# 15B17CI371 – Data Structures Lab

# Week 2-LAB B

# Q1. Given a string ‘s’ containing just the characters '(', ')', '{', '}', '[' and ']',

# determine if the input string is valid. Assume that the string can contain

# parentheses only i.e., only '()[]{}' characters are allowed; and the maximum

# length of string can be, say, 20 characters.

# An input string is valid if:

# 1. Open brackets must be closed by the same type of brackets.

# 2. Open brackets must be closed in the correct order.

# 3. Every close bracket has a corresponding open bracket of the

# same type.

# Example 1:

# Input: s = "()"

# Output: true

# Example 2:

# Input: s = "()[]{}"

# Output: true

# Example 3:

# Input: s = “((]”

# Output: false

# Q2. Given an array of numbers, input one number from this array and find

# if it’s next-greater-element exists to the right of this value in the array. If

# yes, give the position, else print “Not found”. Use stacks to perform this

# operation.

# Example 1:

# Input: arr[] = {1,4,2,5,0,6,7}

# Input: element= 4

# Output: 2 (element 5 is the first next-greater-element which exists to

# the right of the given element in the array, and it is 2 positions far

# from 4).

# Example 2:

# Input: arr[] = {1,4,2,5,0,6,7}

# Input: element= 2

# Output: 1 (element 5 is the first next-greater-element which exists to

# the right of the given element in the array, and it is 1 position far from

# 2).

# Example 3:

# Input: arr[] = {10,4,2,5,0,6,7}

# Input: element= 7

# Output: “Not found” (There are no element to the right of 7 in this

# array)

# Example 4:

# Input: arr[] = {10,6,7,2,5,1,0,4}

# Input: element= 7

# Output: “Not found” (There are elements to the right of 7, but none of

# them are greater than 7).

# Q3. Modify the above question to allow for circular search.

# Example 1:

# Input: arr[] = {1,4,2,5,0,6,7}

# Input: element= 4

# Output: 2 (element 5 is the first next-greater-element which exists to

# the right of the given element in the array, and it is 2 positions far

# from 4).

# Example 2:

# Input: arr[] = {1,4,2,5,0,6,7}

# Input: element= 2

# Output: 1 (element 5 is the first next-greater-element which exists to

# the right of the given element in the array, and it is 1 position far from

# 2).

# Example 3:

# Input: arr[] = {10,4,2,5,0,6,7}

# Input: element= 7

# Output: 1 (Since you are allowing circular search, once the array

# reaches the end, start from the beginning. There now exists 10 which

# is greater than 7 and it can be said to travel 1 hop after 7).

# Example 4:

# Input: arr[] = {10,6,7,2,5,1,0,4}

# Input: element= 7

# Output: 6 (Since you are allowing circular search, once the array

# reaches the end, start from the beginning. There now exists the value

# 10 which is greater than 7 and it can be said to travel 6 hops to reach

# the position where currently 10 resides).

# Q4. You are given a string ‘s’, find the first non-repeating character (a

# character that occurs only once) in it and return its index (position). If it

# does not exist, return -1. Implement using concepts of queues.

# Example 1:

# Input: s = "thisisDSlab"

# Output: Character: t

# Index: 0

# Example 2:

# Input: s = "CodeForDSlabClass"

# Output: Character: d

# (Assumption - ‘d’ and ‘D’ are considered as different characters)

# Index: 2

# Example 3:

# Input: s = "The quick brown fox jumps over a lazy dog"

# Output: Character: None

# Index: -1

# Q1

#include <iostream>

#include <stack>

using namespace std;

bool isPair(char open, char close) {

if (open == '(' && close == ')') return true;

if (open == '{' && close == '}') return true;

if (open == '[' && close == ']') return true;

return false;

}

bool isValid(string s) {

stack<char> st;

for (char c : s) {

if (c == '(' || c == '{' || c == '[') {

st.push(c);

} else {

if (st.empty() || !isPair(st.top(), c)) return false;

st.pop();

}

}

return st.empty();

}

int main() {

string s;

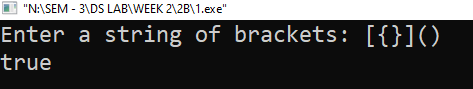
cout << "Enter a string of brackets: ";

cin >> s;

cout << (isValid(s) ? "true" : "false") << endl;

}

Output:



# Q2

#include <iostream>

#include <stack>

#include <vector>

using namespace std;

int ng(vector<int>& a, int x) {

stack<int> s;

int i = -1;

for (int j = a.size() - 1; j >= 0; --j) {

while (!s.empty() && s.top() <= a[j]) s.pop();

if (a[j] == x) {

if (!s.empty()) {

i = j;

break;

} else {

return -1;

}

}

s.push(a[j]);

}

if (i != -1) {

for (int j = i + 1; j < a.size(); ++j) {

if (a[j] == s.top()) {

return j;

}

}

}

return -1;

}

int main() {

int n, x;

cout << "Enter n: ";

cin >> n;

vector<int> a(n);

cout << "Enter elements: ";

for (int i = 0; i < n; ++i) cin >> a[i];

cout << "Enter x: ";

cin >> x;

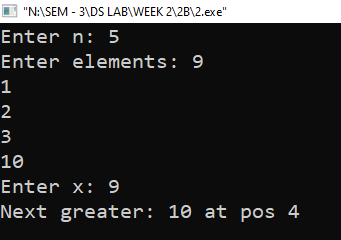
int r = ng(a, x);

if (r != -1) cout << "Next greater: " << a[r] << " at pos " << r << endl;

else cout << "Not found" << endl;

}

Output:



# Q3

#include <iostream>

#include <stack>

#include <vector>

using namespace std;

int ngc(vector<int>& a, int x) {

stack<int> s;

int n = a.size();

int idx = -1;

for (int i = 2 \* n - 1; i >= 0; --i) {

while (!s.empty() && s.top() <= a[i % n]) s.pop();

if (a[i % n] == x) {

idx = i % n;

break;

}

s.push(a[i % n]);

}

if (idx != -1) {

for (int j = idx + 1; j < idx + n; ++j) {

if (a[j % n] > x) {

return j % n;

}

}

}

return -1;

}

int main() {

int n, x;

cout << "Enter n: ";

cin >> n;

vector<int> a(n);

cout << "Enter elements: ";

for (int i = 0; i < n; ++i) cin >> a[i];

cout << "Enter x: ";

cin >> x;

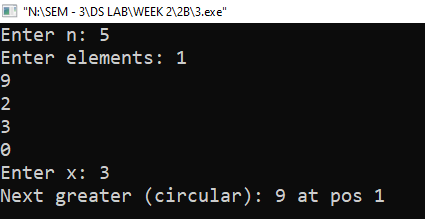
int r = ngc(a, x);

if (r != -1) cout << "Next greater (circular): " << a[r] << " at pos " << r << endl;

else cout << "Not found" << endl;

}

Output:



# Q4

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

pair<char, int> fNR(const string& s) {

const int C = 256;

vector<int> cnt(C, 0);

queue<pair<char, int>> q;

for (int i = 0; i < s.size(); ++i) {

cnt[s[i]]++;

q.push({s[i], i});

}

while (!q.empty()) {

auto f = q.front();

q.pop();

if (cnt[f.first] == 1) {

return f;

}

}

return make\_pair('N', -1);

}

int main() {

string s;

cout << "Enter a string: ";

cin >> s;

auto res = fNR(s);

if (res.first != 'N')

cout << "First non-repeating character: " << res.first << " at index " << res.second << endl;

else

cout << "No non-repeating character found" << endl;

}

Output:

