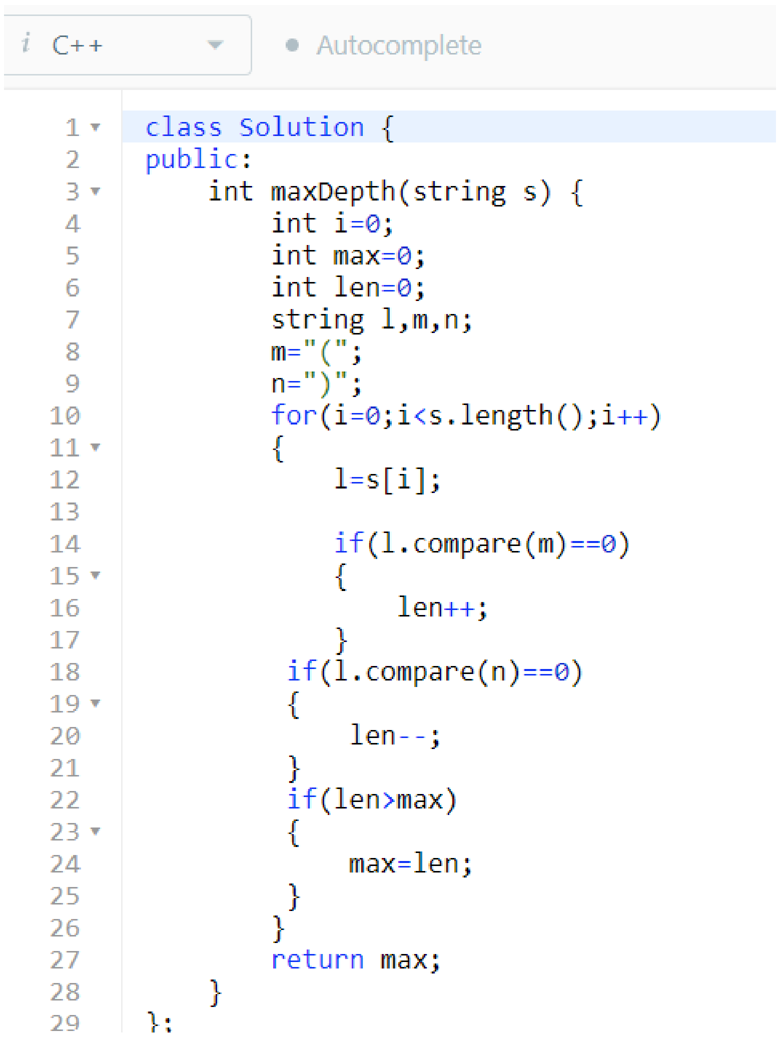
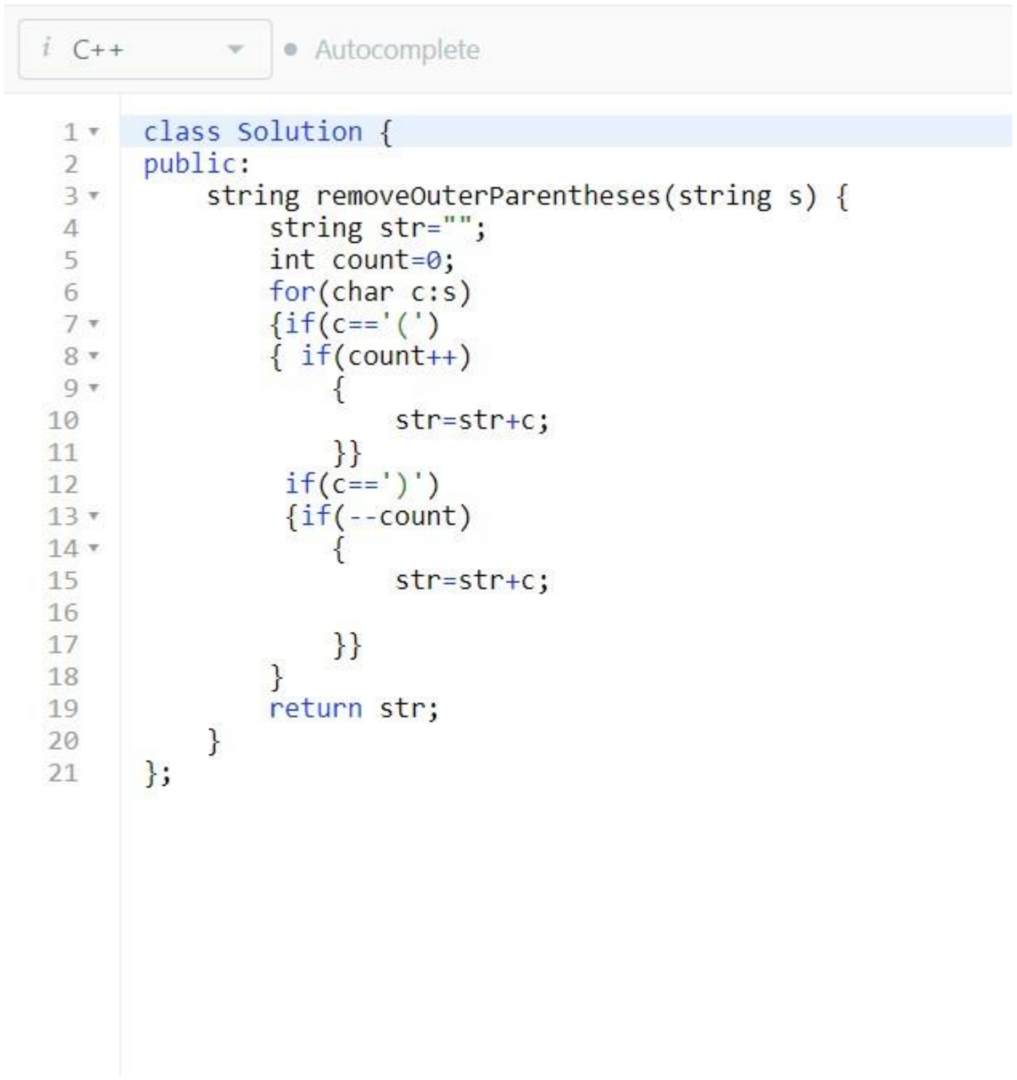
Stack and Queue Implementation

1614. Maximum Nesting Depth of the Parentheses



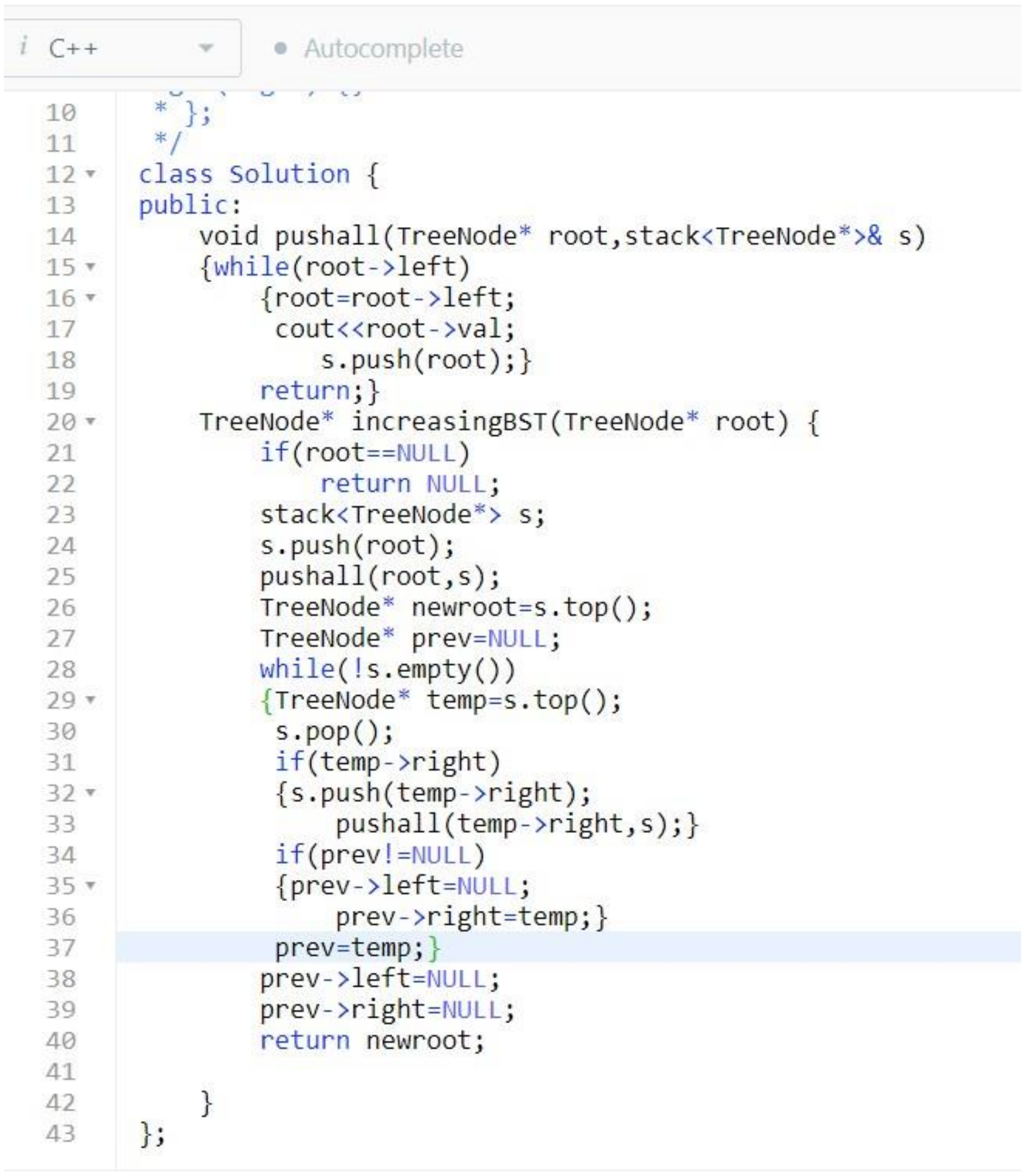
```cpp
class Solution {
public:
    int maxDepth(string s) {
        int i=0;
        int max=0;
        int len=0;
        string l,m,n;
        m="(";
        n=")";
        for(i=0;i<s.length();i++)
        {
            l=s[i];

            if(l.compare(m)==0)
            {
                len++;
            }
         if(l.compare(n)==0)
         {
             len--;
         }
         if(len>max)
         {
             max=len;
         }
        }
        return max;
    }
};
```

