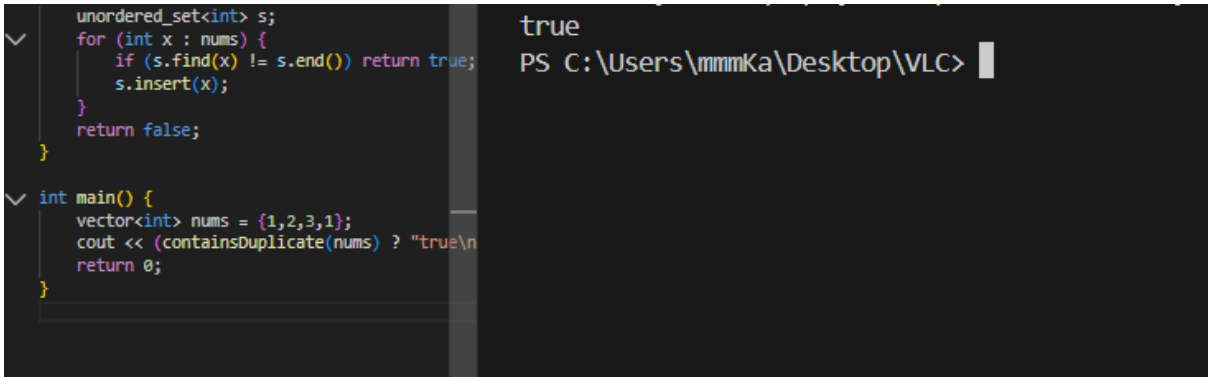
1. Given an integer array, check if it contains any duplicates using hash set

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

bool containsDuplicate(const vector<int>& nums) {
    unordered_set<int> s;
    for (int x : nums) {
        if (s.find(x) != s.end()) return true;
        s.insert(x);
    }
    return false;
}

int main() {
    vector<int> nums = {1,2,3,1};
    cout << (containsDuplicate(nums) ? "true\n" : "false\n");
    return 0;
}
```

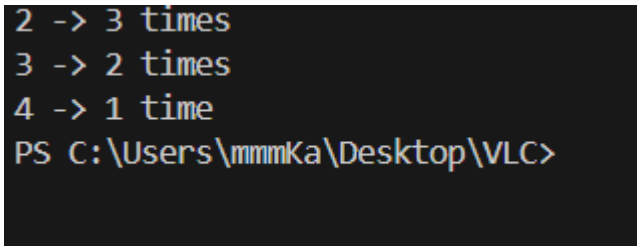


2. Given two arrays, find the common elements using a hash set.

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

vector<int> commonElements(const vector<int>& A, const vector<int>& B) {
    unordered_set<int> sa(A.begin(), A.end());
    unordered_set<int> res;
    for (int x : B) if (sa.count(x)) res.insert(x);
    return vector<int>(res.begin(), res.end());
}

int main() {
    vector<int> A = {1,2,3,4}, B = {3,4,5,6};
    auto c = commonElements(A, B);
    sort(c.begin(), c.end());
    for (size_t i = 0; i < c.size(); ++i) {
        if (i) cout << ", ";
        cout << c[i];
    }
    cout << "\n";
    return 0;
}
```

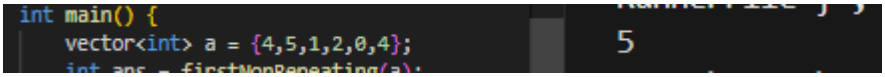
3. Count the frequency of each number in an array using a hash map

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

int main() {
   vector<int> nums = {2,3,2,4,3,2};
   unordered_map<int,int> freq;
   for (int x : nums) freq[x]++;
   // Print in increasing key order for neatness:
   vector<int> keys;
   for (auto &p : freq) keys.push_back(p.first);
   sort(keys.begin(), keys.end());
   for (int k : keys) {
      cout << k << " -> " << freq[k] << (freq[k] == 1 ? " time\n" : " times\n");
   }
   return 0;
}
```



4. Find the first non-repeating element in an array using a hash map.

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

int firstNonRepeating(const vector<int>& a) {
   unordered_map<int,int> cnt;
   for (int x : a) cnt[x]++;
   for (int x : a) if (cnt[x] == 1) return x;
   return INT_MIN; // indicates no non-repeating element
}

int main() {
   vector<int> a = {4,5,1,2,0,4};
   int ans = firstNonRepeating(a);
   if (ans == INT_MIN) cout << "No non-repeating element\n";
   else cout << ans << "\n";
   return 0;
}
```



5. Given a linked list, determine whether it contains a loop (cycle) using a hash set. A loop exists if some node's next pointer points to a previous node in the list

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode(int v): val(v), next(nullptr) {}
};

// Using hash set of pointers
bool hasCycle(ListNode* head) {
    unordered_set<ListNode*> seen;
    for (ListNode* cur = head; cur; cur = cur->next) {
        if (seen.count(cur)) return true;
```

```
        seen.insert(cur);
    }
    return false;
}

int main() {
    // Example 1->2->3->4->(back to 2)
    ListNode* n1 = new ListNode(1);
    ListNode* n2 = new ListNode(2);
    ListNode* n3 = new ListNode(3);
    ListNode* n4 = new ListNode(4);
    n1->next = n2; n2->next = n3; n3->next = n4; n4->next = n2; // cycle
    cout << (hasCycle(n1) ? "true\n" : "false\n");
    // Note: memory leak is fine for this short demo
    return 0;
}
```