Q1 sll

#include<iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int data){

        this->data = data;

        this->next = NULL;

    }

};

class SinglyLinkedList{

private:

    Node\* head;

public:

    SinglyLinkedList() {

        head = NULL;

    }

    void InsertAtHead(int x) {

        Node\* newNode = new Node(x);

        if(head == NULL) {

            head = newNode;

            return;

        }

        newNode->next = head;

        head = newNode;

    }

    void InsertAtTail(int x) {

        Node\* newNode = new Node(x);

        if (head == NULL) {

            head = newNode;

            return;

        }

        Node\* temp = head;

        while(temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

    }

    void InsertBeforePos(int x, int k){

        Node\* newNode = new Node(x);

        if (head == NULL) {

            if ( k == 0) {

                head = newNode;

                return;

            }

            else {

                cout << "Invalid postion" << endl;

                return;

            }

        }

        Node\* temp = head;

        int s = 0;

        while(temp) {

            s++;

            if(s == k-1) {

                break;

            }

            temp = temp->next;

        }

        newNode->next =  temp->next;

        temp->next = newNode;

    }

    void InsertAfterPos(int x, int k) {

        Node\* newNode = new Node(x);

        if (head == NULL){

            if (k == 0){

                head = newNode;

                return;

            }

            else {

                cout << "Invalid position" << endl;

                return;

            }

        }

        Node\* temp = head;

        int s = 0;

        while(temp) {

            s++;

            if (s == k) {

                break;

            }

            temp = temp -> next;

        }

        newNode->next = temp->next;

        temp->next = newNode;

    }

    void DelAtHead() {

        if(head == NULL){

            return;

        }

        if (head->next == NULL){

            delete head;

            return;

        }

        Node \*temp = head;

        head = head->next;

        delete temp;

    }

    void DelAtPos(int k) {

        if(head == NULL) {

            return;

        }

        Node\* temp = head;

        if (k == 0) {

            head = head->next;

            return;

        }

        int s = 0;

        while(temp) {

            s++;

            if (s == k) {

                break;

            }

            temp = temp->next;

        }

        Node\* elem = temp->next;

        temp->next = temp->next->next;

        delete elem;

    }

    void print(){

        Node\* temp = head;

        if(head == NULL) {

                cout  << "The list is empty" << endl;

                return;

        }

        while(temp) {

            cout << temp->data << " ";

            temp = temp->next;

        }

        cout << endl;

    }

};

int main() {

    SinglyLinkedList l1;

    l1.InsertAtHead(2);

    l1.InsertAtHead(9);

    l1.InsertAtHead(4);

    l1.print();

    l1.InsertAtTail(3);

    l1.InsertAtTail(0);

    l1.print();

    l1.InsertBeforePos(10, 4);

    l1.print();

    l1.InsertAfterPos(5, 1);

    l1.print();

    l1.DelAtPos(1);

    l1.print();

    l1.DelAtPos(2);

    l1.print();

    l1.DelAtHead();

    l1.print();

    return 0;

}

Q2 dll

#include<iostream>

using namespace std;

class Node{

    public:

    int data;

    Node\* next;

    Node\* previous;

    Node(int data) {

        this->data = data;

        this ->next = NULL;

        this->previous = NULL;

    }

};

class DoublyLinkedList{

private:

    Node\* head;

public:

    DoublyLinkedList() {

        head == NULL;

    }

    void InsertAtHead(int x) {

        Node\* newNode = new Node(x);

        if(head == NULL) {

            head = newNode;

            return;

        }

        newNode->next = head;

        head->previous = newNode;

        head = newNode;

    }

    void InsertAtTail(int x) {

        Node\* newNode = new Node(x);

        if(head == NULL) {

            head = newNode;

            return;

        }

        Node\* temp = head;

        while(temp->next != 0) {

            temp = temp->next;

        }

        newNode->previous = temp;

        temp->next = newNode;

    }

    void DelAtHead() {

        if(head == NULL) {

            return;

        }

        if(head->next == NULL) {

            delete head;

            return;

        }

        Node\* temp = head;

        head = head->next;

        head->previous = NULL;

        delete temp;

    }

    void DelAtTail() {

        if(head == NULL) {

            return;

        }

        Node\* temp = head;

        while(temp->next != NULL) {

            temp = temp->next;

        }

        temp->previous->next = NULL;

        delete temp;

    }

    void Print() {

        if(head == NULL) {

            cout << "The list is empty" << endl;

            return;

        }

        Node\* temp = head;

        while (temp != NULL) {

            cout << temp->data << " ";

            temp = temp->next;

        }

        cout << endl;

    }

};

int main() {

    DoublyLinkedList d1;

    d1.InsertAtHead(9);

    d1.InsertAtHead(4);

    d1.InsertAtHead(0);

    d1.Print();

    d1.InsertAtTail(6);

    d1.InsertAtTail(3);

    d1.Print();

    d1.DelAtHead();

    d1.Print();

    d1.DelAtTail();

    d1.Print();

    return 0;

