Problem 1: Vector II

Solution Idea

We need to maintain **multiple independent lists** that support three operations:

1. Append a value to a specific list.
2. Print all elements of a specific list.
3. Clear a specific list.

Since each list can grow dynamically and be cleared at any time, the most suitable data structure is a **dynamic array** (vector) which efficiently handles all operations.

Complexity Analysis

- **Append:** O(1) (vector automatically handles resizing)
- **Print:** O(k) where k is the size of the list
- **Clear:** O(k)

Overall complexity depends on total elements processed across all queries.

Reference Code

```
#include<bits/stdc++.h>
using namespace std ;
#define int long long
#define nl "\n"

signed main(){
    int n , q ;
    cin >> n >> q ;
    vector<vector<int>> a(n) ;
    while( q-- ) {
        int op ;
        cin >> op ;
        if( op == 0 ) {
            int t , x ;
            cin >> t >> x ;
            a[t].push_back(x) ;
        }
        else if( op == 1 ) {
            int t ;
            cin >> t ;
            for( int i = 0 ; i < a[t].size() ; i++ ) {
                cout << a[t][i] ;
                if( i != a[t].size() - 1 )
                    cout << " " ;
            }
            cout << nl ;
        }
        else {
            int t ;
            cin >> t ;
            a[t].clear() ;
        }
    }
    return 0;
}
```

Problem 2: Teacher Queries

Solution Idea:

We must maintain each students' corresponding total marks where:

1. **Type 1:** Add marks to a student.
2. **Type 2:** Erase a student's record.
3. **Type 3:** Print a student's marks (print **0** if not present).

Since we need fast insertion, deletion, and searching by **name (string key)** having **marks(int value)**, the appropriate data structure is:

Map<string, int> of STL which stores **key–value pairs** in **sorted order** and provides efficient lookups.

Complexity Analysis

For each query:

- Insert / Update → O(log n)
- Erase → O(log n)
- Search → O(log n)

Where `n` is the number of distinct students stored.

Reference Code:

```
#include<bits/stdc++.h>
using namespace std ;
#define int long long
#define nl "\n"

signed main(){
    int q ;
    cin >> q ;
    map<string, int> mp ;
    while( q-- ) {
        int t ;
        cin >> t ;
        if( t == 1 ) {
            string x ;
            int y ;
            cin >> x >> y ;
            mp[x] += y ;
        }
        else if( t == 2 ) {
            string x ;
            cin >> x ;
            mp.erase(x) ;
        }
        else {
            string x ;
            cin >> x ;
            cout << ( mp.count(x) ? mp[x] : 0 ) << nl ;
        }
    }
    return 0;
}
```

Problem 3: Structure Balance

Solution Idea:

We must determine whether a string containing only () and [] is **balanced**.

A string is correct if:

1. It is empty.
2. If A and B are correct → AB is correct.
3. If A is correct → (A) and [A] are correct.

This is a classic balanced brackets problem, which can be best solved using a **stack/string**.

For every opening bracket, push it into the stack.
For every closing bracket:

- If the stack is empty → string is invalid.
- Otherwise check if the top matches the closing bracket.
- If not matching → invalid.
  At the end, if the stack is empty → valid, otherwise invalid.

Complexity Analysis

- Each string is processed in **O(n)** time.
- Stack operations are **O(1)** each.

Note

Since the input strings may be empty or contain only brackets, we must use:

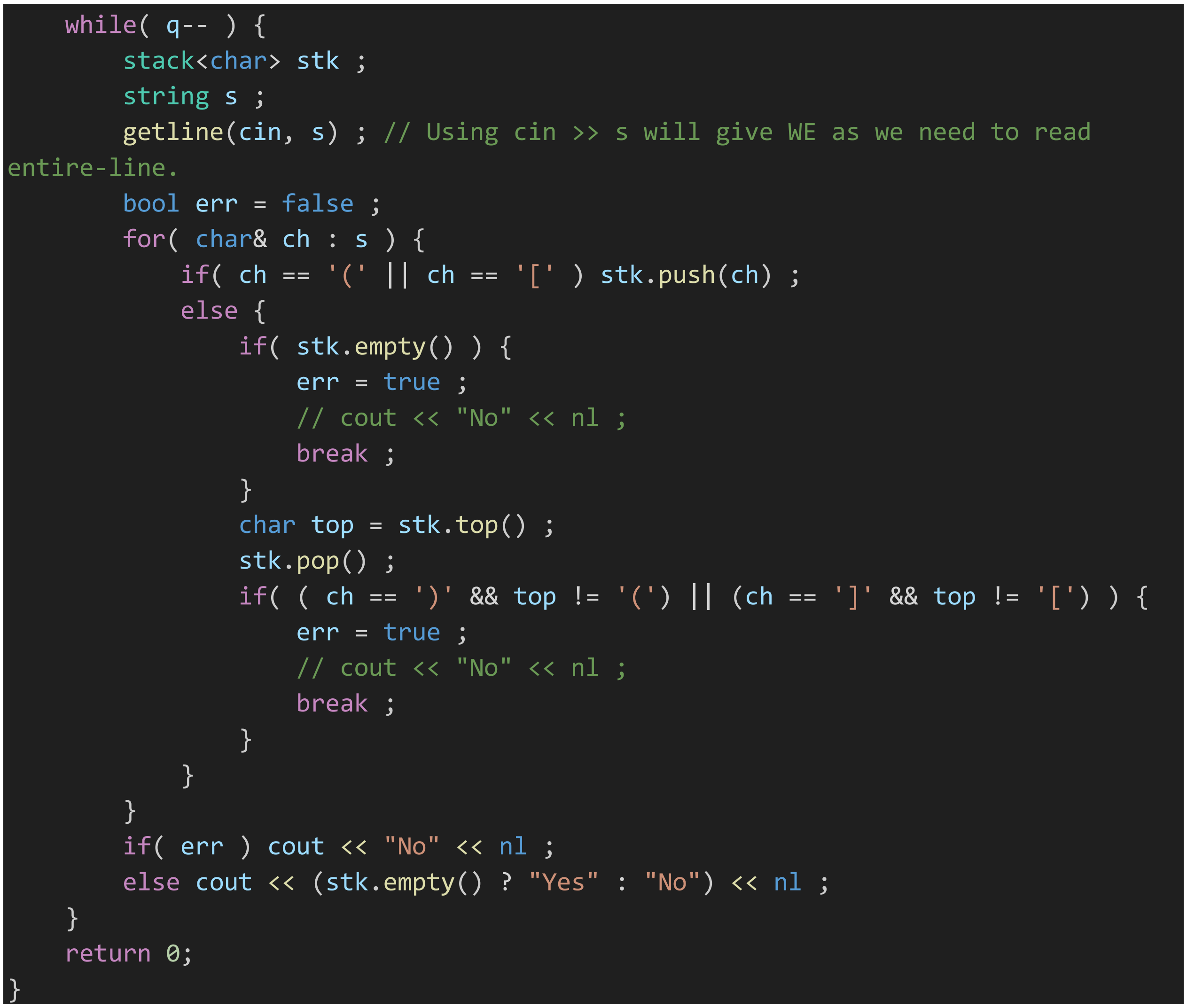
getline(cin, s);

instead of cin >> s to correctly read the full line with empty spaces.

