1.Linklist push pop display

#include<iostream>

using namespace std;

class node{

    public:

    int data;

    node\* next;

    node(int data){

        this->data=data;

        this->next=NULL;

    }

};

class linkedlist{

    public:

    node\* head;

    linkedlist(){

        head=nullptr;

    }

    void push(int data){

        node\* temp=new node(data);

        temp->next=head;

        head=temp;

    }

    void display(){

        node\* curr=head;

        while(curr!=NULL){

            cout<<curr->data<<" ";

            curr=curr->next;

        }

    }

    void insert(int data,int pos){

        if(pos==0){

            node\* temp =new node(data);

            temp->next=head;

            head=temp;

        }

        else{

        node\* prev=head;

        for(int i=0;i<pos-1;i++){

            prev=prev->next;

        }

        node\* temp=new node(data);

        temp->next=prev->next;

        prev->next=temp;

        }

    }

    void deletenode(int pos){

        if(pos==0){

            head=head->next;

        }

        else{

            node\* prev=head;

            for(int i=0;i<pos-1;i++){

            prev=prev->next;

        }

        prev->next=prev->next->next;

        }

    }

    void reverse(){

        node\* prev=NULL;

        node\* curr=head;

        while (curr!=NULL)

        {

            node\* temp=curr->next;

            curr->next=prev;

            prev=curr;

            curr=temp;

        }

        head=prev;

    }

};

int main(){

    linkedlist list;

    list.push(10);

    list.push(20);

    list.push(30);

    list.push(50);

    list.push(60);

    list.push(70);

    list.push(80);

    list.display();

    list.insert(40,2);

    cout<<" \n";

    list.display();

    list.insert(111,0);

    cout<<" \n";

    list.display();

    list.deletenode(2);

    cout<<" \n";

    list.display();

    list.reverse();

    cout<<"\n";

    list.display();

}

2. Add two number in link list

#include <iostream>

struct Node {

    int value;

    Node\* next;

};

Node\* createNode(int value) {

    Node\* newNode = new Node();

    newNode->value = value;

    newNode->next = nullptr;

    return newNode;

}

Node\* addLists(Node\* list1, Node\* list2) {

    Node\* head1 = list1;

    Node\* head2 = list2;

    int carry = 0;

    Node\* resultHead = nullptr;

    Node\* resultTail = nullptr;

    while (head1 || head2 || carry) {

        int value1 = head1 ? head1->value : 0;

        int value2 = head2 ? head2->value : 0;

        // Calculate the sum and carry

        int total = value1 + value2 + carry;

        carry = total / 10;

        int digit = total % 10;

        // Create a new node for the result

        Node\* newNode = createNode(digit);

        if (resultHead == nullptr) {

            resultHead = newNode;

            resultTail = newNode;

        } else {

            resultTail->next = newNode;

            resultTail = newNode;

        }

        // Move to the next nodes in the lists

        if (head1)

            head1 = head1->next;

        if (head2)

            head2 = head2->next;

    }

    return resultHead;

}

void printList(Node\* head) {

    Node\* current = head;

    while (current != nullptr) {

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

        current = current->next;

    }

}

int main() {

    // Create the first list: 7->1->5

    Node\* list1 = createNode(7);

    list1->next = createNode(1);

    list1->next->next = createNode(5);

    // Create the second list: 5->9->2

    Node\* list2 = createNode(5);

    list2->next = createNode(9);

    list2->next->next = createNode(2);

    // Add the lists and store the result

    Node\* resultList = addLists(list1, list2);

    // Print the result list

    printList(resultList);

    return 0;

}

3.. Palindrome linklist

#include <iostream>

#include <stack>

using namespace std;

struct Node {

    int data;

    Node\* next;

};

Node\* createNode(int data) {

    Node\* newNode = new Node();

    newNode->data = data;

    newNode->next = nullptr;

    return newNode;

}

bool isPalin(Node\* head) {

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

        return true;