1021. Remove Outermost Parentheses



```
class Solution {
public:
    string removeOuterParentheses(string s) {
        string str="";
        int count=0;
        for(char c:s)
        {if(c=='(')
        { if(count++)
            {
                str=str+c;
            }}
         if(c==')')
         {if(--count)
            {
                str=str+c;

            }}
        }
        return str;
    }
};
```

897. Increasing Order Search Tree

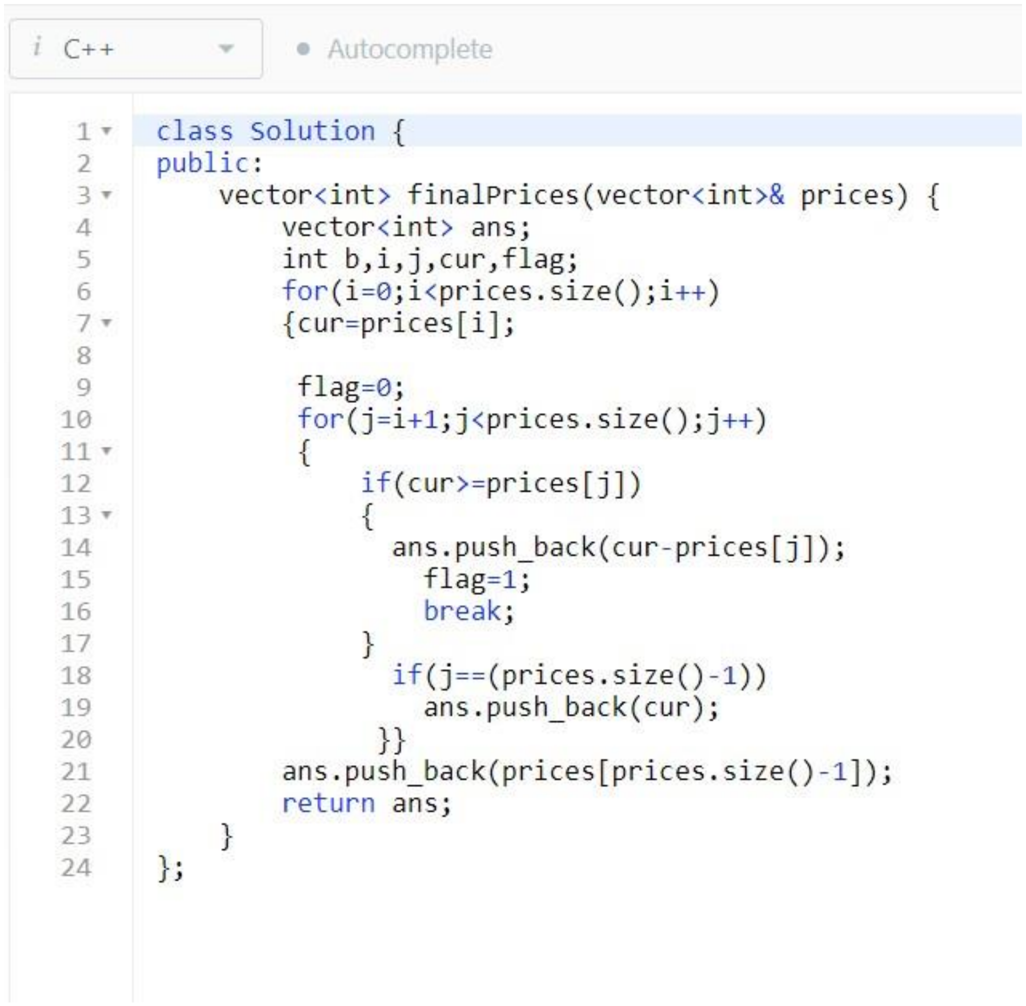


C++

Autocomplete

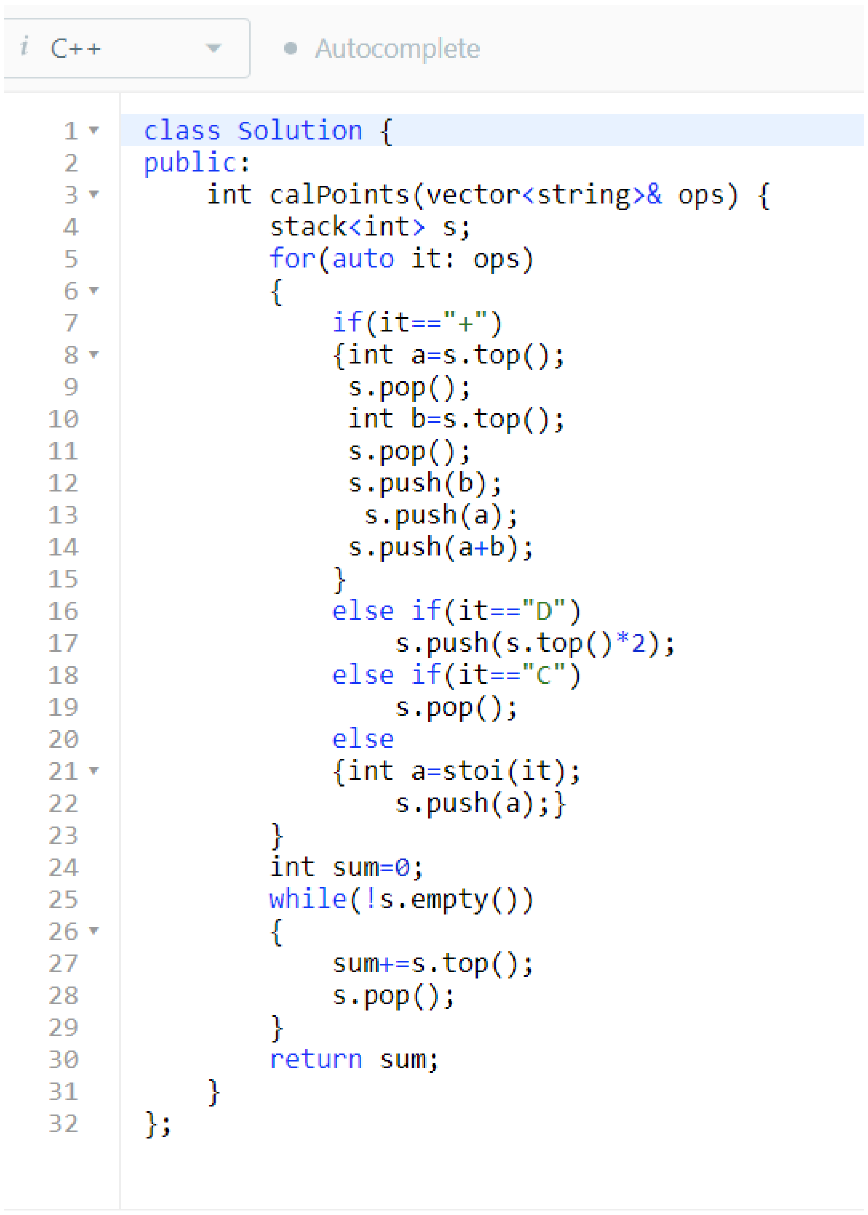
```
 * };
 */
class Solution {
public:
    void pushall(TreeNode* root,stack<TreeNode*>& s)
    {while(root->left)
        {root=root->left;
         cout<<root->val;
            s.push(root);}
        return;}
    TreeNode* increasingBST(TreeNode* root) {
        if(root==NULL)
            return NULL;
        stack<TreeNode*> s;
        s.push(root);
        pushall(root,s);
        TreeNode* newroot=s.top();
        TreeNode* prev=NULL;
        while(!s.empty())
        {TreeNode* temp=s.top();
         s.pop();
         if(temp->right)
         {s.push(temp->right);
             pushall(temp->right,s);}
         if(prev!=NULL)
         {prev->left=NULL;
             prev->right=temp;}
         prev=temp;}
        prev->left=NULL;
        prev->right=NULL;
        return newroot;

    }
};
```

