Sliding Window

NOTE:
Subarray→ contiguous & order it is subpart of array;
Subsequences → It is ordered but may or may not be contiguous;

1. **Fixed Length Sliding window:**

   When to apply:
   —> When we have to find a subarray with certain conditions.
   —> let we have array of length n

   We have to find array of m length and m < n;
   And we have to find array of length m with max sum then apply then;

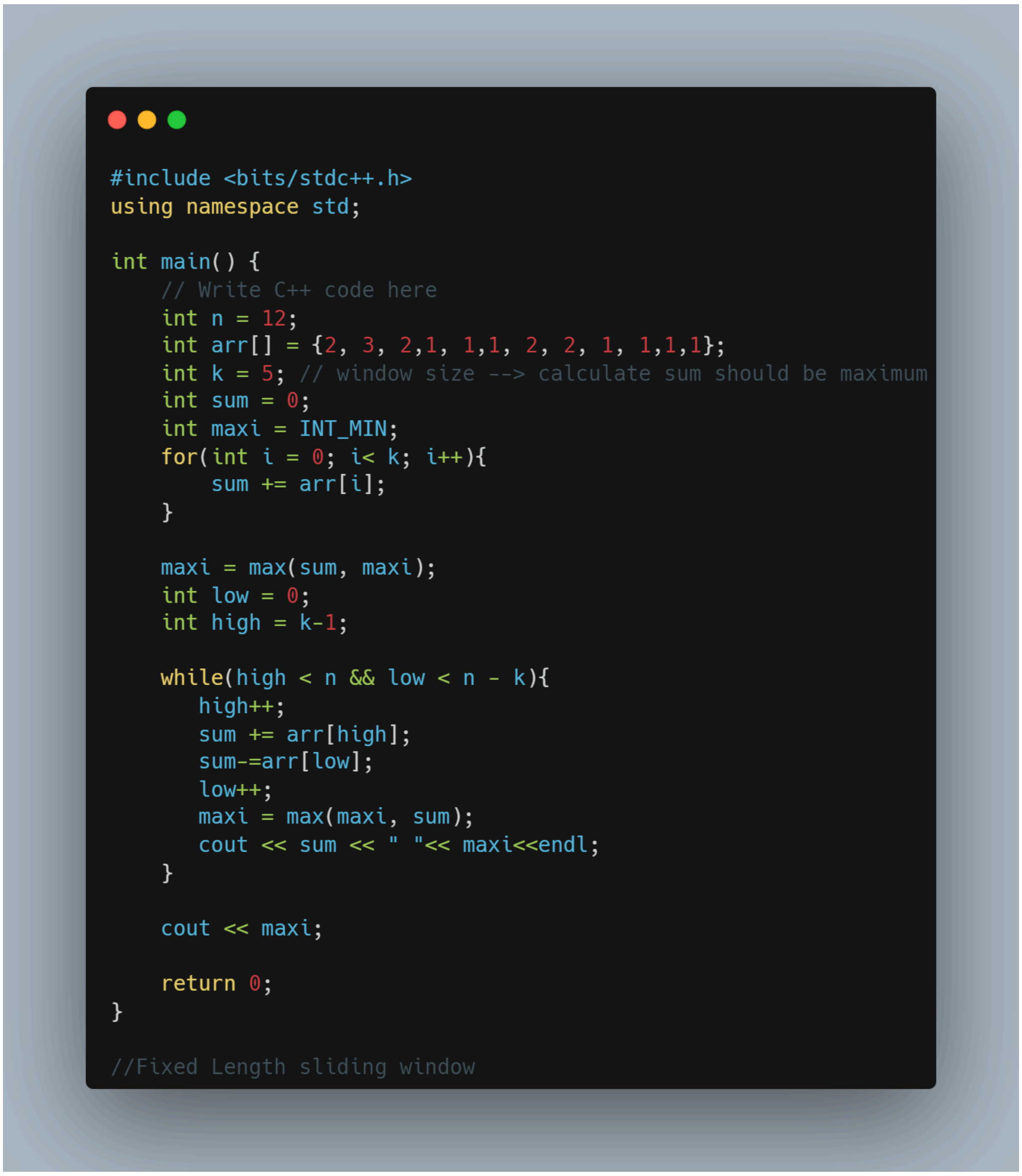
   So conclusion;
   Size of subarray → given;
   We have to maximise → sum;

   Code:

```
// Online C++ compiler to run C++ program online
#include <bits/stdc++.h>
using namespace std;

int main() {
   // Write C++ code here
   int n = 12;
   int arr[] = {2, 3, 2,1, 1,1, 2, 2, 1, 1,1,1};
   int k = 5; // window size --> calculate sum should be maximum
   int sum = 0;
   int maxi = INT_MIN;
   for(int i = 0; i< k; i++){
      sum += arr[i];
   }

   maxi = max(sum, maxi);
   int low = 0;
   int high = k-1;

   while(high < n && low < n - k){
     high++;
```

```
        sum += arr[high];
        sum-=arr[low];
        low++;
        maxi = max(maxi, sum);
        cout << sum << " "<< maxi<<endl;
    }

    cout << maxi;

    return 0;
}
```

Fixed Length sliding window



```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    // Write C++ code here
    int n = 12;
    int arr[] = {2, 3, 2,1, 1,1, 2, 2, 1, 1,1,1};
    int k = 5; // window size --> calculate sum should be maximum
    int sum = 0;
    int maxi = INT_MIN;
    for(int i = 0; i< k; i++){
        sum += arr[i];
    }

    maxi = max(sum, maxi);
    int low = 0;
    int high = k-1;

    while(high < n && low < n - k){
       high++;
       sum += arr[high];
       sum-=arr[low];
       low++;
       maxi = max(maxi, sum);
       cout << sum << " "<< maxi<<endl;
    }

    cout << maxi;

    return 0;
}

//Fixed Length sliding window
```

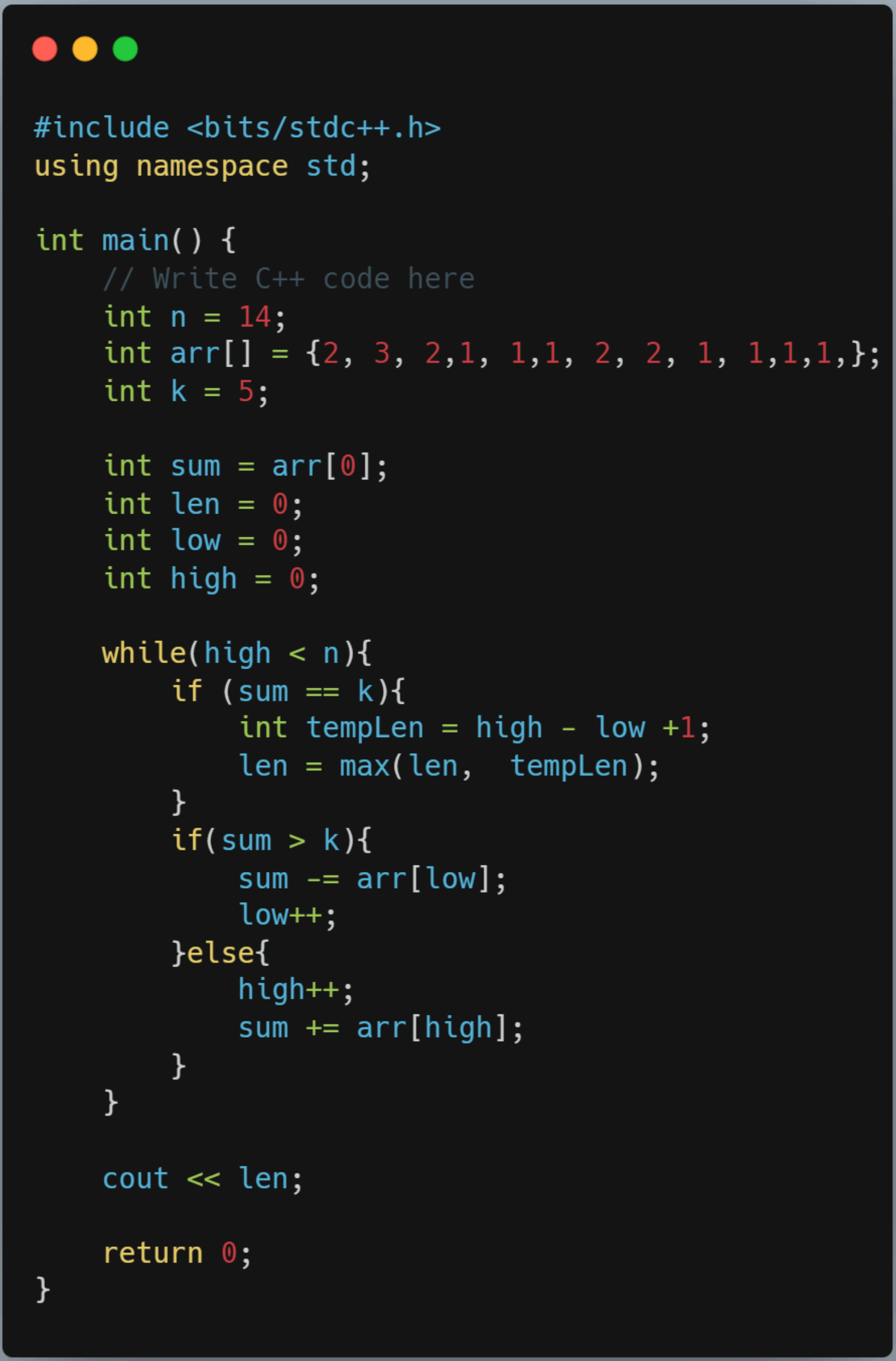
<u>**Variable Length Sliding Window:**</u>

So conclusion;
Size of subarray → maximise;
We have given → sum;

Code:

```
#include <bits/stdc++.h>
using namespace std;

int main() {
   // Write C++ code here
   int n = 14;
   int arr[] = {2, 3, 2,1, 1,1, 2, 2, 1, 1,1,1,};
   int k = 5;

   int sum = arr[0];
   int len = 0;
   int low = 0;
   int high = 0;

   while(high < n){
      if (sum == k){
         int tempLen = high - low +1;
         len = max(len,  tempLen);
      }
      if(sum > k){
         sum -= arr[low];
         low++;
      }else{
         high++;
         sum += arr[high];
      }
   }

   cout << len;

   return 0;
}
```

Variable Length Sliding Window

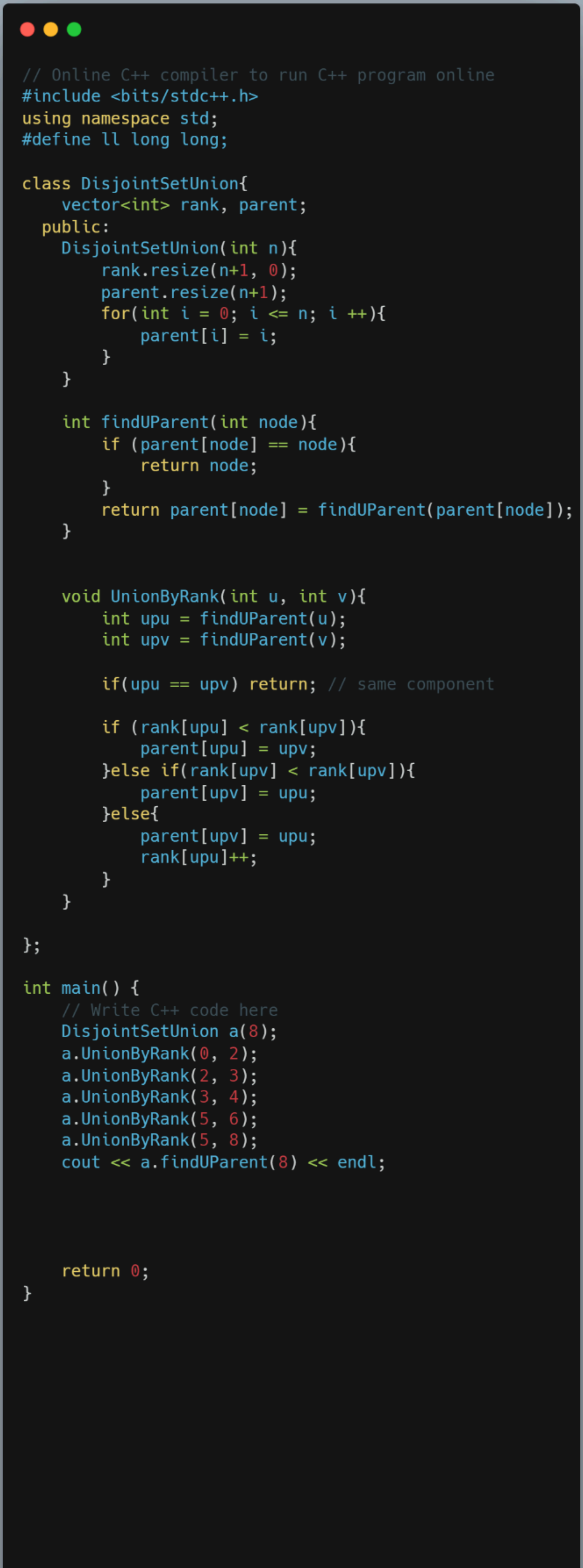


```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    // Write C++ code here
    int n = 14;
    int arr[] = {2, 3, 2,1, 1,1, 2, 2, 1, 1,1,1,};
    int k = 5;

    int sum = arr[0];
    int len = 0;
    int low = 0;
    int high = 0;

    while(high < n){
        if (sum == k){
            int tempLen = high - low +1;
            len = max(len,  tempLen);
        }
        if(sum > k){
            sum -= arr[low];
            low++;
        }else{
            high++;
            sum += arr[high];
        }
    }

    cout << len;

    return 0;
}
```

<u>DSU Data Structure (O(4*Alpha) – TC):</u>
Used for to find parent and make union

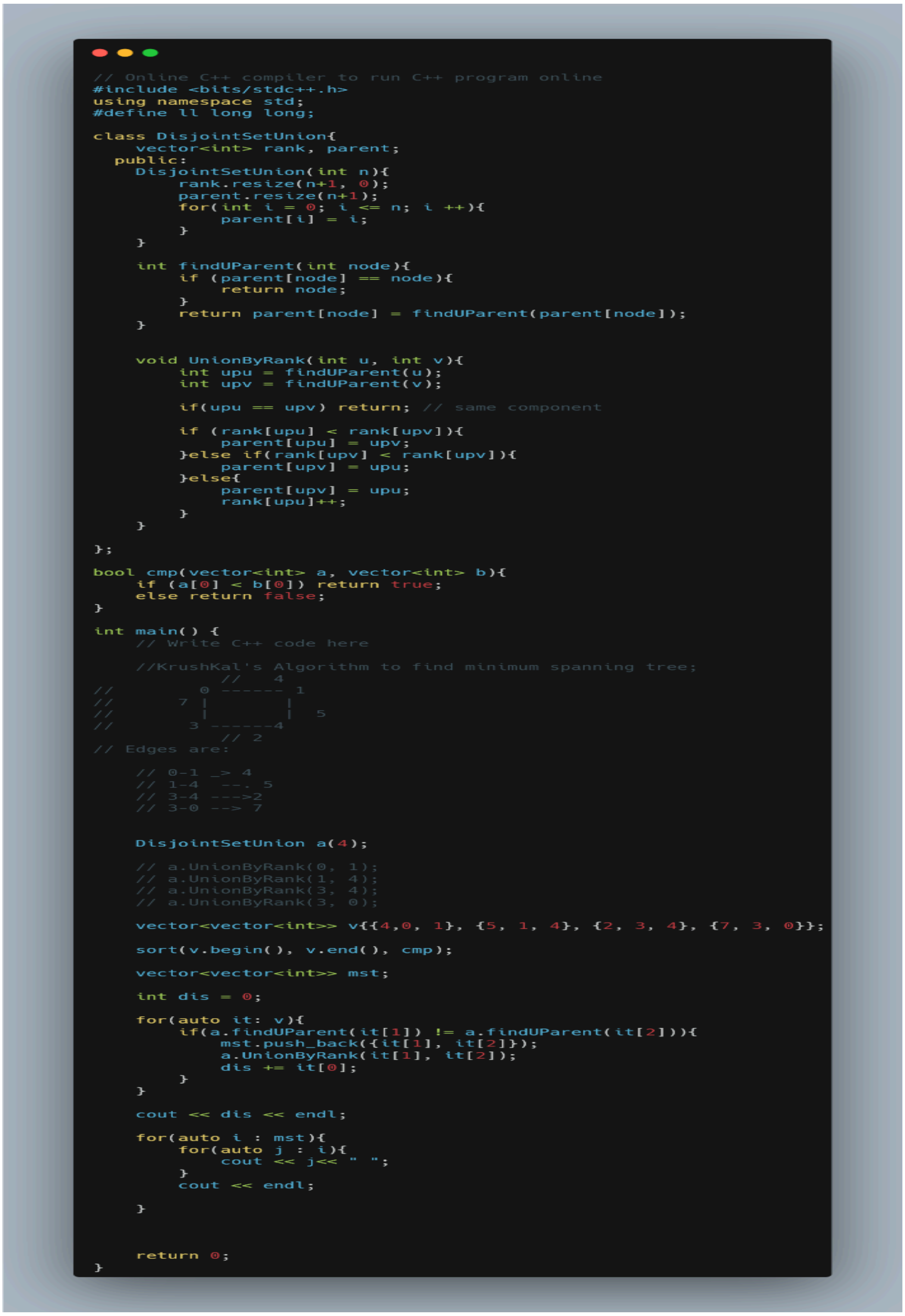
};

```
// Online C++ compiler to run C++ program online
#include <bits/stdc++.h>
using namespace std;
#define ll long long;

class DisjointSetUnion{
    vector<int> rank, parent;
  public:
    DisjointSetUnion(int n){
        rank.resize(n+1, 0);
        parent.resize(n+1);
        for(int i = 0; i <= n; i ++){
            parent[i] = i;
        }
    }

    int findUParent(int node){
        if (parent[node] == node){
            return node;
        }
        return parent[node] = findUParent(parent[node]);
    }


    void UnionByRank(int u, int v){
        int upu = findUParent(u);
        int upv = findUParent(v);

        if(upu == upv) return; // same component

        if (rank[upu] < rank[upv]){
            parent[upu] = upv;
        }else if(rank[upv] < rank[upv]){
            parent[upv] = upu;
        }else{
            parent[upv] = upu;
            rank[upu]++;
        }
    }

};

int main() {
    // Write C++ code here
    DisjointSetUnion a(8);
    a.UnionByRank(0, 2);
    a.UnionByRank(2, 3);
    a.UnionByRank(3, 4);
    a.UnionByRank(5, 6);
    a.UnionByRank(5, 8);
    cout << a.findUParent(8) << endl;




    return 0;
}
```



```cpp
// Online C++ compiler to run C++ program online
#include <bits/stdc++.h>
using namespace std;
#define ll long long;

class DisjointSetUnion{
    vector<int> rank, parent;
  public:
    DisjointSetUnion(int n){
        rank.resize(n+1, 0);
        parent.resize(n+1);
        for(int i = 0; i <= n; i ++){
            parent[i] = i;
        }
    }

    int findUParent(int node){
        if (parent[node] == node){
            return node;
        }
        return parent[node] = findUParent(parent[node]);
    }


    void UnionByRank(int u, int v){
        int upu = findUParent(u);
        int upv = findUParent(v);

        if(upu == upv) return; // same component

        if (rank[upu] < rank[upv]){
            parent[upu] = upv;
        }else if(rank[upv] < rank[upv]){
            parent[upv] = upu;
        }else{
            parent[upv] = upu;
            rank[upu]++;
        }
    }

};

int main() {
    // Write C++ code here
    DisjointSetUnion a(8);
    a.UnionByRank(0, 2);
    a.UnionByRank(2, 3);
    a.UnionByRank(3, 4);
    a.UnionByRank(5, 6);
    a.UnionByRank(5, 8);
    cout << a.findUParent(8) << endl;




    return 0;
}
```

