Problem Set 1
Out of 60 points
Due Tuesday, September 16, 2014 at 11:55pm

Goals

• Get more comfortable with UNIX
• Solve some problems in C

Recommended Reading

• Sections 1 – 7, 9, and 10 of http://www.howstuffworks.com/c.htm
• Chapters 1 – 5, 9, and 11 – 17 of Absolute Beginner’s Guide to C
• Chapters 1 – 6 of Programming in C

1. Getting Started

Configure your programming environment.

a. Download the “Getting Started” document
   (http://marl.smusic.nyu.edu/CProgramming/Fall14/docs/2618_Getting_Started.pdf) and follow the instructions as necessary.

b. Open the terminal program on your computer (on a Mac it can be found in Application/Utilities). You should now be in your home directory.

c. To create a folder for this course in the home directory, type the following at the prompt in your terminal window:

   cd
   mkdir 2618
   cd 2618
   mkdir ps1
   cd ps1

   Your prompt should now resemble this:

   hostname:username ~/2618/ps1 %

   Following the initial “~” is your current working directory. Note that “..” represents your current working directory’s “parent directory.” Go ahead and type the following command to go “up one directory” (i.e., backwards to the parents directory):

   cd ..

   Part of your prompt should now look like this:
Note that . represents your current working directory. Go ahead and type the command below:

cd .

Your prompt should still look the same. If you ever get “lost” within your account, simply type the below to return to the “root” of your home directory:

cd
(which is the same as cd ~)

Now navigate your way back to ~/2618/ps1/. Put all work that you do for this problem set in this directory.

If you have any problems setting up your machine or stepping through this part of the problem set, please go to office hours and get help as soon as possible.

2. **Recreate Your Own “Hello World” (5 points)**

   From within your ps1/ directory, type the following command:

   ```
   vim hello.c
   ```

   If you prefer not to use vim, you can use another text editor to open and create files. However, you want to make sure your program can parse C code and color code text based on syntax for easy viewing and spotting of errors; other popular editors are, e.g. Sublime, XCode, Emacs.

   Now write your own version of “Hello World” that was discussed in class. You can type it character for character, but replace “Hello World” with your own argument to `printf`.

   Now type this command:

   ```
   gcc hello.c
   ```

   If you’ve made no mistakes, you should just see another prompt. If you’ve made a mistake, you’ll instead see an error message. Even if cryptic, think about what it might mean, then go find your mistake.

   Once your code is correct and compiles successfully, look for your program in your current working directory by typing the following command.

   ```
   ls -l
   ```
In addition to listing the contents of your current working directory, this command lists their sizes, dates and times of creation, and more. The output you see should resemble the following:

-rw-xr-x 1 username username 13340 Jan 23 19:49 a.out*
-rw-r--r-- 1 username username 373 Jan 23 19:49 hello.c

The -l is a “switch” (also known as a “flag” or “option”) that controls the behavior of ls. To look up more flags for ls (and its documentation in general), type:

`man ls`

You can scroll up and down through in this manual using your keyboard’s arrow keys and space bar. In general, anytime you’d like more information about some command, try checking its “man page” by executing `man` followed by the command’s name.

Now confirm that your program does work. Execute the command:

`a.out`

If you get a “command not found” error message, it means your working directory is not in your path. Go back to the Getting Started document (available from the course website) and read how to add your working directory to the path. Another option is typing `./` before the name of any executable located in your current directory:

`./a.out`

Before moving on, let’s give your program a more interesting name than a.out. Type the following:

```
gcc -o hello hello.c
```

In this case, `-o` is but a switch for `gcc`. The effect of this switch is to name `gcc`’s output `hello` instead of `a.out`. Let’s now get rid of your first compilation. To delete `a.out`, type the following:

`rm a.out`

If prompted to confirm, hit ‘y’ followed by Enter.

### 3. **Calculating Time (15 points)**

Write a program, in a file called `seconds.c`, that prompts for an integer that represents a number of seconds, and outputs to the user (by way of `printf`) how many hours and minutes are in the given amount of time.
Sample output:

Enter number of seconds: -832
Sorry, that is not a valid value. Try again.
Enter number of seconds: 4325
4325 seconds are equal to 1 hour, 12 minutes, and 5 seconds.

