EECS 22: Assignment 1

Prepared by: Yasaman Samei, Prof. Rainer Dömer

September 26, 2013

Due Monday 7 Oct 2013 at 11:00pm

1 Part1: Login to your Linux account

For this class, you will be doing your assignments by *logging on* to a shared machine (server) running the Linux operating system. Even though you may be using a personal computer or a workstation that is capable of computation locally, you will mainly be using them as *terminals* (clients), whose job is to pass keystrokes to the server and display outputs from the server.

To use a shared machine, first you need an *account* on the machine. EECS support has created an *account* for each student. To retrieve the username and password go to the following website:
`https://newport.eecs.uci.edu/account.py`.
The website asks for your UCInetID and the according password before giving you the account information of your new EECS account. Note that your browser may also ask you to accept a certificate to open the secure website. If you have a problem please contact your EECS 22 TA, (eecs22@eecs.uci.edu).

The name of the instructional server is `crystalcove.eecs.uci.edu.` You can log into your account with your EECS user name and password. Your account also comes with a certain amount of disk space. You can use this space to store homework assignment files, and you don't need to bring your own disks or other storage media.

1.1 Software and commands for remote login

You can connect to `crystalcove.eecs.uci.edu` from virtually any computer anywhere that has internet access. What you need is a client program for *remote login*.

Previously, people used **`rlogin`** or **`telnet`** to connect to the server, and **`ftp`** or **`rcp`** to transfer files. However, these protocols are insecure, because your keystrokes or output are in clear text and can be *snooped* by others. This means your account name and password can be stolen this way. So, for security reasons, do not use either of these programs.

Instead, use **`ssh`** as the primary way to connect to the server. **`ssh`** stands for *secure shell*, and it encrypts your network communication, so that your data cannot be understood by snoopers. For file transfers, use **`sftp`** or **`scp`**, which are secure.

Depending on what computer you use, it may have a different *implementation* of **`ssh`**, but the basic function underneath are all the same. Check out OIT(NACS)'s page on SSH:
`http://www.nacs.uci.edu/support/sysadmin/ssh_info.html`
or check the course web site:
`https://eee.uci.edu/13f/18040/resources.html`

- If you are logging in from a Windows machine, you can use **`SecureCRT`** or **`PuTTY`**.
- MacOS X already has this built-in (use Terminal or X11 to run a Linux shell). Most Linux distributions also bundle **`ssh`**.
- If you are logging in from an X terminal, you can use the command
  ```
  % ssh crystalcove.eecs.uci.edu -X -l yourUserName
  ```
  (note: `%` is the prompt, not part of your command) It will prompt you for your password. Note that the `-X` option allows you to run programs that open X windows on your screen.

1.2 Linux Shell

By now you should be logged in, and you should be looking at the prompt
`crystalcove% _`

Note: in the following writeup, we will show just
`%`
for the prompt, instead of
`crystalcove%`

You should change your password using the **`yppasswd`** command.

Try out the following commands at the shell prompt (See reference to the Linux Guide in section 1.3 for more details about these commands.).

`ls`	list files
`cd`	(change working directory)
`pwd`	(print working directory)
`mkdir`	(make directory)
`mv`	(rename/move files)
`cp`	(copy files)
`rm`	(remove files)
`rmdir`	(remove directory)
`cat`	(print the content of a file)
`more`	(print the content of a file, one screen at a time)
`echo`	(print the arguments on the rest of the command line)

Most commands take one or more file names as parameters. When referring to files, you may need to qualify the file name with directory references, absolute vs. relative paths:

`.`	(current directory)
`..`	(one level higher)
`~`	(home directory)
`/`	the root (top level) directory

1.3 Follow the Linux Guide

The best bet may be to search online for something like "linux user tutorial," "linux user guide," "unix command line" or "unix shell command" and check a few results to see what is agreeable to you. From those links, the following may be reasonable:
`http://linux.org.mt/article/terminal`
`http://www.linux-tutorial.info/modules.php?name=MContent&pageid=49`
`ftp://metalab.unc.edu/pub/Linux/docs/linux-doc-project/users-guide/user-beta-1.pdf.zip` (3.3.1-2, and chapter 4)
`http://www.broadbandexpert.com/guides/ultimate-linux-guide/`
or
`http://www.nacs.uci.edu/help/manuals/uci.unix.guide/`
Learn basic shell commands: list files, change directory, rename files, move files, copy files, show file content.

There is nothing to turn in for this part.

2 Learn to use a text editor

There are three editors that are available on nearly all Linux systems that you may choose from.
`pico` is the easiest to get started with. A guide for **`pico`** can be found at:
`http://www.dur.ac.uk/resources/its/info/guides/17Pico.pdf`.
`vi` is a very powerful editor, but is arguably a bit more difficult to learn. Follow the **`vi`** guide at:
`http://www.nacs.uci.edu/help/manuals/uci.unix.guide/the_vi_editor.html`.

