EECS 2031 SOFTWARE TOOLS, WINTER 2014
LAB 1

PRZEMYSLAW PAWLUK

1. Objective

Cellphones and other electronic devices must be off while you are in the lab. Get familiar with the Unix command line by trying a few commands listed below. Create and compile a simple ANSI-C program to get familiar with the process. Create an ANSI-C program called lab1.c that reads characters from input, classifies them, and outputs information based on the input. Test that your program correctly implements the required functionality Submit your solution electronically before the end of the lab session using the command

```
submit 2031 lab1 lab1.c
```

You may submit your solution more than once. Additional documentation about the submit command can be viewed by typing man submit. Short Reference of useful Unix commands

First, open a command line window/unix prompt by starting an xterm.

1.1. Commands related to directories/folders.

- `ls` lists current directory
- `cd <name>` change current directory to named directory, `cd ..` to move up in the hierarchy, `cd ~` to go back to your home directory
- `pwd` prints location of current directory, i.e., the path where you currently are.
- `mkdir <name>` creates new subdirectory with the given name in current directory

Hint: you can use the tab key for autocompletion of file and directory names. The up/down cursor keys can be used to scroll through the history of commands.

1.2. Commands related to compiling files.

- `gcc -o <name> <name>.c` compiles the named source file `"name.c"` into the executable `"name"`. That executable can then be started by typing `"name"`.
- `cat <name>` prints the contents of a file to the terminal.

Note that you will need to re-compile your program before you can test changes.

2. What to do

First, create a few subdirectories:

- create a subdirectory called "2031" in your home directory
- change the current directory to the newly created directory
- create a subdirectory called "lab1" in 2031

- change the current directory to the newly created directory

If you print the current directory with `pwd`, the system should show you now something like `/cse/home/cse99999/2031/lab1` (with your CSE account number). Then create a simple ANSI-C program and compile it.

Create a text file `lab0.c` your favourite text editor with your favourite text editor. Insert the following text into this file:

```
#include <stdio.h>

main() {
    printf("hello, world\n");
}
```

Use cat lab0.c to check that the file has the correct contents. Compile it with `gcc -o lab0 lab0.c` Run it by typing the command `./lab0` verify that it prints the correct output.

Now create a new ANSI-C program that does the following. For this, you should review these lecture slides. You may want to start with the program "getchar1.c". As far as control statements, such as if, are concerned, and logical operators, such as &&, you can use the same syntax as in Java.

2.1. **Requirements.**

(1) Your program must read individual characters from standard input.
(2) If the character is an uppercase letter ('A'...'Z'), the program must echo the letter to standard output. You can use `putchar()` for this.
(3) If the character is a lowercase letter ('a'...'z'), the program must convert them to uppercase characters.
(4) If the character is a decimal digit, the program must internally sum up the values of all entered digits, but must not generate output.
(5) If the character is a newline character, the program must output a newline character.
(6) For all other characters (including control characters), the program must output the character '.'.
(7) When the user hits `^D`, the `getchar` function will return EOF. In this case, the program must print a newline, followed by the sum of the digits (in decimal format, no leading zeros), followed by another newline.

For the purpose of this lab, you do not need to worry about overflow[1]. Assuming that the program is started with lab1, and given the following input from the keyboard (you have to type this):

```
This is a Test
0123456789
$;
^D
```

[1]If you need details about the encoding of characters in ASCII, refer to the following Wikipedia page http://en.wikipedia.org/wiki/ASCII.

where the last line has the user hitting Ctrl-D. Your terminal will show the following, where input and output is intermixed.

```
This is a Test
THIS.IS.A.TEST
0123456789

$;
..

45
```

The reason behind this behaviour is that input from the terminal is buffered on a line-by-line basis by default. Note the (correctly) generated output from the spaces in the first line.

2.2. **Hints.**

- If you subtract '0' from a decimal digit character, you will get the numerical value.
- It is usually easier to address each individual requirement to your code only after you have verified that the previous requirement is met by your program.
- Testing the program a different way:

To make testing your program a little easier, you should now create a text file with your favourite editor, which contains the input for your program. For your convenience, the above test input is available here as a UNIX text file input.txt. Now you can test your program as follows.

```
lab1 <input.txt
```

This command line syntax makes the program read the standard input from the named file. Not only does this save you the work to type your input every time you want to test your program, it also ensures that you give your program exactly the same input every time. This will reduce the potential for confusion and make it easier to debug your code. Given that the input is not echoed to the terminal when you start your program in this manner, the output will (and should) only be:

```
THIS.IS.A.TEST

..

45
```