList(head); // print linked list

    return 0;

}

1. **Delete the last element of the linked list.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* deleteLast(struct node\* head) {

    if (head == nullptr) {

        return head;

    }

    if (head->next == nullptr) {

        delete head;

        return nullptr;

    }

    struct node\* temp = head;

    while (temp->next->next != nullptr) {

        temp = temp->next;

    }

    struct node\* delNode = temp->next;

    temp->next = nullptr;

    delete delNode;

    return head;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    head = deleteLast(head); // delete last node

    printLinkedList(head); // print linked list

    return 0;

}

1. **Delete the element whose information is x from a linked list.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* deleteNthElement(struct node\* head, int n) {

    if (head == nullptr) {

        return head;

    }

    if (n == 0) {

        struct node\* delNode = head;

        head = head->next;

        delete delNode;

        return head;

    }

    int index = 0;

    struct node\* prev = nullptr;

    struct node\* temp = head;

    while (temp != nullptr) {

        if (index == n) {

            struct node\* delNode = temp;

            prev->next = temp->next;

            delete delNode;

            return head;

        }

        prev = temp;

        temp = temp->next;

        index++;

    }

    return head;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    head = deleteNthElement(head, 4); // delete nth node

    printLinkedList(head); // print linked list

    return 0;

}

1. **Display the contents of the linked list.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);

    printLinkedList(head); // print linked list

    return 0;

}

1. **WAP to find the length of the Linked List using recursion.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

int lengthOfLinkedList(struct node\* head, int n = 0) {

    if (head == nullptr) {

        return n;

    }

    return lengthOfLinkedList(head->next, n + 1);

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    int length = lengthOfLinkedList(head);

    cout << length << endl;

    return 0;

}

1. **WAP to concatenate two Linked Lists.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* concat(struct node\* head1, struct node\* head2) {

    struct node\* head = nullptr;

    struct node\* tail = nullptr;

    if (head1->data < head2->data) {

        head = tail = head1;

        head1 = head1->next;

    } else {

        head = tail = head2;

        head2 = head2->next;

    }

    while (head1 != nullptr && head2 != nullptr) {

        if (head1->data < head2->data) {

            tail->next = head1;

            head1 = head1->next;

        } else {

            tail->next = head2;

            head2 = head2->next;

        }

        tail = tail->next;

    }

    if (head1 != nullptr) {

        tail->next = head1;

    } else {

        tail->next = head2;

    }

    return head;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr1[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int arr2[] = {2, 4, 5, 5, 4, 2};

    int n1 = sizeof(arr1) / sizeof(arr1[0]);

    int n2 = sizeof(arr2) / sizeof(arr2[0]);

    struct node\* head1 = linkedList(arr1, n1);  // create linked list1

    struct node\* head2 = linkedList(arr2, n2);  // create linked list2

    struct node\* head = concat(head1, head2); // concat

    printLinkedList(head); // print linked list

    return 0;

}

1. **WAP to delete duplicate elements from a sorted Linked List.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* removeDuplicates(struct node\* head) {

    struct node\* tail = head;

    while (tail->next != nullptr) {

        if (tail->data == tail->next->data) {

            struct node\* delNode = tail->next;

            tail->next = delNode->next;

            delete delNode;

        }

        tail = tail->next;

    }

    return head;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    head = removeDuplicates(head); // delete duplicates

    printLinkedList(head); // print linked list

    return 0;

}

1. **WAP to delete every alternate element of the Linked List.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* removeAlternate(struct node\* head) {

    struct node\* tail = head;

    while (tail != nullptr && tail->next != nullptr) {

        struct node\* delNode = tail->next;  // The node to be deleted

        tail->next = delNode->next;         // Bypass the node

        delete delNode;  // Free the memory of the deleted node

        tail = tail->next;  // Move to the next node

    }

    return head;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 7, 8, 8, 55};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    head = removeAlternate(head); // delete alternate node

    printLinkedList(head); // print linked list

    return 0;

}

1. **WAP to check whether the Linked List is a palindrome or not.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* reverseLinkedList(struct node\* head) {

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

        return head;  // If the list is empty or has only one node, return the

                      // head (no change needed)

    }

    struct node\* newHead =

        nullptr;  // This will be the head of the new reversed list

    struct node\* current = head;

    while (current != nullptr) {

        struct node\* newNode = new node{

            current->data, nullptr};  // Create a new node with the current data

        newNode->next =

            newHead;        // Point the new node's next to the current newHead

        newHead = newNode;  // Move the newHead to the new node

        current = current->next;  // Move to the next node in the original list

    }

    return newHead;  // Return the head of the new reversed list

}

bool isPalindrome(struct node\* head) {

    struct node\* reversed = reverseLinkedList(head);

    struct node\* temp = head;

    while (reversed != nullptr) {

        if (reversed->data != temp->data) {

            return false;

        }

        reversed = reversed->next;

        temp = temp->next;

    }

    return true;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 5, 3, 1};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    cout << isPalindrome(head);

    return 0;

}

1. **WAP to reverse a Linked List.**

#include <iostream>

using namespace std;

struct node {

    int data;

    struct node\* next;

};

struct node\* linkedList(int arr[], int n) {

    struct node\* head = new node{arr[0], nullptr};

    struct node\* temp = head;

    for (int i = 1; i < n; i++) {

        temp->next = new node{arr[i], nullptr};

        temp = temp->next;

    }

    return head;

}

struct node\* reverseLinkedList(struct node\* head) {

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

        return head;

    }

    struct node\* newHead = nullptr;

    struct node\* current = head;

    while (current != nullptr) {

        struct node\* newNode = new node{current->data, nullptr};

        newNode->next = newHead;

        newHead = newNode;

        current = current->next;

    }

    return newHead;

}

void printLinkedList(struct node\* head) {

    while (head != nullptr) {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main() {

    int arr[] = {1, 3, 5, 4, 6, 9};

    int n = sizeof(arr) / sizeof(arr[0]);

    struct node\* head = linkedList(arr, n);  // create linked list

    head = reverseLinkedList(head);

    printLinkedList(head);  // print reversed linked list

    return 0;

}