Reference Code:

```
#include<bits/stdc++.h>
using namespace std ;
#define int long long
#define nl "\n"

signed main(){
    int q ;
    cin >> q ;
    cin.ignore() ;
```



```cpp
    while( q-- ) {
        stack<char> stk ;
        string s ;
        getline(cin, s) ; // Using cin >> s will give WE as we need to read
entire-line.
        bool err = false ;
        for( char& ch : s ) {
            if( ch == '(' || ch == '[' ) stk.push(ch) ;
            else {
                if( stk.empty() ) {
                    err = true ;
                    // cout << "No" << nl ;
                    break ;
                }
                char top = stk.top() ;
                stk.pop() ;
                if( ( ch == ')' && top != '(') || (ch == ']' && top != '[') ) {
                    err = true ;
                    // cout << "No" << nl ;
                    break ;
                }
            }
        }
        if( err ) cout << "No" << nl ;
        else cout << (stk.empty() ? "Yes" : "No") << nl ;
    }
    return 0;
}
```

Problem 4: Broken Keyboard

Solution Idea

While typing, two special characters affect the cursor position:

- ' [' → Move cursor to **beginning** (Home key)
- '] ' → Move cursor to **end** (End key)

We must reconstruct the final text efficiently.
Since the string length can be up to 100,000, normal string insertion at the front would be too slow (O(n^2)). So we need a data structure that supports **fast insertion at both ends** and also in between.

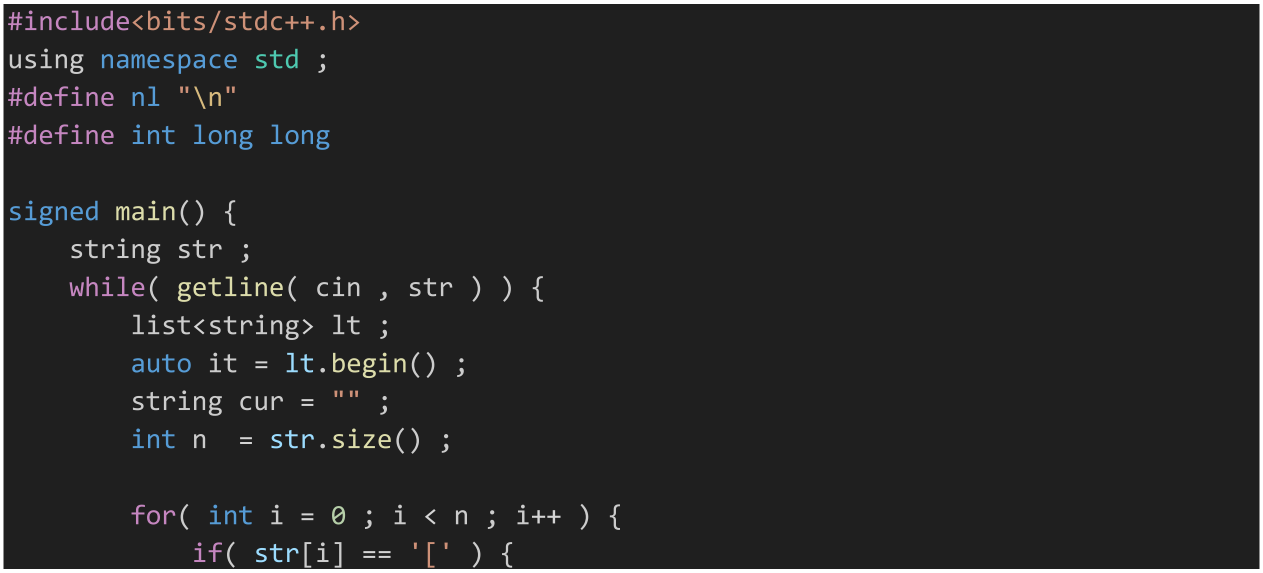
Using list<string> (Doubly Linked List)

Note: You can use **deque<string>** also since its push/pop on both sides is O(1) time. OR you can use your custom **struct Node** based LinkedList also.

- Maintain a list<string>.
- Keep an iterator it to represent the current cursor position.
- When:
  - ‘ [’ → move iterator to begin()
  - ‛] ’ → move iterator to end()
- Insert completed segments at iterator position and move cursor accordingly.

Insertion in STL list is O(1).