1475. Final Prices With a Special Discount in a Shop



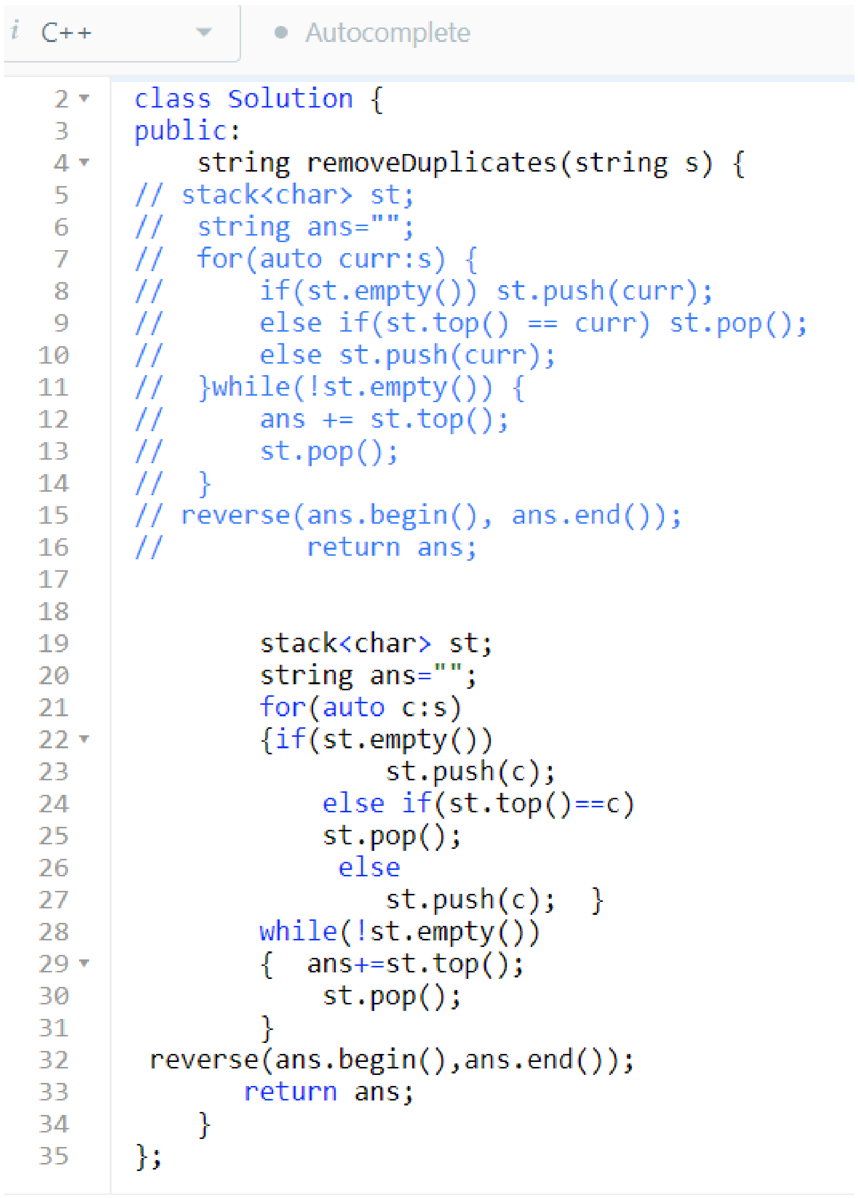
```cpp
class Solution {
public:
    vector<int> finalPrices(vector<int>& prices) {
        vector<int> ans;
        int b,i,j,cur,flag;
        for(i=0;i<prices.size();i++)
        {cur=prices[i];

         flag=0;
         for(j=i+1;j<prices.size();j++)
         {
             if(cur>=prices[j])
             {
               ans.push_back(cur-prices[j]);
                 flag=1;
                 break;
             }
               if(j==(prices.size()-1))
                 ans.push_back(cur);
              }}
        ans.push_back(prices[prices.size()-1]);
        return ans;
    }
};
```

682. Baseball Game



```cpp
class Solution {
public:
    int calPoints(vector<string>& ops) {
        stack<int> s;
        for(auto it: ops)
        {
            if(it=="+")
            {int a=s.top();
             s.pop();
             int b=s.top();
             s.pop();
             s.push(b);
              s.push(a);
             s.push(a+b);
            }
            else if(it=="D")
                s.push(s.top()*2);
            else if(it=="C")
                s.pop();
            else
            {int a=stoi(it);
                s.push(a);}
        }
        int sum=0;
        while(!s.empty())
        {
            sum+=s.top();
            s.pop();
        }
        return sum;
    }
};
```

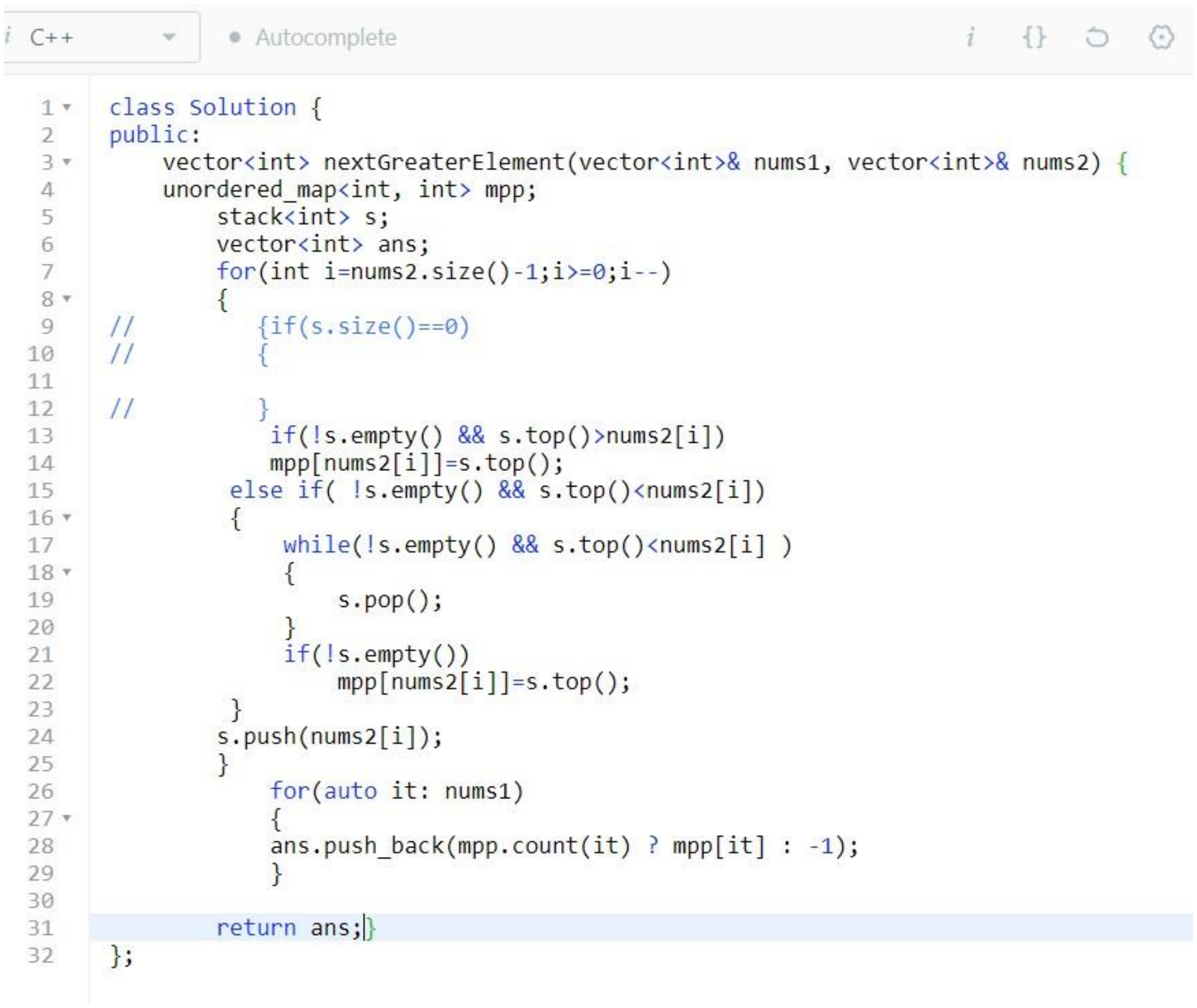
1047. Remove All Adjacent Duplicates In String



