Name  : Rudraksh Mishra
UID   : 22BCS10607
Class : KPIT - 901 / A

Q1. Given a string s, find the first non-repeating character in it and return its index. If it does not exist, return -1.

Q2. A bracket is considered to be any one of the following characters:
- ( , )
- { , }
- [ , ]

Two brackets are considered to be a matched pair if an opening bracket i.e., (, [, or { occurs to the left of a closing bracket i.e., ), ], or } of the exact same type. There are three types of matched pairs of brackets: [], {}, and (). A matching pair of brackets is not balanced if the set of brackets it encloses are not matched.

For example, {[(])} is not balanced because the contents in between { and } are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, (, and the pair of parentheses encloses a single, unbalanced closing square bracket, ]. By this logic, we say a sequence of brackets is balanced if the following conditions are met:
- It contains no unmatched brackets.
- The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Given n strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, return YES. Otherwise, return NO.

Q3. Given a queue, write a recursive function to reverse it.
Standard operations allowed :
- Enqueue(x) : Add an item ‘x’ to rear of queue.
- dequeue() : Remove an item from front of queue.
- empty() : Checks if a queue is empty or not.

Q4. You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Return the max sliding window.

Q5. You are given an integer array nums of length n and an integer array called queries. Let gcdPairs denote an array obtained by calculating the GCD of all possible pairs (nums[i], nums[j]), where 0 <= i < j < n, and then sorting these values in ascending order. For each query queries[i], you need to find the element at index queries[i] in gcdPairs. Return an integer array answer, where answer[i] is the value at gcdPairs[queries[i]] for each query. The term gcd(a, b) denotes the greatest common divisor of a and b.

Solutions :

A1. First non-repeating character in string

```
#include <iostream>
#include <unordered_map>
#include <algorithm>

using namespace std;

void non_repeating(string s) {
    unordered_map<char, int> mp;
    for (const auto& c : s) {
        mp[c]++;
    }

    for (const auto& c : s) {
        if (mp[c] == 1) {
            auto itr = find(s.begin(), s.end(), c);
            cout << itr - s.begin() << endl;
            break;
        }
    }

    cout << -1;
}

int main(int argc, char* argv[]) {
    string s = "aabb";
    non_repeating(s);
    return 0;
}
```

Output :

```
-1
0
2
```

A2. Brackets balance

```
#include <iostream>
#include <stack>

using namespace std;

bool is_balanced(string s) {
    stack<char> stk;

    for (const auto& c : s) {
        if (c == '(' || c == '[' || c == '{') {
            stk.push(c);
        } else if (c == ')') {
            if (stk.empty() || stk.top() != '(') return false;
            stk.pop();
        } else if (c == ']') {
```

```
            if (stk.empty() || stk.top() != '[') return false;
            stk.pop();
        } else if (c == '}') {
            if (stk.empty() || stk.top() != '{') return false;
            stk.pop();
        }
    }

    if (!stk.empty()) { return false; }

    return true;
}

int main(int argc, char* argv[]) {
    string s = "([{}])";
    cout << is_balanced(s) << endl;
    s = "([{}]";
    cout << is_balanced(s) << endl;
    s = "{[(])}";
    cout << is_balanced(s);
    return 0;
}
```

Output :

```
([{}])  Balanced ? : True
([{}]   Balanced ? : False
{[(])}  Balanced ? : False
```

A3. Reverse a queue recursively

```
#include <iostream>
#include <queue>

using namespace std;

void r_reverse_queue(queue<int>& q) {
    if (q.empty()) { return; }

    int t = q.front();
    q.pop();
    r_reverse_queue(q);
    q.push(t);
}

int main(int argc, char* argv[]) {
    queue<int> q;
    q.push(1); q.push(2); q.push(3); q.push(4); q.push(5);

    r_reverse_queue(q);

    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
    return 0;
}
```

Output :

```
5 4 3 2 1
```

A4. Maximum value in sliding window

```
#include <vector>
#include <deque>
#include <iostream>

using namespace std;

vector<int> max_sliding_windows(vector<int>& nums, int k) {
    vector<int> res;
    deque<int> dq;

    for (int i = 0; i < nums.size(); ++i) {
        if (!dq.empty() && dq.front() == i - k) { dq.pop_front(); }

        while (!dq.empty() && nums[dq.back()] < nums[i]) { dq.pop_back(); }
        dq.push_back(i);

        if (i >= k - 1) { res.push_back(nums[dq.front()]); }
    }

    return res;
}

int main() {
    vector<int> nums = {1, 3, -1, -3, 5, 3, 6, 7};
    int k = 3;

    vector<int> result = max_sliding_windows(nums, k);

    for (int val : result) {
        cout << val << " ";
    }

    return 0;
}
```

Output :

```
3 3 5 5 6 7
```

A5. Sorted GCD Pair Queries

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

vector<int> gcdValues(vector<int>& nums, vector<long long>& queries) {
        int mx = *max_element(nums.begin(), nums.end());
        vector<int>        cnt(mx + 1);
        vector<long long> cntG(mx + 1);
```

```
        for (int x : nums) { ++cnt[x]; }
        for (int i = mx; i; --i) {
            long long v = 0;
            for (int j = i; j <= mx; j += i) {
                v += cnt[j];
                cntG[i] -= cntG[j];
            }
            cntG[i] += 1LL * v * (v - 1) / 2;
        }
        for (int i = 2; i <= mx; ++i) { cntG[i] += cntG[i - 1]; }
        vector<int> ans;
        for (auto&& q : queries) {
            ans.push_back(
                upper_bound(
                    cntG.begin(),
                    cntG.end(), q
                ) - cntG.begin()
            );
        }
        return ans;
    }

int main(int argc, char* argv[]) {
    vector<int> nums          = {4, 4, 2, 1};
    vector<long long> queries = {5, 3, 1, 0};

    vector<int> result = gcdValues(nums, queries);

    cout <<   "Nums    : ";
    for (int val : nums)    { cout << val << " "; };
    cout << "\nQueries : ";
    for (int val : queries) { cout << val << " "; };
    cout << "\nResult  : ";
    for (int val : result)  { cout << val << " "; };

    return 0;
}
```

Output :

```
Nums    : 4 4 2 1
Queries : 5 3 1 0
Result  : 4 2 1 1
```