Finally, **`emacs`** is another editor that you may use. **`emacs`** is also a powerful editor, but is a bit easier to learn than **`vi`**. Follow the **`emacs`** guide at:
`http://www.nacs.uci.edu/help/manuals/uci.unix.guide/editing_with_gnu_emacs.html`.

Learn how to edit a file, move the cursor, insert text, insert text from file, delete words, delete lines, cut/paste, save changes, save to another file, quit without saving.

There is nothing to turn in for this part. However, it is critical that you get enough practice with your editor, so that you can do the homework for this class.

3 Part 2: Rabbit Ecosystem and Demography [100 points]

Write a C program that analyzes rabbit and wolf population in an island and its influence on the grass area. Grass area in the island grows at a constant rate every year through rains, but is gradually depleted due to continuous consumption by rabbits. Rabbit population grows at a constant rate every year but is constrained by wolves attack. Wolf population grows at a constant rate every year but rapid decline occur in regular intervals through spread of epidemic diseases.

Your input section should look like the following:

```
Enter wolf population (initial): 10
Enter rabbit population (initial): 2300
Enter total grass area, initially fertile (in sq yards): 40000
Enter wolf annual growth rate (in percentage): 20
Enter rabbit annual growth rate (in percentage): 30
Enter grass area annual growth rate (in percentage): 5
```

These inputs along with below constraints will be used to calculate wolf population, rabbit population and available grass area, for each year.

Wolf Population (2nd Column): Wolf population grows annually at the specific growth rate input by user. Apart from this, every 5 years wolf population decreases to half of previous year population due to widespread epidemic diseases. Given initial wolf population, calculate wolf population for each year (Hint: Use modulo operator for computing population in disease spreading years).
Every year, wolf population needs to be updated based on its growth rate as follows.

```
WolfPopulation = PreviousWolfPopulation*(1+WolfGrowthRate/100)
```

Further, once in every 5 years except year 1 (i.e., only in years 6,11,16), Wolf population must be made half of previous year's Wolf population.

Rabbit Population (3rd column): Rabbit population grows annually at the specific growth rate input by user. Apart from this, each wolf kills 50 rabbits every year for their survival, apart from other prey.

```
RabbitPopulation = PreviousRabbitPopulation*(1+RabbitGrowthRate/100)
                   - PreviousWolfPopulation*50
```

Available grass area (4th column): Grass area grows annually at the specific growth rate input by user. Rabbits in this island consume grass continuously, and every year, each rabbit depletes grass in 1.2 sq yards of fertile grass area, and make it barren (due to rabbit consumption and insufficient rains for regrowth).

```
GrassArea = PreviousGrassArea*(1+GrassGrowthRate/100)
            - PreviousRabbitPopulation*1.2
```

Your program should calculate and print the following data in a table format: Year, Wolf population in specific year, Rabbit population in specific year, Available grass area (in sq yards). Your final program output should look like the following

```
Enter wolf population (initial): 10
Enter rabbit population (initial): 2300
Enter total grass area, initially fertile (in sq yards): 40000
Enter wolf annual growth rate (in percentage): 20
Enter rabbit annual growth rate (in percentage): 30
Enter grass area annual growth rate (in percentage): 5
Year            Wolf population Rabbit population       Available Grass Area
 0              10              2300                    40000.00
 1              12              2490                    39240.00
 2              14              2637                    38214.00
 3              16              2728                    36960.30
 4              19              2746                    35534.72
 5              22              2619                    34016.25
 6              11              2304                    32574.26
 7              13              2445                    31438.18
 8              15              2528                    30076.09
 9              18              2536                    28546.29
10              21              2396                    26930.40
11              10              2064                    25401.72
12              12              2183                    24195.01
13              14              2237                    22785.16
14              16              2208                    21240.02
15              19              2070                    19652.42
16               9              1741                    18151.04
17              10              1813                    16969.39
18              12              1856                    15642.26
19              14              1812                    14197.18
20              16              1655                    12732.63
```

NOTES: Use a loop so that table is printed for 20 consecutive years as shown above. For year 0, only initial values are displayed. For available grass area, use double type variables, and print out exactly 2 digits after the decimal point.

Also ensure that all the numbers in the output table line up nicely so that the decimal points are all at the same column position. Use suitable number of tab characters in print statements to align the table output in same fashion as above. Wolf and rabbit population are integers, and fractions due to growth rate multiplications need to be ignored (equivalent to flooring).

You should submit your program code as file **rabbit.c**, a text file **rabbit.txt** briefly explaining how you designed your program, and a typescript **rabbit.script** which shows that you compile your program and run it.

Your script file should contain output for following two scenarios.

```
A. Wolf population = 10, Rabbit population = 2300, Initial grass area = 40000
   Wolf growth rate = 20%, Rabbit growth rate = 30%, Grass growth rate = 5%
B. Wolf population = 15, Rabbit population = 6500, Initial grass area = 90000
   Wolf growth rate = 25%, Rabbit growth rate = 20%, Grass growth rate = 10%
```

4 Bonus: Minimum and maximum population [5 Points]

Calculate and print the minimum and maximum population of rabbits and wolves over these 20 years. Along with the minimum and maximum values, also print the corresponding years. Maintain variables for minimum and maximum wolf population and rabbit population seperately. For each case, maintain same number of variables for storing corresponding years. In your loop, update your maximum and minimum population by comparing with calculated population values. Also update corresponding year, when you update the population values.