    // Find the middle of the linked list

    Node\* slow = head;

    Node\* fast = head;

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

        slow = slow->next;

        fast = fast->next->next;

    }

    // Reverse the second half of the linked list

    Node\* prev = nullptr;

    Node\* curr = slow->next;

    while (curr != nullptr) {

        Node\* nextNode = curr->next;

        curr->next = prev;

        prev = curr;

        curr = nextNode;

    }

    // Compare the first half with the reversed second half

    Node\* firstHalf = head;

    Node\* secondHalf = prev;

    while (secondHalf != nullptr) {

        if (firstHalf->data != secondHalf->data)

            return false;

        firstHalf = firstHalf->next;

        secondHalf = secondHalf->next;

    }

    return true;

}

int main() {

    Node\* list1 = createNode(7);

    list1->next = createNode(1);

    list1->next->next = createNode(2);

    list1->next->next->next = createNode(7);

    // list1->next->next->next->next = createNode(7);

    bool result = isPalin(list1);

    if (result)

        cout << "isPalindrome is true\n";

    else

        cout << "isPalindrome is false\n";

    return 0;

}

4. String matching

#include <iostream>

using namespace std;

bool isMatch(string text, string pattern) {

    int n = text.length();

    int m = pattern.length();

    for (int i = 0; i <= n - m; i++) {

        int j;

        for (j = 0; j < m; j++) {

            if (text[i + j] != pattern[j])

                break;

        }

        if (j == m)

            return true; // Match found

    }

    return false; // No match found

}

int main() {

    string text;

    string pattern;

    cout<<"Input Your String : ";

    getline(cin,text);

    cout<<"Pattern : ";

    getline(cin,pattern);

    if (isMatch(text, pattern))

        cout << "Pattern found in the text.\n";

    else

        cout << "Pattern not found in the text.\n";

    return 0;

}

5.Characters are sorted in a form of a list (non continuous).  We have to check if the list of characters has a given character in it or not.

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

    int val;

    Node \*next;

    Node(){

        this->val = INT\_MIN;

        this->next = NULL;

    }

    Node(int x){

        val = x;

        this->next = NULL;

    }

};

class LL{

    Node \*head;

public:

    LL(){

        this->head = NULL;

    }

    void insertAtBegin(int x){

        Node \*n1 = new Node(x);

        n1->next = head;

        head = n1;

    }

    void insertAtEnd(int x){

        Node \*n1 = new Node(x);

        if(head==NULL){

            head=n1;

            return ;

        }

        Node \*temp = head;

        while(temp->next){

            temp = temp->next;

        }

        temp->next = n1;

    }

    bool isFound(int x){

        Node \*temp = head;

        while(temp){

            if(temp->val == x){

                return true;

            }

            temp = temp->next;

        }

        return false;

    }

    void printList(){

        if(!head){

            cout<<"Underflow: No data to print.";

            return ;

        }

        Node \*temp = head;

        while(temp){

            cout<<temp->val<<"->";

            temp = temp->next;

        }

    }

};

int main(){

    LL l1,l2,l3;

    l1.insertAtEnd(1);

    l1.insertAtEnd(5);

    l1.insertAtEnd(7);

    cout<<(l1.isFound(5) ? "Found\n" : "Not Found\n");

    return 0;

}

6. Common elements in a list

#include<bits/stdc++.h>

using namespace std;

int isPresent(vector<int>& arr, int k) {

    int n = arr.size();

    int l = 0, r = n - 1, ans = 0;

    while (l <= r) {

        int m = (l + r) / 2;

        if (arr[m] == k) {

            ans = 1;

            break;

        }

        else if (arr[m] < k) {

            l = m + 1;

        }

        else {

            r = m - 1;

        }

    }

    return ans;

}

void print1d(vector<int>& v) {

    for (auto& x : v) {

        cout << x << " ";

    }

    cout << "\n";

