

Problem Set 1: C

out of 55 points

due by 7:00 P.M. on Friday, 5 October 2007

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

Goals.

The goals of this problem set are to:

- Get you more comfortable with Linux.
- Have you solve some problems in C.

Recommended Reading.

Per the syllabus, no books are required for this course. If you feel that you would benefit from some supplementary reading, though, below are some recommendations.

- 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*.

Academic Honesty.

All work that you do toward fulfillment of this course's expectations must be your own unless collaboration is explicitly allowed (*e.g.*, by some problem set or the final project). Viewing or copying another individual's work (even if left by a printer, stored in an executable directory, or accidentally shared in the course's virtual terminal room) or lifting material from a book, magazine, website, or other source—even in part—and presenting it as your own constitutes academic dishonesty, as does showing or giving your work, even in part, to another student.

Similarly is dual submission academic dishonesty: you may not submit the same or similar work to this class that you have submitted or will submit to another. Moreover, submission of any work that you intend to use outside of the course (*e.g.*, for a job) must be approved by the staff.

All forms of cheating will be dealt with harshly.

You are welcome to discuss the course's material with others in order to better understand it. You may even discuss problem sets with classmates, but you may not share code. In other words, you may communicate with classmates in English, but you may not communicate in, say, C. If in doubt as to the appropriateness of some discussion, contact the staff.

Getting Started.

0. Visit the forum of **Final Project Ideas** on the course website's **Bulletin Board** and read over the ideas that have already been posted. Feel free to post zero or more of your own!
1. Assuming you already configured your FAS account for use in this course for Problem Set 0, SSH to `nice.fas.harvard.edu` and execute the following sequence of commands.¹

```
cd ~/cs50
mkdir ps1
cd ps1
```

Your prompt should now resemble the below.

```
username@nice (~ /cs50/ps1) :
```

Within your prompt's parentheses, recall, is a reminder of your current working directory. Know that `..` represents your current working directory's "parent directory." Go ahead and type the command below to go "up one directory" (*i.e.*, backwards).

```
cd ..
```

Your prompt should now resemble the below.

```
username@nice (~ /cs50) :
```

It's also worth knowing that `.` represents your current working directory. Go ahead and type the command below.

```
cd .
```

Your prompt should still look the same. Pointless, eh? Trust us, though, `.` does have its uses. Know that if you ever get "lost" within your account, simply type the below to be whisked back to the "root" of your home directory.

```
cd
```

Anyhow, navigate your way back to `~/cs50/ps1/`. (Remember how?) All of the work that you do for this problem set must ultimately reside in that directory for submission.

2. (5 points.) We've all printed things out before (including, perhaps, this problem set), but have you ever thought about how the process works? In a paragraph of your own words in a file called `printers.txt`, how do laser printers and, by contrast, inkjet printers work?²

¹ Recall that you created your `ps0/` directory with a somewhat different command for Problem Set 0. There are multiple ways to get the job done!

² For this question, you're welcome to consult *How Computers Work*, Google, Wikipedia, a friend, or anyone else, so long as your words are ultimately your own! Create `printers.txt` just like you did `questions.txt` for Problem Set 0!

Let's C What You Can Do.

3. (5 points.) Let's get you warmed up. From within your `~/cs50/ps1/` directory, go ahead and type the command below.

```
nano hello.c
```

Proceed to write your own version of “hello, world.” It suffices to re-type, character for character, Week 1's `hello1.c`, but do at least replace “hello, world” with your own argument to `printf`.

Once done with your recreation, hit `ctrl-x` to save, followed by `Enter`, and you should be returned to your prompt. Proceed to execute the command below.

```
gcc hello.c
```

If you've made no mistakes, you should just see another prompt. If you've made some mistake, you'll instead see some error message. Even if cryptic, think about what it might mean, then go find your mistake. To edit `hello.c`, re-execute Nano as before.

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

```
ls -l
```

More than just list 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 below.

```
-rwx----- 1 username student 6886 2007-09-27 18:04 a.out*  
-rw----- 1 username student  373 2007-09-27 18:03 hello.c
```

The `-l` is a “switch” that controls the behavior of `ls`. To look up more switches for `ls` (and its documentation in general), execute the command below.

```
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!

Let's now confirm that your program does work. Execute the command below.