Krushkal’s Algorithm: (Minimum Spanning tree) (E4Aplhs)



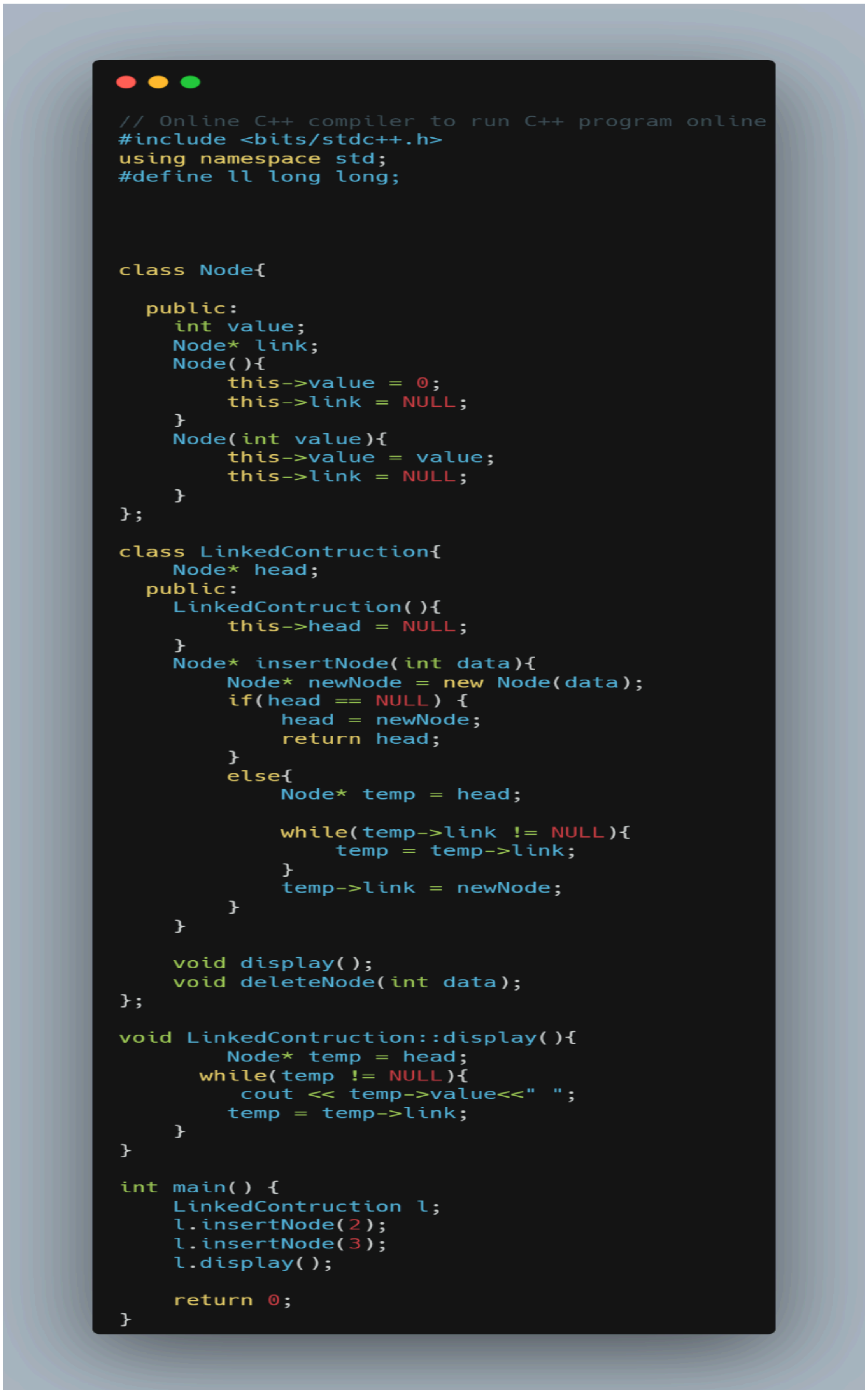
```
// Online C++ compiler to run C++ program online
#include <bits/stdc++.h>
using namespace std;
#define ll long long;

class DisjointSetUnion{
        vector<int> rank, parent;
    public:
        DisjointSetUnion(int n){
            rank.resize(n+1, 0);
            parent.resize(n+1);
            for(int i = 0; i <= n; i ++){
                parent[i] = i;
            }
        }

        int findUParent(int node){
            if (parent[node] == node){
                return node;
            }
            return parent[node] = findUParent(parent[node]);
        }


        void UnionByRank(int u, int v){
            int upu = findUParent(u);
            int upv = findUParent(v);

            if(upu == upv) return; // same component

            if (rank[upu] < rank[upv]){
                parent[upu] = upv;
            }else if(rank[upv] < rank[upv]){
                parent[upv] = upu;
            }else{
                parent[upv] = upu;
                rank[upu]++;
            }
        }
};

bool cmp(vector<int> a, vector<int> b){
        if (a[0] < b[0]) return true;
        else return false;
}

int main() {
        // Write C++ code here

        //KrushKal's Algorithm to find minimum spanning tree;
                //    4
//          0 ------- 1
//        7 |         |
//          |         |   5
//         3 -------4
                // 2
// Edges are:

        // 0-1 _> 4
        // 1-4  --. 5
        // 3-4 --->2
        // 3-0 --> 7


        DisjointSetUnion a(4);

        // a.UnionByRank(0, 1);
        // a.UnionByRank(1, 4);
        // a.UnionByRank(3, 4);
        // a.UnionByRank(3, 0);

        vector<vector<int>> v{{4,0, 1}, {5, 1, 4}, {2, 3, 4}, {7, 3, 0}};

        sort(v.begin(), v.end(), cmp);

        vector<vector<int>> mst;

        int dis = 0;

        for(auto it: v){
            if(a.findUParent(it[1]) != a.findUParent(it[2])){
                mst.push_back({it[1], it[2]});
                a.UnionByRank(it[1], it[2]);
                dis += it[0];
            }
        }

        cout << dis << endl;

        for(auto i : mst){
            for(auto j : i){
                cout << j<< " ";
            }
            cout << endl;

        }


        return 0;
}
```

Code:

```
// Online C++ compiler to run C++ program online
#include <bits/stdc++.h>
using namespace std;
#define ll long long;

class DisjointSetUnion{
    vector<int> rank, parent;
  public:
    DisjointSetUnion(int n){
        rank.resize(n+1, 0);
        parent.resize(n+1);
        for(int i = 0; i <= n; i ++){
            parent[i] = i;
        }
    }

    int findUParent(int node){
        if (parent[node] == node){
            return node;
        }
        return parent[node] = findUParent(parent[node]);
    }


    void UnionByRank(int u, int v){
        int upu = findUParent(u);
        int upv = findUParent(v);

        if(upu == upv) return; // same component

        if (rank[upu] < rank[upv]){
            parent[upu] = upv;
        }else if(rank[upv] < rank[upv]){
            parent[upv] = upu;
        }else{
            parent[upv] = upu;
            rank[upu]++;
        }
    }

};
```

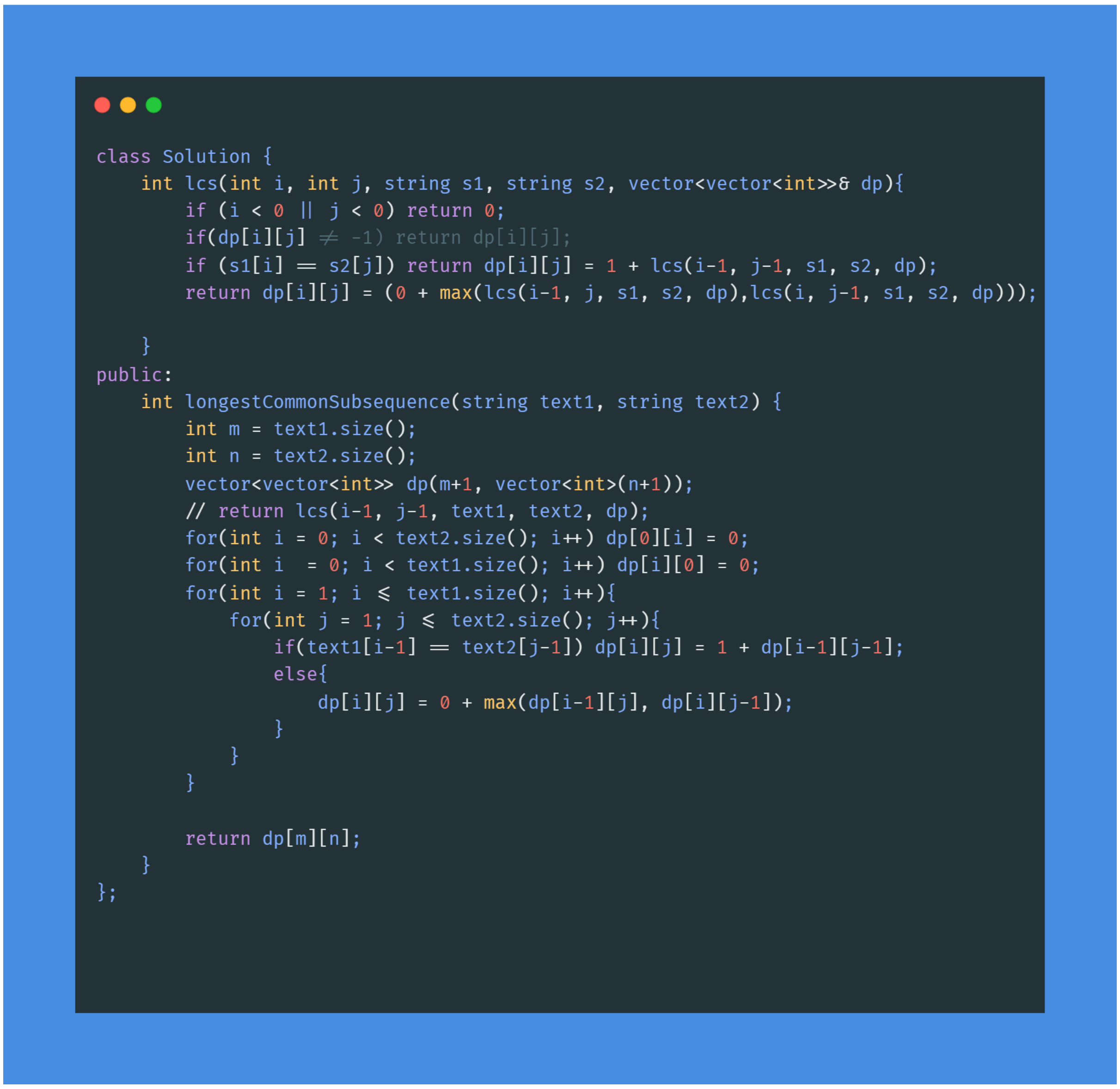
```
bool cmp(vector<int> a, vector<int> b){
    if (a[0] < b[0]) return true;
    else return false;
}

int main() {
    // Write C++ code here

    //KrushKal's Algorithm to find minimum spanning tree;
            //   4
//        0 ------ 1
//      7 |        |
//        |        |  5
//       3 ------4
            // 2
// Edges are:

    // 0-1 _> 4
    // 1-4  --. 5
    // 3-4 --->2
    // 3-0 --> 7


    DisjointSetUnion a(4);

    // a.UnionByRank(0, 1);
    // a.UnionByRank(1, 4);
    // a.UnionByRank(3, 4);
    // a.UnionByRank(3, 0);

    vector<vector<int>> v{{4,0, 1}, {5, 1, 4}, {2, 3, 4}, {7, 3, 0}};

    sort(v.begin(), v.end(), cmp);

    vector<vector<int>> mst;

    int dis = 0;

    for(auto it: v){
        if(a.findUParent(it[1]) != a.findUParent(it[2])){
            mst.push_back({it[1], it[2]});
            a.UnionByRank(it[1], it[2]);
            dis += it[0];
        }
    }

    cout << dis << endl;

    for(auto i : mst){
        for(auto j : i){
            cout << j<< " ";
        }
        cout << endl;

    }



    return 0;
}
```

Linked List:



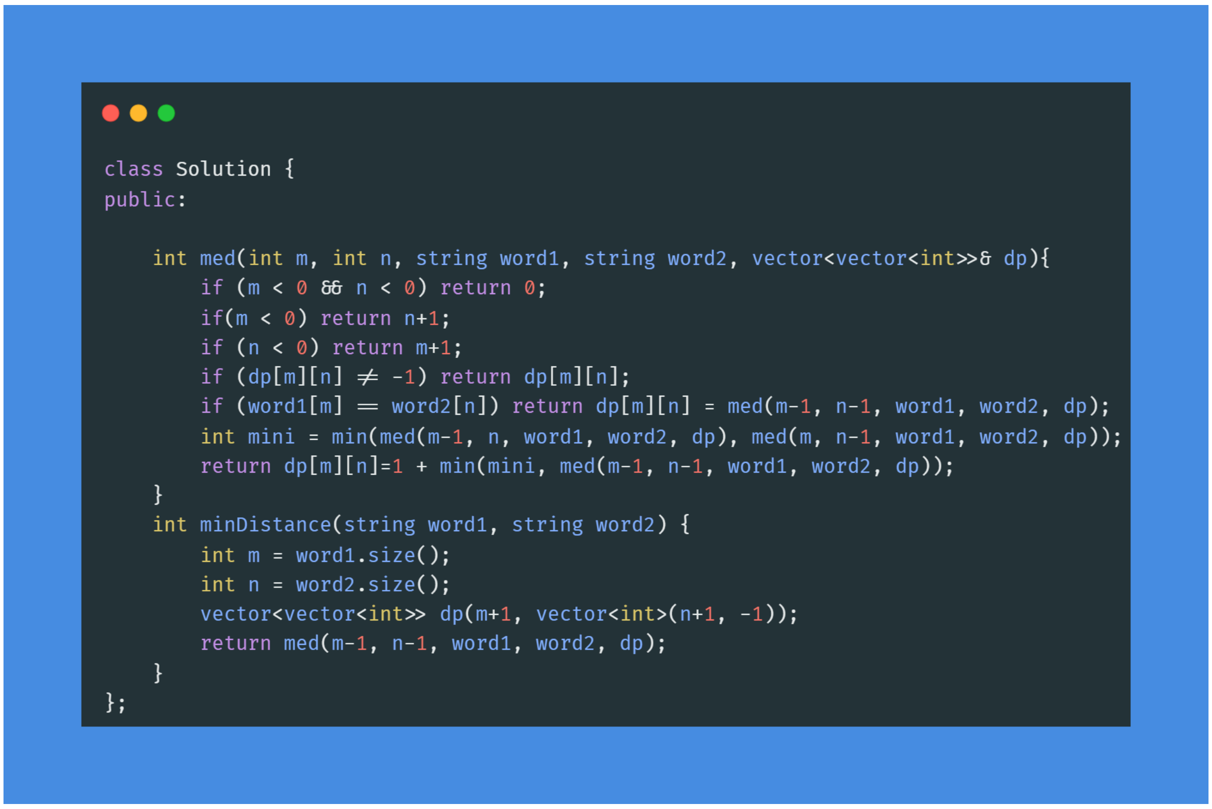
```cpp
// Online C++ compiler to run C++ program online
#include <bits/stdc++.h>
using namespace std;
#define ll long long;



class Node{

  public:
    int value;
    Node* link;
    Node(){
        this->value = 0;
        this->link = NULL;
    }
    Node(int value){
        this->value = value;
        this->link = NULL;
    }
};

class LinkedContruction{
    Node* head;
  public:
    LinkedContruction(){
        this->head = NULL;
    }
    Node* insertNode(int data){
        Node* newNode = new Node(data);
        if(head == NULL) {
            head = newNode;
            return head;
        }
        else{
            Node* temp = head;

            while(temp->link != NULL){
                temp = temp->link;
            }
            temp->link = newNode;
        }
    }

    void display();
    void deleteNode(int data);
};

void LinkedContruction::display(){
        Node* temp = head;
      while(temp != NULL){
         cout << temp->value<<" ";
        temp = temp->link;
    }
}

int main() {
    LinkedContruction l;
    l.insertNode(2);
    l.insertNode(3);
    l.display();

    return 0;
}
```