}

int main() {

    vector<int> v1 = { 1,2,3,4,5,8,9 };

    vector<int> v2 = { 2,4,8,9,11,13 };

    vector<int> v3 = { 1,2,3,5,9,12,18,18 };

    vector<int> ans;

    int n1 = v1.size(), n2 = v2.size(), n3 = v3.size();

    int n = min({ n1,n2,n3 });

    if (n1 == n) {

        for (int i = 0; i < n; i++) {

            if (isPresent(v2, v1[i]) && isPresent(v3, v1[i])) {

                ans.push\_back(v1[i]);

            }

        }

    }

    else if (n2 == n) {

        for (int i = 0; i < n; i++) {

            if (isPresent(v1, v2[i]) && isPresent(v3, v2[i])) {

                ans.push\_back(v2[i]);

            }

        }

    }

    else {

        for (int i = 0; i < n; i++) {

            if (isPresent(v2, v3[i]) && isPresent(v1, v3[i])) {

                ans.push\_back(v3[i]);

            }

        }

    }

    print1d(ans);

    return 0;

}

7. Longest Subsecqunce in a string

#include <iostream>

#include <vector>

#include <algorithm>

std::string longestCommonSubsequence(const std::string& str1, const std::string& str2) {

    int m = str1.length();

    int n = str2.length();

    // Create a 2D vector to store the lengths of LCS for substrings

    std::vector<std::vector<int>> dp(m + 1, std::vector<int>(n + 1, 0));

    // Build the dp table

    for (int i = 1; i <= m; ++i) {

        for (int j = 1; j <= n; ++j) {

            if (str1[i - 1] == str2[j - 1]) {

                dp[i][j] = dp[i - 1][j - 1] + 1;

            } else {

                dp[i][j] = std::max(dp[i - 1][j], dp[i][j - 1]);

            }

        }

    }

    // Backtrack to find the LCS

    std::string lcs;

    int i = m, j = n;

    while (i > 0 && j > 0) {

        if (str1[i - 1] == str2[j - 1]) {

            lcs = str1[i - 1] + lcs;

            --i;

            --j;

        } else if (dp[i - 1][j] > dp[i][j - 1]) {

            --i;

        } else {

            --j;

        }

    }

    return lcs;

}

int main() {

    std::string str1 = "ABCDGH";

    std::string str2 = "AEDFHR";

    std::string lcs = longestCommonSubsequence(str1, str2);

    std::cout << "Longest Common Subsequence: " << lcs << std::endl;

    return 0;

}

8.Count Sort

#include <iostream>

#include <vector>

void countingSort(std::vector<int>& arr) {

    int n = arr.size();

    // Find the maximum element in the array

    int maxElement = arr[0];

    for (int i = 1; i < n; ++i) {

        if (arr[i] > maxElement) {

            maxElement = arr[i];

        }

    }

    // Create a count array to store the frequency of each element

    std::vector<int> count(maxElement + 1, 0);

    // Calculate the frequency of each element

    for (int i = 0; i < n; ++i) {

        count[arr[i]]++;

    }

    // Update the count array to store the cumulative count of elements

    for (int i = 1; i <= maxElement; ++i) {

        count[i] += count[i - 1];

    }

    // Create a temporary array to store the sorted elements

    std::vector<int> sortedArr(n);

    // Build the sorted array

    for (int i = n - 1; i >= 0; --i) {

        sortedArr[count[arr[i]] - 1] = arr[i];

        count[arr[i]]--;

    }

    // Copy the sorted elements back to the original array

    for (int i = 0; i < n; ++i) {

        arr[i] = sortedArr[i];

    }

}

int main() {

    std::vector<int> arr = {4, 2, 0, 2, 5, 1, 4, 5};

    countingSort(arr);

    std::cout << "Sorted Array: ";

    for (int num : arr) {

        std::cout << num << " ";

    }

    std::cout << std::endl;

    return 0;

}

9. Quicksort

#include <iostream>

#include <vector>

int partition(std::vector<int>& arr, int low, int high) {

    // Choose the rightmost element as the pivot

    int pivot = arr[high];

