# Introduction to Programming

## Assignment: Task 3

### The Problem

For better or worse, most access to computer systems is still reliant on passwords. Various mechanisms exist to manage keeping records of users, their passwords, and the associated permissions. On traditional Unix systems the basic information (username, real name, password, etc) is stored in a plain text file usually called `/etc/passwd`.

Three command-line programs are used in the Unix world to maintain password information.

- `adduser` creates a new user, in the form of a new entry in the password file.
- `deluser` does the opposite, by removing the entry from the password file.
- `passwd` allows a user to change their password.

When a user attempts to access a system, the username and password they provide are checked against this file. This is done with a program called `login`.

This task involves developing implementations of each of these programs.

### The Task

Your task is to implement the four commands above.

You will provide implementations that will work for a simplified password file. The format of the file is as follows. (In this example `password_string` marks where the encrypted passwords are (they would obviously be different on each line)).

```
rms:Richard Stallman:password_string
ada:Ada Lovelace:password_string
jb:Justin Bieber:password_string
homer:Homer Simpson:password_string
```

The first field is the username, which always consists of lowercase letters. The second field is the user's real name, an arbitrary string. The third is their encrypted password. The three fields are separated with colons (`:`).

**This file is generated by the programs that maintain it (a default version is created when the system is installed), so it is safe to assume that it is always formatted as expected, which is as shown. The order of the lines in the file is not significant.**

A sample file is included in this repo. In this example the passwords are "encrypted" with ROT-13.

Your programs should work as follows:

adduser

Adds a new entry to the file. It can be added anywhere. The user should enter the desired username, the real name, and the password. If the username already exists, an error should be shown and no change should be made. It is fine for several users to have the same real name, or even the same password.

deluser

Removes an entry from the file. This should be based on the username, which is known to be unique. If the user is not found, the program should display a message to say nothing was changed.

passwd

Changes a user's password. The user should enter their username, and their new password. As is customary, for verification, the user should first enter their current password, and then the new password should be entered twice. No password should appear on the screen as it is typed. If the username is not found, the current password is invalid, or the passwords do not match, no change to the file should be made.

The password itself should be stored in some encrypted form. Anything is fine, even a simple substitution or trivial obfuscation. There are no extra marks for doing something more complicated!

Your fourth program should be a simple "login" where the user enters a username and password, and the program reports whether or not they would be allowed to access the system. The password should not be displayed as it is typed, but you may want to print it out for testing purposes.

In all these programs it is fine to hard-code the name of the file. This will be needed by several of the programs, but you should still store it in only one place. **Remember that it is safe to assume that the file exists, and can be read.**

Likewise, it is quite possible that one or more of your programs could require the same functions. Such code should **not** be duplicated in multiple programs. (Most likely, a well designed solution will contain one module with many functions, and four quite short programs that use these.)

The files you will be working with are quite small, so there is no need to be too concerned about the efficiency of your solution. Certainly you should assume that the whole file can be held in memory without any performance hit.

*In completing the tasks above you may use any module from the Python Standard Library that you find will help. Indeed, it will be very difficult to achieve good results without doing so!*

*If you want to use serious encryption for passwords, you are welcome to use something like the* `cryptography` *module from PyPi, but there is no need to do so. Nor are there any extra marks for doing so. If you use any modules from PyPi you should use a Virtual Environment* *(which should obviously not be in your repo), and include in your repo a clear statement of what you have used (the usual `requirements.txt`) file is fine.*

## Examples

Starting with an empty file, we use `adduser` to make an entry. (Note the password *is* displayed here for debugging and because this would normally be done by a trusted and reliable Sysadmin).

```
$ cat passwd.txt
$ ./adduser.py
Enter new username: ada
Enter real name:    Ada B Lovelace
Enter password:     babbage
User Created.
$ cat passwd.txt
ada:Ada B Lovelace:onoontr
```

Ada can now login (the password is not displayed, but was entered correctly):

```
$ ./login.py
User:     ada
Password:
Access granted.
```

Other users cannot:

```
$ ./login.py
User:     rms
Password:
Access denied.
```

A user with the same username as Ada cannot be added:

```
$ ./adduser.py
Enter new username: ada
Enter real name:    Ada Byron King
Enter password:     difference
Cannot add. Most likely username already exists.
```

Of course, that user can be added if they choose a different username:

```
$ ./adduser.py
Enter new username: abking
Enter real name:    Ada Byron King
Enter password:     difference
User Created.
$ cat passwd.txt
ada:Ada B Lovelace:onoontr
abking:Ada Byron King:qvssrerapr
```

The original user can be deleted. And `abking` can change their password (here it is changed to ada's original password):

```
$ ./deluser.py
Enter username: ada
User Deleted.
$ cat passwd.txt
abking:Ada Byron King:qvssrerapr
$ ./passwd.py
User:             abking
Current Password:
New Password:
Confirm:
Password changed.
$ cat passwd.txt
abking:Ada Byron King:onoontr
```

These examples do not cover every possibility. You should ask if you are unsure what should happen in a particular case.