i C++ ▾ ● Autocomplete

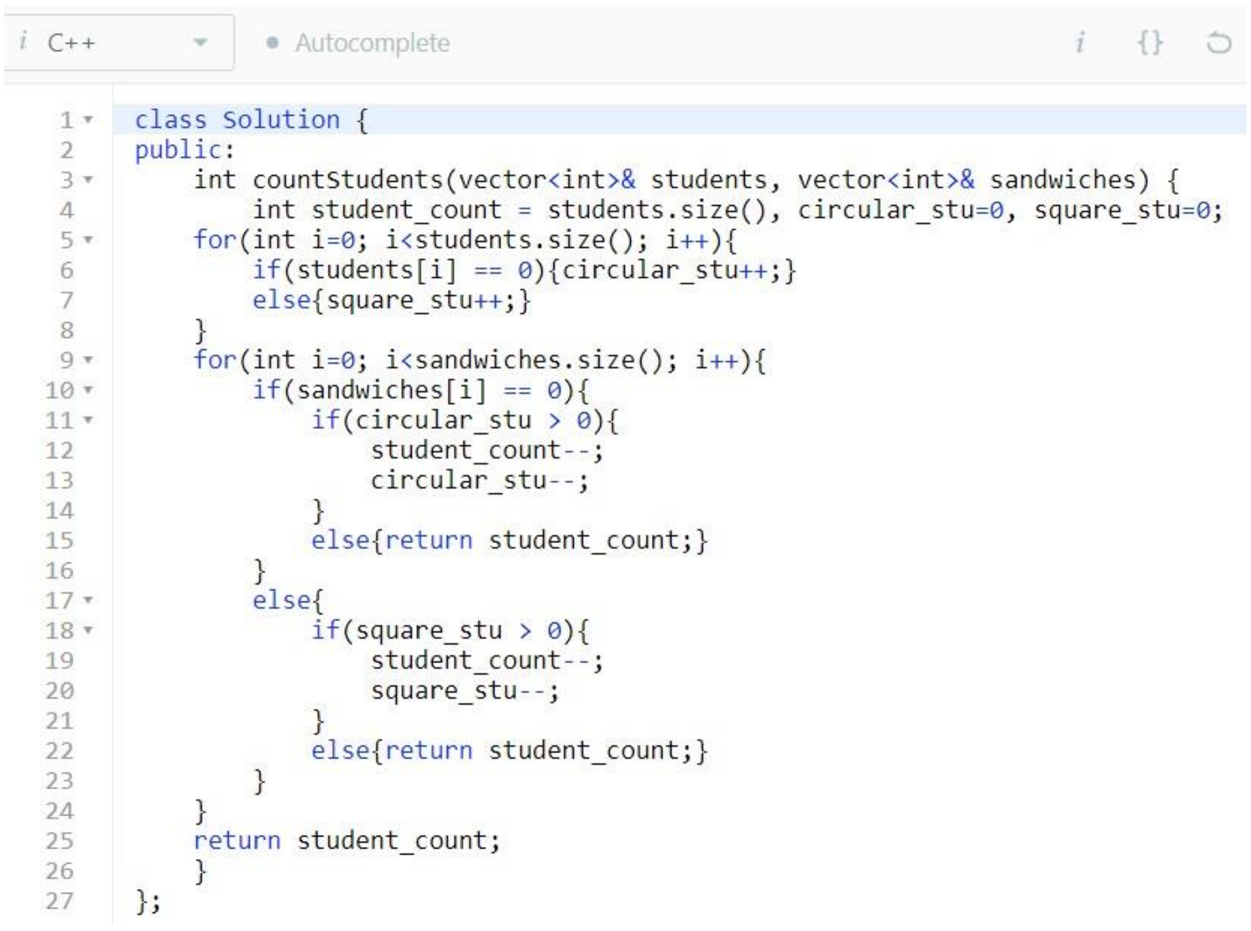
```
class Solution {
public:
    string removeDuplicates(string s) {
//  stack<char> st;
//   string ans="";
//   for(auto curr:s) {
//       if(st.empty()) st.push(curr);
//       else if(st.top() == curr) st.pop();
//       else st.push(curr);
//   }while(!st.empty()) {
//       ans += st.top();
//       st.pop();
//   }
// reverse(ans.begin(), ans.end());
//          return ans;


        stack<char> st;
        string ans="";
        for(auto c:s)
        {if(st.empty())
                st.push(c);
            else if(st.top()==c)
            st.pop();
             else
                st.push(c);   }
        while(!st.empty())
        {  ans+=st.top();
            st.pop();
        }
 reverse(ans.begin(),ans.end());
      return ans;
    }
};
```

496. Next Greater Element I



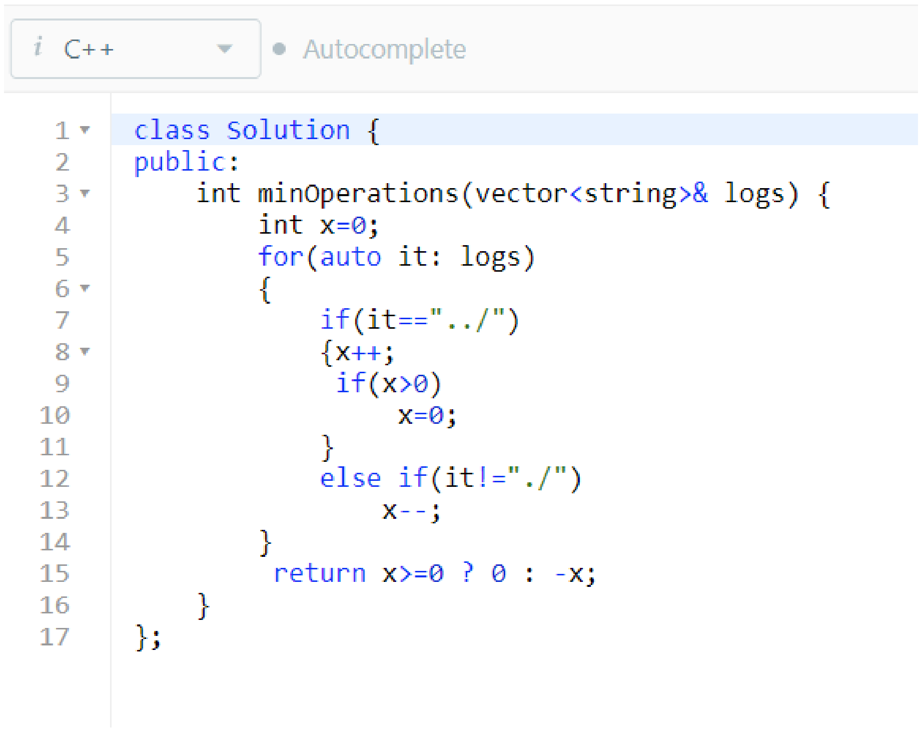
```
class Solution {
public:
    vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {
    unordered_map<int, int> mpp;
        stack<int> s;
        vector<int> ans;
        for(int i=nums2.size()-1;i>=0;i--)
        {
//           {if(s.size()==0)
//           {

//           }
            if(!s.empty() && s.top()>nums2[i])
            mpp[nums2[i]]=s.top();
         else if( !s.empty() && s.top()<nums2[i])
         {
             while(!s.empty() && s.top()<nums2[i] )
             {
                 s.pop();
             }
             if(!s.empty())
                 mpp[nums2[i]]=s.top();
         }
        s.push(nums2[i]);
        }
            for(auto it: nums1)
            {
            ans.push_back(mpp.count(it) ? mpp[it] : -1);
            }

        return ans;}
};
```