Longest Common Subsequence DP



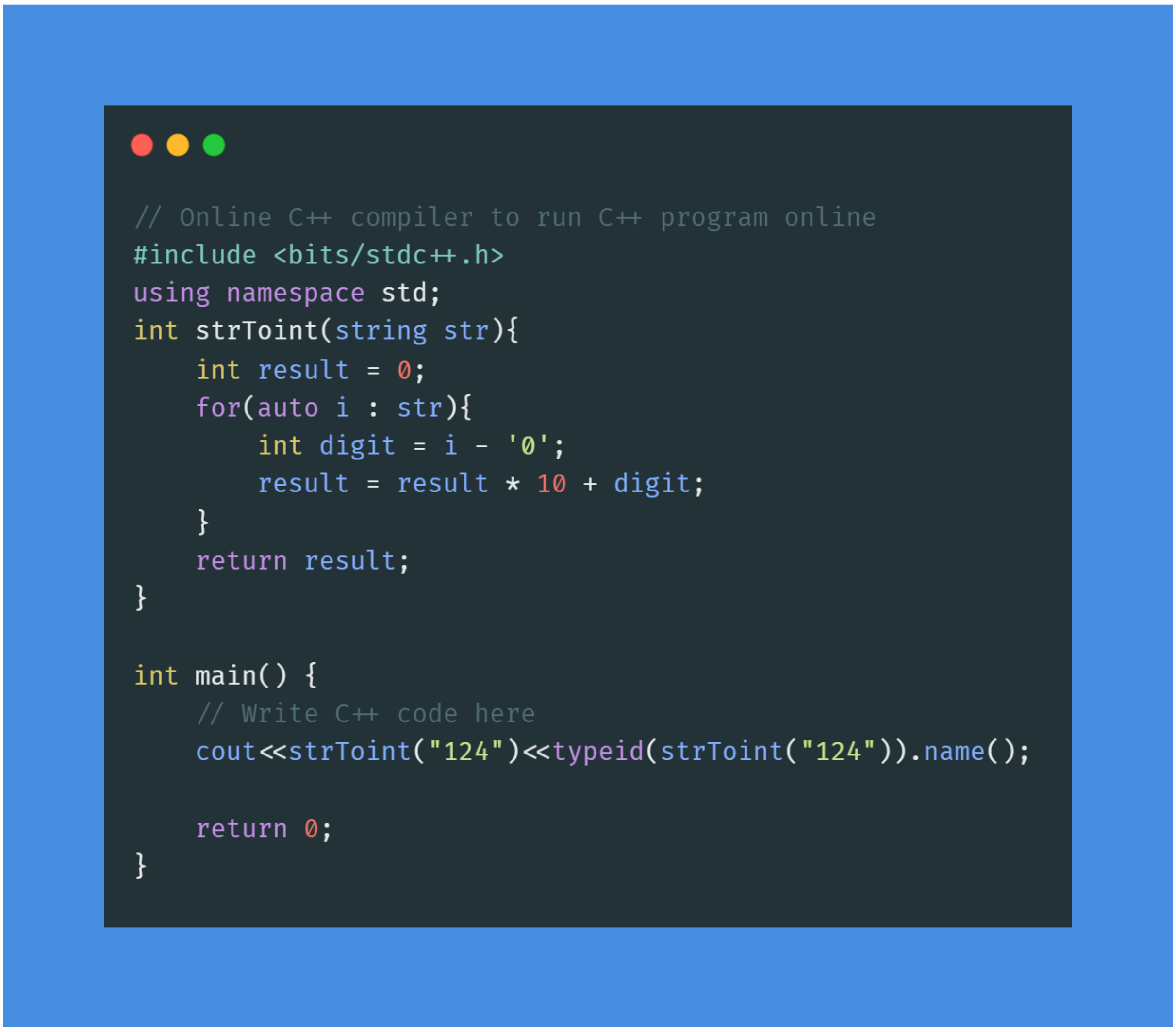
```
class Solution {
    int lcs(int i, int j, string s1, string s2, vector<vector<int>>& dp){
        if (i < 0 || j < 0) return 0;
        if(dp[i][j] != -1) return dp[i][j];
        if (s1[i] == s2[j]) return dp[i][j] = 1 + lcs(i-1, j-1, s1, s2, dp);
        return dp[i][j] = (0 + max(lcs(i-1, j, s1, s2, dp),lcs(i, j-1, s1, s2, dp)));

    }
public:
    int longestCommonSubsequence(string text1, string text2) {
        int m = text1.size();
        int n = text2.size();
        vector<vector<int>> dp(m+1, vector<int>(n+1));
        // return lcs(i-1, j-1, text1, text2, dp);
        for(int i = 0; i < text2.size(); i++) dp[0][i] = 0;
        for(int i  = 0; i < text1.size(); i++) dp[i][0] = 0;
        for(int i = 1; i <= text1.size(); i++){
            for(int j = 1; j <= text2.size(); j++){
                if(text1[i-1] == text2[j-1]) dp[i][j] = 1 + dp[i-1][j-1];
                else{
                    dp[i][j] = 0 + max(dp[i-1][j], dp[i][j-1]);
                }
            }
        }

        return dp[m][n];
    }
};
```

Minimum Edit Distance



```cpp
class Solution {
public:

    int med(int m, int n, string word1, string word2, vector<vector<int>>& dp){
        if (m < 0 && n < 0) return 0;
        if(m < 0) return n+1;
        if (n < 0) return m+1;
        if (dp[m][n] != -1) return dp[m][n];
        if (word1[m] == word2[n]) return dp[m][n] = med(m-1, n-1, word1, word2, dp);
        int mini = min(med(m-1, n, word1, word2, dp), med(m, n-1, word1, word2, dp));
        return dp[m][n]=1 + min(mini, med(m-1, n-1, word1, word2, dp));
    }
    int minDistance(string word1, string word2) {
        int m = word1.size();
        int n = word2.size();
        vector<vector<int>> dp(m+1, vector<int>(n+1, -1));
        return med(m-1, n-1, word1, word2, dp);
    }
};
```

String to int



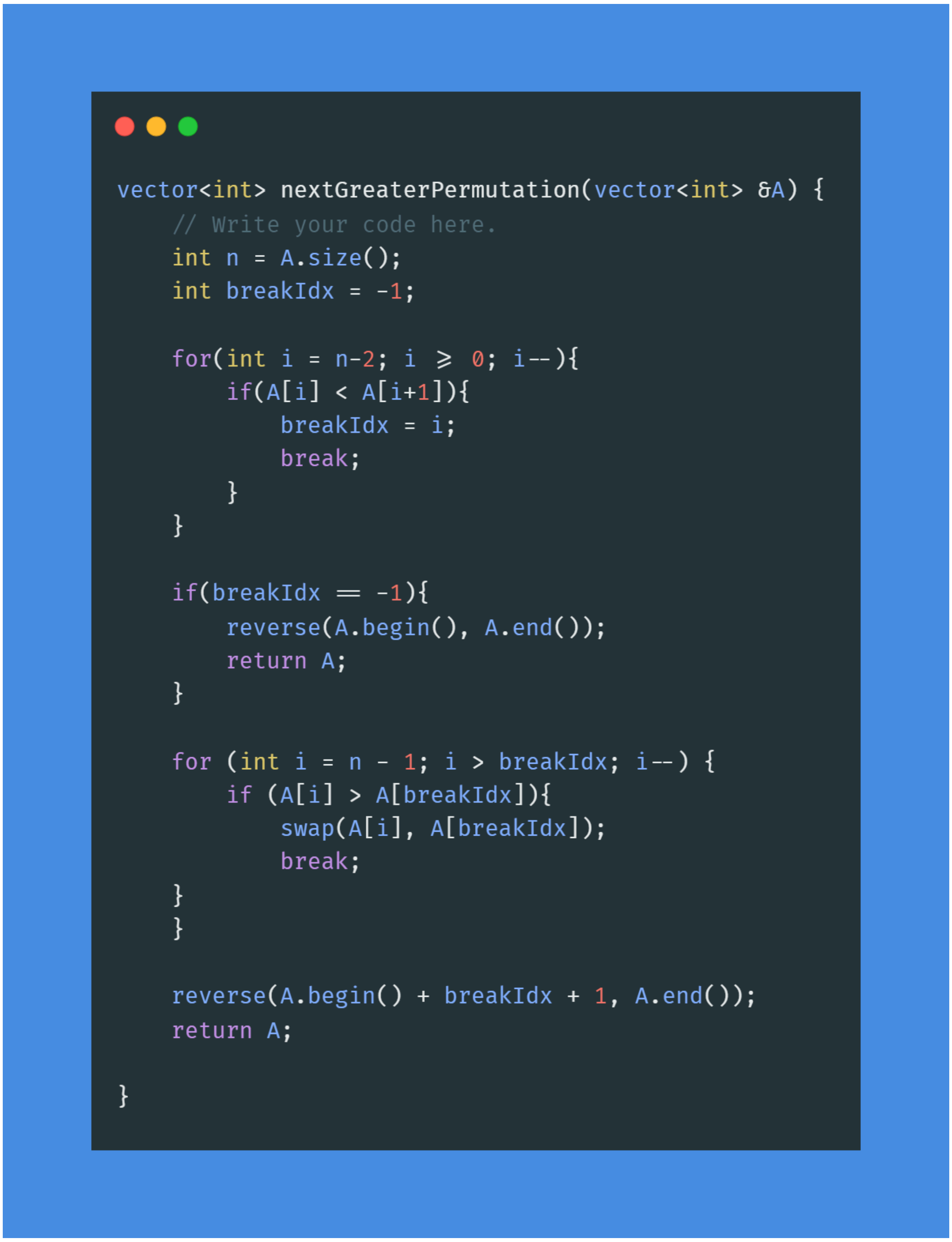
```cpp
// Online C++ compiler to run C++ program online
#include <bits/stdc++.h>
using namespace std;
int strToint(string str){
    int result = 0;
    for(auto i : str){
        int digit = i - '0';
        result = result * 10 + digit;
    }
    return result;
}

int main() {
    // Write C++ code here
    cout<<strToint("124")<<typeid(strToint("124")).name();

    return 0;
}
```

Next Permutation

Algorithm:

1. **Find the break point means point where sudden decrease occur let say (1 , 2, 5, 4, 3, 2, 1) so here 5, 4, 3, 2, 1, are in increasing order but after 5 sudden decrease occurred (To get maximum prefix point)**
2. **Find out Just greater element from that break point to end (reverse iterate as it sorted) and swap**
3. **Reverse from breakpoint + 1 to end.**

```
vector<int> nextGreaterPermutation(vector<int> &A) {
    // Write your code here.
    int n = A.size();
    int breakIdx = -1;

    for(int i = n-2; i >= 0; i--){
        if(A[i] < A[i+1]){
            breakIdx = i;
            break;
        }
    }

    if(breakIdx == -1){
        reverse(A.begin(), A.end());
        return A;
    }

    for (int i = n - 1; i > breakIdx; i--) {
        if (A[i] > A[breakIdx]){
            swap(A[i], A[breakIdx]);
            break;
    }
    }

    reverse(A.begin() + breakIdx + 1, A.end());
    return A;

}
```



```cpp
vector<int> nextGreaterPermutation(vector<int> &A) {
    // Write your code here.
    int n = A.size();
    int breakIdx = -1;

    for(int i = n-2; i >= 0; i--){
        if(A[i] < A[i+1]){
            breakIdx = i;
            break;
        }
    }

    if(breakIdx == -1){
        reverse(A.begin(), A.end());
        return A;
    }

    for (int i = n - 1; i > breakIdx; i--) {
        if (A[i] > A[breakIdx]){
            swap(A[i], A[breakIdx]);
            break;
    }
    }

    reverse(A.begin() + breakIdx + 1, A.end());
    return A;

}
```

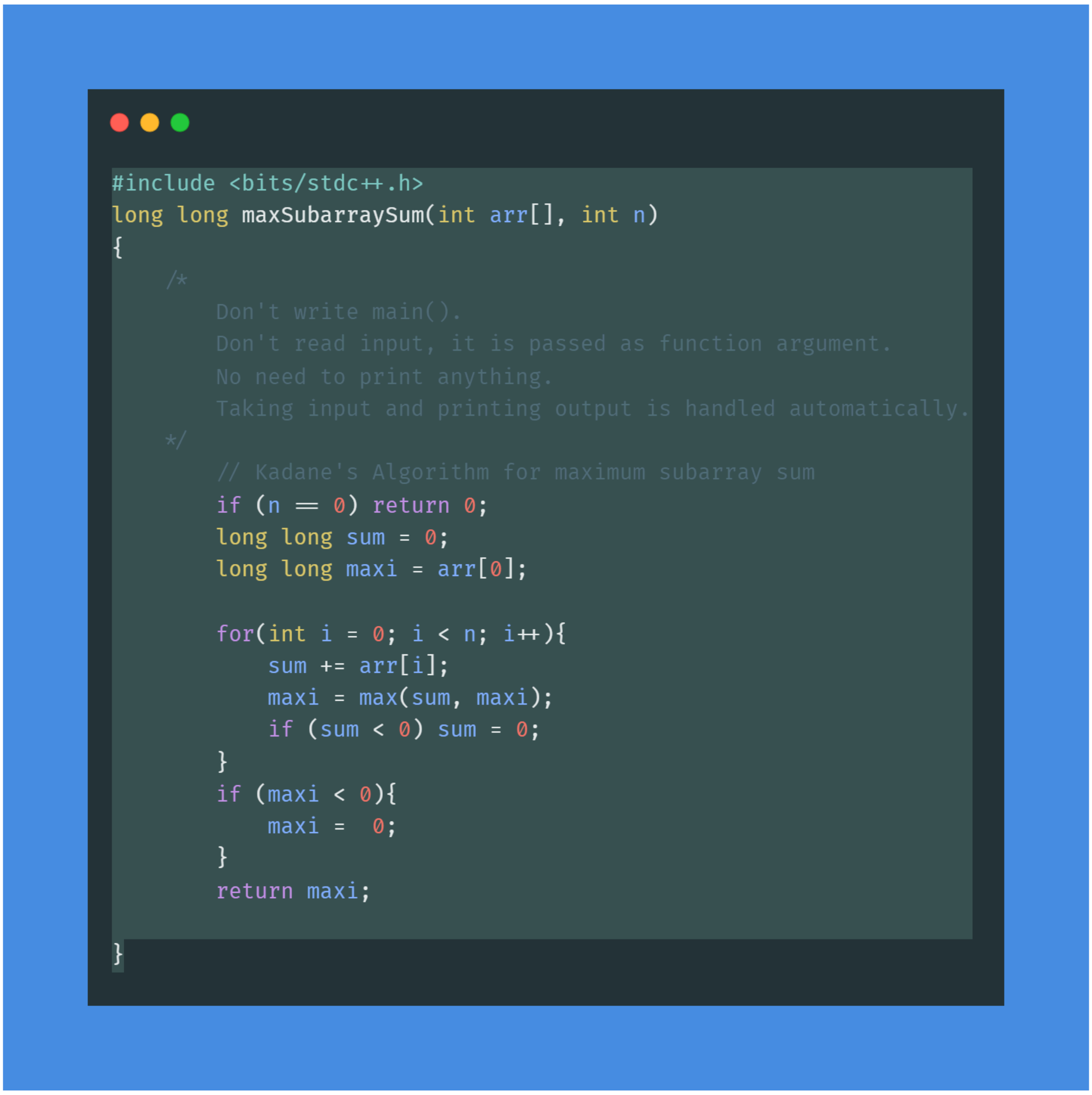
Kadane’s Algorithm

To Find out maximum subarray sum in negative value containing array with handling empty subarray

