Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Năm, 29 tháng 2 2024, 5:29 PM
Kết thúc lúc	Chủ Nhật, 10 tháng 3 2024, 5:14 PM
Thời gian thực hiện	9 Các ngày 23 giờ

Câu hỏi **1**
Đúng
Đạt điểm 1,00

You are keeping score for a basketball game with some new rules. The game consists of several rounds, where the scores of past rounds may affect future rounds' scores.

At the beginning of the game, you start with an empty record. You are given a list of strings **ops**, where **ops[i]** is the operation you must apply to the record, with the following rules:

- A non-negative integer **x (from 0 to 9)** - record a new score of **x**
- **'+'** - Record a new score that is the sum of the previous two scores. It is guaranteed there will always be two previous scores.
- **'D'** - Record a new score that is double the previous score. It is guaranteed there will always be a previous score.
- **'C'** - Invalidate the previous score, removing it from the record. It is guaranteed there will always be a previous score.

Finally, return the sum of all scores in the record.

For example:

ops = "52CD+"

- '5' - add to the record. Record now is [5]
- '2' - add to the record. Record now is [5,2]
- 'C' - invalid the previous score (2). Record now is [5]
- 'D' - Record new score that is double of previous score (5*2). Record now is [5,10]
- '+' - Record a new score that is the sum of the previous two scores. Record now is [5,10,15]

Return the sum: 5+10+15 = 30

For example:

Test	Result
`cout << baseballScore("52CD+");`	30
`cout << baseballScore("524CD9++");`	55

Answer: (penalty regime: 0 %)

Reset answer

```
int baseballScore(string ops) {
    stack<int> scores;

    for (char op : ops) {
        if (isdigit(op)) {
            scores.push(op - '0');
        } else if (op == '+') {
            int prev1 = scores.top();
            scores.pop();
            int prev2 = scores.top();
            scores.push(prev1);
            scores.push(prev1 + prev2);
        } else if (op == 'D') {
            scores.push(scores.top() * 2);
        } else if (op == 'C') {
            scores.pop();
        }
    }

    int sum = 0;
    while (!scores.empty()) {
        sum += scores.top();
        scores.pop();
    }

    return sum;
}

```

	Test	Expected	Got	
✓	`cout << baseballScore("52CD+");`	30	30	✓
✓	`cout << baseballScore("524CD9++");`	55	55	✓
✓	`cout << baseballScore("5C4C2C11+D3");`	11	11	✓

Passed all tests! ✓

Câu hỏi **2**

Đúng

Đạt điểm 1,00

Implement all methods in class **Stack** with template type **T**. The description of each method is written as comment in frame code.

```
#ifndef STACK_H
#define STACK_H
#include "DLinkedList.h"
template<class T>
class Stack {
protected:
    DLinkedList<T> list;
public:
    Stack() {}
    void push(T item) ;
    T pop() ;
    T top() ;
    bool empty() ;
    int size() ;
    void clear() ;
};

#endif
```

You can use all methods in class **DLinkedList** without implementing them again. The description of class **DLinkedList** is written as comment in frame code.

```
template <class T>
class DLinkedList
{
public:
    class Node;     //forward declaration
protected:
    Node* head;
    Node* tail;
    int count;
public:
    DLinkedList() ;
    ~DLinkedList();
    void add(const T& e);
    void add(int index, const T& e);
    T removeAt(int index);
    bool removeItem(const T& removeItem);
    bool empty();
    int size();
    void clear();
    T get(int index);
    void set(int index, const T& e);
    int indexOf(const T& item);
    bool contains(const T& item);
};
```

For example:

Test	Result

`Stack<int> stack;`
`cout << stack.empty() << " " << stack.size();`
	`1 0`

`Stack<int> stack;`
 	
`int item[] = { 3, 1, 4, 5, 2, 8, 10, 12 };`
`for (int idx = 0; idx < 8; idx++) stack.push(item[idx]);`
 	
`assert(stack.top() == 12);`
 	
`stack.pop();`
`stack.pop();`
 	
`cout << stack.top();`
	`8`

Answer: (penalty regime: 0 %)