1700. Number of Students Unable to Eat Lunch



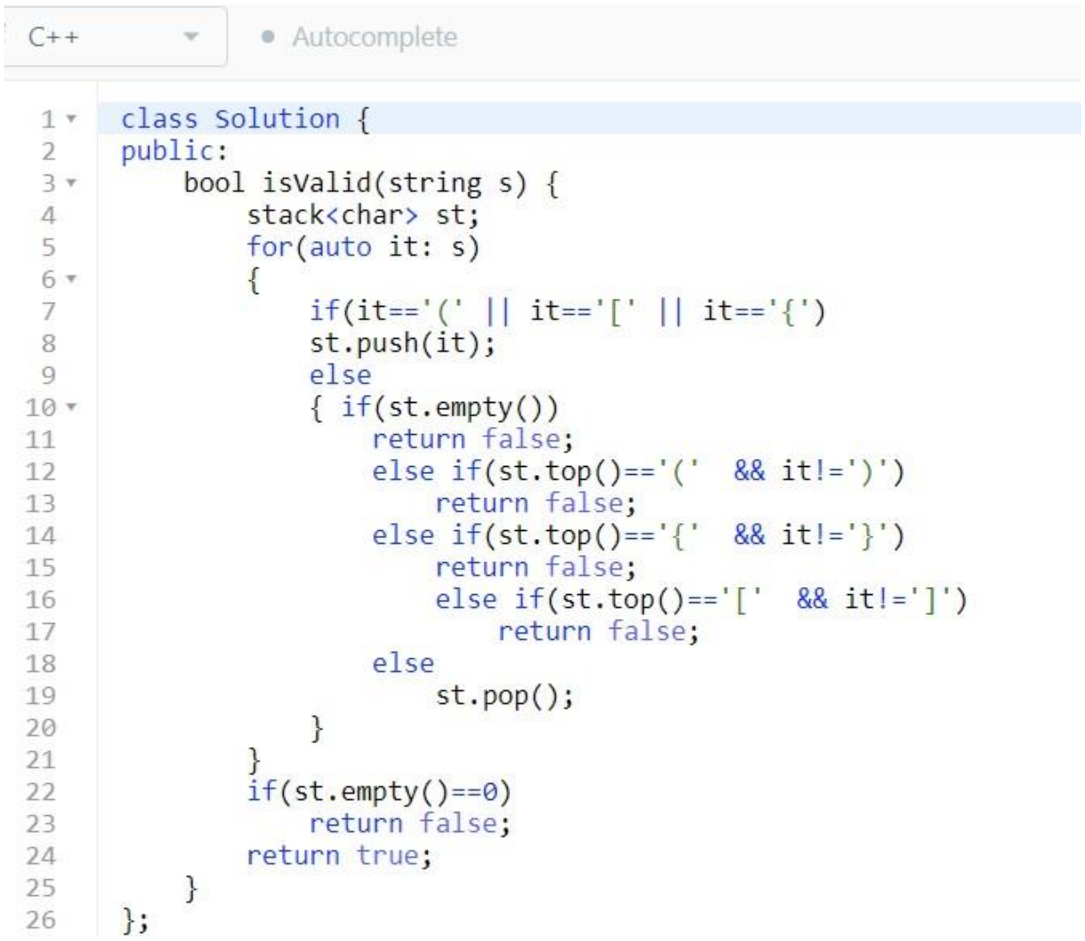
```cpp
class Solution {
public:
    int countStudents(vector<int>& students, vector<int>& sandwiches) {
        int student_count = students.size(), circular_stu=0, square_stu=0;
    for(int i=0; i<students.size(); i++){
        if(students[i] == 0){circular_stu++;}
        else{square_stu++;}
    }
    for(int i=0; i<sandwiches.size(); i++){
        if(sandwiches[i] == 0){
            if(circular_stu > 0){
                student_count--;
                circular_stu--;
            }
            else{return student_count;}
        }
        else{
            if(square_stu > 0){
                student_count--;
                square_stu--;
            }
            else{return student_count;}
        }
    }
    return student_count;
    }
};
```

1598. Crawler Log Folder



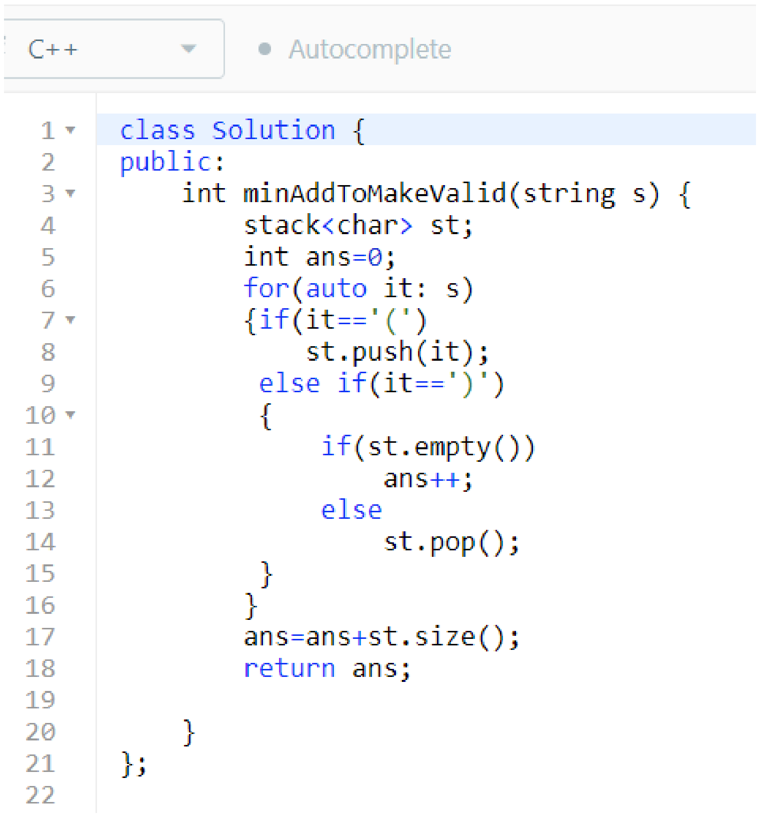
```cpp
class Solution {
public:
    int minOperations(vector<string>& logs) {
        int x=0;
        for(auto it: logs)
        {
            if(it=="../")
            {x++;
             if(x>0)
                 x=0;
            }
            else if(it!="./")
                x--;
        }
         return x>=0 ? 0 : -x;
    }
};
```

20. Valid Parentheses



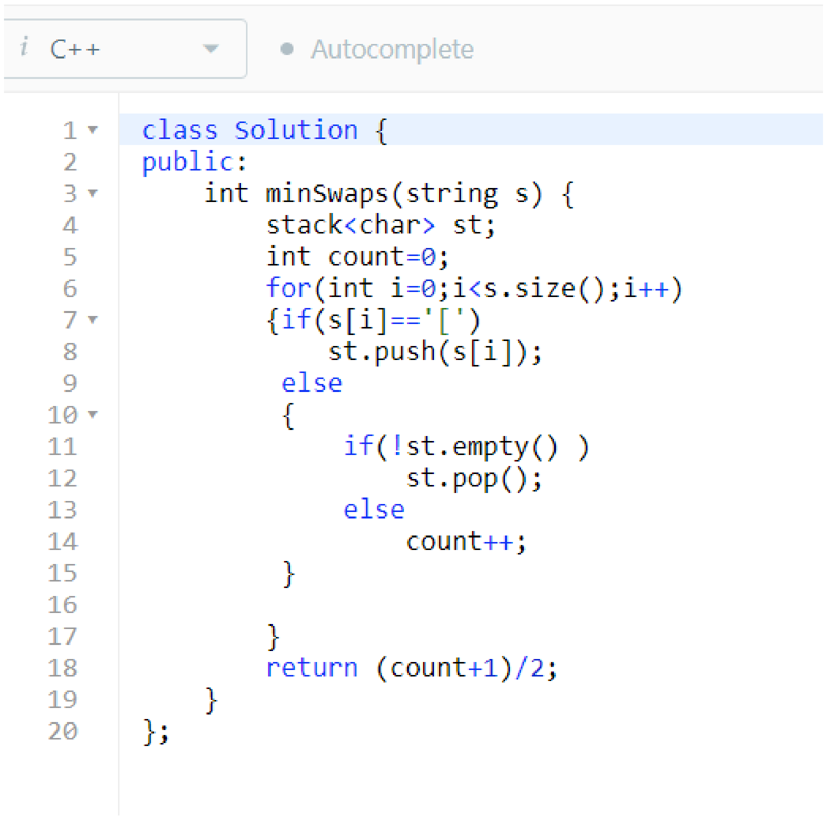
```cpp
class Solution {
public:
    bool isValid(string s) {
        stack<char> st;
        for(auto it: s)
        {
            if(it=='(' || it=='[' || it=='{')
            st.push(it);
            else
            { if(st.empty())
                return false;
                else if(st.top()=='('  && it!=')')
                    return false;
                else if(st.top()=='{'  && it!='}')
                    return false;
                    else if(st.top()=='['  && it!=']')
                        return false;
                else
                    st.pop();
            }
        }
        if(st.empty()==0)
            return false;
        return true;
    }
};
```