Main thing → sum < 0 then sum = 0;

Code:

```
#include <bits/stdc++.h>
long long maxSubarraySum(int arr[], int n)
{
    /*
        Don't write main().
        Don't read input, it is passed as function argument.
        No need to print anything.
        Taking input and printing output is handled automatically.
    */
        if (n == 0) return 0;
        long long sum = 0;
        long long maxi = arr[0];

        for(int i = 0; i < n; i++){
            sum += arr[i];
            maxi = max(sum, maxi);
            if (sum < 0) sum = 0;
        }
        if (maxi < 0){
            maxi =  0;
        }
        return maxi;

}
```



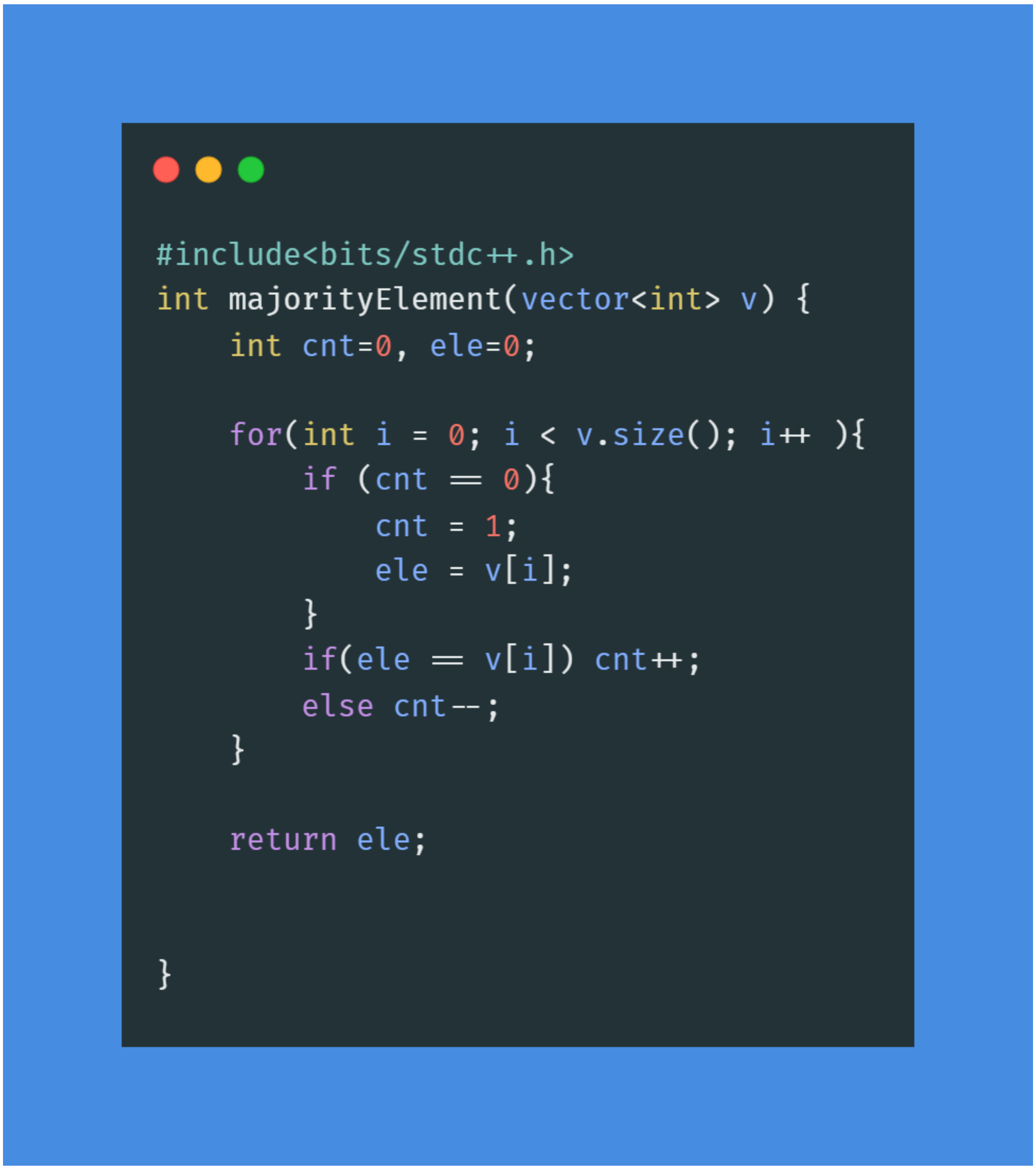
```cpp
#include <bits/stdc++.h>
long long maxSubarraySum(int arr[], int n)
{
    /*
        Don't write main().
        Don't read input, it is passed as function argument.
        No need to print anything.
        Taking input and printing output is handled automatically.
    */
        // Kadane's Algorithm for maximum subarray sum
        if (n == 0) return 0;
        long long sum = 0;
        long long maxi = arr[0];

        for(int i = 0; i < n; i++){
            sum += arr[i];
            maxi = max(sum, maxi);
            if (sum < 0) sum = 0;
        }
        if (maxi < 0){
            maxi =  0;
        }
        return maxi;

}
```

More’s Algorithm

More’s Algorithm for Majority Element among array element when we definitely know there is elements more than n/2 which is major

Code:

```
#include<bits/stdc++.h>
int majorityElement(vector<int> v) {
    int cnt=0, ele=0;

    for(int i = 0; i < v.size(); i++ ){
        if (cnt == 0){
            cnt = 1;
            ele = v[i];
        }
        if(ele == v[i]) cnt++;
        else cnt--;
    }

    return ele;


}
```



```cpp
#include<bits/stdc++.h>
int majorityElement(vector<int> v) {
    int cnt=0, ele=0;

    for(int i = 0; i < v.size(); i++ ){
        if (cnt == 0){
            cnt = 1;
            ele = v[i];
        }
        if(ele == v[i]) cnt++;
        else cnt--;
    }

    return ele;


}
```

Pascal’s Triangle

E.g

```
                    1
                 1     1
              1     2     1
           1     3     3     1
        1     4     6     4     1
     1     5     10    10    5     1
```

For n = 6 (row)

1. To find any value at any specific row and col like we have to find 6 then formula will be

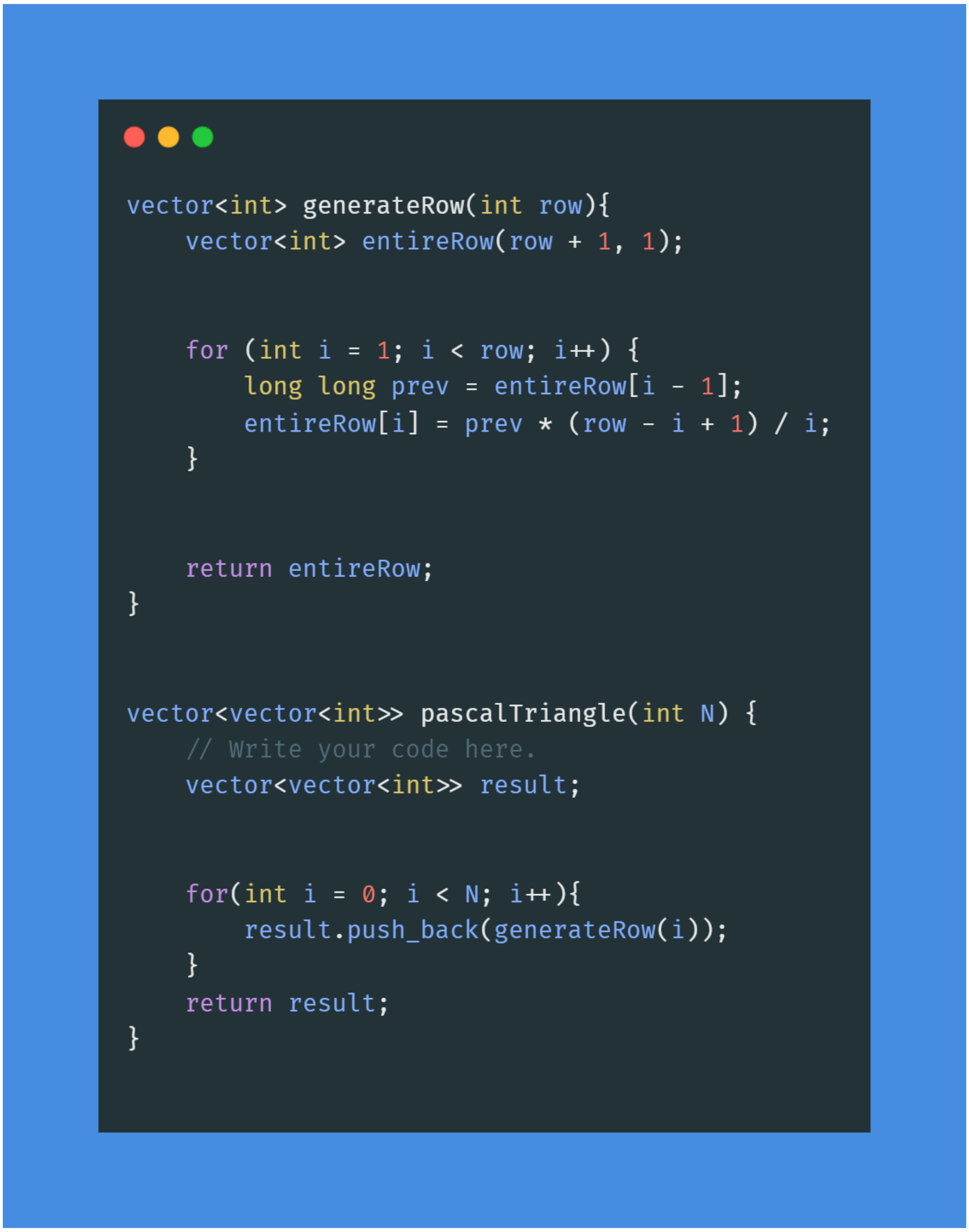
   $$^{n-1}C_{r-1} = \frac{(n-1)!}{(r-1)!(n-1-r+1)!}$$

Where n = row and r = col

Code: To Print entire pascal triangle

```
vector<int> generateRow(int row){
    vector<int> entireRow(row + 1, 1);

    for (int i = 1; i < row; i++) {
        long long prev = entireRow[i - 1];
        entireRow[i] = prev * (row - i + 1) / i;
    }

    return entireRow;
}

vector<vector<int>> pascalTriangle(int N) {
    // Write your code here.
    vector<vector<int>> result;

    for(int i = 0; i < N; i++){
        result.push_back(generateRow(i));
    }
    return result;
}
```

To print row fomula is : previous * (row - 1 - col)/col



```
vector<int> generateRow(int row){
    vector<int> entireRow(row + 1, 1);


    for (int i = 1; i < row; i++) {
        long long prev = entireRow[i - 1];
        entireRow[i] = prev * (row - i + 1) / i;
    }


    return entireRow;
}


vector<vector<int>> pascalTriangle(int N) {
    // Write your code here.
    vector<vector<int>> result;


    for(int i = 0; i < N; i++){
        result.push_back(generateRow(i));
    }
    return result;
}
```

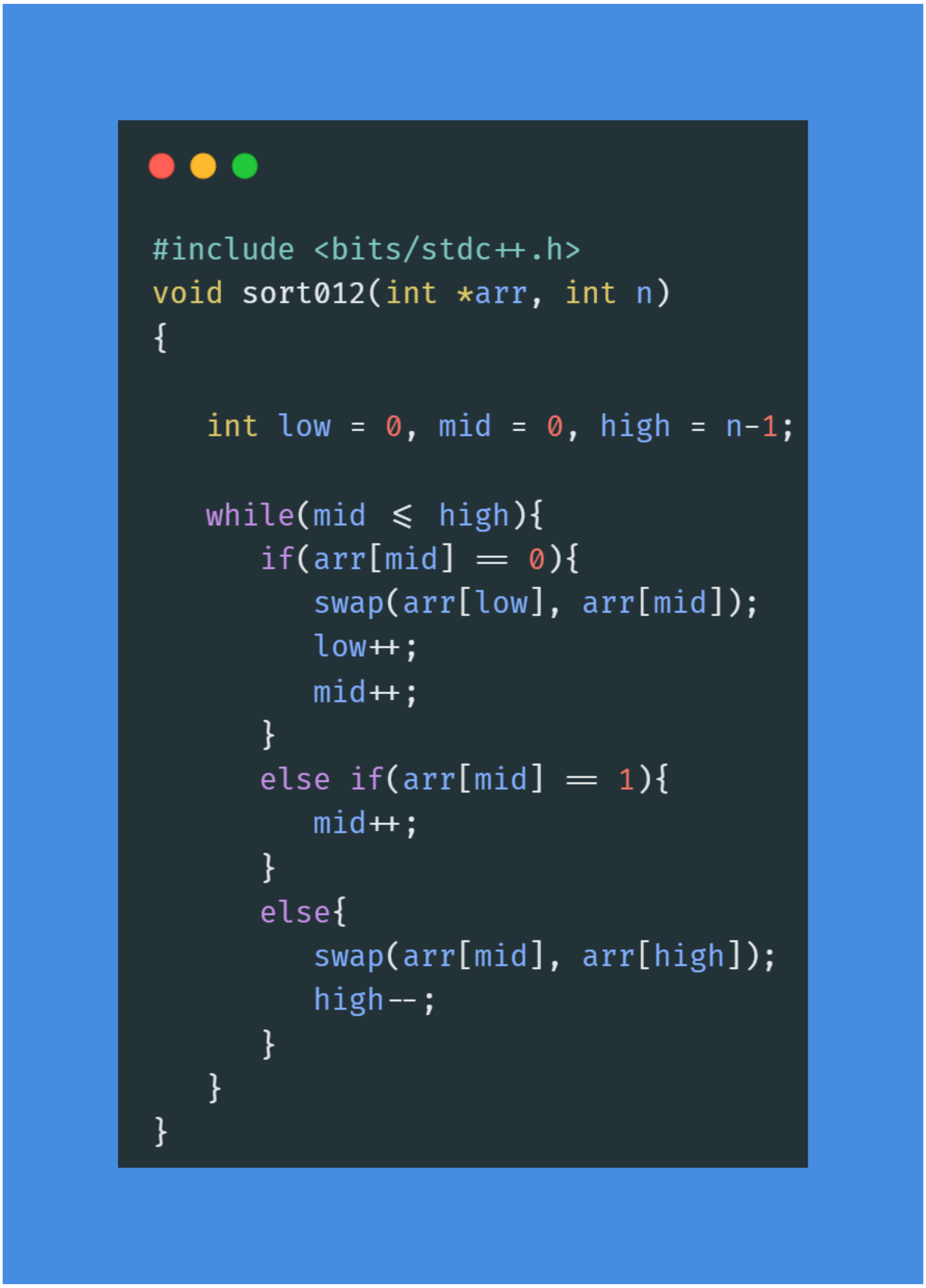
Dutch National flag algorithm → 0, 1, 2 array sort

0 to low-1 | low to mid-1 | mid to high | high +1 to   n-1
0000000 | 111111111111 | unsorted(0/2/1)| 2 22 2 2 2 2

Without DNF

```
int cnt0 = 0;
   int cnt1 = 0;
   int cnt2 = 0;

   for(int i = 0; i < n; i++){
      if (arr[i] == 0) cnt0++;
      if (arr[i] == 1) cnt1++;
      if(arr[i] == 2) cnt2++;
   }

   for(int i = 0; i < n; i++){
      if(i < cnt0) arr[i] = 0;
      else if(i >= cnt0 & i < cnt0 + cnt1) arr[i] = 1;
      else arr[i] = 2;
   }
```

