Лабораторная работа 7. Структуры данных, часть 1.

1. Построить собственную минимальную реализацию АТД «динамический массив» для работы с целыми (int) числами.

Для хранения элементов использовать обычный массив. Позволять хранить не более 10000 элементов

*(*доп. – решение с рациональным использованием памяти: оперативное наращивание и освобождение в соответствии с реальной длиной массива, оценивается доп.баллами)*

Обеспечить выполнение операций:
- добавление элемента
- удаление элемента
- чтение элемента по индексу
- запись элемента по индексу
- возврат максимального значения (индекса)
- возврат второго максимума (индекса)
- проверка на переполнение
- возврат актуального количества элементов

а) В тестирующей программе из произвольного текстового файла прочитать все имеющиеся там строки, их длины занести в динамический массив. Вывести номера двух самых длинных строк.

б) решить задачу п. а), используя ArrayList из java.util

2. Создайте собственную реализацию стека.

Заполните стек числовыми значениями длин строк, прочитанных из текстового файла.
а) Распечатайте на экране содержимое стека.
б) Выведите номер и длину самой короткой строки файла,
обеспечьте выполнение этой операции за O(1).

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

3. Используя собственную реализацию стека проверьте правильность скобочной последовательности

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

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

В тестирующей программе выводить на экран
“YES” – если скобки 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]} ,l) -> “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 хранить в стеке структуру, которая содержит скобку и ее позицию в строке; или же 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. Создайте собственную реализацию АТД «очередь» со строковым информационным полем.

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 с клавиатуры или из файла на ваше усмотрение).
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.

в)*доп. Построить реализацию такой очереди на двух стеках. Обеспечить в ней операцию возврата максимального элемента за O(1). Протестировать на подходящих примерах.

5. Создайте собственную реализацию АТД «дек» со строковым информационным полем.
(можно за основу взять решение зад.4)

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 начало дека
- любая строчная латинская буква – поместить эту букву в конец дека
- плюс – извлечь из начала дека и вывести на экран очередной элемент
- звездочка – извлечь из конца дека и вывести на экран очередной элемент
- другие символы игнорируются.

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