921. Minimum Add to Make Parentheses Valid



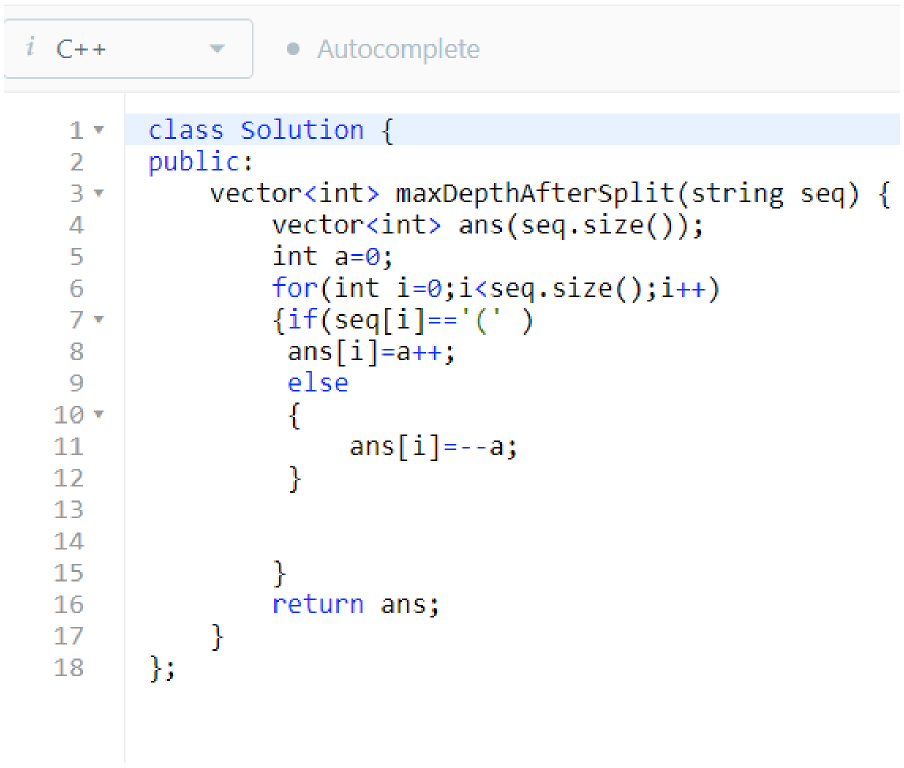
```
class Solution {
public:
    int minAddToMakeValid(string s) {
        stack<char> st;
        int ans=0;
        for(auto it: s)
        {if(it=='(')
            st.push(it);
         else if(it==')')
         {
             if(st.empty())
                 ans++;
             else
                 st.pop();
         }
        }
        ans=ans+st.size();
        return ans;

    }
};
```

1963. Minimum Number of Swaps to Make the String Balanced



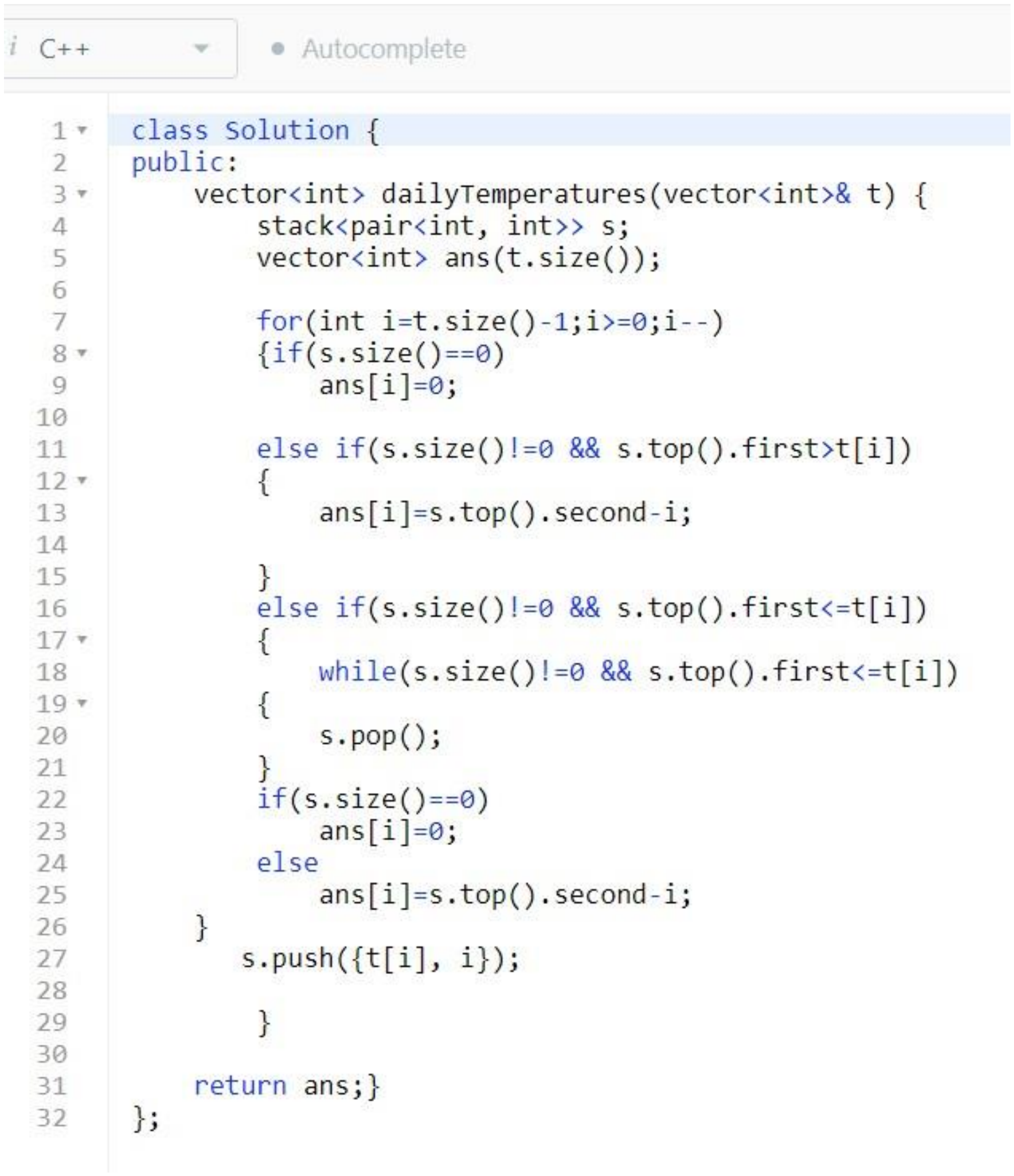
```cpp
class Solution {
public:
    int minSwaps(string s) {
        stack<char> st;
        int count=0;
        for(int i=0;i<s.size();i++)
        {if(s[i]=='[')
            st.push(s[i]);
         else
         {
             if(!st.empty() )
                 st.pop();
             else
                 count++;
         }

        }
        return (count+1)/2;
    }
};
```

1111. Maximum Nesting Depth of Two Valid Parentheses Strings



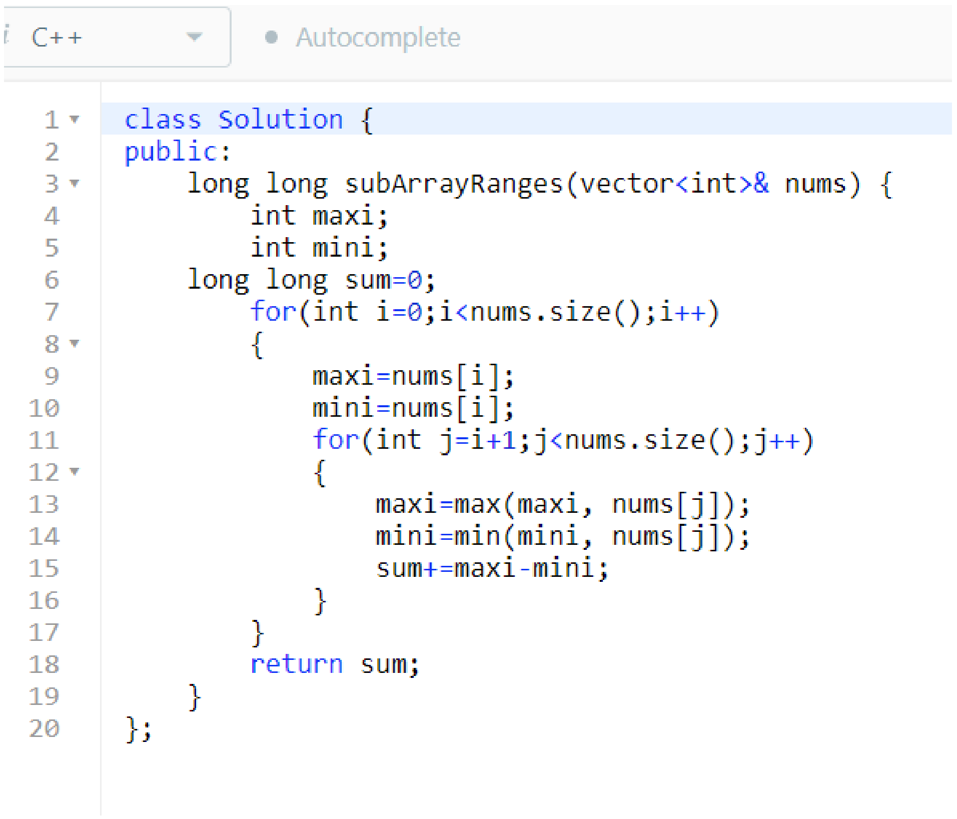
```
class Solution {
public:
    vector<int> maxDepthAfterSplit(string seq) {
        vector<int> ans(seq.size());
        int a=0;
        for(int i=0;i<seq.size();i++)
        {if(seq[i]=='(' )
         ans[i]=a++;
         else
         {
             ans[i]=--a;
         }


        }
        return ans;
    }
};
```

