

Лабораторная работа 09. Структуры данных, стеки, очереди.

I. Стеки

0. Создать интерфейс для стека из целых чисел со следующими методами:

```
int Pop();           // извлечь элемент из вершины стека
void Push(int value); // поместить элемент в стек
int Peek();          // показать элемент в вершине, не удаляя его из стека

bool IsEmpty();       // проверить, что стек пуст
bool IsFull();        // проверить что стек полон
```

1. Создать собственную реализацию стека из целых чисел на основе массива, этот класс должен реализовать интерфейс из п. 0.

Протестировать стек на примере:

Из текстового файла прочитать все имеющиеся числа, разместить их в стеке в порядке чтения.

Вывести все содержимое стека на экран (числа должны выводиться в порядке обратном, тому, что были в файле, стек в конце операции должен стать пустым)

2. Создать собственную реализацию стека из целых чисел на основе связанного списка, этот класс должен реализовать интерфейс из п. 0.

Протестировать стек на примере:

Из текстового файла читать строки, содержащие команды. Каждая строка содержит одну команду.

Команда — это либо "+ ", либо "-".

Команда "+ " означает добавление в стек числа.

Команда "-" означает изъятие элемента из стека.

Гарантируется, что цепочка команд не приведет к необходимости извлечением из пустого стека.

Гарантируется, что в процессе выполнения команд стек не будет переполнен.

Формат входного файла

В первой строке входного файла содержится число команд.

Каждая последующая строка исходного файла содержит ровно одну команду.

Формат вывода на экран

Вывести числа, которые удаляются из стека с помощью команды "-", по одному в каждой строке. Числа нужно вывести в том порядке, в котором они были извлечены из стека.

Пример

input.txt	output
6	10
+ 1	1234
+ 10	
-	
+ 2	
+ 1234	
-	

3. Использовать стек из п. 1 или из п.2. для решения задачи.

Проверить правильность скобочной последовательности

Реализовать решение для скобок вида (), [], { }

(*доп. – предложить удобный механизм добавления других видов скобок)

В тестирующей программе из входного текстового файла читать строки, содержащие только ASCII-символы.

а) Выводить на экран

“YES” – если скобки в строке расставлены верно или NO, в противном случае

Пример

input.txt	output
()()	YES
([])	YES
([])	NO
(([])	NO
)(NO

б) выводить на экран

“YES” – если скобки в строке расставлены верно или порядковый номер символа в строке (номер скобки), если он создает ошибочную комбинацию

То есть вывести

- номер первой закрывающей скобки, для которой нет соответствующей открывающей.

Если такой нет, то

- то номер первой открывающей скобки, для которой нет соответствующей закрывающей.

Некоторые тестовые последовательности и результаты для них (последовательность -> результат):

```
[()][()][()]) -> "YES"
{*} -> "YES"
{} -> "YES"
-> "YES"
*{} -> "YES"
[] -> "YES"
{}[] -> "YES"
[()] -> "YES"
(()) -> "YES"
{[]}() -> "YES"
[()][()][()]) -> "YES"
foo(bar); -> "YES"
(slkj, {lk[lve]} ,1) -> "YES"
```

```
()[] -> 5
{[()] -> 7
{{{[[]]} -> 3
*{[]} -> 3
[[]*] -> 2
{[ -> 2
} -> 1
{{{*[]]} -> 3
() ( {} -> 3
[] ( [] -> 3
((([])) -> 2
}{[] -> 3
```

```
(slkj{lk[lsj]} -> 1
dasdsadsadas]] -> 13
[]([] -> 3
{{{[[]]} -> 3
{ -> 1
{[] -> 3
()[] -> 5
{{{[]]} -> 7
foo(bar[i]); -> 10
[]([] -> 3
```

Подсказка:

Можно использовать два стека.

Сначала возвращайте ошибки связанные с закрывающими скобками, это позиция в строке плюс один.

Проверяйте стек на пустоту, когда встречается закрывающая скобка.

В конце проверяйте стек на пустоту, возвращать надо позицию скобки, которая на верхушке стека.

4. Использовать стек из п. 1 или из п.2. для решения задачи.

Из текстового файла прочитать все имеющиеся числа, разместить их в стеке в порядке чтения.

Выведите самое маленькое число и его порядковый номер в файле (относительно других чисел).

Обеспечьте решение задачи за $O(1)$.

Нужен будет стек с поддержкой минимума(максимума), достаточно просто реализуется двумя стековыми структурами.

II. Очереди

0. Создать интерфейс для очереди из целых чисел со следующими методами:

```
String Dequeue();           // извлечь элемент из головы очереди
void Enqueue(String value); // поместить элемент в очередь
String Peek();              // показать элемент в голове, не удаляя его

bool IsEmpty();             // проверить, что очередь пуста
bool IsFull();              // проверка на заполнение
```

1. Создать собственную реализацию интерфейса «очередь» из п. 0. на основе массива строк.

а) В тестирующей программе должны обрабатываться строки, содержащие символы с ASCII от 32 до 127

Пример:

```
EAS * Yes * QUE * * * stop * * * IO * to * N *
```

В них отдельные символы интерпретируются как команды:

- любая заглавная латинская буква – поместить эту букву в очередь
- звездочка – извлечь из очереди и вывести на экран очередной элемент
- другие символы игнорируются.