}

Q3 cll

#include <iostream>

using namespace std;

class Node{

public:

    int data;

    Node\* next;

    Node(int data) {

        this->data = data;

        this->next = NULL;

    }

};

class CircularLinkedList{

private:

    Node\* tail;

public:

    CircularLinkedList() {

        tail == NULL;

    }

    void InsertAtHead(int x) {

        Node\* newNode = new Node(x);

        if (tail == NULL) {

            tail = newNode;

            newNode->next = newNode;

            return;

        }

        newNode->next = tail->next;

        tail->next = newNode;

    }

    void InsertAtTail(int x) {

        Node\* newNode = new Node(x);

        if(tail == NULL) {

            tail = newNode;

            newNode->next = newNode;

            return;

        }

        newNode->next = tail->next;

        tail->next = newNode;

        tail = newNode;

    }

    void DelAtHead() {

        if(tail == NULL) {

            return;

        }

        if(tail->next == NULL) {

            delete tail;

            return;

        }

        Node\* temp = tail->next;

        tail->next = temp->next;

        delete temp;

    }

    void DelAtTail() {

        if(tail == NULL) {

            return;

        }

        Node\* temp = tail;

        Node\* c = tail->next;

        while ( c->next != tail) {

            c = c->next;

        }

        c->next = temp->next;

        delete temp;

        tail = c;

    }

    Node\* search(int x) {

        Node\* temp = tail->next;

        if (tail == NULL) {

            cout << "cannot find the element as the list is empty" << endl;

            return NULL;

        }

        do{

            if (temp->data == x) {

                return temp;

            }

            temp = temp->next;

        } while (temp != tail->next);

        return NULL;

    }

    void print() {

        if (tail == NULL) {

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

            return;

        }

        Node\* temp = tail->next;

        do {

            cout << temp->data << " ";

            temp = temp->next;

        } while(temp != tail->next);

        cout << endl;

    }

};

int main() {

    CircularLinkedList c1;

    c1.InsertAtHead(2);

    c1.InsertAtHead(5);

    c1.print();

    c1.DelAtTail();

    c1.print();

    c1.InsertAtTail(7);

    c1.InsertAtTail(1);

    c1.print();

    c1.DelAtHead();

    c1.print();

    Node\* result = c1.search(2);

    if(result) {

        cout << result << endl;

    } else {

        cout << "element 2 is not found" << endl;

    }

    result = c1.search(7);

    if(result) {

        cout << result << endl;

    } else {

        cout << "element 7 is not found" << endl;

    }

    return 0;

}

Q4 stack

#include<iostream>

using namespace std;

class Node{

public:

    int data;

    Node\* next;

    Node(int data) {

        this->data = data;

        this->next = nullptr;

    }

};

class stack{

private:

    Node\* top;

    int size = 0;

    int n;

public:

    stack(int x) {

        top == NULL;

        n = x;

    }

    void push(int x){

        Node\* newNode = new Node(x);

        if (size == n) {

            cout << "Stack is full" << endl;

            return;

        }

        if (top == NULL) {

            top = newNode;

            size++;

            return;

        }

        newNode->next = top;

        top = newNode;

        size++;

    }

    void pop() {

        if (top == NULL) {

            cout << "Stack is empty" << endl;

            return;

        }

        Node\* temp = top;

        top = top->next;

        size--;

    }

    void print() {

        Node\* temp = top;

        while (temp) {

            cout << temp->data << " ";

            temp = temp->next;

        }

        cout << endl;

    }

};

int main() {

    stack s(5);

    s.push(1);

    s.push(2);

    s.push(3);

    s.push(4);

    s.print();

    s.pop();

    s.print();

    return 0;

}

Q5 queue

#include<iostream>

using namespace std;

class Node{

public:

    int data;

    Node\* next;

    Node(int data) {

        this->data = data;

        this->next = NULL;

    }

};

class queue {

private:

    Node\* front;

    Node\* rear;

    int size = 0;

public:

    queue() {

        front = rear = NULL;

    }

    void push(int x) {

        Node\* newNode = new Node(x);

        if (front == NULL) {

            front = rear = newNode;

            size++;

        }

        else{

            rear->next = newNode;

            rear = newNode;

            size++;

        }

    }

    void pop() {

        if (front == NULL) {

            cout << "Queue is empty" << endl;

            return;

        }

        front = front->next;

        size--;

    }

    void print() {

        Node\* temp = front;

        while (temp) {

            cout << temp->data << " ";

            temp = temp->next;

        }

        cout << endl;

    }

};

int main() {

    queue q;

    q.push(1);

    q.push(2);

    q.push(3);

    q.push(4);

    q.print();

    q.pop();

    q.print();

    return 0;

}

Q6 bst

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* left;

    Node\* right;

    Node(int data) {

        this->data = data;

        this->left = nullptr;

        this->right = nullptr;

    }

};

class BST {

private:

    Node\* root;

public:

    BST() {

        root == NULL;

