CS 271 Computer Architecture and Assembly Language
Programming Assignment #7

Objectives:

1) Designing, implementing, and calling low-level I/O procedures
2) Implementing recursion
    a. parameter passing on the system stack
    b. maintaining activation records (stack frames)

Problem Definition:

A system is required for statistics students to use for drill and practice in combinatorics. In particular, the system will ask the student to calculate the number of combinations of ***r*** items taken from a set of ***n*** items (using the symbol C_n^r). The system generates random problems with n in [3, 12] and r in [1, n]. The student enters his/her answer, and the system reports the correct answer and an evaluation of the student's answer. The system repeats until the student chooses to quit.

Requirements:

1) The calculation must use the formula of combinations: $C_n^r = C(n,r) = \frac{n!}{r!(n-r)!}$ .The factorial calculation, like f(k) = k!, must be done recursively.
2) User's numeric input must be validated the hard way: Read the user's input as a string, convert the string to numeric form. If the user enters non-digits, an error message should be displayed.
3) All parameters must be passed on the system stack.
4) Used registers must be saved and restored by the called procedure.
5) The stack must be "cleaned up" by the called procedure.
6) The program must be modularized into at least the following procedures:
    a. *main*: mostly pushing parameters and calling procedures.
    b. *introduction*: display title, programmer name, and instructions.
    c. *showProblem*: generates the random numbers and displays the problem
        • *showProblem* accepts addresses of ***n*** and ***r***.
    d. *getData*: prompt / get the user's answer.
        • *answer* should be passed to *getData* by address (of course!).
    e. *combinations*, *factorial*: do the calculations.
        • *combinations* accepts *n* and *r* by value and *result* by address.
        • *combinations* calls *factorial* (3 times) to calculate *n!*, *r!*, and *(n-r)!*.
        • *combinations* calculates $C_n^r = C(n,r) = \frac{n!}{r!(n-r)!}$, and stores the value in *result*.
    f. *showResults*: display the student's *answer*, the calculated *result*, and a brief statement about the student's performance
        • *showResults* accepts the values of *n*, *r*, *answer*, and *result*.
7) You should use a string display macro to display strings.
8) The usual requirements regarding documentation, readability, user-friendliness, etc., apply.
9) Turn in your submission to Canvas by the due date.

What to turn in:

1. Your source code files (*.asm*) that can be compiled by Visual Studio.
2. A video of a quick overview of your code and a quick demonstration of your program by compiling and running through it.
3. Do NOT put them into a zip file. Please leave them out separately.

Notes:
1) It is OK to use strings as globals.

2) The limits are chosen to keep calculations within the limitations of DWORD

3) You are required to handle non-numeric input. You may use Irvine's *ReadString* to get the user's input, but you must validate / convert the string to numeric data.

Example (user input in ***italics***):

```
Welcome to the Combinations Calculator
Implemented by Fred Flintstone

I'll give you a combinations problem. You enter your answer, and I'll let you know if
you're right.

Problem:
Number of elements in the set: 10
Number of elements to choose from the set: 6
How many ways can you choose? 250

There are 210 combinations of 6 items from a set of 10.
You need more practice.

Another problem? (y/n): OK
Invalid response. Another problem? (y/n): Y

Problem:
Number of elements in the set: 9
Number of elements to choose from the set: 4
How many ways can you choose? 126

There are 126 combinations of 4 items from a set of 9.
You are correct!

Another problem? (y/n): n
OK ... goodbye.
```

Optional challenges:
1) Numbering each problem and keeping score. When the student quits, report number right/wrong, etc.
2) Computing factorials in the floating-point unit to expand the limits.