Reset answer

```
void push(T item) {
    // TODO: Push new element into the top of the stack
    list.add(0, item);
}

T pop() {
    // TODO: Remove an element on top of the stack
    T tmp = list.get(0);
    list.removeAt(0);
    return tmp;
}

T top() {
    // TODO: Get value of the element on top of the stack
    return list.get(0);
}

bool empty() {
    // TODO: Determine if the stack is empty
    return list.empty();
}

int size() {
    // TODO: Get the size of the stack
    return list.size();
}

void clear() {
    // TODO: Clear all elements of the stack
    list.clear();
}
```

	Test	Expected	Got	

✓	`Stack<int> stack;`
	`cout << stack.empty() << " " << stack.size();`
		1 0	1 0	✓

✓	`Stack<int> stack;`
	
	`int item[] = { 3, 1, 4, 5, 2, 8, 10, 12 };`
	`for (int idx = 0; idx < 8; idx++) stack.push(item[idx]);`
	
	`assert(stack.top() == 12);`
	
	`stack.pop();`
	`stack.pop();`
	
	`cout << stack.top();`
		8	8	✓

Passed all tests! ✓

Câu hỏi **3**
Đúng
Đạt điểm 1,00

Given an array nums[] of size N having distinct elements, the task is to find the next greater element for each element of the array
Next greater element of an element in the array is the nearest element on the right which is greater than the current element.
If there does not exist a next greater of a element, the next greater element for it is -1

Note: iostream, stack and vector are already included

Constraints:
1 <= nums.length <= 10^5
0 <= nums[i] <= 10^9

Example 1:
Input:
nums = {15, 2, 4, 10}
Output:
{-1, 4, 10, -1}

Example 2:
Input:
nums = {1, 4, 6, 9, 6}
Output:
{4, 6, 9, -1, -1}

For example:

Test	Input	Result

`int N;`
`cin >> N;`
`vector<int> nums(N);`
`for(int i = 0; i < N; i++) cin >> nums[i];`
`vector<int> greaterNums = nextGreater(nums);`
`for(int i : greaterNums)`
`    cout << i << ' ';`
`cout << '\n';`
	`4`
	`15 2 4 10`
		`-1 4 10 -1`

`int N;`
`cin >> N;`
`vector<int> nums(N);`
`for(int i = 0; i < N; i++) cin >> nums[i];`
`vector<int> greaterNums = nextGreater(nums);`
`for(int i : greaterNums)`
`    cout << i << ' ';`
`cout << '\n';`
	`5`
	`1 4 6 9 6`
		`4 6 9 -1 -1`

Answer: (penalty regime: 0 %)

Reset answer

```cpp
vector<int> nextGreater(vector<int>& arr) {
    int n = arr.size();
    vector<int> result(n, -1);
    stack<int> st;
    for (int i = n - 1; i >= 0; --i) {
        while (!st.empty() && arr[i] >= arr[st.top()]) {
            st.pop();
        }
        if (!st.empty()) {
            result[i] = arr[st.top()];
        }
        st.push(i);
    }
    return result;
}
```

	Test	Input	Expected	Got	

✓	`int N;`
	`cin >> N;`
	`vector<int> nums(N);`
	`for(int i = 0; i < N; i++) cin >> nums[i];`
	`vector<int> greaterNums = nextGreater(nums);`
	`for(int i : greaterNums)`
	`    cout << i << ' ';`
	`cout << '\n';`
		4
		15 2 4 10
			-1 4 10 -1	-1 4 10 -1	✓

✓	`int N;`
	`cin >> N;`
	`vector<int> nums(N);`
	`for(int i = 0; i < N; i++) cin >> nums[i];`
	`vector<int> greaterNums = nextGreater(nums);`
	`for(int i : greaterNums)`
	`    cout << i << ' ';`
	`cout << '\n';`
		5
		1 4 6 9 6
			4 6 9 -1 -1	4 6 9 -1 -1	✓

Passed all tests! ✓

Câu hỏi **4**
Đúng
Đạt điểm 1,00

Given string **S** representing a **postfix expression**, the task is to evaluate the expression and find the final value. Operators will only include the basic arithmetic operators like ***, /, + and -**.

Postfix expression: The expression of the form "a b operator" (ab+) i.e., when a pair of operands is followed by an operator.

