CS 271   Computer Architecture and Assembly Language
Programming Assignment #4

Objectives:
1) Designing and implementing procedures
2) Designing and implementing loops
3) Writing nested loops
4) Understanding data validation

Problem Definition:
Write a program to calculate prime numbers. First, the user is instructed to enter the number of primes to be displayed, and is prompted to enter an integer in the range [1 .. 200]. The user enters a number, *n*, and the program verifies that $1 \leq n \leq 200$. If *n* is out of range, the user is re-prompted until s/he enters a value in the specified range. The program then calculates and displays the all of the prime numbers up to and including the nth prime. The results should be displayed 10 primes per line with at least 3 spaces between the numbers.

Requirements:
1) The programmer's name must appear in the output.
2) The counting loop (1 to *n*) must be implemented using the MASM *loop* instruction.
3) The *main* procedure must consist (mostly) of procedure calls. It should be a readable "list" of <u>what</u> the program will do.
4) Each procedure will implement a section of the program logic, i.e., each procedure will specify <u>how</u> the logic of its section is implemented. The program must be modularized into at least the following procedures and sub-procedures :
   * *introduction*
   * *getUserData*
      * *validate*
   * *showPrimes*
      * *isPrime*
   * *farewell*
5) The upper limit should be defined and used as a <u>constant</u>.
6) Data validation is required. If the user enters a number outside the range [1 .. 200] an error message should be displayed and the user should be prompted to re-enter the number of primes.
7) The usual requirements regarding documentation, readability, user-friendliness, etc., apply.
8) Submit your text code file (*.asm*) to Canvas by the due date.

Notes:
1) For this program, you may use global variables instead of passing parameters. This is a one-time relaxation of the standards so that you can get accustomed to using procedures.
2) A number *k* is prime if it is an integer with exactly 2 distinct divisors: 1 and *k*. Note that this implies that
   a. 1 is not prime.
   b. *k* must be positive.
3) There are several ways to make your *isPrime* procedure efficient. (Discussion in class)
4) See next page for an example execution

Example (user input in *italics*):
```
Prime Numbers      Programmed by Euclid

Enter the number of prime numbers you would like to see.
I'll accept orders for up to 200 primes.

Enter the number of primes to display [1 .. 200]: 301
Out of range.  Try again.
Enter the number of primes to display [1 .. 200]: 0
Out of range.  Try again.
Enter the number of primes to display [1 .. 200]: 31

2   3   5   7   11   13   17   19   23   29
31   37   41   43   47   53   59   61   67   71
73   79   83   89   97   101   103   107   109   113
127

Results certified by Euclid.  Goodbye.
```

Extra Credit:
1) Align the output columns.
2) Display more primes, but show them one page at a time. The user can "Press any key to continue …" to view the next page. Since length of the numbers will increase, it's OK to display fewer numbers per line.
3) One way to make the program more efficient is to use only prime divisors, which requires saving all of the primes that have already been generated. It's easy in a high-level language, but you will have to look ahead in the textbook to figure out how to do it in assembly language.