    // Index of the smaller element

    int i = low - 1;

    for (int j = low; j < high; ++j) {

        // If current element is smaller than or equal to the pivot

        if (arr[j] <= pivot) {

            // Swap arr[i] and arr[j]

            std::swap(arr[++i], arr[j]);

        }

    }

    // Swap the pivot element with the element at the correct position

    std::swap(arr[i + 1], arr[high]);

    // Return the partitioning index

    return i + 1;

}

void quickSort(std::vector<int>& arr, int low, int high) {

    if (low < high) {

        // Partition the array into two halves

        int pivotIndex = partition(arr, low, high);

        // Recursively sort the two halves

        quickSort(arr, low, pivotIndex - 1);

        quickSort(arr, pivotIndex + 1, high);

    }

}

int main() {

    std::vector<int> arr = {4, 2, 0, 2, 5, 1, 4, 5};

    quickSort(arr, 0, arr.size() - 1);

    std::cout << "Sorted Array: ";

    for (int num : arr) {

        std::cout << num << " ";

    }

    std::cout << std::endl;

    return 0;

}

10. 0/1 Knapsack

#include <iostream>

#include <vector>

using namespace std;

// Structure to represent an item

struct Item {

    int weight;

    int value;

};

// Function to solve the 0/1 Knapsack problem

int knapsack(vector<Item>& items, int capacity) {

    int n = items.size();

    vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));

    for (int i = 1; i <= n; i++) {

        for (int w = 1; w <= capacity; w++) {

            // If the current item's weight is less than or equal to the remaining capacity

            // include it in the knapsack by checking its value

            if (items[i - 1].weight <= w) {

                dp[i][w] = max(items[i - 1].value + dp[i - 1][w - items[i - 1].weight], dp[i - 1][w]);

            } else {

                // If the current item's weight is greater than the remaining capacity

                // exclude it from the knapsack

                dp[i][w] = dp[i - 1][w];

            }

        }

    }

    return dp[n][capacity];

}

int main() {

    vector<Item> items = {

        {2, 10},

        {3, 15},

        {5, 20},

        {7, 25}

    };

    int capacity = 10;

    int maxVal = knapsack(items, capacity);

    cout << "Maximum value that can be obtained: " << maxVal << endl;

    return 0;

}

11. Fractional knapsack

#include <iostream>

#include <vector>

#include <algorithm>

struct Item {

    int weight;

    int value;

};

bool compare(Item item1, Item item2) {

    double ratio1 = static\_cast<double>(item1.value) / item1.weight;

    double ratio2 = static\_cast<double>(item2.value) / item2.weight;

    return ratio1 > ratio2;

}

double fractionalKnapsack(int capacity, std::vector<Item>& items) {

    std::sort(items.begin(), items.end(), compare);

    double totalValue = 0.0;

    for (const auto& item : items) {

        if (capacity >= item.weight) {

            totalValue += item.value;

            capacity -= item.weight;

        } else {

            totalValue += static\_cast<double>(item.value) \* capacity / item.weight;

            break;

        }

    }

    return totalValue;

}

int main() {

    int capacity;

    std::cout << "Enter the capacity of the knapsack: ";

    std::cin >> capacity;

    int numItems;

    std::cout << "Enter the number of items: ";

    std::cin >> numItems;

    std::vector<Item> items(numItems);

    for (int i = 0; i < numItems; ++i) {

        std::cout << "Enter weight and value for item " << (i + 1) << ": ";

        std::cin >> items[i].weight >> items[i].value;

    }

    double maxTotalValue = fractionalKnapsack(capacity, items);

    std::cout << "The maximum total value in the knapsack is: " << maxTotalValue << std::endl;

    return 0;

}

12. Implement stack using two queues

// implementing stack using two queues

#include<iostream>

#include<queue>

using namespace std;

class stack{

    public:

    queue<int> q1;

    queue<int> q2;

    void push(int data){

        while(!q1.empty()){

            q2.push(q1.front());

            q1.pop();