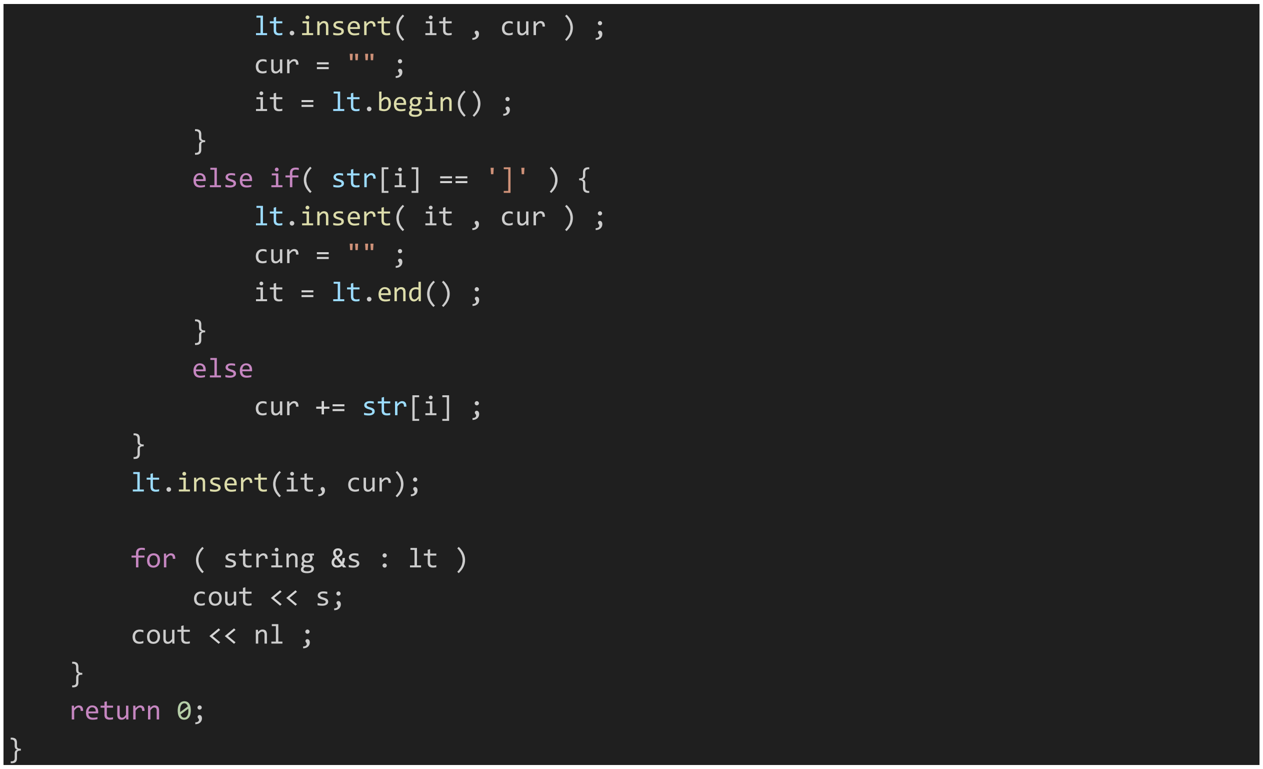
Reference Code:



```
#include<bits/stdc++.h>
using namespace std ;
#define nl "\n"
#define int long long

signed main() {
    string str ;
    while( getline( cin , str ) ) {
        list<string> lt ;
        auto it = lt.begin() ;
        string cur = "" ;
        int n  = str.size() ;

        for( int i = 0 ; i < n ; i++ ) {
            if( str[i] == '[' ) {
```



```
                lt.insert( it , cur ) ;
                cur = "" ;
                it = lt.begin() ;
            }
            else if( str[i] == ']' ) {
                lt.insert( it , cur ) ;
                cur = "" ;
                it = lt.end() ;
            }
            else
                cur += str[i] ;
        }
        lt.insert(it, cur);

        for ( string &s : lt )
            cout << s;
        cout << nl ;
    }
    return 0;
}
```

Problem 5: A Simple Math Problem

Solution Idea

We are given:

- $x + y = a$
- $lcm(x,\ y) = b$

We must determine whether such positive integers x and y exist.

