Assignment Two
(100 points)

Introduction
In this assignment you will write a single shell script **chk** to output certain kinds of information about two different types of Unix objects: files and users. This will require processing of command-line arguments and the use of complex, nested conditional statements (if-then-else) to control program flow.
Note: The -u option of chk will only work on hills.

Note: This program gets its information from the command-line only. It should never ask the user a question.

Description
chk can be used to check a file if it is invoked as

chk -f filepath

or to check a user if it is invoked as

chk -u user

Here is the information it should report:

chk -f filepath

If **filepath** exists, output in readable sentences

•

if it is a symbolic link to anything, say it is a symlink, even if it is a broken symbolic link. You do not have to continue and report the permissions.

- if it doesn't exist, say so. Don't continue to report the permissions
- report what it is: file directory , or something else and continue to report the permissions
- report what combination of read, write and execute access rights your program has for the data. Note that this is dependent on who runs your program. Do not attempt to do this by looking at the permissions as output by **ls –l.** You must use the test operators to do this.

If **filepath** does not exist (and is not a symbolic link), your program should report this instead in an informative error message. In this case, you should exit with an error.

chk -u user

If the user exists on the system, report

- the path to the user's home directory
- if the user is currently logged in, say so. Otherwise, report when they last logged in. (Take some care so that this is generated reliably and quickly.)

If the user doesn't exist, report this in an informative error message, and exit with an error.
You may not use the finger or pinky programs to get your information for chk –u.

Your program must, of course, check its arguments well and report any inconsistencies. If your program detects an error it should give the user sufficient information to diagnose the error and rerun the program successfully. It should also set the exit status appropriately.

This program will be much easier if you think and map out its steps before you start coding!

To Hand In

Hand in a copy of your shell script and a standard script session with runs of your program on the following test cases. (These test cases are not exhaustive - do your own testing!)

chk -f
chk -f chk
chk -f .
chk -f /dev/null
chk -f <on a symbolic link that you create>
chk -f "my resume"
 # this file should not exist
chk –u jstrick2

Note: user names may be very similar. Your program should be very careful in how it obtains its information so as to get only the user name it wants. (To see the problem, run the finger command as finger gboyd) .) The purpose of the large number of test cases above is to discover bugs that occur with unexpected input. You may well have to do some work after running the test cases, so start your testing early!

Transfer a copy of your shell script as chk and of your script session showing **chk's** testing as **asmt02.script** using the standard procedure.

A copy of one version of **chk** 's output can be found in the **asmt02** directory on hills.

A bit of help

A pair of functions for you to use in **chk** may be helpful. Although we will not cover functions yet, these are very simple and are very easy to use, as described below. They will also save you some typing. The purpose of these functions are to help you to handle errors when your program diagnoses them and to allow you to output error messages in a standard format that can be changed in a central place. In addition, these functions write their error messages to standard error; another technique that we will cover later.

- The first function, named **error** outputs the error message you pass it to standard error in a standard format.
- The second function, named **fatal**, simply calls the **error** function with the error message you give it, then exits the program with an exit status to indicate failure.

I suggest you customize the contents of the functions to output the kind of information you want. Functions are called as if they are commands. The following calls to the function **error** generate the same error message:

error "File $file does not exist"
error File "$file" does not exist

you will get a slightly nicer error message with the form

error "File '$file' does not exist"

Each invocation of the **error** function above will output the error message and continue your program. If you wanted to exit the program instead, simply replace the function name with **fatal**

The functions are found in the file **asmt02/errfns.** . You can use them or not use them as you wish.