THE UNIVERSITY OF SYDNEY

COMP2129 Assignment 1

Due: 6:00pm Wednesday, 1 April 2015

This assignment is worth 8% of your final assessment

Task description

In this assignment you will develop some simple C programs and shell scripts that interact with a transportation database. Commuters use their transport cards to tap on at the beginning each trip, and tap off at the end of each trip. The card is similar to the Opal card used in Sydney. The system will be used to validate transport cards, calculate the cost of each trip and find commuters who owe money.

All programs written throughout this assignment must be built using a shell script called **`build.sh`**. You can assume that no line of input will be longer than 100 characters, including the newline.

If you have any questions about any Unix commands or C functions, then refer to the corresponding **`man`** pages. You can ask questions about this assignment on ed. As with any assignment, make sure that your work is your own, and that you do not share your code or solutions with other students.

Fetching the assignment files

To obtain the files for this assignment, run the following command on the lab machines or `ucpu{1,2}`

```
$ /labcommon/comp2129/bin/get-assignment1
```

Academic declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Information Technologies, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of IT for the purpose of future plagiarism checking.

Task 1 Checksum validation (4 marks)

Every card in the system has a 10 digit identifier. The first 9 digits uniquely identify the card, while the last digit is a checksum. The checksum is used to validate cards and detect incomplete card reads.

The card identifier is only valid if it is 10 digits long, composed only of the digits 0-9, and the checksum digit is equal to the last digit of the sum of the first 9 digits. Here are some examples:

```
1234567895 has (1+2+3+4+5+6+7+8+9) = 45 and last digit 5 => valid
0011223346 has (0+0+1+1+2+2+3+3+4) = 16 and last digit 6 => valid
0011223347 has (0+0+1+1+2+2+3+3+4) = 16 and last digit 7 => invalid
```

Write a C program called **`checksum`** that reads a line at a time from `stdin` until `EOF` and writes the line to `stdout` if the first 10 characters forms a valid card identifier. Otherwise, ignore the line.

Here are some sample interactions with the program:

```
$ echo 1234567895 | ./checksum
1234567895

$ echo 1234567890 | ./checksum

$ cat cards.txt
1111111111
1234567890
1234567895
9999999991:sample

$ ./checksum < cards.txt
1234567895
9999999991:sample
```

Important – you must include a command in your `build.sh` script that compiles your C file `checksum.c` to an executable file `checksum`. Otherwise, your program will not be marked.

Task 2 — Fare calculator (4 marks)

A GPS system installed on every vehicle records the tap on and tap off locations of each commuter. Due to the low quality GPS modules, the locations are recorded in an unusual coordinate system (u, v). The (u, v) coordinates can be transformed into the true (x, y) coordinates using the formula:

$$x = au + bv$$
$$y = cu + dv$$

where a, b, c and d are the coordinate modifiers that are supplied as parameters to your program.

The commuter's fare in cents is then calculated using the Manhattan distance between their tap on and tap off coordinates. If the commuter taps on at (x_1, y_1) and taps off at (x_2, y_2), the fare would be:

$$\text{fare} = |x_1 - x_2| + |y_1 - y_2|$$

Write a C program called **`farecalc`** that takes four command line arguments that specify the values of a, b, c and d, as well as input from the `stdin` until `EOF`. The input should be read line by line, with each line containing four space separated integers u_1, v_1, u_2, v_2 where (u_1, v_1) are are tap on coordinates, and (u_2, v_2) are the tap off coordinates. For each line of the input, calculate the fare and write it to `stdout`. You may assume that the input will only be provided in the format below.

Here are some sample interactions with the program:

```
$ echo 1 1 2 2 | ./farecalc 1 0 0 1
2

$ cat locations.txt
1 2 3 4
0 -4 2 6

$ ./farecalc 1 2 3 4 < locations.txt
20
68

$ ./farecalc 1 -1 1 -1 < locations.txt
0
16
```

Important – you must include a command in your `build.sh` script that compiles your C file `farecalc.c` to an executable file `farecalc`. Otherwise, your program will not be marked.

Querying the database

Tasks 3 and 4 will both use a database file that stores entries in the following format:

```
<card identifier>:<customer name>:<balance>:<u1>,<v1>:<u2>,<v2>
```

Each entry contains information about one trip made by a particular customer and their given identifier. The balance shows how much money in cents is stored on the customer's card when they tapped on, the first pair (u_1, v_1) is the tap on location and the second pair(u_2, v_2) is the tap off location.

Here is an example of a database file:

```
0000000011:tim:987:1,2:3,4
0000000112:jony:65:5,6:7,8
0000001113:eddy:43:9,8:7,6
0000011114:phil:21:5,4:3,2
0000111115:craig:0:1,2:3,4
```

You can assume the database file exists, and the name field will not contain any colons or commas.

We suggest that you write tasks 3 and 4 as shell scripts, that use the programs that you developed for tasks 1 and 2 along with common Unix utilities such as `tr`, `bc`, `cut`, `paste`, `grep`, `awk` and so on.

Task 3 Total balance of all valid cards (3 marks)

Write a program called **`balance`** that calculates the total amount of money stored on all valid cards before any fares were charged. The program will take one argument: the path to the database file.

Here is an example using the database file above:

```
$ ./balance database.txt
1116
```

Task 4 Customers with an outstanding balance (3 marks)

Write a program called **`outstanding`** that outputs the names of the customers with valid cards who did not have enough money stored on their card to pay their fare. The program will take five arguments: the path to the database file and the coordinate modifiers a, b, c, d described in task 2.

Here is an example using the database file above:

```
$ ./outstanding database.txt 2 3 4 5
phil
craig
```

Submission details

Your submission must contain a `build.sh` script that will run without any arguments or input. The build script will prepare the binaries and scripts required for this assignment. After running your build script, if any of your programs are not in the correct format below then they will not be marked.

Here is a sample directory structure before building:

```
scott at ucpu2 in ~/comp2129/assignment1
  tests/
  build.sh
  Makefile
  source files: *.c, *.h, *.sh
```

Here is a sample directory structure after running `./build.sh`:

```
scott at ucpu2 in ~/comp2129/assignment1
  tests/
  build.sh
  Makefile
  source files: *.c, *.h, *.sh
      checksum: executable C binary
      farecalc: executable C binary
       balance: executable C binary or shell script
   outstanding: executable C binary or shell script
```

The Makefile

We have provided you with a Makefile that can be used to build and submit your code.
The included Makefile will only run correctly on the lab machines and undergraduate servers.

- **`$ make`**
  will use your own **`build.sh`** script to build your C programs and shell scripts.
- **`$ make submission`**
  will check your submission is valid, and submit a compressed archive of your assignment.

You are encouraged to submit multiple times, but only your last submission will be marked.

Marking

14 **marks** are assigned based on automatic tests for the *correctness* of your program.
This component will use our own hidden test cases that cover every aspect of the specification.

2 **marks** are assigned based on a manual inspection of the style and quality of your code.

Warning: Any attempts to deceive the automatic marking will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not properly follow the assignment specification, or your code is unnecessarily or deliberately obfuscated.