Source File: `~/public_html/lab17.php`
Input: HTML/PHP Form or URL
Output: Standard output (HTML Code)
Value: 4

In this assignment take aspects from Labs 15 and 16 to create a new application that will allow a user to enter a Jumble word and be shown all permutations of the letters of the word. The word will be provided via input from a form or a URL. The permutations should be numbered and appear in lexicographically increasing order.

Given n characters $\{1, 2, \ldots, n\}$, generate and list all permutations of those characters. For example, if $n = 4$ and the digits from 1 through 4 are the characters, the permutations are

```
1234    2134    3124    4123
1243    2143    3142    4132
1324    2314    3214    4213
1342    2341    3241    4231
1423    2413    3412    4312
1432    2431    3421    4321
```

There are $4! = 24$ permutations listed, and in general there are $n!$ permutations of n objects.

What is needed is an algorithm that generates one permutation from the immediately preceding permutation without generating any patterns that have to be discarded. For example, how might we operate on the four-digit permutation 1432 to produce 2134, the permutation of next higher (apparent) value? Looking at only this single example is unlikely to suggest an answer, but inspecting the entire sequence of 24 four-digit permutations listed earlier allows us to deduce the following algorithm for transforming a given permutation p, considered as a one-dimensional array (vector) of characters, into the permutation $p\prime$, the next higher permutation:

1. Scan p from right to left until an element is found that is less than its neighbor to the right. Call the index position of this element i. If this condition cannot be satisfied, the algorithm terminates.
2. Scan again from right to left until an element is found that is greater than the one at i. Call the position of this element j.
3. Exchange the i^{th} and j^{th} elements.
4. Reverse the elements from the $(i + 1)^{\text{st}}$ position to the right.

For $n = 4$ and $p = 1432$, the first element less than its right neighbor is the 1 in the first position. Then, scanning again from right to left, the first element greater than the 1 in the first position is the 2 in the fourth position. Exchange these elements to get 2431. Now reverse all elements from the second position on, to obtain 2134, which is indeed the successor to 1432 in the earlier listing of all 24 four-digit permutations. For $n = 7$ and $p = 3176542$, the first element less than its right neighbor is the 1 at the second position. Scan again from right to left to find the first element greater than 1, which is the 2 at the seventh position. Exchange the 1 and 2 to obtain 3276541. Now reverse all elements to the right of the second position to obtain $p\prime = 3214567$.

If the n characters being permuted are distinct, there are $n!$ permutations; however, if the n characters are not distinct, there are fewer permutations. For example, suppose the characters `abba` are to be permuted. The above algorithm can be used and shows the permutations to be:

```
aabb    baab
abab    baba
abba    bbaa
```

The program needs to be able to handle the following error cases:

- variable `word` missing
- variable `word` empty
- variable `word` contains non-alphabetic characters
- variable `word` has length less than four (4) or greater than seven (7)

The output should be formatted as shown in the instructor's version of this program.

Some additional notes for this assignment:

- Insert an HTML comment at the top of the document identifying you as the author, the class, and the assignment number.
- Add an echo statement to the beginning of the script section that will display your name, the course number, and the assignment number.
- Since this assignment uses several PHP code blocks, it's always a good idea to check for syntax errors. You can do this by using the `-l` option to the `php` command at the command line as in

```
1 newuser@csunix ~/public_html> php -l lab17.php
2 No syntax errors detected in lab17.php
```

- You should always validate the rendered HTML code. The validator is discussed near the top of p. 6 and in Appendix A on pp. 629–631. By including the following link and image, a user will be able to click the image and receive a report from the validator.

```
1 <?php
2   $location = 'https://' . $_SERVER['HTTP_HOST'] . $_SERVER['REQUEST_URI'];
3   $location = urlencode($location);
4   echo '<a href="https://validator.w3.org/nu/?doc=' . $location . '">';
5 ?>
6   <img src="https://www.w3.org/QA/Tools/I_heart_validator"
7    alt="I heart Validator logo" height="31" width="80" />
8 </a>
```

After the document is valid, open it in your Web browser to see how it renders.

Upon completion of this assignment, submit your source file via Blackboard. Only submit your PHP source; do **not** submit the form (your program will be invoked via a URL).