**15B17CI371 – Data Structures Lab**

**ODD 2024**

**Week 2-LAB B**

**Practice Lab**

**[CO: C270.1]**

Q1.

#include <iostream>

#include <string>

using namespace std;

bool isValid(const string& s) {

char stack[100];

int top = -1;

char opening[] = "({[";

char closing[] = ")}]";

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

char c = s[i];

bool isOpening = false;

for (int j = 0; j < 3; ++j) {

if (c == opening[j]) {

stack[++top] = c;

isOpening = true;

break;

}

}

if (!isOpening) {

bool isValid = false;

for (int j = 0; j < 3; ++j) {

if (c == closing[j]) {

if (top == -1 || stack[top] != opening[j]) {

return false;

}

top--;

isValid = true;

break;

}

}

if (!isValid) {

return false;

}

}

}

return top==-1;

}

int main()

{

string s1 = "()";

string s2 = "()[]{}";

string s3 = "((]";

cout << boolalpha;

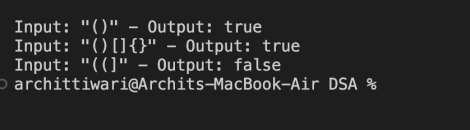
cout << "Input: \"" << s1 << "\" - Output: " << isValid(s1) << endl;

cout << "Input: \"" << s2 << "\" - Output: " << isValid(s2) << endl;

cout << "Input: \"" << s3 << "\" - Output: " << isValid(s3) << endl;

return 0;

}



2.

#include <iostream>

using namespace std;

struct Node {

int data;

int index;

Node\* next;

};

class LinkedListStack {

private:

Node\* top;

public:

LinkedListStack() : top(nullptr) {}

void push(int value, int index) {

Node\* newNode = new Node();

newNode->data = value;

newNode->index = index;

newNode->next = top;

top = newNode;

}

Node\* pop() {

if (isEmpty()) {

return nullptr;

}

Node\* temp = top;

top = top->next;

return temp;

}

bool isEmpty() const {

return top == nullptr;

}

Node\* peek() const {

if (isEmpty()) {

return nullptr;

}

return top;

}

~LinkedListStack() {

while (!isEmpty()) {

Node\* temp = pop();

delete temp;

}

}

};

int findNextGreaterElementPosition(const int arr[], int size, int element) { LinkedListStack stack;

int nextGreater[size];

int elementIndex = -1;

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

nextGreater[i] = -1;

}

for (int i = size - 1; i >= 0; --i) {

while (!stack.isEmpty() && stack.peek()->data <= arr[i]) { stack.pop();

}

if (!stack.isEmpty()) {

nextGreater[i] = stack.peek()->index;

}

stack.push(arr[i], i);

}

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

if (arr[i] == element) {

elementIndex = i;

break;

}

}

if (elementIndex == -1) {

return -1;

}

int nextGreaterIndex = nextGreater[elementIndex]; if (nextGreaterIndex != -1) {

return nextGreaterIndex - elementIndex - 1;

} else {

return -1;

}

}

int main() {

int arr1[] = {1, 4, 2, 5, 0, 6, 7};

int size1 = sizeof(arr1) / sizeof(arr1[0]);

int element1 = 4;

int result1 = findNextGreaterElementPosition(arr1, size1, element1); if (result1 != -1) {

cout << "Output: " << result1 << endl;

} else {

cout << "Output: Not found" << endl;

}

int arr2[] = {1, 4, 2, 5, 0, 6, 7};

int size2 = sizeof(arr2) / sizeof(arr2[0]);

int element2 = 2;

int result2 = findNextGreaterElementPosition(arr2, size2, element2); if (result2 != -1) {

cout << "Output: " << result2 << endl;

} else {

cout << "Output: Not found" << endl;

}

int arr3[] = {10, 4, 2, 5, 0, 6, 7};

int size3 = sizeof(arr3) / sizeof(arr3[0]);

int element3 = 7;

int result3 = findNextGreaterElementPosition(arr3, size3, element3); if (result3 != -1) {

cout << "Output: " << result3 << endl;

} else {

cout << "Output: Not found" << endl;

}

int arr4[] = {10, 6, 7, 2, 5, 1, 0, 4};

int size4 = sizeof(arr4) / sizeof(arr4[0]);

int element4 = 7;

int result4 = findNextGreaterElementPosition(arr4, size4, element4); if (result4 != -1) {

cout << "Output: " << result4 << endl;

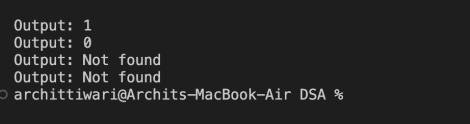
} else {

cout << "Output: Not found" << endl;

}

return 0;

}

3.