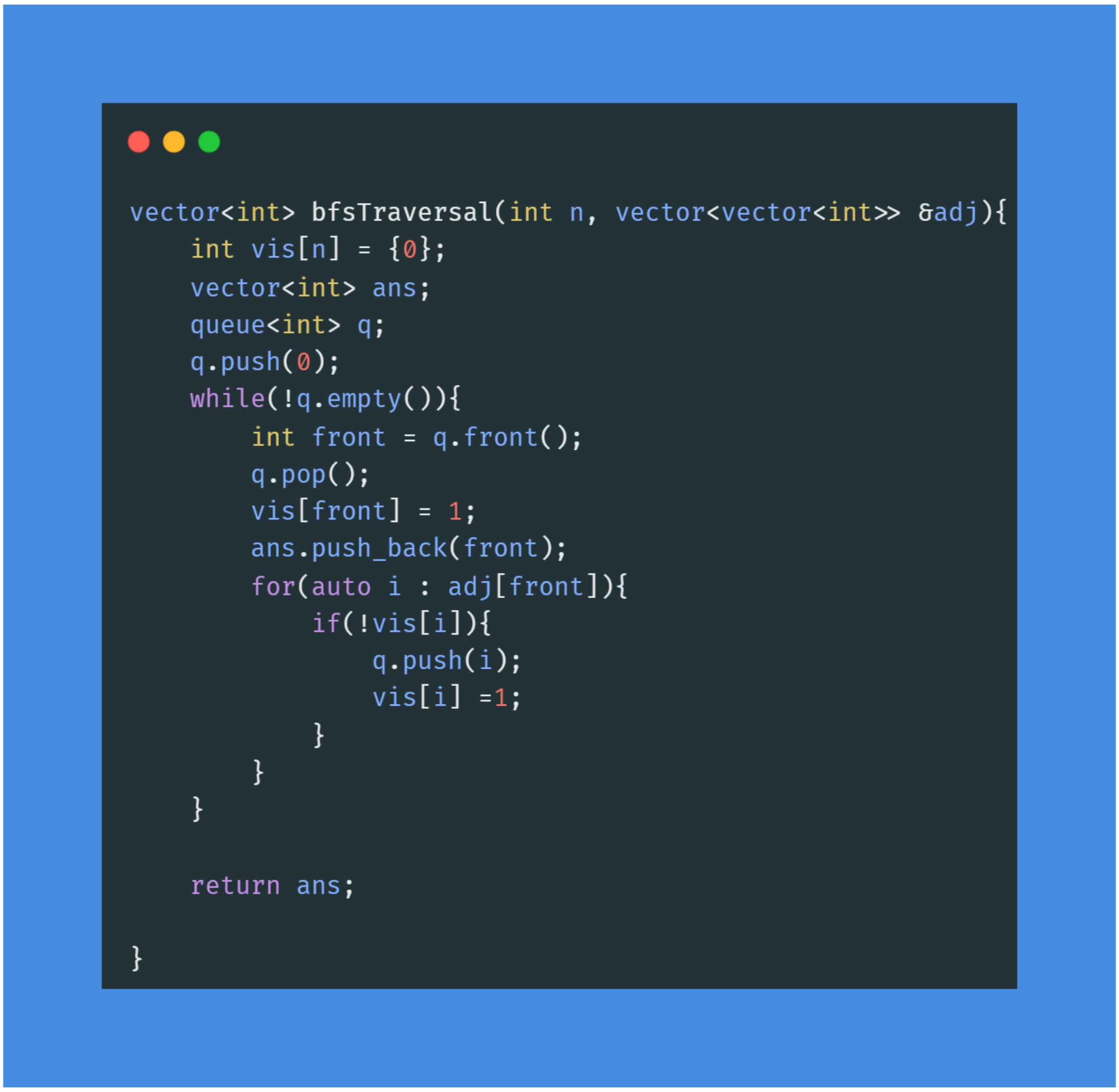
DNF algo



```cpp
#include <bits/stdc++.h>
void sort012(int *arr, int n)
{

    int low = 0, mid = 0, high = n-1;

    while(mid <= high){
        if(arr[mid] == 0){
            swap(arr[low], arr[mid]);
            low++;
            mid++;
        }
        else if(arr[mid] == 1){
            mid++;
        }
        else{
            swap(arr[mid], arr[high]);
            high--;
        }
    }
}
```

BFS

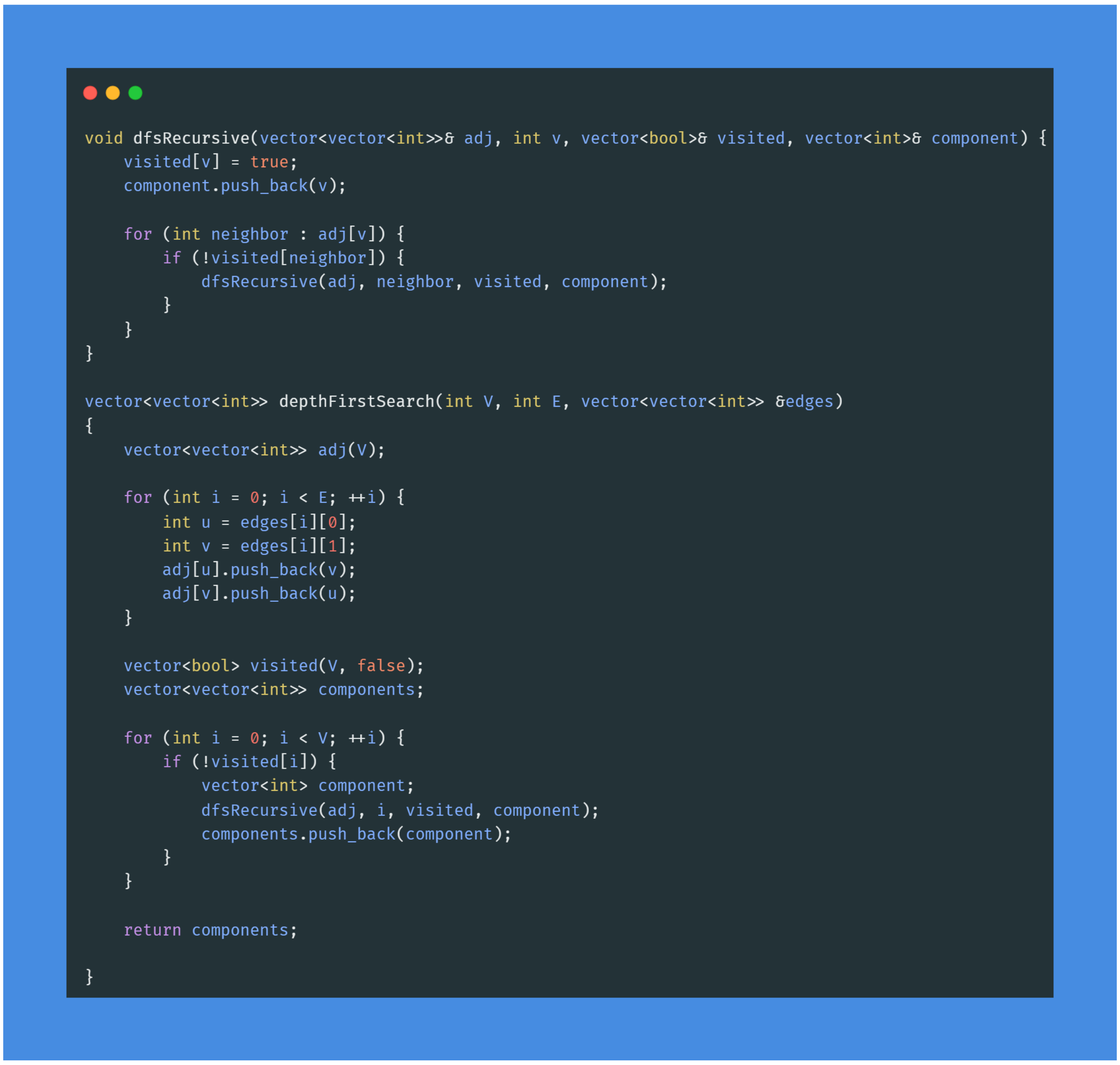
```
vector<int> bfsTraversal(int n, vector<vector<int>> &adj){
    int vis[n] = {0};
    vector<int> ans;
    queue<int> q;
    q.push(0);
    while(!q.empty()){
        int front = q.front();
        q.pop();
        vis[front] = 1;
        ans.push_back(front);
        for(auto i : adj[front]){
            if(!vis[i]){
                q.push(i);
                vis[i] =1;
            }
        }
    }

    return ans;

}
```



```cpp
vector<int> bfsTraversal(int n, vector<vector<int>> &adj){
    int vis[n] = {0};
    vector<int> ans;
    queue<int> q;
    q.push(0);
    while(!q.empty()){
        int front = q.front();
        q.pop();
        vis[front] = 1;
        ans.push_back(front);
        for(auto i : adj[front]){
            if(!vis[i]){
                q.push(i);
                vis[i] =1;
            }
        }
    }

    return ans;

}
```

DFS

```cpp
void dfsRecursive(vector<vector<int>>& adj, int v, vector<bool>&
visited, vector<int>& component) {
    visited[v] = true;
    component.push_back(v);

    for (int neighbor : adj[v]) {
        if (!visited[neighbor]) {
            dfsRecursive(adj, neighbor, visited, component);
        }
    }
}

vector<vector<int>> depthFirstSearch(int V, int E, vector<vector<int>>
&edges)
{
    vector<vector<int>> adj(V);

    for (int i = 0; i < E; ++i) {
        int u = edges[i][0];
        int v = edges[i][1];
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    vector<bool> visited(V, false);
    vector<vector<int>> components;

    for (int i = 0; i < V; ++i) {
        if (!visited[i]) {
            vector<int> component;
            dfsRecursive(adj, i, visited, component);
            components.push_back(component);
        }
    }

    return components;

}
```



```cpp
void dfsRecursive(vector<vector<int>>& adj, int v, vector<bool>& visited, vector<int>& component) {
    visited[v] = true;
    component.push_back(v);

    for (int neighbor : adj[v]) {
        if (!visited[neighbor]) {
            dfsRecursive(adj, neighbor, visited, component);
        }
    }
}

vector<vector<int>> depthFirstSearch(int V, int E, vector<vector<int>> &edges)
{
    vector<vector<int>> adj(V);

    for (int i = 0; i < E; ++i) {
        int u = edges[i][0];
        int v = edges[i][1];
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    vector<bool> visited(V, false);
    vector<vector<int>> components;

    for (int i = 0; i < V; ++i) {
        if (!visited[i]) {
            vector<int> component;
            dfsRecursive(adj, i, visited, component);
            components.push_back(component);
        }
    }

    return components;

}
```

pow(x, y) using binary exponentiation (O(log2(n)) complexity

```
double myPow(double x, int n) {

    // Binary Exponentiation method O(lon2n)
    long long ntemp = n;
    if(ntemp < 0) ntemp = -1 * ntemp;

    double  ans = 1.0;

    while(ntemp){
        if(ntemp % 2){
            ans = ans * x;
            ntemp -= 1;
        }else{
            x *= x;
            ntemp = ntemp/2;
        }
    }

    if(n < 0) ans = 1/ ans;
    return ans;

}
```

Patterns of recursive calls;

1. Print all the subsequences

[1, 2, 3]

```
Base case (n == 0) {
        Print subsequence;
        return
}

take element in ans;
Call for next element
Not take just call for next element;
```

2. 2nd pattern:

Print any one subsequence

```
Base case: n == 0
        If condition satisfied return true
        Otherwise return false;

if(call fuction fo take ele == true) return true;
if(call function for not take ele == true) return true;

return false;
```

3. Count all subsequences with some condition

```
Base case:
        If condition satisfied return 1;
        Else return 0;

Left  = call for take case;
Right = call for not take case;
Return left + right;
```

If multiple recursive call

```
Then
S = 0;
for(int i -> n)
        S += f()

Return s;
```

Return Subsets Sum to K

Code 👍

```
void generateSubset(vector<int> &arr, vector<vector<int>> &ans,
vector<int> &currentSubset, int i, int sum, int n){
    //Base case
    if(i == n){
        if(sum == 0){
            // sort(currentSubset.begin(), currentSubset.end());
            ans.push_back(currentSubset);
        }
        return;
    }

    currentSubset.push_back(arr[i]);
    generateSubset(arr, ans, currentSubset, i+1, sum - arr[i], n);
    currentSubset.pop_back();
    //Not Take case
    generateSubset(arr, ans, currentSubset, i+1, sum, n);
}

vector<vector<int>> findSubsetsThatSumToK(vector<int> arr, int n, int
k)
{
    // Write your code here.
    vector<vector<int>> ans;
    vector<int> currentSubset;
    generateSubset(arr, ans, currentSubset, 0, k, n);
    return ans;
}
```

Combination Sum II

Combination Sum I

Given an array of distinct integers candidates and a target integer target, return a list of all unique combinations of candidates where the chosen numbers sum to target. You may return the combinations in any order.

The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to the target is less than 150 combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used
multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

Code:

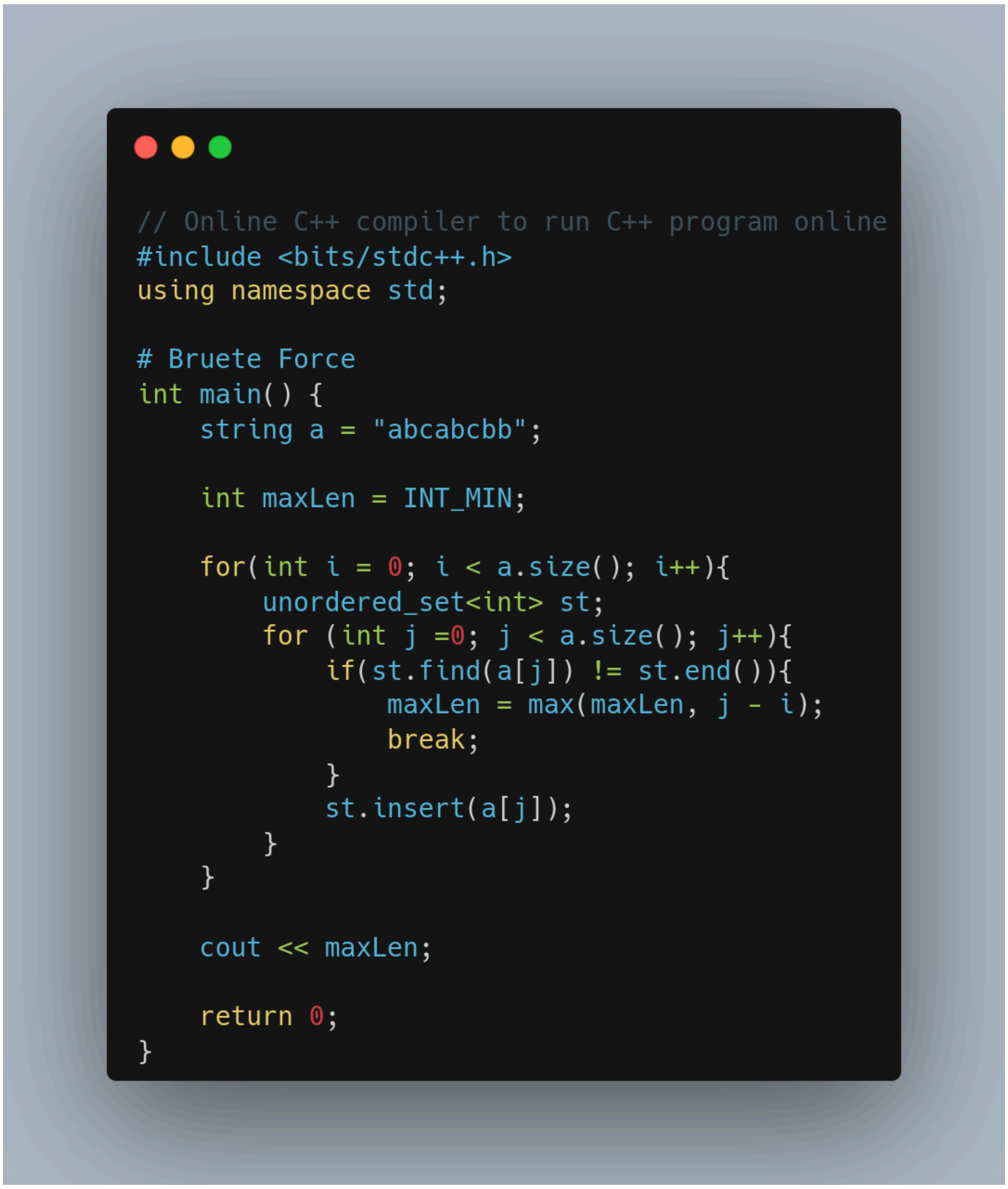
```cpp
class Solution {

    void generateSubset(vector<int> &arr, vector<vector<int>> &ans,
vector<int> &currentSubset, int i, int sum, int n){
    //Base case
    if(i == n){
        if(sum == 0){
            // sort(currentSubset.begin(), currentSubset.end());
            ans.push_back(currentSubset);
        }
        return;
    }
    if(arr[i] <= sum){
        //Pick
        currentSubset.push_back(arr[i]);
        generateSubset(arr, ans, currentSubset, i, sum - arr[i], n);
        currentSubset.pop_back();
    }

    //Not Take case
    generateSubset(arr, ans, currentSubset, i+1, sum, n);
}

public:
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> ans;
        int n = candidates.size();
        vector<int> currentSubset;
        generateSubset(candidates, ans, currentSubset, 0, target, n);
        return ans;
    }
};
```

Combination sum II

```cpp
class Solution {
    void generateSubset(vector<int> &candidates, vector<vector<int>> &ans,
vector<int> &currentSubset, int idx, int target, int n){
        if(target == 0){
            ans.push_back(currentSubset);
            return;
        }
      // Avoid extra calls for negative
        if(target<0){
            return;
        }
        for(int i = idx; i < candidates.size();i++){
            if(i > idx && candidates[i] == candidates[i - 1]) continue;
            currentSubset.push_back(candidates[i]);
            generateSubset(candidates, ans, currentSubset, i +1 , target -
candidates[i], n);
            currentSubset.pop_back();
        }
    }
public:
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target)
{
        vector<vector<int>> ans;
        vector<int> currentSubset;
        sort(candidates.begin(), candidates.end());
        int n = candidates.size();
        generateSubset(candidates, ans, currentSubset, 0, target, n);
        // sort(ans.begin(), ans.end());
        return ans;
    }
};
```