```
a.out
```

You should see your greeting. Before moving on, let's give your program a more interesting name than `a.out`. Go ahead and execute the following command.

```
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`, execute the following command.

```
rm a.out
```

If prompted to confirm, hit `y` followed by `Enter`.

Welcome to Linux and C.

4. (15 points.) Most years have 365 days. But “leap years” have 366. Any year that is evenly divisible by 4 but not by 100 (unless it is also evenly divisible by 400) is a leap year. Put another way, any year that is evenly divisible by 4 is a leap year, unless it happens to be a centennial year, in which case it must also be evenly divisible by 400 in order to have 366 days. Accordingly, 1996 and 2000 were leap years, but 1900 was not.

Write, in a file called `leap.c`, a program that prompts the user for a positive integer and then informs the user (by way of `printf`) whether or not that year is a leap year. If the user fails to provide a positive integer, your program must not simply quit but instead keep prompting the user for a positive integer.³ You might be inclined to declare some variable as `unsigned`, but don't, lest you be unable to determine whether the user provided a negative number or a really big, positive one. (Why?) Seeing as most of us won't live until the year $2^{63} - 1$, it's fine to treat the user's input as an `int` as opposed to, say, a `long long`.

In fact, you'll probably want to use `GetInt` from CS 50's library for this program, in which case you'll need to put

```
#include <cs50.h>
```

at the top of your file. And you'll also need to compile your program with `-lcs50`, as in the command below.

```
gcc -o leap leap.c -lcs50
```

³ In the event you accidentally induce an infinite loop in this or any other program, know that you can usually terminate it by hitting `ctrl-c`.

Because of how GCC works, `-lcs50` must appear after the name of the file using the library (*i.e.*, at the end of this command). Assuming it compiled, run your program with the command below.

```
leap
```

Test it out with a whole bunch of inputs. If it misbehaves (or doesn't even compile), welcome to the world of debugging!

5. (15 points.) According to the Center for Disease Control (CDC), Body Mass Index (BMI) is a “reliable indicator of body fatness for most people and is used to screen for weight categories that may lead to health problems.” An adult’s 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. Adults’ weights can be categorized by BMI per the table below.

BMI	Status
< 18.5	Underweight
18.5 – 24.9	Normal
25.0 – 29.9	Overweight
> 29.9	Obese

Write, in a file called `bmi.c`, 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 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, do require that users’ inputs be non-negative; rather than quit upon invalid input, let the user re-try again and again. Also round your program’s floating-point output to one decimal place.

You’re on your own as to how to compile and run this program!

6. (15 points.) Toward the end of World 1-1 in Nintendo's Super Mario Brothers, Mario must ascend a "half-pyramid" of blocks before leaping (if he wants to maximize his score) toward a flag pole. Below is that very scene.



Write, in a file called `mario.c`, a program that recreates this half-pyramid using asterisks (*) for blocks. However, to make things more interesting, first prompt the user for the half-pyramid's height. (The height of the half-pyramid pictured above happens to be 8.) Then, generate (with the help of `printf` and one or more loops) the desired half-pyramid. Assume that the user's terminal (*i.e.*, SecureCRT or Terminal) window is exactly 80 characters wide by 24 characters tall. (Best to ensure that your own window boasts exactly those dimensions.) So that a blinking prompt still fits on the screen after a half-pyramid's generation, demand that the user provide a non-negative integer no greater than 23. Take care to align your half-pyramid, no matter its height, in the window's bottom-right corner (*i.e.*, 80 characters over and 23 characters down). Note that the rightmost two columns of blocks must be of the same height. No need generate the pipe, clouds, or Mario himself. Just the half-pyramid!

You're again on your own as to how to compile and run this program!

7. That's it for this problem set! No free point this time!

Submitting Your Work.

8. Ensure that your work is in `~/cs50/ps1/`. Submit your work by executing the command below.

```
cs50submit ps1
```

Thereafter, follow any on-screen instructions until you receive visual confirmation of your work's successful submission. You will also receive a "receipt" via email to your FAS account, which you should retain until term's end. You may re-submit as many times as you'd like; each resubmission will overwrite any previous submission. But take care not to re-submit after the problem set's deadline, as only your latest submission's timestamp is retained.