For example: Given string S is "2 3 1 * + 9 -". If the expression is converted into an infix expression, it will be 2 + (3 * 1) – 9 = 5 – 9 = -4.

Requirement: Write the function to evaluate the value of postfix expression.

For example:

Test	Result
`cout << evaluatePostfix("2 3 1 * + 9 -");`	-4
`cout << evaluatePostfix("100 200 + 2 / 5 * 7 +");`	757

Answer: (penalty regime: 0 %)

Reset answer

```cpp
#include <sstream>
#include <stack>
#include <string>

using namespace std;
int evaluatePostfix(string expr) {
    stack<int> operands;
    auto isOperator = [](char c) {
        return c == '+' || c == '-' || c == '*' || c == '/';
    };
    istringstream iss(expr);
    string token;
    while (iss >> token) {
        if (isdigit(token[0])) {
            operands.push(stoi(token));
        } else if (isOperator(token[0])) {
            int operand2 = operands.top();
            operands.pop();
            int operand1 = operands.top();
            operands.pop();
            switch (token[0]) {
                case '+':
                    operands.push(operand1 + operand2);
                    break;
                case '-':
                    operands.push(operand1 - operand2);
                    break;
                case '*':
                    operands.push(operand1 * operand2);
                    break;
                case '/':
                    operands.push(operand1 / operand2);
                    break;
            }
        }
    }
    return operands.top();
}
```

	Test	Expected	Got	
✓	`cout << evaluatePostfix("2 3 1 * + 9 -");`	-4	-4	✓
✓	`cout << evaluatePostfix("100 200 + 2 / 5 * 7 +");`	757	757	✓

Passed all tests! ✓

Câu hỏi **5**
Đúng
Đạt điểm 1,00

A Maze is given as 5*5 binary matrix of blocks and there is a rat initially at the upper left most block i.e., maze[0][0] and the rat wants to eat food which is present at some given block in the maze (fx, fy). In a maze matrix, 0 means that the block is a dead end and 1 means that the block can be used in the path from source to destination. The rat can move in any direction (not diagonally) to any block provided the block is not a dead end.

Your task is to implement a function with following prototype to check if there exists any path so that the rat can reach the food or not:

`bool canEatFood(int maze[5][5], int fx, int fy);`

Template:

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <stack>
#include <vector>
using namespace std;

class node {
public:
    int x, y;
    int dir;
    node(int i, int j)
    {
        x = i;
        y = j;

        // Initially direction
        // set to 0
        dir = 0;
    }
};
```

Some suggestions:

```
- X : x coordinate of the node
- Y : y coordinate of the node
- dir : This variable will be used to tell which all directions we have tried and which to choose next. We will try all the
directions in anti-clockwise manner starting from up.
```

- If dir=0 try up direction.
- If dir=1 try left direction.
- If dir=2 try down direction.
- If dir=3 try right direction.

For example:

Test	Result

```
// Maze matrix
int maze[5][5] = {
    { 1, 0, 1, 1, 0 },
    { 1, 1, 1, 0, 1 },
    { 0, 1, 0, 1, 1 },
    { 1, 1, 1, 1, 0 },
    { 1, 0, 0, 1, 0 }
};

// Food coordinates
int fx = 1, fy = 4;

cout << canEatFood(maze, fx, fy);
```
	1

```
// Maze matrix
int maze[5][5] = {
    { 1, 0, 1, 1, 0 },
    { 1, 1, 1, 0, 0 },
    { 0, 1, 0, 1, 1 },
    { 0, 1, 0, 1, 0 },
    { 0, 1, 1, 1, 0 }
};

// Food coordinates
int fx = 2, fy = 3;

cout << canEatFood(maze, fx, fy);
```
	1

Answer: (penalty regime: 0 %)

Reset answer