Length of longest substring without repeating character

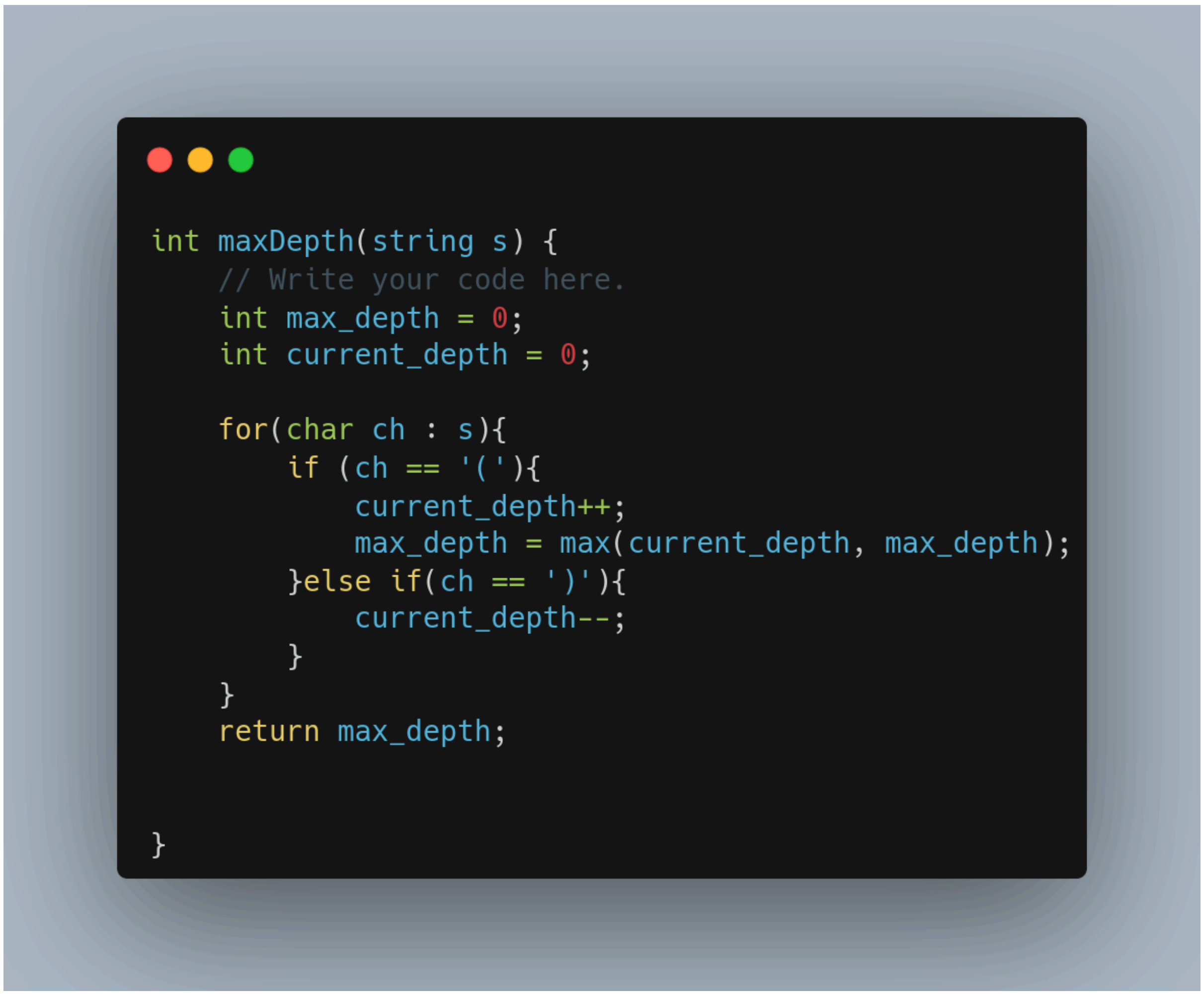
Code:
```
// Online C++ compiler to run C++ program online
#include <bits/stdc++.h>
using namespace std;
int main() {
    string a = "abcabcbb";

    int maxLen = INT_MIN;

    for(int i = 0; i < a.size(); i++){
        unordered_set<int> st;
        for (int j =0; j < a.size(); j++){
            if(st.find(a[j]) != st.end()){
                maxLen = max(maxLen, j - i);
                break;
            }
            st.insert(a[j]);
        }
    }

    cout << maxLen;

    return 0;
}
```



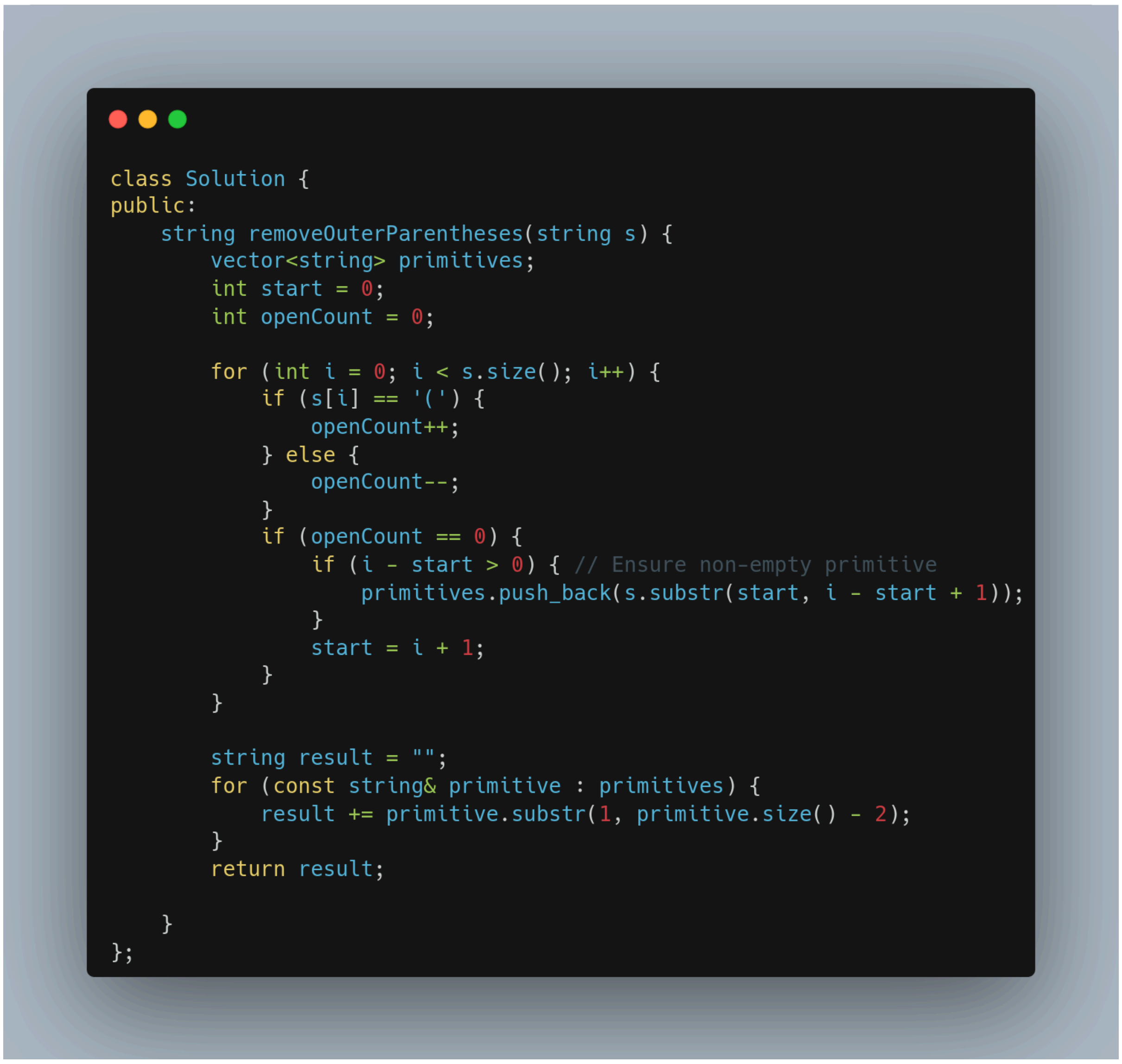
```cpp
// Online C++ compiler to run C++ program online
#include <bits/stdc++.h>
using namespace std;

# Bruete Force
int main() {
    string a = "abcabcbb";

    int maxLen = INT_MIN;

    for(int i = 0; i < a.size(); i++){
        unordered_set<int> st;
        for (int j =0; j < a.size(); j++){
            if(st.find(a[j]) != st.end()){
                maxLen = max(maxLen, j - i);
                break;
            }
            st.insert(a[j]);
        }
    }

    cout << maxLen;

    return 0;
}
```

String Problems

1. Maximum Nesting Depth Of The Parentheses

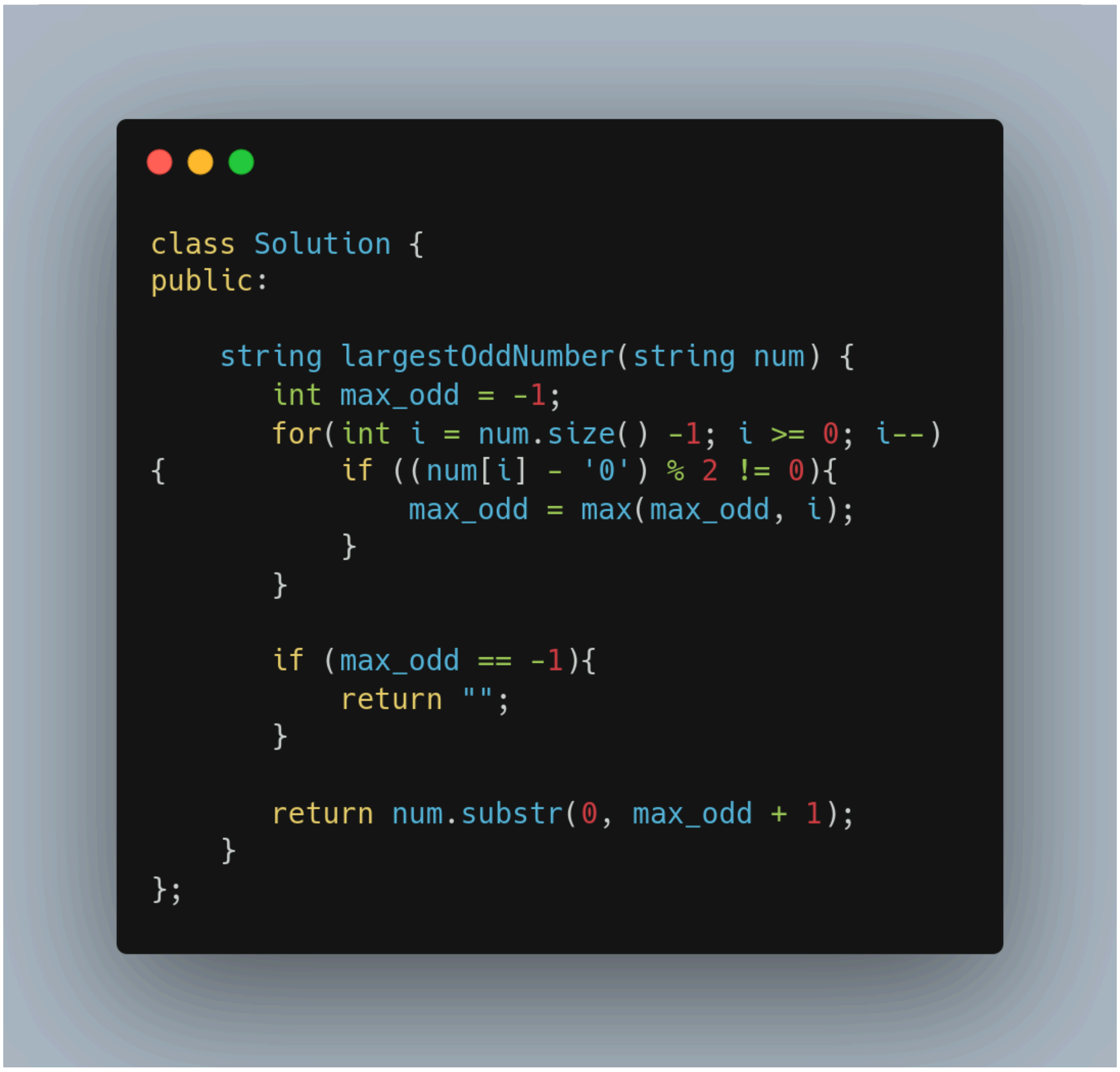


```cpp
int maxDepth(string s) {
    // Write your code here.
    int max_depth = 0;
    int current_depth = 0;

    for(char ch : s){
        if (ch == '('){
            current_depth++;
            max_depth = max(current_depth, max_depth);
        }else if(ch == ')'){
            current_depth--;
        }
    }
    return max_depth;


}
```



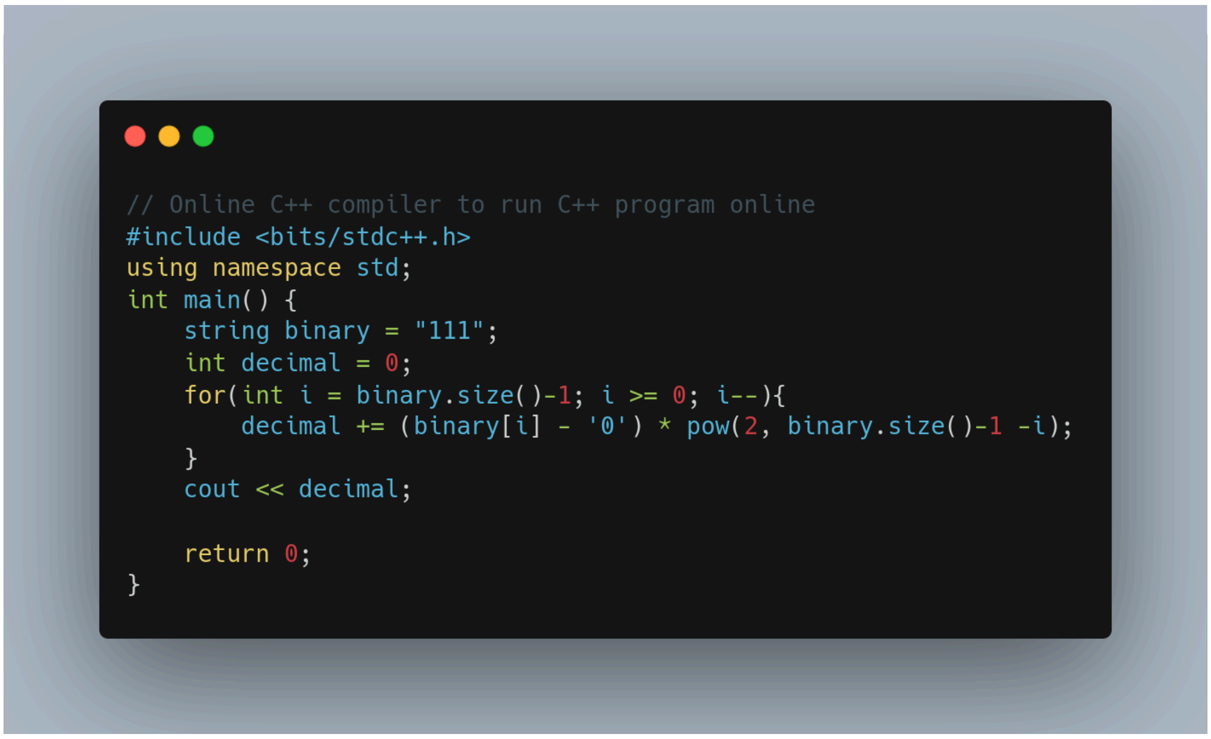
```cpp
class Solution {
public:
    string removeOuterParentheses(string s) {
        vector<string> primitives;
        int start = 0;
        int openCount = 0;

        for (int i = 0; i < s.size(); i++) {
            if (s[i] == '(') {
                openCount++;
            } else {
                openCount--;
            }
            if (openCount == 0) {
                if (i - start > 0) { // Ensure non-empty primitive
                    primitives.push_back(s.substr(start, i - start + 1));
                }
                start = i + 1;
            }
        }

        string result = "";
        for (const string& primitive : primitives) {
            result += primitive.substr(1, primitive.size() - 2);
        }
        return result;

    }
};
```