739. Daily Temperatures



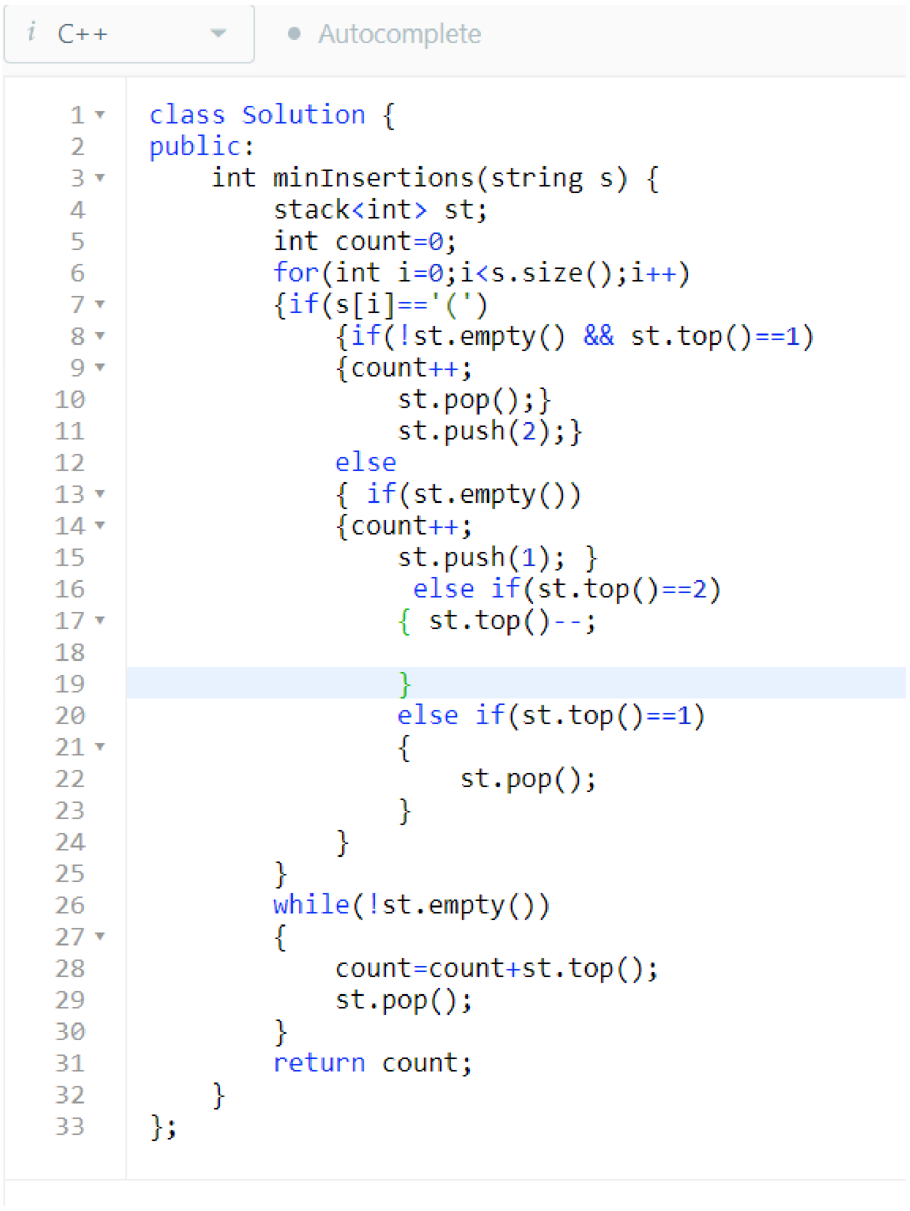
```cpp
class Solution {
public:
    vector<int> dailyTemperatures(vector<int>& t) {
        stack<pair<int, int>> s;
        vector<int> ans(t.size());

        for(int i=t.size()-1;i>=0;i--)
        {if(s.size()==0)
            ans[i]=0;

        else if(s.size()!=0 && s.top().first>t[i])
        {
            ans[i]=s.top().second-i;

        }
        else if(s.size()!=0 && s.top().first<=t[i])
        {
            while(s.size()!=0 && s.top().first<=t[i])
        {
            s.pop();
        }
        if(s.size()==0)
            ans[i]=0;
        else
            ans[i]=s.top().second-i;
    }
       s.push({t[i], i});

        }

    return ans;}
};
```

2104. Sum of Subarray Ranges



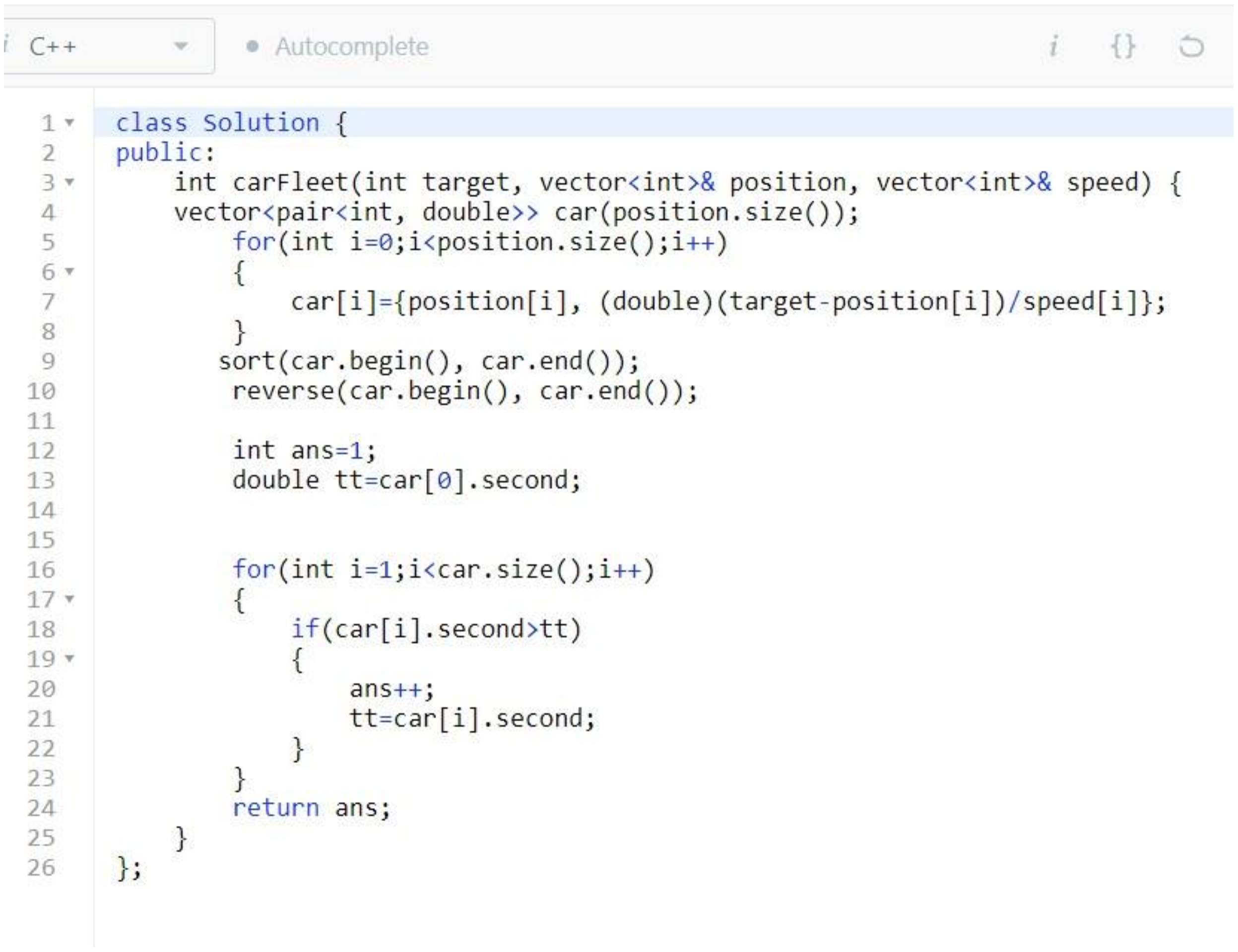
```cpp
class Solution {
public:
    long long subArrayRanges(vector<int>& nums) {
        int maxi;
        int mini;
    long long sum=0;
        for(int i=0;i<nums.size();i++)
        {
            maxi=nums[i];
            mini=nums[i];
            for(int j=i+1;j<nums.size();j++)
            {
                maxi=max(maxi, nums[j]);
                mini=min(mini, nums[j]);
                sum+=maxi-mini;
            }
        }
        return sum;
    }
};
```