В программе ввести с консоли строку, вывести результаты ее обработки таким командным интерпретатором.

б) В тестирующей программе, поместить в очередь все введенные строки (вводить из текстового файла). Используя только операции *enqueue* и *dequeue* выполнить циклический сдвиг элементов в очереди так, чтобы в ее начале (в голове, front) был расположен наибольший элемент – самая длинная строка.

2. Создать собственную реализацию интерфейса «очередь» из п. 0. на основе линейного списка.

В дополнение к стандартным операциям очереди, необходимо также отвечать на запрос о минимальном элементе из тех, которые сейчас находится в очереди. Для каждой операции запроса минимального элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо «+ », либо «-», либо «?». Команда «+ » означает добавление в очередь числа , по модулю не превышающего . Команда «-» означает изъятие элемента из очереди. Команда «?» означает запрос на поиск минимального элемента в очереди.

Формат входного файла

В первой строке содержится число команд. В последующих строках содержатся команды, по одной в каждой строке.

Формат вывода

Для каждой операции поиска минимума в очереди выведите её результат. Результаты должны быть выведены в том порядке, в котором эти операции встречаются во входном файле. Гарантируется, что операций извлечения или поиска минимума для пустой очереди не производится.

Пример

input.txt	output
7	1
+ 1	1
?	10
+ 10	
?	
-	
?	
-	

III. *ДОП. Язык Quack — забавный язык, который фигурирует в одной из задач с [Internet Problem Solving Contest](#). В этой задаче требуется написать интерпретатор языка Quack.

Виртуальная машина, на которой исполняется программа на языке Quack, имеет внутри себя очередь, содержащую целые числа по модулю 65536 (то есть, числа от 0 до 65535, соответствующие беззнаковому 16-битному целому типу). Слово `get` в описании операций означает извлечение из очереди, `put` — добавление в очередь. Кроме того, у виртуальной машины есть 26 регистров, которые обозначаются буквами от 'a' до 'z'. Изначально все регистры хранят нулевое значение. В языке Quack существуют следующие команды (далее под и подразумеваются некие абстрактные временные переменные):

+	Сложение: <code>get , get , put () mod 65536</code>
-	Вычитание: <code>get , get , put () mod 65536</code>
*	Умножение: <code>get , get , put () mod 65536</code>
/	Целочисленное деление: <code>get , get , put div</code> (будем считать, что <code>div 0 = 0</code>)
%	Взятие по модулю: <code>get , get , put mod</code> (будем считать, что <code>mod 0 = 0</code>)
>[register]	Положить в регистр: <code>get ,</code> установить значение [register] в
<[register]	Взять из регистра: <code>put</code> значение [register]
P	Напечатать: <code>get ,</code> вывести в стандартный поток вывода и перевести строку
P[register]	Вывести значение регистра [register] в стандартный поток вывода и перевести строку
C	Вывести как символ: <code>get ,</code> вывести символ с ASCII-кодом <code>mod 256</code> в стандартный поток вывода
C[register]	Вывести регистр как символ: вывести символ с ASCII-кодом <code>mod 256</code> (где — значение регистра [register]) в стандартный поток вывода
:[label]	Метка: эта строка программы имеет метку [label]
J[label]	Переход на строку с меткой [label]
Z[register][label]	Переход если 0: если значение регистра [register] равно нулю, выполнение программы продолжается с метки [label]
E[register1][register2][label]	Переход если равны: если значения регистров [register1] и [register2] равны, исполнение программы продолжается с метки [label]
G[register1][register2][label]	Переход если больше: если значение регистра [register1] больше, чем значение регистра [register2], исполнение программы продолжается с метки [label]
Q	Завершить работу программы. Работа также завершается, если выполнение доходит до конца программы
[number]	Просто число во входном файле — <code>put</code> это число

Формат входного файла

Входной файл содержит синтаксически корректную программу на языке Quack. Известно, что программа завершает работу не более чем за шагов. Программа содержит не менее одной и не более инструкций. **Метки имеют длину от 1 до 10 и состоят из цифр и латинских букв.**

Формат выходного файла

Выведите содержимое стандартного потока вывода виртуальной машины в выходной файл.

Примеры

input.txt	output
100 0 :start >a Zaend <a <a 1 + - >b <b Jstart :end P	5050

Второй пример подразумевает UNIX-переводы строки в ответе (один символ с кодом 10).

input.txt	output
-----------	--------

58	58
49	49
10	10
62	62
97	97
10	10
80	80
97	97
10	10
90	90
97	97
50	50
10	10
60	60
97	97
10	10
74	74
49	49
10	10
58	58
50	50
10	10
48	48
10	10
58	58
51	51
10	10
62	62
97	97
10	10
90	90
97	97
52	52
10	10
67	67
97	97
10	10
74	74
51	51
10	10
58	58
52	52
10	10
0	0
:1	:1
>a	>a
Pa	Pa
Za2	Za2
<a	<a
J1	J1
:2	:2
0	0
:3	:3
>a	>a
Za4	Za4
Ca	Ca
J3	J3
:4	:4