## Largest Odd number in string:



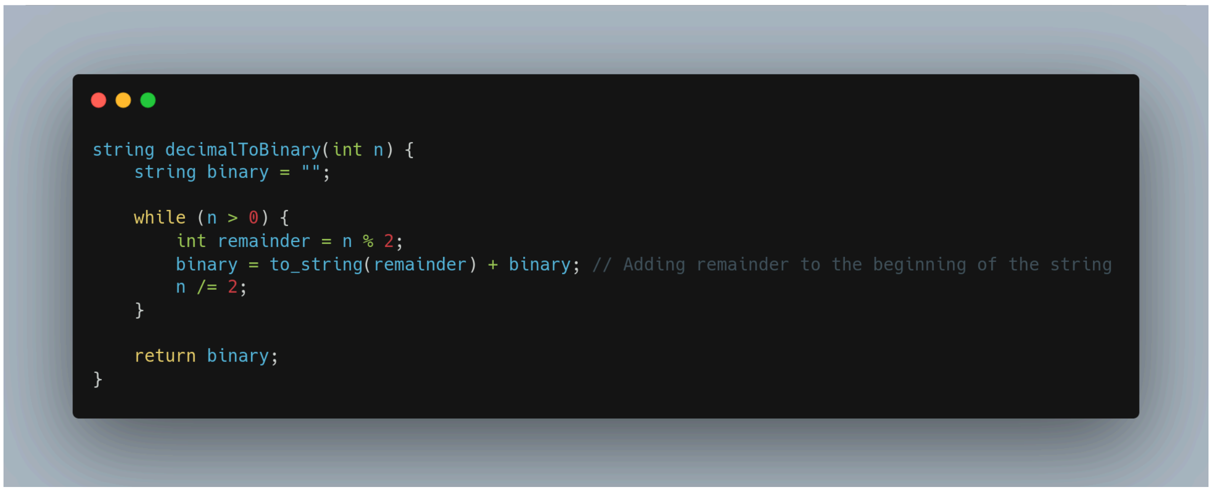
```
class Solution {
public:

    string largestOddNumber(string num) {
        int max_odd = -1;
        for(int i = num.size() -1; i >= 0; i--)
{           if ((num[i] - '0') % 2 != 0){
                max_odd = max(max_odd, i);
            }
        }

        if (max_odd == -1){
            return "";
        }

        return num.substr(0, max_odd + 1);
    }
};
```

Binary to decimal:



```cpp
// Online C++ compiler to run C++ program online
#include <bits/stdc++.h>
using namespace std;
int main() {
    string binary = "111";
    int decimal = 0;
    for(int i = binary.size()-1; i >= 0; i--){
        decimal += (binary[i] - '0') * pow(2, binary.size()-1 -i);
    }
    cout << decimal;

    return 0;
}
```

## Decimal to binary

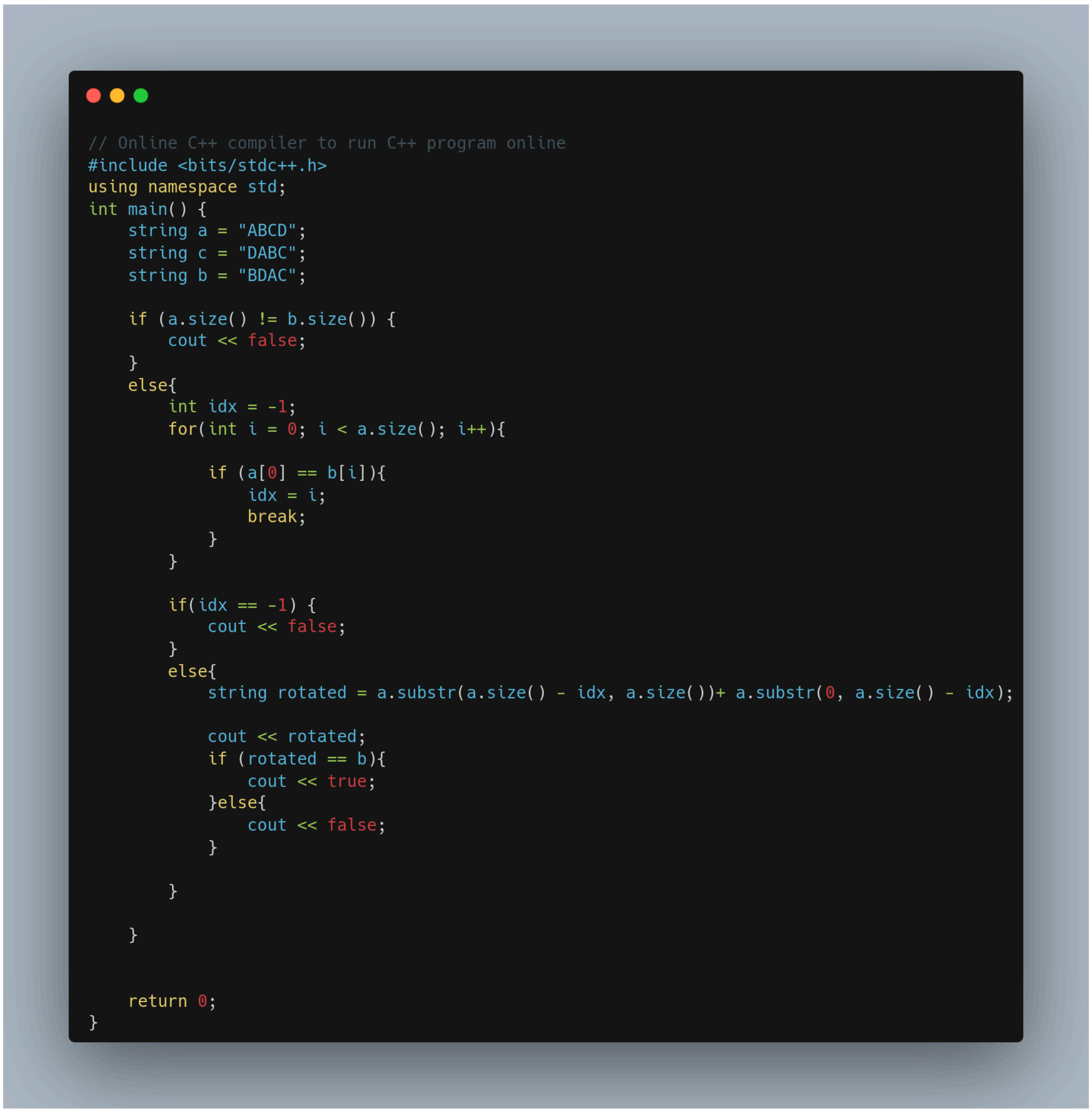


```cpp
string decimalToBinary(int n) {
    string binary = "";

    while (n > 0) {
        int remainder = n % 2;
        binary = to_string(remainder) + binary; // Adding remainder to the beginning of the string
        n /= 2;
    }

    return binary;
}
```

Check Given String are rotation of each other:

Approach 1:
→ Create a rotated string just by finding the location of first character of string 1 in string 2
→ then using that location create a rotated string and check whether string 2 and rotated string are equal.

Approach 2:
-> concatenate str1 + str1 and check str2 is present in concat or not



```cpp
// Online C++ compiler to run C++ program online
#include <bits/stdc++.h>
using namespace std;
int main() {
    string a = "ABCD";
    string c = "DABC";
    string b = "BDAC";

    if (a.size() != b.size()) {
        cout << false;
    }
    else{
        int idx = -1;
        for(int i = 0; i < a.size(); i++){

            if (a[0] == b[i]){
                idx = i;
                break;
            }
        }

        if(idx == -1) {
            cout << false;
        }
        else{
            string rotated = a.substr(a.size() - idx, a.size())+ a.substr(0, a.size() - idx);

            cout << rotated;
            if (rotated == b){
                cout << true;
            }else{
                cout << false;
            }

        }

    }


    return 0;
}
```

## Isomorphic String:

Sample Input 1 :

aab
xxy

Sample Output 1 :

1

Explanation of sample input 1:

The character ‘a’ maps to ‘x’ and ‘b’ maps to ‘y’. Hence, the answer is 1 in this case.

Sample Input 2 :

aab
xyz

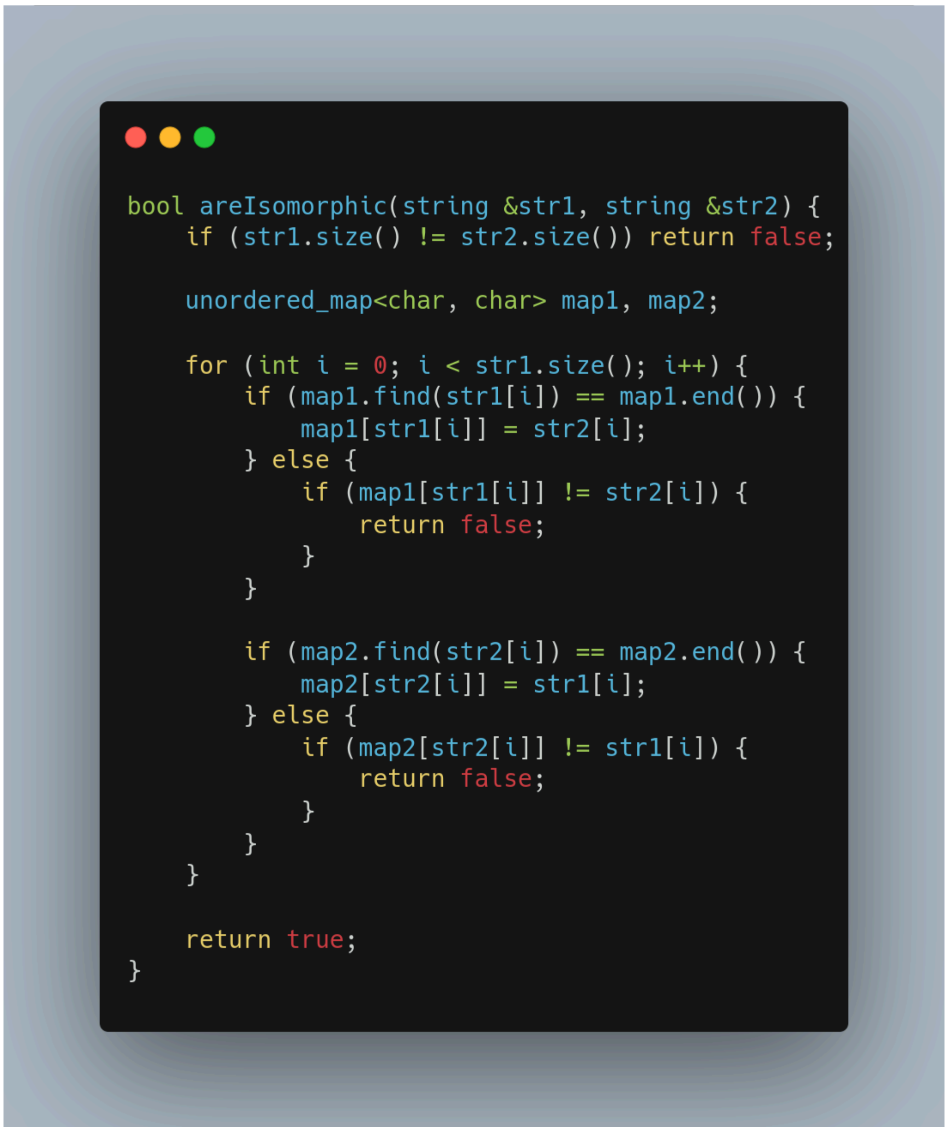
Sample Output 2 :

0

Constraints :

1 <= |str1|, |str2| <= 10^3

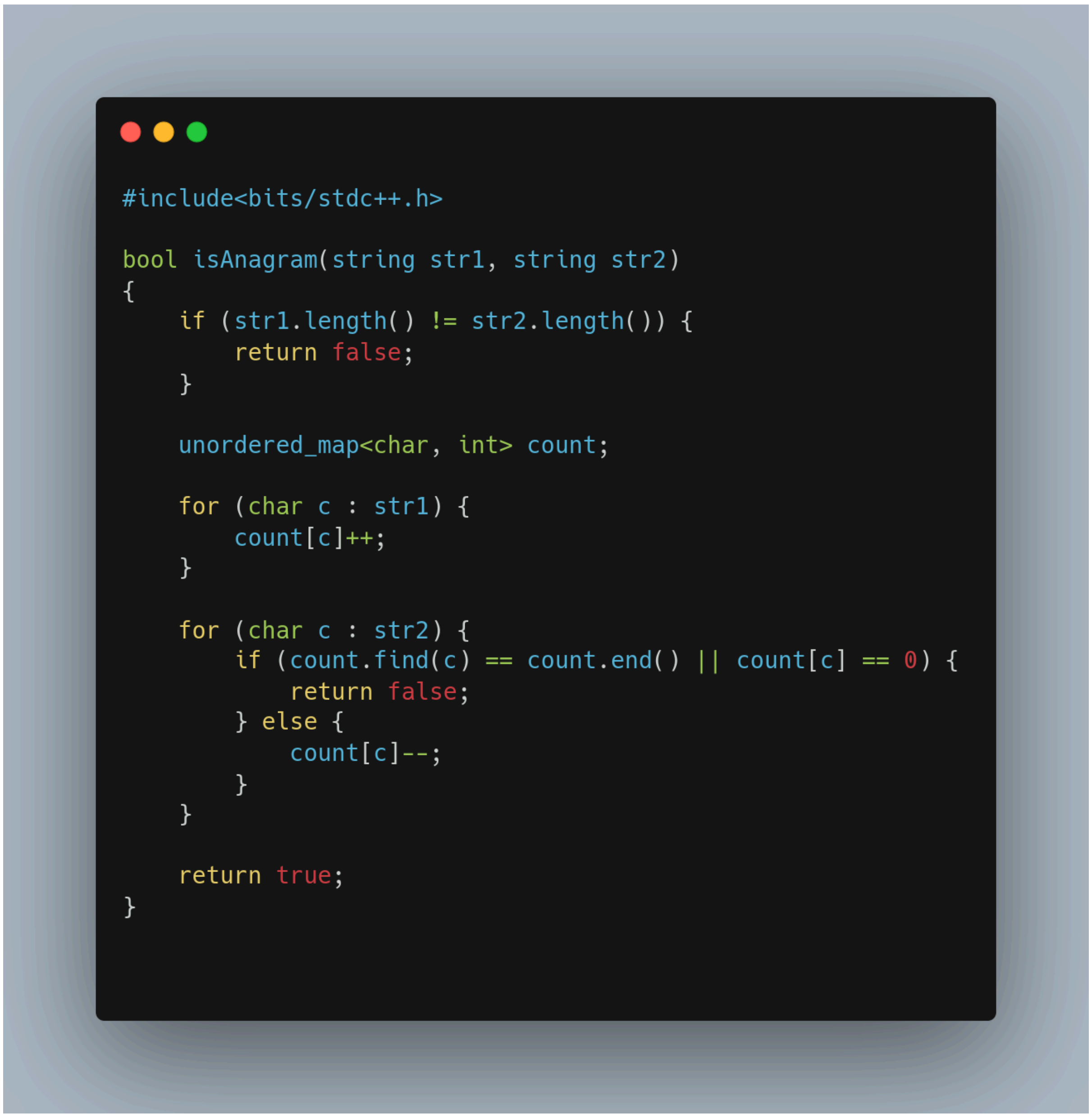
# Approach create two maps to keep track of each mapping from both side means left to right and right to left
If that mapping already present then check wheather the mapping wrong the return false other wise after loop return true



```
bool areIsomorphic(string &str1, string &str2) {
    if (str1.size() != str2.size()) return false;

    unordered_map<char, char> map1, map2;

    for (int i = 0; i < str1.size(); i++) {
        if (map1.find(str1[i]) == map1.end()) {
            map1[str1[i]] = str2[i];
        } else {
            if (map1[str1[i]] != str2[i]) {
                return false;
            }
        }

        if (map2.find(str2[i]) == map2.end()) {
            map2[str2[i]] = str1[i];
        } else {
            if (map2[str2[i]] != str1[i]) {
                return false;
            }
        }
    }

    return true;
}
```