```
bool canEatFood(int maze[5][5], int fx, int fy) {
    bool visited[5][5] = {false};
    int dx[] = {-1, 0, 1, 0};
    int dy[] = {0, -1, 0, 1};

    stack<pair<int, int>> stk;
    stk.push({0, 0});
    visited[0][0] = true;

    while (!stk.empty()) {
        int x = stk.top().first;
        int y = stk.top().second;
        stk.pop();
        if (x == fx && y == fy) {
            return true;
        }
        for (int i = 0; i < 4; i++) {
            int nextX = x + dx[i];
            int nextY = y + dy[i];
            if (nextX >= 0 && nextX < 5 && nextY >= 0 && nextY < 5 && maze[nextX][nextY]
                stk.push({nextX, nextY});
                visited[nextX][nextY] = true;
            }
        }
    }
    return false;
}

```

	Test	Expected	Got	

✓	`// Maze matrix`
	`int maze[5][5] = {`
	`    { 1, 0, 1, 1, 0 },`
	`    { 1, 1, 1, 0, 1 },`
	`    { 0, 1, 0, 1, 1 },`
	`    { 1, 1, 1, 1, 0 },`
	`    { 1, 0, 0, 1, 0 }`
	`};`
	
	`// Food coordinates`
	`int fx = 1, fy = 4;`
	
	`cout << canEatFood(maze, fx, fy);`
		1	1	✓

✓	`// Maze matrix`
	`int maze[5][5] = {`
	`    { 1, 0, 1, 1, 0 },`
	`    { 1, 1, 1, 0, 0 },`
	`    { 0, 1, 0, 1, 1 },`
	`    { 0, 1, 0, 1, 0 },`
	`    { 0, 1, 1, 1, 0 }`
	`};`
	
	`// Food coordinates`
	`int fx = 2, fy = 3;`
	
	`cout << canEatFood(maze, fx, fy);`
		1	1	✓

Passed all tests! ✓

Câu hỏi **6**

Đúng

Đạt điểm 1,00

Given a string **S** of characters, a *duplicate removal* consists of choosing two adjacent and equal letters, and removing them.

We repeatedly make duplicate removals on **S** until we no longer can.

Return the final string after all such duplicate removals have been made.

Included libraries: vector, list, stack

For example:

Test	Result
`cout << removeDuplicates("abbaca");`	ca
`cout << removeDuplicates("aab");`	b

Answer: (penalty regime: 0 %)

Reset answer

```
string removeDuplicates(string S) {
    stack<char> stk;
    for (char c : S) {
        if (!stk.empty() && stk.top() == c) {
            stk.pop();
        } else {
            stk.push(c);
        }
    }
    string result = "";
    while (!stk.empty()) {
        result = stk.top() + result;
        stk.pop();
    }