    }

    void insert(Node\* root, int x) {

        Node\* newNode = new Node(x);

        if(root == nullptr) {

            root = newNode;

        }

        else if(x < root->data) {

            insert(root->left, x);

        }

        else{

            insert(root->right, x);

        }

    }

    Node\* delete1(Node\* root, int x) {

        if (root == NULL) {

            return NULL;

        }

        if (x == root->data) {

            Node\* temp = root->right;

            delete root;

            return temp;

        }

        if(x < root->data) {

            delete1(root->left, x);

        }

        else if(x > root->data) {

            delete1(root->right, x);

        }

        else {

            if(root->right == NULL) {

                Node\* temp = root->left;

                delete root;

                return temp;

            } else if(root->left == NULL) {

                Node\* temp = root->right;

                delete root;

                return temp;

            }

        }

    }

    bool search(Node\* root, int x) {

        if (root == nullptr) {

            return false;

        }

        if (x == root->data) {

            return true;

        }

        if(x < root->data) {

            search(root->left, x);

        } else if(x > root->data) {

            search(root->right, x);

        }

    }

    void preorder(Node\* root) {

        if(root == nullptr){

            return;

        }

        cout << root->data << " ";

        preorder(root->left);

        preorder(root->right);

    }

    void inorder (Node\* root) {

        inorder(root->left);

        cout << root->data << " ";

        inorder(root->right);

    }

    void postorder (Node\* root) {

        postorder(root->left);

        postorder(root->right);

        cout << root->data << " ";

    }

};

int main() {

    BST bst;

    bst.insert(50);

    bst.insert(30);

    bst.insert(70);

    bst.insert(20);

    bst.insert(100);

    bst.insert(60);

    bst.inorder();

    cout << endl;

    bst.preorder();

    cout << endl;

    bst.postorder()

    cout << endl;

    bst.delete1(70);

    cout << endl;

    bst.search(60);

}

Q7 linear search

#include <iostream>

using namespace std;

int linearSearch(int arr[], int size, int key) {

    for (int i = 0; i < size; i++) {

        if (arr[i] == key) {

            return i;

        }

    }

    return -1;

}

int main() {

    int n, key;

    cout << "Enter the number of elements in the array: ";

    cin >> n;

    int arr[n];

    cout << "Enter the elements of the array: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    cout << "Enter the element to search: ";

    cin >> key;

    int result = linearSearch(arr, n, key);

    if (result != -1) {

        cout << "Element found at index " << result << "." << endl;

    } else {

        cout << "Element not found in the array." << endl;

    }

    return 0;

}

Q8 binary search

#include <iostream>

using namespace std;

int binarySearch(int arr[], int size, int key) {

    int low = 0, high = size - 1;

    while (low <= high) {

        int mid = low + (high - low) / 2;

        if (arr[mid] == key) {

            return mid;

        } else if (arr[mid] < key) {

            low = mid + 1;

        } else {

            high = mid - 1;

        }

    }

    return -1;

}

int main() {

    int n, key;

    cout << "Enter the number of elements in the array: ";

    cin >> n;

    int arr[n];

    cout << "Enter the elements of the array in sorted order: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    cout << "Enter the element to search: ";

    cin >> key;

    int result = binarySearch(arr, n, key);

    if (result != -1) {

        cout << "Element found at index " << result << "." << endl;

    } else {

        cout << "Element not found in the array." << endl;

    }

    return 0;

}

Q9 insertion sort

#include <iostream>

using namespace std;

void insertionSort(int arr[], int n) {

    for (int i = 1; i < n; i++) {

        int key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = key;

    }

}

int main() {

    int n;

    cout << "Enter the number of elements in the array: ";

    cin >> n;

    int arr[n];

    cout << "Enter the elements of the array: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    insertionSort(arr, n);

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << endl;

    return 0;

}

Q10 factorial

#include <iostream>

using namespace std;

int factorial(int n) {

    if (n == 0 || n == 1) {

        return 1;

    } else {

        return n \* factorial(n - 1);

    }

}

int main() {

    int number = 5;

    int result = factorial(number);

    cout << "The factorial of " << number << " is " << result << "." << endl;

    return 0;

}

Q11 power

#include <iostream>

using namespace std;

double power(double base, int exponent) {

    if (exponent == 0) {

        return 1;

    } else if (exponent < 0) {

        return 1 / power(base, -exponent);

    } else {

        return base \* power(base, exponent - 1);

    }

}

int main() {

    double base = 2.0;

    int exponent = 3;

    double result = power(base, exponent);

    cout << base << " raised to the power of " << exponent << " is " << result << "." << endl;

    return 0;

}

Q12 fibonacci

#include <iostream>

using namespace std;

int fibonacci(int n) {

    if (n == 0) {

        return 0;

    } else if (n == 1) {

        return 1;

    } else {

        return fibonacci(n - 1) + fibonacci(n - 2);

    }

}

int main() {

    int number = 6;

    int result = fibonacci(number);

    cout << "Fibonacci of " << number << " is " << result << "." << endl;

    return 0;

}