**Lab task: the Stack and Introduction to Procedure**

1. Write a program to print the reversed string of user input. You should take inputs until user press CR. [hint: Use stack]. For example, **"COA SPRING AIUB"** becomes **"BUIA GNIRPS AOC"**.
2. Write a program that lets the user type some text, consisting of words separated by blanks, ending with a carriage return, and displays the text in the same word order as entered, but with the letters in each word reversed. For example, **"this is a test"** becomes **"siht si a tset"**. [Hint: modify the previous program or in the Example in section 8.3.]
3. Write program to create a simple calculator to facilitate ADD (), SUB (), MUL () and DIV () features. You should create four different procedures and call all the procedures form main based on the user instruction.
4. Write a program to find all the even and odd numbers between 1 and 100. You must use procedure to print EVEN or ODD numbers.
5. Write a program that displays a '?', calls READ to read a binary number, and calls RANDOM and WRITE to compute and display 100 random numbers. The numbers should be displayed **four per line**, with **four blanks** separating the numbers.
6. Scenario: A problem in elementary algebra is to decide if an expression containing several kinds of brackets, such as, [,],{,},(,), is correctly bracketed. This is the case if
   1. there are the same number of left and right brackets of each kind, and
   2. When a right bracket appears, the most recent preceding unmatched left bracket should be of the same type. For example,

(a+[b-{c \* (d-e)}] + f) is correctly bracketed, but

(a+ [b-{c\*(d-e)]}+f) is NOT

Correct bracketing can be decided by using a stack. The expression is scanned left to right. When a left bracket is encountered, it is pushed onto the stack. When a right bracket is encountered, the stack is popped (if the stack is empty, there are too many right brackets) and the brackets are compared. If they are of the same type, the scanning continues. If there is a mismatch, the expression is incorrectly bracketed. At the end of the expression, if the stack is empty the expression is correctly bracketed. If the stack is not empty, there are too many left brackets.

TASK: Write a program that lets the user type in an algebraic expression, ending with a carriage return, contains **round (parentheses), square, and curly brackets**. As the expression is being typed in, the program evaluates each character. If at any point the expression is incorrectly bracketed (too many right brackets or a mismatch between left and right brackets), the program tells the user to start over. After the carriage return is typed, If the expression is correct, the program displays **"expression Is correct."** If not, the program displays **"too many left brackets"**. In both cases, the program asks the user if he or she wants to continue. *If the user types 'Y', the program runs again.* Your program does not need to store the input string, **only check it for correctness**.

**Sample execution:**

ENTER AN ALGEBRAIC EXPRESSION: (a + b]]: **TOO MANY RIGHT BRACKETS. BEGIN AGAIN**!

ENTER AN ALGEBRAIC EXPRESSION (a + [b - c] \* d) **EXPPESSION IS CORRECT**

TYPE Y IF YOU WANT TO CONTINUE:**N**