Shell Programming - Points: 68 (12 of 35%)

(I will pick up this file online from your account after the test ends.) Write an executable shell script named **`test4.sh`** that will do the following actions, in the exact order given below. You will write approximately 40-45 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):

> This script expects zero or one source file arguments. If the argument source file is missing, prompt and read it from the user. Detect whether the source file is a C language or C++ language program, and compile the program with the correct compiler. Save the warning messages for later review.

---

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: 7]* Verify that there is either zero or one command line argument; otherwise, print an error message and exit the script with an error status. (i.e. Exit if there is more than one argument.)
3. *[Points: 7]* If there is no command line argument, prompt the user to enter the missing source file name and read the name from standard input. Put the source file name (either the command line argument or what the user entered from standard input) into the variable **`myprog`** for use in the rest of the script.
4. *[Points: 2]* Display this exactly punctuated sentence on stdout: **`Processing file 'XXX'.`** where **`XXX`** is the contents of the **`myprog`** variable containing the file name that the user specified.
5. *[Points: 14]* Verify that the path named by the **`myprog`** variable is a file; otherwise, exit non-zero with an error message. (i.e. Exit if the path is not a file.) Verify that the file named by the **`myprog`** variable is readable; otherwise, exit non-zero with an error message. (i.e. Exit if the file is not readable.)
6. *[Points: 10]* If the file named by the **`myprog`** variable contains the (9-character) string **`<iostream`** then compile the file using the C++ language compiler. Turn on all warnings during the compile. Name the compiled output program **`newcpp`**. Save any error or warning messages that may be generated on standard error by the C++ compiler into a file named **`warnings.out`** in the current directory. Save the exit status of the C++ compiler in a variable named **`cstatus`**.
7. *[Points: 2]* If the file named by the **`myprog`** variable contains the (6-character) string **`<stdio`** then compile the file using the C language compiler. Perform the all same functions as when you compiled the C++ program above, except name the compiled output program **`newccc`**. Avoid duplicating code.
8. *[Points: 7]* If the source file contains neither string, print an error message saying the program cannot be compiled and exit non-zero. Combine the above compile steps with this to avoid duplicate code.
9. *[Points: 3]* If the compiler has an error exit status, display this exact error message:
   `Compiler YYY exit status was ZZZ.`
   where **`YYY`** is the name of the compiler you used and **`ZZZ`** is the compiler error exit status.
10. *[Points: 4]* If the file **`warnings.out`** is empty, remove it, print a message on stdout saying **`Output is in AAA`**, where **`AAA`** is the name of the compiled program, and exit the script with a good status.
11. *[Points: 7]* Rename the file **`warnings.out`** to be **`warnings.txt`** and change the file permissions so that only the owner can read it. Display this exactly punctuated message on standard output:
    `Number of warning lines for 'XXX' is NNN.`
    where **`XXX`** is the name of the source file and **`NNN`** is the count of lines in the warnings file.

You will find sample C and C++ programs in the course notes: **`argv.c.txt`** and **`argv.c++.txt`**

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