If the user fails to provide a positive integer, your program must simply quit (return 1) with an error message. You might be inclined to declare some variable as unsigned, but don’t—you would be unable to determine whether the user provided a negative number or a really big, positive one. (Why would this matter?)

Compile your program by typing

```
gcc -o seconds seconds.c
```

Assuming it compiled, run your program by typing

```
seconds
```

Test it out with a whole bunch of inputs to make sure it doesn’t misbehave. If you accidentally induce an infinite loop in this or any other program, you can usually terminate it by hitting ctrl-c.

Be sure that your code is thoroughly commented so that the each line’s functionality is apparent from comments alone. You will lose points for poor or no commenting.

4. Converting from MIDI Value to Pitch (10 points)

The Musical Instrument Digital Interface (MIDI) was invented in the early 80s, and has since become the standard protocol for communication between computers and various electronic music devices. Note values are represented in MIDI by 7-bit integer values ranging from 0 to 127, where the value 60 is equivalent to middle C on the piano; 59 represents B right below middle C; 61 represents C# right above middle C; 72 represents C an octave above middle C, etc.

In a file called midi.c, write a program that prompts the user for a MIDI note value between 0 and 127 and then prints out a note name for the value entered. Some sample output:

```
Please enter a MIDI note value: 130
Please enter a value between 0 and 127.  Retry: -2
Please enter a value between 0 and 127.  Retry:
The MIDI value 50 is equivalent to the pitch D.
```

Make sure you error check and comment your code thoroughly.
5. Calculating BMI (15 points)

An adult’s Body Mass Index (BMI) is a function of his or her weight and height, the formula for which is

$$\frac{w}{h^2} \times 703,$$

where $w$ is weight in pounds and $h$ is height in inches. According to the National Institutes of Health, adults’ weights can be categorized by BMI per the table below.

<table>
<thead>
<tr>
<th>BMI</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 18.5</td>
<td>Underweight</td>
</tr>
<tr>
<td>18.5 – 24.9</td>
<td>Normal</td>
</tr>
<tr>
<td>25.0 – 29.9</td>
<td>Overweight</td>
</tr>
<tr>
<td>30 or greater</td>
<td>Obese</td>
</tr>
</tbody>
</table>

In a file called bmi.c, write a program that first prompts users for their weight and height and then informs them of their BMI and status. Americans tend to think in terms of feet, so let users provide their height in feet plus inches. The aesthetics of your program are largely up to you, but your program must prompt users for input in this order: pounds then feet then inches. Functionally, then, your program must resemble the sample output below. Underlined are some sample inputs.

Weight in pounds: 165
Height (feet): -6
I doubt that.
Height (feet): 6
Height (inches): 2

Your BMI is 21.2. You are normal.

As implied by the above, you must require that users’ inputs be non-negative; rather than quit upon invalid input, let the user retry again and again. Also round your program’s floating-point output to one decimal place.

Be sure that your code is thoroughly commented to such an extent that lines’ functionality is apparent from comments alone.

6. The Resizable H (15 points)

Write a C program, in a file called h.c that reads a number from the keyboard, say $n$, and prints out a block letter H on the screen with sides of size $n$. $n$ must be an integer number from 1 to 50. Here you have some examples for $n = 3$, 2, and 1:
Make sure there is error checking on user input. Be sure you comment your code thoroughly.

7. Submitting Your Work

All assignments are submitted through New Classes. Students registered for the course are automatically given access. See course website under “Problem Sets” for more information on New Classes submission.

In order to submit, you first need to rename your ps1 directory and then compress it by doing the following: first cd to the directory above ps1. Rename your ps1 directory by typing

```
mv ps1 ps1_Lastname_Firstname
```

Then compress the newly renamed directory by typing

```
tar cvzf ps1_Lastname_Firstname.tgz ps1_Lastname_Firstname
```

Now submit ps1_Lastname_Firstname.tgz via the course NYU Classes page. Take care to submit before the problem set deadline, or you will not be graded. Be warned that the timestamps for submitted work are based on the server’s clock, not your computer’s clock.