#include <iostream>

using namespace std;

struct Node {

int data;

int index;

Node\* next;

};

class CircularLinkedListStack {

private:

Node\* top;

Node\* tail;

public:

CircularLinkedListStack() : top(nullptr), tail(nullptr) {}

void push(int value, int index) {

Node\* newNode = new Node();

newNode->data = value;

newNode->index = index;

if (top == nullptr) {

top = newNode;

tail = newNode;

newNode->next = top;

} else {

newNode->next = top;

top = newNode;

tail->next = top;

}

}

Node\* pop() {

if (isEmpty()) {

return nullptr;

}

Node\* temp = top;

if (top == tail) {

top = nullptr;

tail = nullptr;

} else {

tail->next = top->next;

top = top->next;

}

return temp;

}

bool isEmpty() const {

return top == nullptr;

}

Node\* peek() const {

if (isEmpty()) {

return nullptr;

}

return top;

}

~CircularLinkedListStack() {

while (!isEmpty()) {

Node\* temp = pop();

delete temp;

}

}

};

int findNextGreaterElementPosition(const int arr[], int size, int element) { CircularLinkedListStack stack;

int nextGreater[size];

int elementIndex = -1;

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

nextGreater[i] = -1;

}

for (int i = size - 1; i >= 0; --i) {

while (!stack.isEmpty() && stack.peek()->data <= arr[i]) { stack.pop();

}

if (!stack.isEmpty()) {

nextGreater[i] = stack.peek()->index;

}

stack.push(arr[i], i);

}

for (int i = size - 1; i >= 0; --i) {

if (nextGreater[i] == -1) {

int j = (i + size - 1) % size;

while (j != i) {

if (arr[j] > arr[i]) {

nextGreater[i] = j;

break;

}

j = (j + size - 1) % size;

}

}

}

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

if (arr[i] == element) {

elementIndex = i;

break;

}

}

if (elementIndex == -1) {

return -1;

}

int nextGreaterIndex = nextGreater[elementIndex]; if (nextGreaterIndex != -1) {

return (nextGreaterIndex - elementIndex + size) % size - 1; } else {

return -1;

}

}

int main() {

int arr1[] = {1, 4, 2, 5, 0, 6, 7};

int size1 = sizeof(arr1) / sizeof(arr1[0]);

int element1 = 4;

int result1 = findNextGreaterElementPosition(arr1, size1, element1); if (result1 != -1) {

cout << "Output: " << result1 << endl;

} else {

cout << "Output: Not found" << endl;

}

int arr2[] = {1, 4, 2, 5, 0, 6, 7};

int size2 = sizeof(arr2) / sizeof(arr2[0]);

int element2 = 2;

int result2 = findNextGreaterElementPosition(arr2, size2, element2); if (result2 != -1) {

cout << "Output: " << result2 << endl;

} else {

cout << "Output: Not found" << endl;

}

int arr3[] = {10, 4, 2, 5, 0, 6, 7};

int size3 = sizeof(arr3) / sizeof(arr3[0]);

int element3 = 7;

int result3 = findNextGreaterElementPosition(arr3, size3,

element3); if (result3 != -1) {

cout << "Output: " << result3 << endl;

} else {

cout << "Output: Not found" << endl;

}

int arr4[] = {10, 6, 7, 2, 5, 1, 0, 4};

int size4 = sizeof(arr4) / sizeof(arr4[0]);

int element4 = 7;

int result4 = findNextGreaterElementPosition(arr4, size4,

element4); if (result4 != -1) {

cout << "Output: " << result4 << endl;

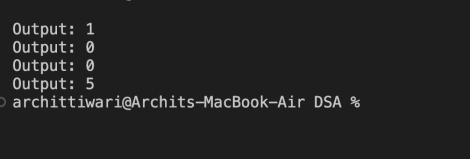
} else {

cout << "Output: Not found" << endl;

}

return 0;

}

4.

#include <iostream>

#include <string>

using namespace std;

const int MAX\_CHARS = 256;

struct Queue {

char data[MAX\_CHARS];

int front = 0;

int rear = 0;

void enqueue(char c) {

if (rear < MAX\_CHARS) {

data[rear++] = c;

}

}

char dequeue() {

if (front == rear) {

return '\0';

}

return data[front++];

}

bool isEmpty() {

return front == rear;

}

char peek() {

if (isEmpty()) {

return '\0';

}

return data[front];

}

};

int findFirstNonRepeatingCharacter(string& s) {

int frequency[MAX\_CHARS] = {0};

Queue q;

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

char c = s[i];

frequency[c]++;

q.enqueue(c);

}

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

char c = q.dequeue();

if (frequency[c] == 1) {

return i;

}

}

return -1;

}

int main() {

string s1 = "thisisDSlab";

string s2 = "CodeForDSlabClass";

string s3 = "The quick brown fox jumps over a lazy dog";

cout << "Input: \"" << s1 << "\" - ";

int index1 = findFirstNonRepeatingCharacter(s1);

if (index1 != -1) {

cout << "Character: " << s1[index1] << ", Index: " << index1 << endl; } else {

cout << "Character: None, Index: -1" << endl;

}

cout << "Input: \"" << s2 << "\" - ";

int index2 = findFirstNonRepeatingCharacter(s2);

if (index2 != -1) {

cout << "Character: " << s2[index2] << ", Index: " << index2 << endl;

} else {

cout << "Character: None, Index: -1" << endl;

}

cout << "Input: \"" << s3 << "\" - ";

int index3 = findFirstNonRepeatingCharacter(s3);

if (index3 != -1) {

cout << "Character: " << s3[index3] << ", Index: " << index3 << endl;

}else{

cout<<"Character: None, Index: -1"<<endl;

}

return 0;

}

