Problem: Given an integer as input calculate the sum of the squares of all integers from 1 to that number. Secondly, find the next integer greater than or equal to the sum of the squares that is a prime number.

Example Execution #1:
Enter a positive integer: 5
Sum of squares from 1 to 5: 55
Next prime number is: 59

Example Execution #2:
Enter a positive integer: 10
Sum of squares from 1 to 10: 385
Next prime number is: 389

Example Execution #3:
Enter a positive integer: 60
Sum of squares from 1 to 60: 73810
Next prime number is: 73819

Example Execution #4:
Enter a positive integer: 100
Sum of squares from 1 to 100: 338350
Next prime number is: 338369

Example Execution #5:
Enter a positive integer: 1000
Sum of squares from 1 to 1000: 333833500
Next prime number is: 333833503

Example Execution #6:
Enter a positive integer: 10000
Sum of squares from 1 to 10000: 333383335000
Next prime number is: 333383335037

Example Execution #7 (input validation demonstrated):
Enter a positive integer: -3
Error! Positive values only!!
Enter a positive integer: -4
Error! Positive values only!!
Enter a positive integer: 0
Error! Positive values only!!
Enter a positive integer: 21
Sum of squares from 1 to 21: 3311
Next prime number is: 3313

Example Execution #8 (one is considered prime):
Enter a positive integer: 1
Sum of squares from 1 to 1: 1
Next prime number is: 1

Academic Integrity Reminder:

• Please review the policies of the course as they relate to academic integrity. The assignment you submit should be your own original work. You are to be consulting only course staff regarding your specific algorithm for assistance. Collaboration is not permitted on individual homework assignments.
Additional Requirements:

1. Add the homework (head_hw) assignment header file to the top of your program. A description of your program will need to be included in the assignment header.
   ○ This particular header can be added to your file by entering hhw while in command mode in vi.

2. Each example execution represents a single test of your program. The seven examples provided imply that the program was run seven different times.
   ○ Your program is expected to accept input and produce output in the same manner demonstrated above.
   ○ See example #6 for input validation expectations. The user will always enter an integer (int).

3. For this assignment you will be required to implement the user-defined functions (from chapter 4). Failing to follow course standards as they relate to good user-defined function use will result in a zero for this assignment.

4. Course standards prohibit the use of programming concepts not yet introduced in lecture. For this assignment you can consider all material in the first SIX chapters of the book, notes, and lectures to be acceptable for use.
   ○ The use of any arrays will result in no credit being awarded for your effort.

5. A program MUST compile to be considered for partial credit. The submission script will reject the submission of any file that does not successfully compile on the guru server.

Course Programming and Documentation Standards Reminders:

• The use of { and } for the body of relevant selection and repetition constructs is required.
• All code found within the body of a selection or repetition construct should be indented two additional spaces.
• See problem 5-7 on page 258 of your C programming text for an example of proper indenting within a switch construct.
• Control-forcing statements such as exit, break, continue, and the use of multiple return statements in a single function are not acceptable practices according to course standards.
• Use the course function header (head_fx vi shortcut hfx while in command mode) for every user-defined function in your program.
  ○ List and comment all parameters to a function, one per line, in the course function header.
  ○ All function declarations will appear in the global declaration section of your program.
  ○ The user-defined function definitions will appear in your program after the main function.
• Place a single space between all operators and operands.
• Comment all variables to the right of each declaration. Declare only one variable per line.
• At no point during the semester should the two sections of a function overlap.
  ○ Note that many programs in the C programming text make use a single line comment to indicate the start of the local declaration and executable statement sections of the main function. You may consider adopting this habit of commenting the start of each section to help you avoid this mistake.
• Select meaningful identifiers (names) for all variables in your program.
• Indent all code found within the main function exactly two spaces.
• Do not single (or double) space the entire program, use blank lines when appropriate.
• Remove any diagnostic print statements from your code, even if they are commented out (inactive), unless you believe your program to be logically incorrect or incomplete as a way to demonstrate to your lab instructor how much of the problem you were able to solve and attempted to implement.

When you submit... only the final attempt of a submission is kept for grading. All other submissions are over-written and cannot be recovered. You may make multiple submissions but only the last attempt is retained and graded.

• Verify in the confirmation e-mail sent to you by the course that you have submitted the correct file, to the correct assignment (hw05), and to the correct section.
• Leave time prior to the due date to seek assistance should you experience difficulties completing or submitting this assignment.
• All attempts to submit via a method other than through the guru server as set up during the first week of the semester will be denied consideration.
Assignment deadlines... are firm and the electronic submission will disable promptly as advertised. We can only grade what you submit as expected prior to the assignment deadline.

All course programming and documentation standards are in effect for this and each assignment this semester. Please review this document!