        }

        q1.push(data);

        while(!q2.empty()){

            q1.push(q2.front());

            q2.pop();

        }

    }

    int pop(){

        if(!q1.empty()){

            int res=q1.front();

            q1.pop();

            return res;

        }

        return 0;

    }

    int peek(){

        return q1.front();

    }

    bool empty(){

        return q1.empty();

    }

};

int main()

{

stack s;

s.push(10);

s.push(20);

s.push(30);

s.push(40);

cout<<s.pop();

cout<<s.pop();

}

13. implement queue using two stacks

// implementing queue using two stacks

#include <iostream>

#include <stack>

using namespace std;

class queue

{

public:

    stack<int> s1;

    stack<int> s2;

    void enqueue(int data)

    {

        s1.push(data);

    }

    void dequeue()

    {

        if(s1.empty()){

            cout<<"no elements";

        }

        while (!s1.empty())

        {

            s2.push(s1.top());

            s1.pop();

        }

        s2.pop();

        while (!s2.empty())

        {

            s1.push(s2.top());

            s2.pop();

        }

    }

    int peek()

    {

        while (!s1.empty())

        {

            s2.push(s1.top());

            s1.pop();

        }

        int res = s2.top();

        while (!s2.empty())

        {

            s1.push(s2.top());

            s2.pop();

        }

        return res;

    }

    bool empty()

    {

        if (!s1.empty())

        {

            return false;

        }

        return true;

    }

};

int main()

{

    queue q;

    q.enqueue(1);

    //  cout<<q.peek()<<"\n";

    q.enqueue(2);

    //  cout<<q.peek()<<"\n";

    q.enqueue(3);

    //  cout<<q.peek()<<"\n";

    q.enqueue(4);

    //  cout<<q.peek()<<"\n";

    cout<<q.peek()<<"\n";

    q.dequeue();

    cout<<q.peek()<<"\n";

    q.dequeue();

    cout<<q.peek()<<"\n";

    q.dequeue();

    cout<<q.peek()<<"\n";

    q.dequeue();

    q.dequeue();

}

14.in,pre,post , level

// level order traversal

#include<iostream>

#include<queue>

using namespace std;

class node{

    public:

    int data;

    node\* right;

    node\* left;

    node(int data){

        this->data=data;

        this->right=NULL;

        this->left=NULL;

    }

};

node\* createtree(node\* root){

    int data;

    cout<<"Enter data :";

    cin>>data;

    if(data==-1)

        return NULL;

    root=new node(data);

    cout<<"for left of "<<root->data<<" ";

    root->left=createtree(root->left);

    cout<<"for right of "<<root->data<<" ";

    root->right=createtree(root->right);

    return root;

}

void inorder(node\* root){

    if(root==NULL){

        return ;

    }

    inorder(root->left);

    cout<<root->data<<" ";

    inorder(root->right);

}

void preorder(node\* root){

    if(root==NULL){

        return ;

    }

    cout<<root->data<<" ";

    preorder(root->left);

    preorder(root->right);

}

void postorder(node\* root){

    if(root==NULL){

        return ;

    }

    postorder(root->left);

    postorder(root->right);

    cout<<root->data<<" ";

}

void levelorder(node\* root){

    queue<node\*> q;

    q.push(root);

    q.push(NULL);

    while(!q.empty()){

        node\* curr=q.front();

        q.pop();

        if(curr==NULL){

            cout<<"\n";

            if(!q.empty())

                q.push(NULL);

        }

        else{

        cout<<curr->data<<" ";

        if(curr->left!=NULL)

            q.push(curr->left);

        if(curr->right!=NULL)

            q.push(curr->right);

        }

    }

}

int main(){

    node\* root=NULL;

    root=createtree(root);

    inorder(root);

    cout<<"\n";

    preorder(root);

    cout<<"\n";

    postorder(root);

    cout<<"\n";

    levelorder(root);

}