1541. Minimum Insertions to Balance a Parentheses String



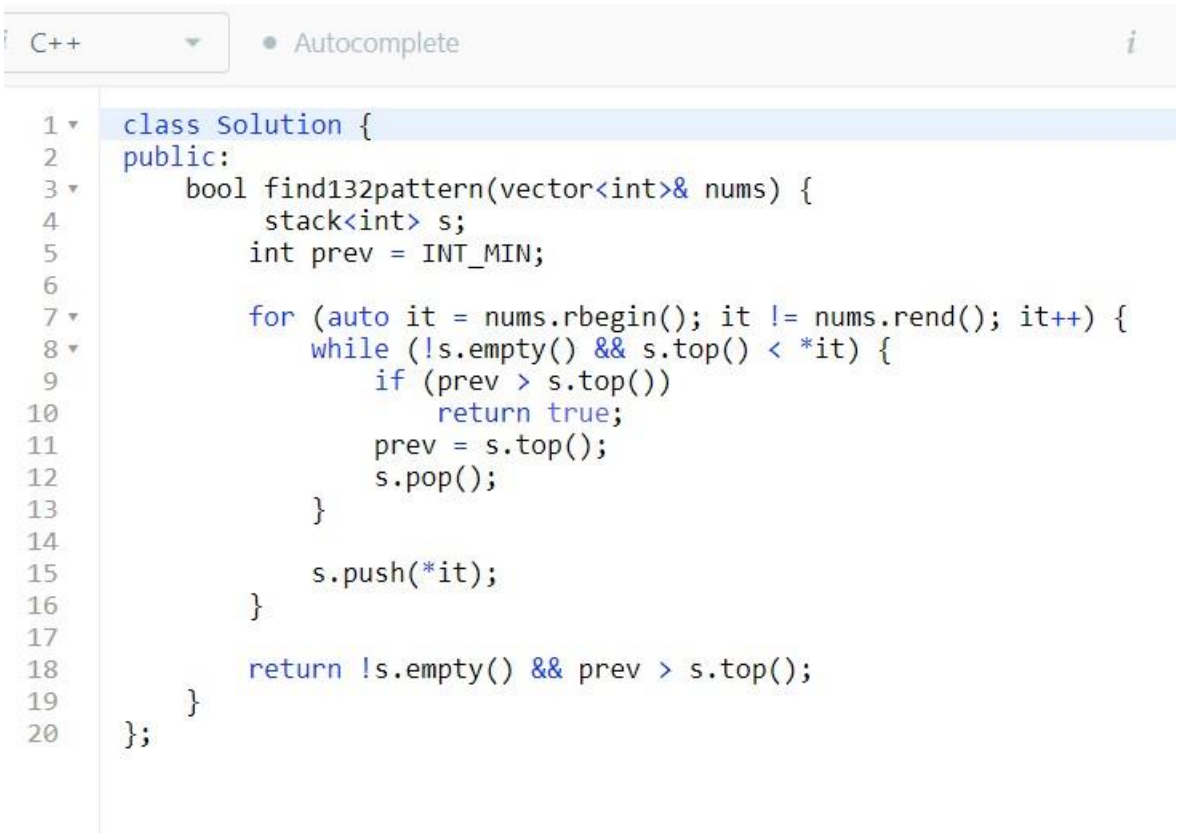
```cpp
class Solution {
public:
    int minInsertions(string s) {
        stack<int> st;
        int count=0;
        for(int i=0;i<s.size();i++)
        {if(s[i]=='(')
            {if(!st.empty() && st.top()==1)
            {count++;
                st.pop();}
                st.push(2);}
            else
            { if(st.empty())
            {count++;
                st.push(1); }
                 else if(st.top()==2)
                { st.top()--;

                }
                else if(st.top()==1)
                {
                    st.pop();
                }
            }
        }
        while(!st.empty())
        {
            count=count+st.top();
            st.pop();
        }
        return count;
    }
};
```

853. Car Fleet



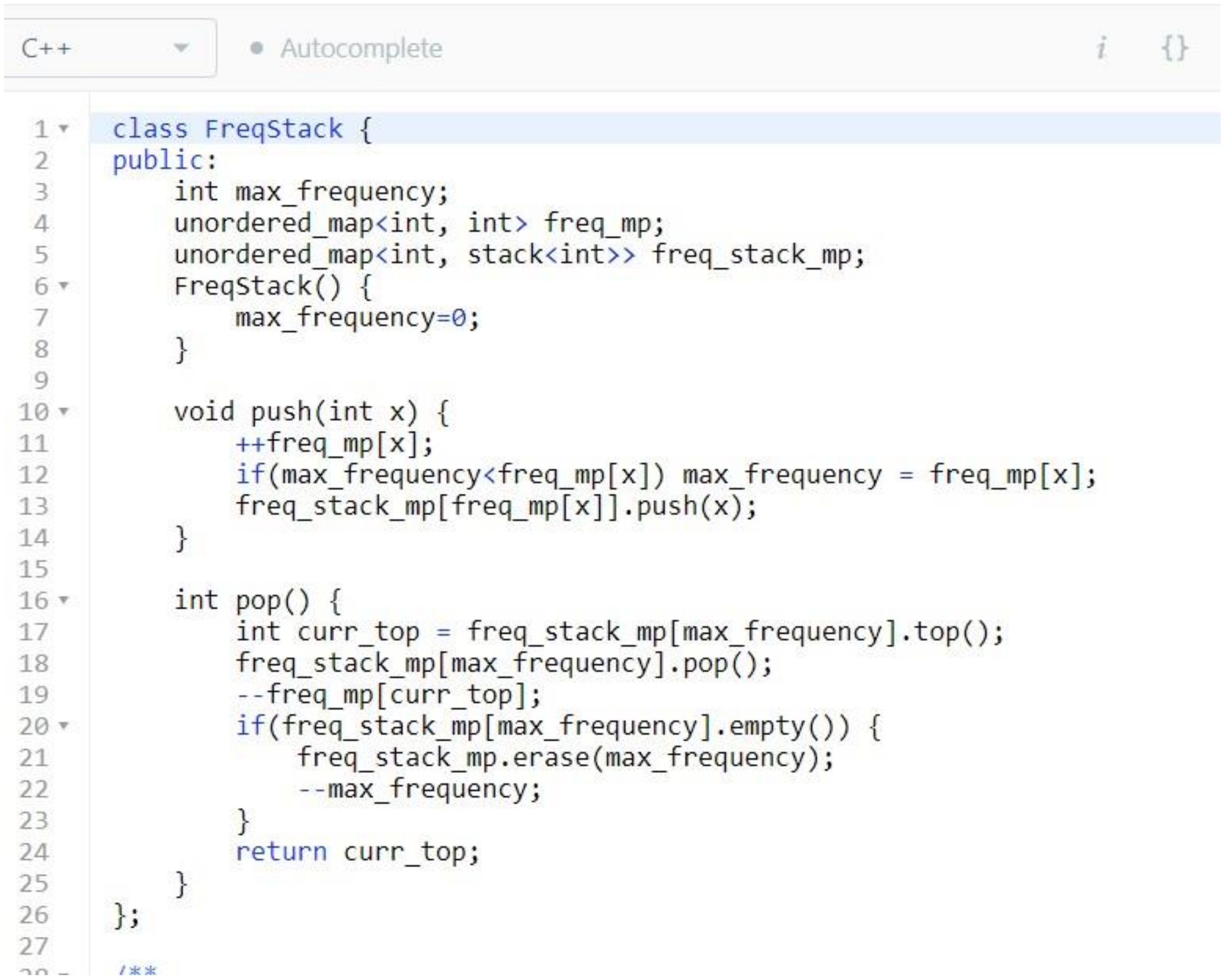
```
class Solution {
public:
    int carFleet(int target, vector<int>& position, vector<int>& speed) {
    vector<pair<int, double>> car(position.size());
        for(int i=0;i<position.size();i++)
        {
            car[i]={position[i], (double)(target-position[i])/speed[i]};
        }
       sort(car.begin(), car.end());
        reverse(car.begin(), car.end());

        int ans=1;
        double tt=car[0].second;


        for(int i=1;i<car.size();i++)
        {
            if(car[i].second>tt)
            {
                ans++;
                tt=car[i].second;
            }
        }
        return ans;
    }
};
```

456. 132 Pattern



```
class Solution {
public:
    bool find132pattern(vector<int>& nums) {
         stack<int> s;
        int prev = INT_MIN;

        for (auto it = nums.rbegin(); it != nums.rend(); it++) {
            while (!s.empty() && s.top() < *it) {
                if (prev > s.top())
                    return true;
                prev = s.top();
                s.pop();
            }

            s.push(*it);
        }

        return !s.empty() && prev > s.top();
    }
};
```

895. Maximum Frequency Stack



C++ ● Autocomplete i {}

```
class FreqStack {
public:
    int max_frequency;
    unordered_map<int, int> freq_mp;
    unordered_map<int, stack<int>> freq_stack_mp;
    FreqStack() {
        max_frequency=0;
    }

    void push(int x) {
        ++freq_mp[x];
        if(max_frequency<freq_mp[x]) max_frequency = freq_mp[x];
        freq_stack_mp[freq_mp[x]].push(x);
    }

    int pop() {
        int curr_top = freq_stack_mp[max_frequency].top();
        freq_stack_mp[max_frequency].pop();
        --freq_mp[curr_top];
        if(freq_stack_mp[max_frequency].empty()) {
            freq_stack_mp.erase(max_frequency);
            --max_frequency;
        }
        return curr_top;
    }
};

/**
```