Using the identity → $\mathrm{X} \times \mathrm{Y} = \mathrm{gcd}(\mathrm{X}, \mathrm{Y}) \times \mathrm{lcm}(\mathrm{X}, \mathrm{Y})$; we get the simultaneous quadratic eq as:

$$a^2 - \mathrm{ax} + \mathrm{b} \times \mathrm{gcd}(\mathrm{x}, \mathrm{y}) = 0$$

The discriminant:

$$\mathrm{D} = a^2 - 4\ \times\ \mathrm{b}\ \times\ \mathrm{gcd}\ (\mathrm{x}, \mathrm{y})$$

For valid integer solutions:

1. $D\ \geq\ 0$
2. $a\ \neq 0$ to avoid infinity
3. D must be a **perfect square**
4. The roots must be integers

If these conditions fail → **No Solution**

Otherwise:

$$X = \frac{a - \sqrt{D}}{2}\ ,\ Y = \frac{a + \sqrt{D}}{2}$$

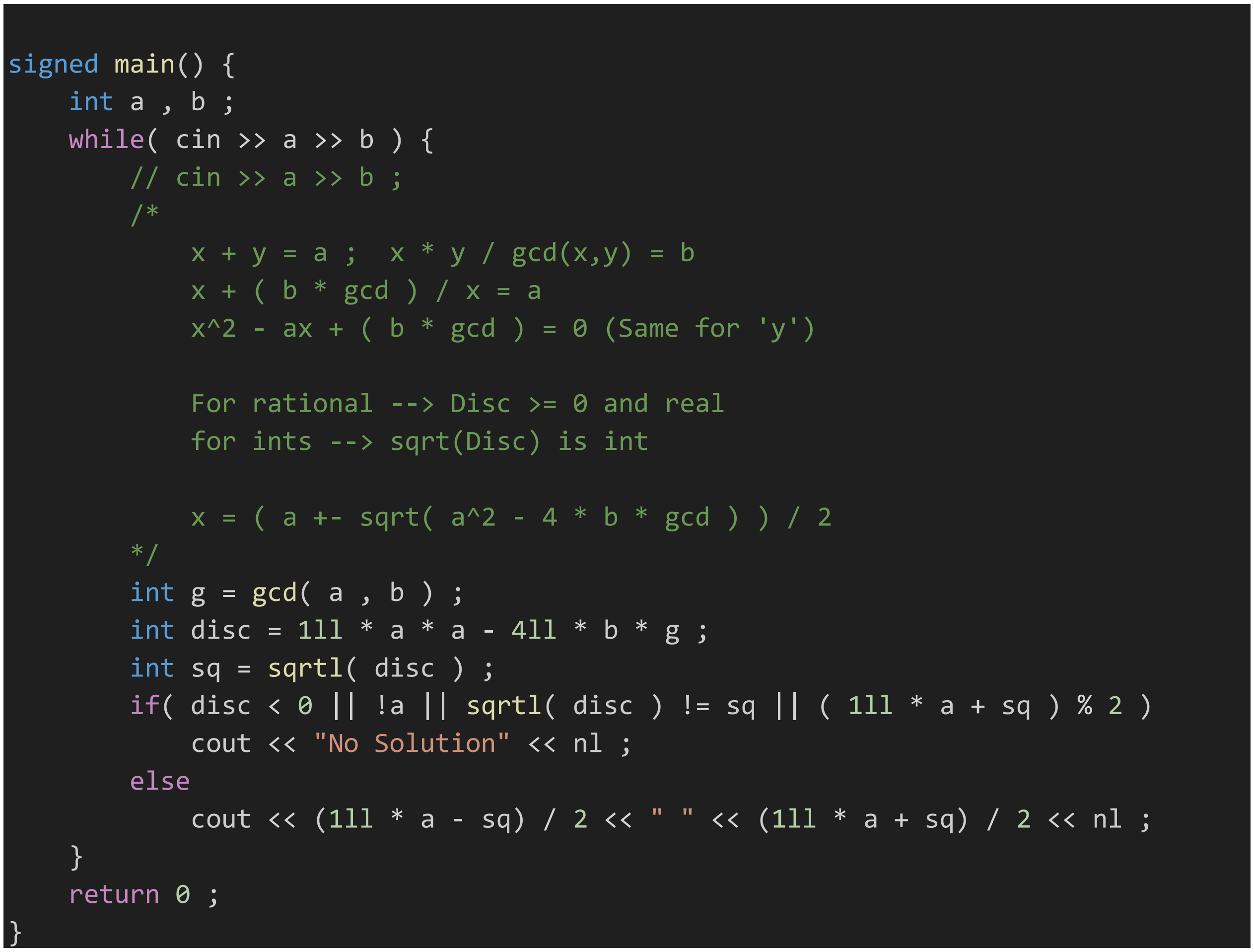
Complexity Analysis

- gcd → O(log min(a,b))

- constant arithmetic checks

Reference Code:

```
#include <bits/stdc++.h>
using namespace std ;
#define int long long
#define nl "\n"
```



```
signed main() {
    int a , b ;
    while( cin >> a >> b ) {
        // cin >> a >> b ;
        /*
            x + y = a ;  x * y / gcd(x,y) = b
            x + ( b * gcd ) / x = a
            x^2 - ax + ( b * gcd ) = 0 (Same for 'y')

            For rational --> Disc >= 0 and real
            for ints --> sqrt(Disc) is int

            x = ( a +- sqrt( a^2 - 4 * b * gcd ) ) / 2
        */
        int g = gcd( a , b ) ;
        int disc = 1ll * a * a - 4ll * b * g ;
        int sq = sqrtl( disc ) ;
        if( disc < 0 || !a || sqrtl( disc ) != sq || ( 1ll * a + sq ) % 2 )
            cout << "No Solution" << nl ;
        else
            cout << (1ll * a - sq) / 2 << " " << (1ll * a + sq) / 2 << nl ;
    }
    return 0 ;
}
```

Problem 6: Berland National Library

Solution Idea

We are given a log of people entering (+ r) and leaving (- r) a reading room.

However:

- The room **may already contain visitors** before logging starts.
- The room **may still contain visitors** after logging ends.
- The log is guaranteed not contradictory.

We must determine the **minimum possible capacity** of the room (maximum number of people inside at any moment).

We simulate the process while tracking:

- cur → current number of people in the room.
- res → maximum number of people ever present (our answer).
- A set to track who is currently inside (Can use array also)

Important Case

If we see: - r and r is not currently inside, that means:

- This person must have been inside **before the system started**.
- So the initial number of people must increase.

Hence, we increment res (minimum required capacity).

