Shell Programming - Points: 68 (10 of 20%)

On the Course Linux Server, write and submit as **`datsubmit 33`** an executable shell script named **`test3.sh`** that will do the following actions, in the exact order given below. You will write approximately 45-50 lines of executable code. For full marks, you must put a one-line comment containing the step number in front of the executable code in each step. Do not put a Step Number comment as the first line of the file!

Summary and Purpose (what this script will do):

> The first argument to this script is a positive integer number of lines. The second argument is an optional message string. The third argument is an optional file name. The script will compare the number of lines in the file to the given number and output the message.

---

1. *[Points: 5]* Structure your script using the standard nine-part format given in DAT2330 Notes file **`script_style.txt`** ; however, do *not* include the Purpose or Assignment Label (parts 4&5).
2. *[Points: 9]* If the number of arguments is not one, two, or three, issue a good error message (follow the DAT2330 guidelines for error messages) and exit the script with status 2.
3. *[Points: 2]* Put the first argument (an integer number of lines) into a variable named **`basenum`** .
4. *[Points: 7]* If the number in variable **`basenum`** is not positive (greater than zero), issue a good error message and exit the script with status 3.
5. *[Points: 5]* If there is a second argument (an optional message string), put the second argument into a variable named **`message`** ; otherwise, put the string **`Test File`** into the variable.
6. *[Points: 6]* Make sure the string in variable **`message`** is not zero length (null); otherwise, issue a good error message and exit the script with status 4.
7. *[Points: 8]* If there is a third argument (an optional file name), put the third argument into a variable named **`myfile`** ; otherwise, prompt and get the missing file name from the user and put the file name entered by the user into variable **`myfile`** . Quick-exit the script with status 5 if the user signals EOF to the script. (No error message is needed.)
8. *[Points: 4]* Make the name in variable **`myfile`** lower-case. Quick-exit the script (no message) with status 6 if doing this fails.
9. *[Points: 7]* Make sure the name in variable **`myfile`** is the name of a plain file; otherwise, issue a good error message and exit the script with status 7.
10. *[Points: 3]* Put the count of lines contained in the user's file into a variable named **`count`** .
11. *[Points: 2]* Output one line containing the file name and number of lines it contains on standard output.
12. *[Points: 10]* Compare the number of lines in the user's file against the user's input number and output **one** message on standard output using **one** of the the following format templates:

```
MMM XXX is larger than base number YYY
MMM XXX is the same size as base number YYY
MMM XXX is smaller than base number YYY
```

where **`MMM`** is replaced in the template by the message string, **`XXX`** is replaced by the name of the user's file and **`YYY`** is replaced by the value of the user's number. These message templates are examples. Do not output the place-holder strings **`MMM`** , **`XXX`** or **`YYY`** . Output only **one** of the messages, for example:

```
Test File foo.txt is larger than base number 123
```

Put a one-line comment containing the step number on the line above the executable code in each step.