Your program output should have an additional section (after the table in Problem 1), which should look like the following:

```
Wolf population was minimum as 9 in year 16
Wolf population was maximum as 22 in year 5
Rabbit population was minimum as 1655 in year 20
Rabbit population was maximum as 2746 in year 4
```

To submit, use the same files as in Part 2, i.e. **rabbit.c**, **rabbit.txt**, and **rabbit.script**. Just add your code lines for this bonus part in these files. Your script file should have bonus section output for both scenarios as discussed in Prob 1.

4.1 Writing your code

First create a subdirectory named `hw1` (for homework one). Change into the created directory `hw1`. Then, use your editor to create a C file named `rabbit.c`. Do not use a word processor and transfer or paste the content. The C file should state your name and exercise number as a comment at the top of the file.

4.2 Compiling your code

To test your program, it must be compiled with the **`gcc`** command. This command will report any errors in your code. To call **`gcc`**, use the following template:

```
% gcc sourcefile -o targetfile
```

Then, simply execute the compiled file by typing the following:

```
% ./targetfile
```

Note: Please compile your C code using **-ansi -Wall** options as below to specify ANSI code with all warnings:

Below is an example of how you would compile and execute your program:

```
% gcc rabbit.c -ansi -Wall -o rabbit
% ./rabbit
program executes
% _
```

5 Submission

To submit your work, you have to be logged in the server `crystalcove`.

Here is a checklist of the files you should have:

In the `hw1` directory, you should have the following files in your linux account:

- `rabbit.c`
- `rabbit.txt`
- `rabbit.script`

We do require these *exact* file names. If you use different file names, we will not see your files for grading. Now, you should change the current directory to the directory containing the `hw1` directory. Then type the command:

```
%  /ecelib/bin/turnin22
```

which will guide you through the submission process.
You will be asked if you want to submit the script file. Type yes or no. If you type "n" or "y" or just plain return, they will be ignored and be taken as a no. You can use the same command to update your submitted files until the submission deadline.
Below is an example of how you would submit your homework:

```
% ls # This step is just to make sure that you are in the correct directory that contains hw1/
hw1/
% /ecelib/bin/turnin22
==============================================================
EECS 22 FALL 2013:
Project "hw1" submission for eecs22
Due date:  Mon Oct 7 23:00:00 2013
* Looking for files:
* rabbit.c
* rabbit.txt
* rabbit.script
==============================================================
Please confirm the following:                              *
"I have read the Section on Academic Honesty in the        *
UCI Catalogue of Classes (available online at              *
http://www.editor.uci.edu/catalogue/appx/appx.2.htm#gen0)  *
and submit myoriginal work accordingly."                   *
Please type YES to confirm.  y
==============================================================
change mode
Submit rabbit.c [yes, no]?  y
File rabbit.c has been submitted
Submit rabbit.script [yes, no]?  y
File rabbit.script has been submitted
Submit rabbit.txt [yes, no]?  y
File rabbit.txt has been submitted
==============================================================
Summary:// ==============================================================
Submitted on Tue Sep 17 12:38:10 2013
You just submitted file(s):
rabbit.c
rabbit.script
rabbit.txt

% _
```

5.1 Verify your submission

This step is optional, but recommended. If you want to confirm which files you have submitted, call the following command:

```
% /users/grad2/doemer/eecs22/bin/listfiles.py
==============================================================
EECS 22 Fall 2013:  "hw1" listing for eecs22
==============================================================
Files submitted for assignment "hw1":
rabbit.c
rabbit.script
rabbit.txt

% _
```

This command lists your submitted files. Don't worry if you submitted too many files. We will only look at the files with defined names (here: `rabbit.c`, `rabbit.txt` and `rabbit.script`) and ignore other files.

6 Typescript

A typescript is a text file that captures an interactive session with the Linux shell. Very often you are required to turn in a typescript to show that your program runs correctly. To create a typescript, use the **`script`** command. Here is an example:

- Type the command
  ```
  % script
  ```
  into the shell. It should say
  ```
  Script started, file is typescript
  % _
  ```
  This means it is recording every key stroke and every output character into a file named "typescript", until you hit **`^D`** or type **`exit`**.

- Type some shell commands. But don't start a text editor!

- Stop recording the typescript by typing **`exit`**.
  ```
  % exit
  Script done, file is typescript
  % _
  ```

- Now you should have a text file named `typescript`. Make sure it looks correct.
  ```
  % more typescript
  Script started on Wed 26 Sep 2013 03:12:57 PM PDT
  ...
  ...
  ```

You should immediately rename the typescript to another file name. Otherwise, if you run **`script`** again, it will overwrite the `typescript` file.

Note: If you backspace while in script, it will show the `^H` (control-H) character in your typescript. This is normal. If you use **`more`** to view the typescript, then it should look normal.