**Objective:**

- Exploring the LIFO behavior. Getting a drip on usage of stack in different problems.

**Task-1:**
You are to develop an Undo/Redo behavior as in all word processing applications.
All these applications can remember a limited number of operations. Let’s try to explore the behavior of Undo/Redo in MSWord through following example:

**Example: MSWord 2007**

In MSWord 2007, whenever you perform an operation it somehow uniquely identify those operations and memorize them. I don’t know the maximum number of operations that MSWord can memorize but let’s assume that it cam memorize up to N = 10 operations.

So If I type the following in the MSWord:

```
An
Undo
Redo
Facility
```

I typed the above in small caps but MSWord automatically does the auto correction as well. So there are not only four typing operations rather the sequence of operations is as follows:

Typing, auto correction, Typing, auto correction, Typing, auto correction, Typing, auto correction,

Notice that we haven’t done any undo/redo so far. So when I do undo, the last operation will be undone i.e. in the above text the ‘Facility’ will become ‘facility’

Another point to note is that what if I do more N=10 operations in that case 11th operations will become 10th, 10th will become 9th and so on... And 1st operations will be dropped / forgotten.

You can explore the behavior of Undo/Redo yourself.

In the following ADT we have to exhibit the same behavior. For us an operation is an integer which will be sent to ‘memoriseOperation’. ‘undo’ method will exhibit the behavior of undo as in MSWord and ‘redo’ will exhibit redo behavior.

```
class UndoRedo
{
        int capacity; //N // number of operations that can be memorized
        //decide rest of the data structure yourself
public:
        UndoRedo (int c = 10);
        void memoriseOperation(int op);
        int undo();
        int redo();
};
```

**Task-2:**

*It is advised to solve the problems in order.*

**Q. # 1.** Validate Hetrogeneous brackets

**Q. # 2.** Check whether a string is of the form $a^nb^n$ where n = 0, 1, 2, 3, 4, …….
E.g. the string aaabbb is of $a^nb^n$ form but bbbaaa, aabbb are not.

**Q. # 3.** Check whether a string is of the form $a^sb^t$ where s = 0, 1, 2, 3, 4, ……. and t = 0, 1, 2, 3, 4, …….
E.g. the string aaaaaabbb is of $a^sb^t$ form.

**Q. # 4.** Check whether a string is of the form $a^nb^nc^n$ where n = 0, 1, 2, 3, 4, …….

**Q. # 5.** Reverse the order of elements in stack S
  a. Using two additional stacks
  b. Using one additional stack and some additional non-array variables

**Q. # 6.** Review the algorithm 'addingLargeNumbers()' discussed in Section 4.1 of Text Book-B

**Q. # 7.** Put the elements on stack S in ascending order using one additional stack and some additional non-array variables.

**Q. # 8.** Transfer the elements of stack S1 to Stack S2 so that the elements from S2 are in the same order as on S1
  a. using one additional Stack
  b. Using no additional stack but only some additional non-array variables

**Q. # 9.** Convert infix to prefix

**Q. # 10.** Checks whether a string is palindrome or not

**Q. # 11.** Reverse the words in a given string.

**Q. # 12.** Implement a function that converts prefix/postfix expression into infix form.

**Q. # 13.** Function Call Stack
You can browse the following for this:
http://en.wikipedia.org/wiki/Call_stack

***NOTE:***

- While implementing the above function you are not allowed to change the standard ADT of stack discussed in class, so all the function asked above will only be able to access the public part of Stack ADT discussed in class.
- Most of the problems have been taken from Text Book B.