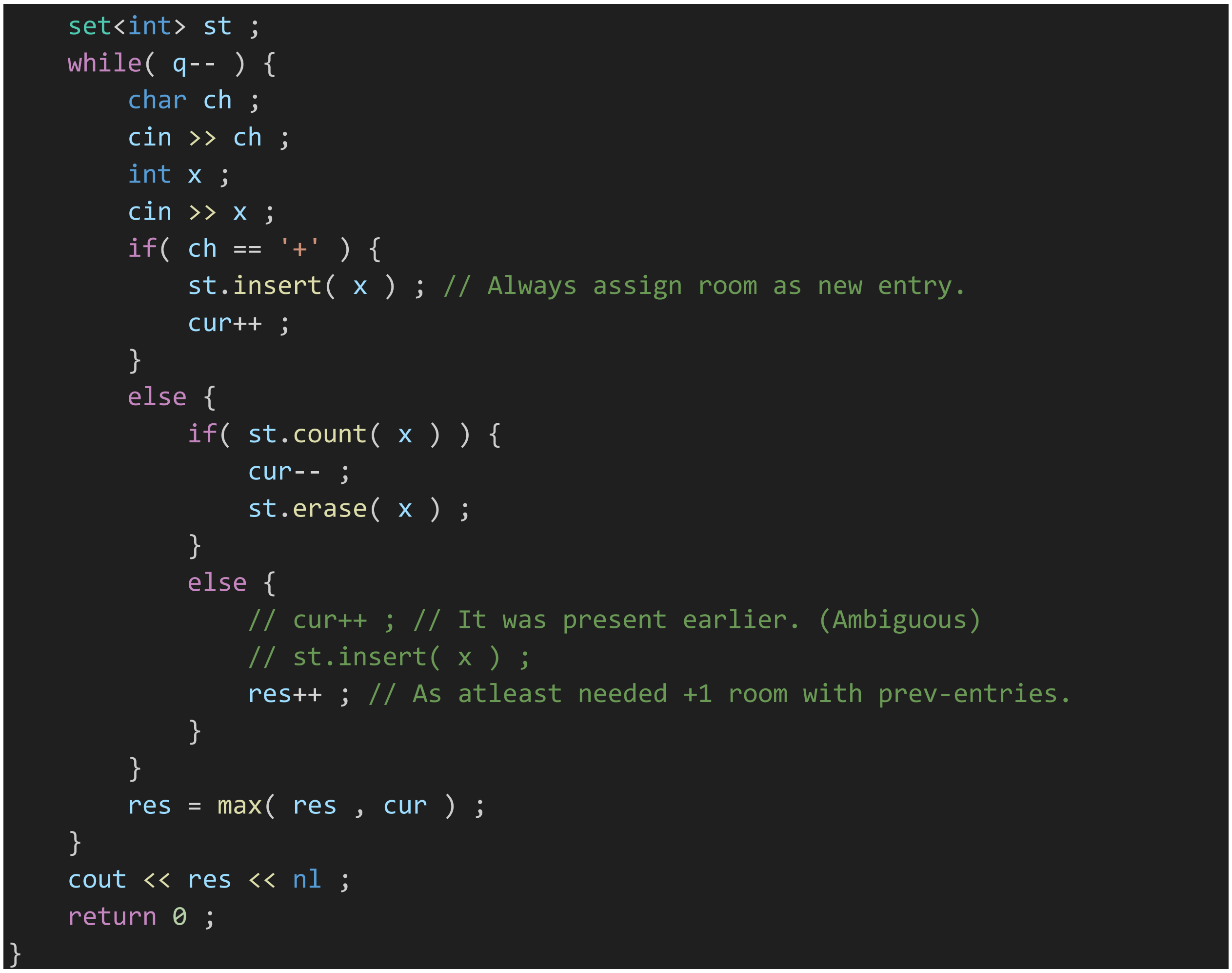
Complexity

- Each operation on set → O(log n)
- Overall complexity: **O(n log n)**

Reference Code:

```
#include <bits/stdc++.h>
using namespace std ;
#define int long long
#define nl "\n"

signed main() {
    int q , cur = 0 , res = 0 ;
    cin >> q ;
```



```cpp
    set<int> st ;
    while( q-- ) {
        char ch ;
        cin >> ch ;
        int x ;
        cin >> x ;
        if( ch == '+' ) {
            st.insert( x ) ; // Always assign room as new entry.
            cur++ ;
        }
        else {
            if( st.count( x ) ) {
                cur-- ;
                st.erase( x ) ;
            }
            else {
                // cur++ ; // It was present earlier. (Ambiguous)
                // st.insert( x ) ;
                res++ ; // As atleast needed +1 room with prev-entries.
            }
        }
        res = max( res , cur ) ;
    }
    cout << res << nl ;
    return 0 ;
}
```