Problem: Given an integer as a starting value, identify the next thirty prime numbers that are greater than or equal to the input value. The results will be stored in a thirty-element array and must be displayed on two separate lines, those at even-indexes first and odd-indexes second.

Example Execution #1:

Enter starting value: 23

Even-index elements: 23 31 41 47 59 67 73 83 97 103 109 127 137 149 157
Odd-index elements: 29 37 43 53 61 71 79 89 101 107 113 131 139 151 163

Example Execution #2:

Enter starting value: 377

Even-index elements: 379 389 401 419 431 439 449 461 467 487 499 509 523 547 563
Odd-index elements: 383 397 409 421 433 443 457 463 479 491 503 521 541 557 569

Example Execution #3:

Enter starting value: 0

Even-index elements: 1 3 7 13 19 29 37 43 53 61 71 79 89 101 107
Odd-index elements: 2 5 11 17 23 31 41 47 59 67 73 83 97 103 109

Example Execution #4:

Enter starting value: -4

Error! Non-negative values only!!!

Enter starting value: -5

Error! Non-negative values only!!!

Enter starting value: 91

Even-index elements: 97 103 109 127 137 149 157 167 179 191 197 211 227 233 241
Odd-index elements: 101 107 113 131 139 151 163 173 181 193 199 223 229 239 251

Additional Requirements:

1. **Add the homework 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. The example executions provided each represent a single test of your program. We will test your final submission with the data used in the test case above and with several unpublished test cases.
   - Your program is expected to accept input, validate input, and produce output in the exact same manner demonstrated above.
   - You may assume that the use will always enter an integer and no value generated will exceed that which can be stored in an int type variable.
   - Do not add any “bonus” features not demonstrated in the example executions provided.

3. **Course standards prohibit** the use of programming concepts not yet introduced in lecture. For this assignment you can consider all material in the first eight chapters of the book, notes, and lectures to be acceptable for use.
Additional Requirements (continued):

4. 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.

With our ability to implement user-defined functions **only the following** will be permitted in the main function:

1. Declaration of variables to be passed to functions.
2. Calls to user-defined functions by main.
3. A limited amount of control structures (see chapters 5 and 6) to retain the previous two tasks within the main function.

Additionally, each user-defined function may **represent a single task** in your larger program. The failure to make a good use of user-defined functions as described here and in the course standards will result in a loss of ALL points on those assignments that require user-defined functions.

5. A program **MUST** compile, be submitted through the guru server as demonstrated during the first week of the semester in lab, and submitted prior to the posted due date to be considered for partial credit. **It cannot be said enough, start your work early!**

Course Programming and Documentation Standards Reminders:

- 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.**
- Indent all code found within the main and all user-defined functions **exactly** two spaces.
- Indent all code found within the body of relevant selection and repetition constructs exactly two additional spaces.
- Make use of `{ and } with all relevant section and repetition constructs.
- 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 (must be named `hw06.c`), to the correct assignment (`hw06`), and to the correct lab section (CRN value).
- 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 lab 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.

**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.

---

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