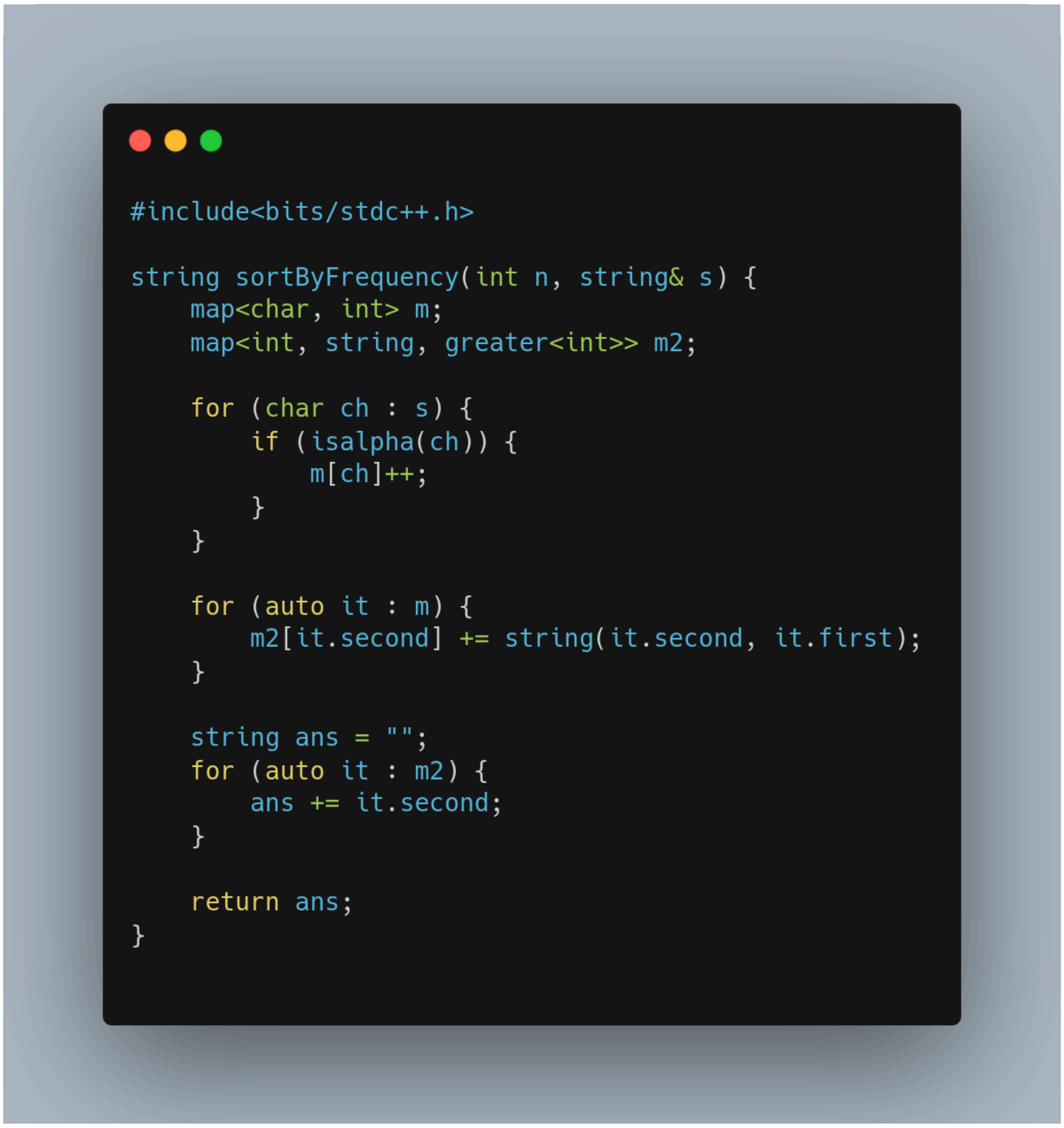
## Anagram Pairs

Character Frequency Count:
Decrement Frequency Count:



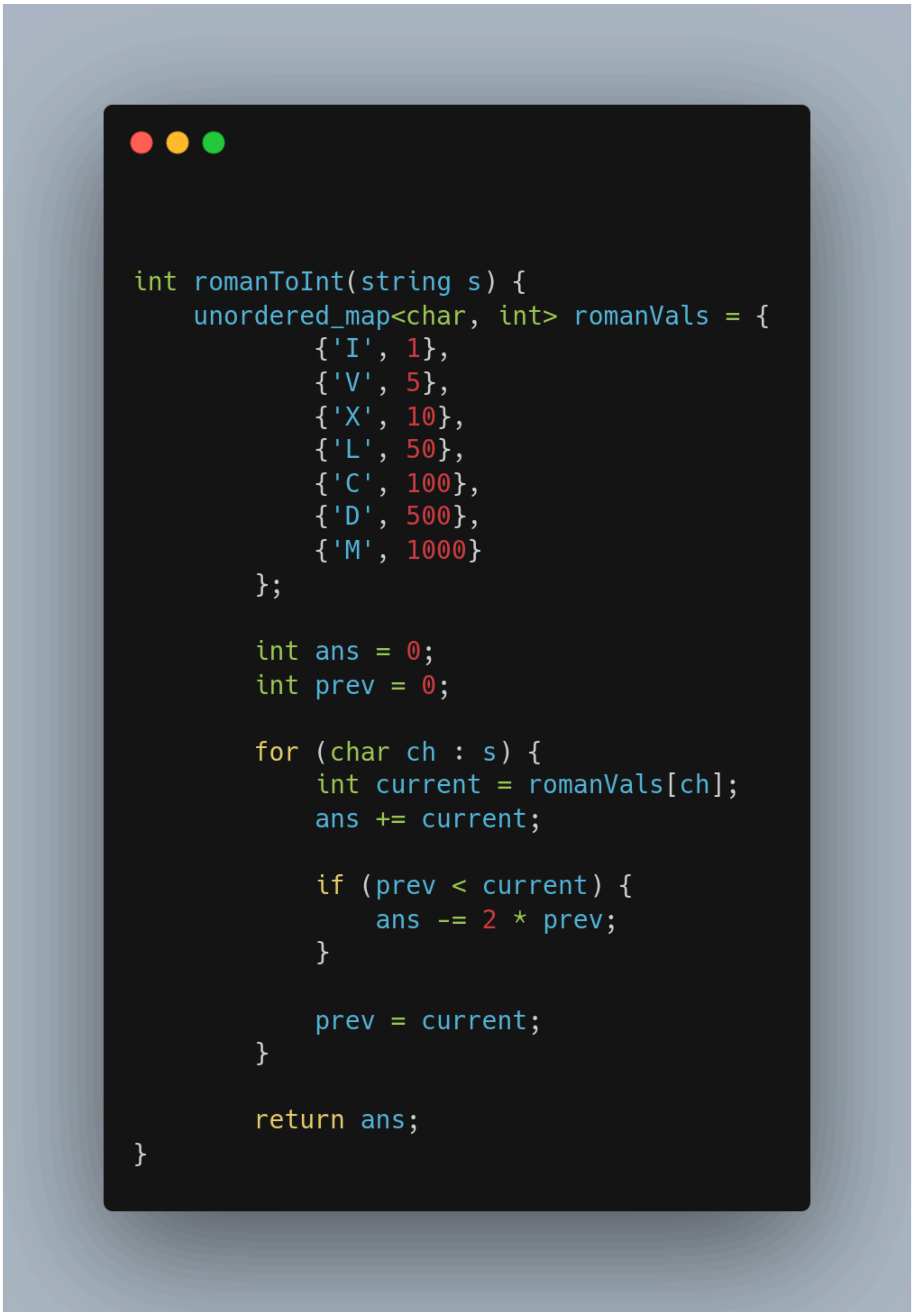
```
#include<bits/stdc++.h>

bool isAnagram(string str1, string str2)
{
    if (str1.length() != str2.length()) {
        return false;
    }

    unordered_map<char, int> count;

    for (char c : str1) {
        count[c]++;
    }

    for (char c : str2) {
        if (count.find(c) == count.end() || count[c] == 0) {
            return false;
        } else {
            count[c]--;
        }
    }

    return true;
}
```

## Sorting by frequency



```cpp
#include<bits/stdc++.h>

string sortByFrequency(int n, string& s) {
    map<char, int> m;
    map<int, string, greater<int>> m2;

    for (char ch : s) {
        if (isalpha(ch)) {
            m[ch]++;
        }
    }

    for (auto it : m) {
        m2[it.second] += string(it.second, it.first);
    }

    string ans = "";
    for (auto it : m2) {
        ans += it.second;
    }

    return ans;
}
```

# To convert roman to integer



```
int romanToInt(string s) {
    unordered_map<char, int> romanVals = {
            {'I', 1},
            {'V', 5},
            {'X', 10},
            {'L', 50},
            {'C', 100},
            {'D', 500},
            {'M', 1000}
        };

        int ans = 0;
        int prev = 0;

        for (char ch : s) {
            int current = romanVals[ch];
            ans += current;

            if (prev < current) {
                ans -= 2 * prev;
            }

            prev = current;
        }

        return ans;
}
```

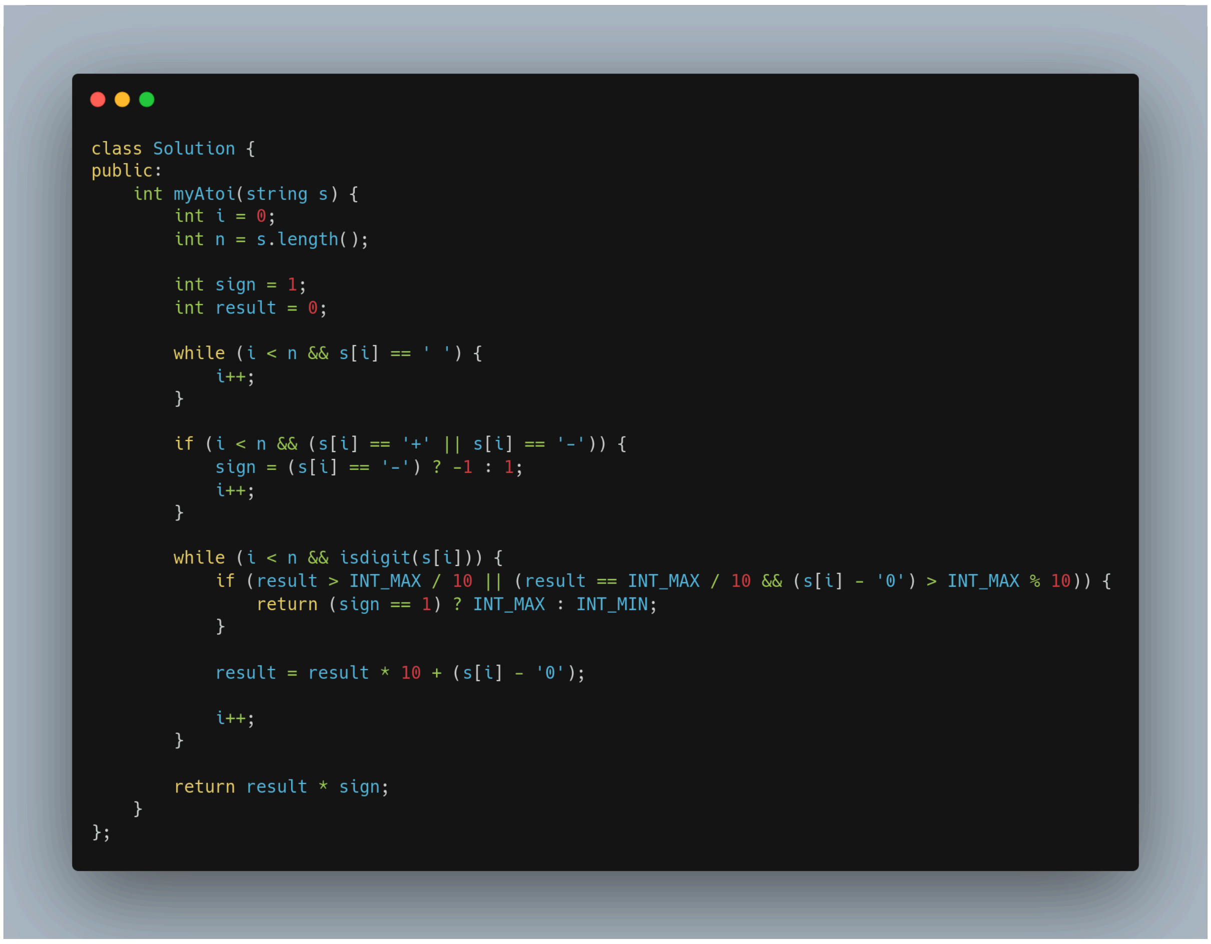
# Atoi→ String to integer

Sample Input 1 :

-546+er

Sample Output 1 :

-546



```cpp
class Solution {
public:
    int myAtoi(string s) {
        int i = 0;
        int n = s.length();

        int sign = 1;
        int result = 0;

        while (i < n && s[i] == ' ') {
            i++;
        }

        if (i < n && (s[i] == '+' || s[i] == '-')) {
            sign = (s[i] == '-') ? -1 : 1;
            i++;
        }

        while (i < n && isdigit(s[i])) {
            if (result > INT_MAX / 10 || (result == INT_MAX / 10 && (s[i] - '0') > INT_MAX % 10)) {
                return (sign == 1) ? INT_MAX : INT_MIN;
            }

            result = result * 10 + (s[i] - '0');

            i++;
        }

        return result * sign;
    }
};
```

Longest Palindromic substring:

(n3) → brute force solution

```cpp
class Solution {
public:
   bool isPalindrome(string s){
       int left = 0, right = s.size() - 1;
       while(left < right){
           if (s[left] != s[right]){
               return false;
           }
           ++left;
           --right;
       }
       return true;
   }
   string longestPalindrome(string s) {
       int maxLength = 0;
       string result = "";
       for (int i = 0; i < s.size(); i++){
           for(int j = i + 1; j <= s.length(); ++j){
               string substr = s.substr(i, j - i);
               if (isPalindrome(substr) && substr.length() > maxLength) {
                   maxLength = substr.length();
                   result = substr;
               }
           }
       }
       return result;
   }
};
```

(n2)

Expand string from middle:

Bab —> so let say we are at a then we expand from both side of middle till its equal

## Sum of beauty of substrings:

Example 1:
Input: s = "aabcb"
Output: 5
Explanation: The substrings with non-zero beauty are ["aab","aabc","aabcb","abcb","bcb"], each with beauty equal to 1.
Example 2:
Input: s = "aabcbaa"
Output: 17

```
int sumOfBeauty(string s) {
   int totalBeauty = 0;
   int n = s.length();

   for (int i = 0; i < n; ++i) {
       vector<int> count(26, 0);
       for (int j = i; j < n; ++j) {
           count[s[j] - 'a']++;
           int maxFreq = 0, minFreq = n;
           for (int k = 0; k < 26; ++k) {
               if (count[k] > 0) {
                   maxFreq = max(maxFreq, count[k]);
                   minFreq = min(minFreq, count[k]);
               }
           }

           totalBeauty += (maxFreq - minFreq);
       }
   }

   return totalBeauty;
}
```

Link list:

Linked List Cycle

```
/**
* Definition for singly-linked list.
* struct ListNode {
*     int val;
*     ListNode *next;
*     ListNode(int x) : val(x), next(NULL) {}
* };
*/
class Solution {
public:
   bool hasCycle(ListNode *head) {
       if (head == nullptr || head->next == nullptr) return false;

       ListNode* slow = head;
       ListNode* fast = head;

       while(fast != nullptr && fast->next != nullptr){
           slow = slow->next;
           fast = fast->next->next;
           if (slow == fast){
               return true;
           }
       }
       return false;
   }
};
```

Reverse Link List:

```
/*********************************************************************

   Following is the class structure of the Node class:

       class Node
       {
       public:
           int data;
           Node *next;
           Node()
           {
               this->data = 0;
               next = NULL;
           }
           Node(int data)
           {
               this->data = data;
               this->next = NULL;
           }
           Node(int data, Node* next)
           {
               this->data = data;
               this->next = next;
           }
       };


*****************************************************************/

Node* reverseLinkedList(Node *head)
{
   Node* prev = nullptr;
   Node* current = head;
   Node* next = nullptr;
   while(current != nullptr){
       next = current->next;

       current->next = prev;

       prev = current;
       current = next;
   }

   head = prev;
   return head;
}
```

Middle of link list:

```cpp
/**
* Definition for singly-linked list.
* struct ListNode {
*     int val;
*     ListNode *next;
*     ListNode() : val(0), next(nullptr) {}
*     ListNode(int x) : val(x), next(nullptr) {}
*     ListNode(int x, ListNode *next) : val(x), next(next) {}
* };
*/
class Solution {
public:
   ListNode* middleNode(ListNode* head) {
       if (head == NULL && head->next == NULL) return head;

       ListNode* slow = head;
       ListNode* fast = head;
       while(fast != NULL && fast->next != NULL){
           slow = slow->next;
           fast = fast->next->next;
       }
       return slow;
   }
};
```

Find Start of cycle:

```cpp
/**
* Definition for singly-linked list.
* struct ListNode {
*     int val;
*     ListNode *next;
*     ListNode(int x) : val(x), next(NULL) {}
* };
*/
class Solution {
public:
   ListNode *detectCycle(ListNode *head) {
       if(head == nullptr || head->next == nullptr) return nullptr;

       ListNode* slow = head;
       ListNode* fast = head;
       while(fast != nullptr && fast->next != nullptr){
           slow = slow->next;
           fast = fast->next->next;
```

```
            if(slow == fast){
                fast = head;

                while(slow != fast){
                    slow = slow->next;
                    fast = fast->next;
                }
                return slow;
            }
        }
        return nullptr;
    }
};
```

```
// Online C++ compiler to run C++ program online
#include <iostream>
using namespace std;
int main() {
    int numDigit = 3;
    int sumDigit = 9;

    string ans(numDigit, '0');
    cout << ans;
    ans[0] = '1';
    sumDigit -= 1;
    for(int i = numDigit-1; i >= 1; i--){
        int digit = min(sumDigit, 9);
        char Cdigit = '0' + digit;
        ans[i] = Cdigit;
        sumDigit -= digit;
    }
    int remain = sumDigit;
    cout << remain << endl;

    if(remain != 0){
        ans[0] = '0' + remain;
    }

    cout << ans;

    return 0;
}
```

Convert char to int

```
    int a = 'digit_char' - '0';
    cout << a;
```

Convert digit to char

```
char a = '0' +  digit;
   cout << a;
```