    return result;
}

```

	Test	Expected	Got	
✓	`cout << removeDuplicates("abbaca");`	ca	ca	✓
✓	`cout << removeDuplicates("aab");`	b	b	✓

Passed all tests! ✓

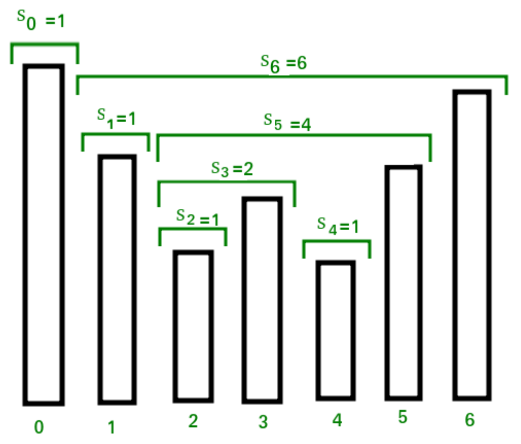
Câu hỏi **7**
Đúng
Đạt điểm 1,00

Vietnamese version:

Bài toán stock span là một bài toán về chủ đề kinh tế tài chính, trong đó ta có thông tin về giá của một cổ phiếu qua từng ngày. Mục tiêu của bài toán là tính *span* của giá cổ phiếu ở từng ngày.

Span của giá cổ phiếu tại ngày thứ *i* (ký hiệu là S_i) được định nghĩa là *số ngày liên tục nhiều nhất liền trước ngày thứ i có giá cổ phiếu thấp hơn, cộng cho 1 (cho chính nó)*.

Ví dụ, với chuỗi giá cổ phiếu là [100, 80, 60, 70, 60, 75, 85].



1. Ngày thứ 0 không có ngày liền trước nên S0 bằng 1.
2. Ngày thứ 1 có giá nhỏ hơn giá ngày thứ 0 nên S1 bằng 1.
3. Ngày thứ 2 có giá nhỏ hơn giá ngày thứ 1 nên S2 bằng 1.
4. Ngày thứ 3 có giá **lớn hơn** giá ngày thứ 2 nên S3 bằng 2.
5. Ngày thứ 4 có giá nhỏ hơn giá ngày thứ 3 nên S4 bằng 1.
6. Ngày thứ 5 có giá **lớn hơn giá ngày thứ 4, 3, 2** nên S5 bằng 4.
7. Ngày thứ 6 có giá **lớn hơn giá ngày thứ 5, 4, 3, 2, 1** nên S6 bằng 6.

Kết quả sẽ là [1, 1, 1, 2, 1, 4, 6].

Yêu cầu. Viết chương trình tính toán chuỗi span từ chuỗi giá cổ phiếu từ đầu vào.

Input. Các giá trị giá cổ phiếu, cách nhau bởi các ký tự khoảng trắng, được đưa vào standard input.

Output. Các giá trị span, cách nhau bởi một khoảng cách, được xuất ra standard ouput.

(Nguồn: Geeks For Geeks)

=================================

Phiên bản tiếng Anh:

The stock span problem is a financial problem where we have a series of daily price quotes for a stock and we need to calculate the span of the stock's price for each day.

The span S_i of the stock's price on a given day i is defined as the maximum number of consecutive days just before the given day, for which the price of the stock on the current day is less than its price on the given day, plus 1 (for itself).

For example: take the stock's price sequence [100, 80, 60, 70, 60, 75, 85]. (See image above)

The given input span for 100 will be 1, 80 is smaller than 100 so the span is 1, 60 is smaller than 80 so the span is 1, 70 is greater than 60 so the span is 2 and so on.

Hence the output will be [1, 1, 1, 2, 1, 4, 6].

Requirement. Write a program to calculate the spans from the stock's prices.

Input. A list of whitespace-delimited stock's prices read from standard input.

Output. A list of space-delimited span values printed to standard output.

(Source: Geeks For Geeks)
For example:

Input	Result
`100 80 60 70 60 75 85`	`1 1 1 2 1 4 6`
`10 4 5 90 120 80`	`1 1 2 4 5 1`

Answer: (penalty regime: 0 %)

Reset answer

```
vector<int> stock_span(const vector<int>& ns) {
    // STUDENT ANSWER
    int n = ns.size();
    vector<int> spans(n,1);
    for (int i = 0; i < n; i++){
        int counter = 1;
        int j = i - 1;
        while (j >= 0 && ns[i] > ns[j]){
            counter += spans[j];
            j -= spans[j];
        }
        spans[i] = counter;
    }
    return spans;
}

```

	Input	Expected	Got	
✓	`100 80 60 70 60 75 85`	`1 1 1 2 1 4 6`	`1 1 1 2 1 4 6`	✓
✓	`10 4 5 90 120 80`	`1 1 2 4 5 1`	`1 1 2 4 5 1`	✓

Passed all tests! ✓

Câu hỏi **8**
Đúng
Đạt điểm 1,00

Given a string **s** containing just the characters **'(', ')', '[', ']', '{', and '}'**. Check if the input string is valid based on following rules:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

For example:

- String **"[]()"** is a valid string, also **"[()]"**.
- String "**[])**" is **not** a valid string.

Your task is to implement the function

```
bool isValidParentheses (string s){
    /*TODO*/
}
```

```
Note: The library stack of C++ is included.
```

For example:

Test	Result
`cout << isValidParentheses("[]");`	1
`cout << isValidParentheses("[]()");`	1
`cout << isValidParentheses("[)");`	0

Answer: (penalty regime: 0 %)

Reset answer

```
bool isValidParentheses(string s) {
    stack<char> stk;
    for (char c : s) {
        if (c == '(' || c == '[' || c == '{') {
            stk.push(c);
        } else {
            if (stk.empty() || (c == ')' && stk.top() != '(') || (c == ']' && stk.top()
                return false;
            }
            stk.pop();
        }
    }
    return stk.empty();
}

```

	Test	Expected	Got	
✓	`cout << isValidParentheses("[]()");`	1	1	✓
✓	`cout << isValidParentheses("[)");`	0	0	✓

Passed all tests! ✓