CSC161 2011S *Imperative Problem Solving*

Assignment 4: Fun with Bits, Bytes, and Beyond

Assigned: Wednesday, 16 February 2011
Due: 11:00 p.m., Wednesday, 23 February 2011

Summary: In this assignment, you will explore the underlying representation of some types and the consequences of C's type systems.

Purposes: To give you more experience with these key concepts in C.

Expected Time: Three to four hours.

Collaboration: I encourage you to work in groups of two or three students. However, you may work on your own or in groups of up to size four. You may discuss the assignment with anyone you wish, provided you clearly document such discussions.

Submitting: Submit tarballs of your solutions on Pioneerweb.

Warning: So that this assignment is a learning experience for everyone, I may spend class time publicly critiquing your work.

Assignment

Problem 1: Bit Flags

As you may recall, Unix file permissions are represented as an integer using bit flags. We might define the flags as follows.

```
#define OTHER_READ    04
#define OTHER_WRITE   02
#define OTHER_EXEC    01
#define GROUP_READ   040
#define GROUP_WRITE  020
#define GROUP_EXEC   010
#define USER_READ   0400
#define USER_WRITE  0200
#define USER_EXEC   0100
```

Using these definitions, write a procedure, `void printPerms(int perms)`, that prints out permissions in a form similar to `ls -l`. For example,

```
printPerms (USER_READ | USER_WRITE | GROUP_READ);
```
    rw-r-----
```
printPerms (0755);
```
    rwxr-xr-x

```
printPerms (0664);
```
rw-rw-r--

Problem 2: Examining Bits

Write a function, `void show_int_bits (int i)`, the prints out the bits in integer `i`. For example, given 5 as input, your program should output something like

```
00000000000000000000000000000101
```

or

```
00000000 00000000 00000000 00000101
```

or

```
0000 0000 0000 0000 0000 0000 0000 0101
```

or

```
00 000 000 000 000 000 000 000 000 000 101
```

To print the *i*th bit of a number, you can shift it right by *i* spaces and then bitwise "and" it with 1.

Problem 3: Interpreting Bits

Write a program, `twoscomp BBBBBBBB`, that reads a string of bits, interprets it in two's complement, and prints out the result.

For example,

```
$ twoscomp 00000000
0
$ twoscomp 00000101
5
$ twoscomp 11111011
-5
```

Problem 4: Representing RGB Colors

As you know, we traditionally represent an RGB color as a triplet of three values, representing the red, green, and blue components of the color. Each value is between 0 and 255.

Since each value has a restricted range, we can represent it as a single byte. Hence, we can fit all three components (and a little more information) into the typical integer, which is four bytes. That is, we can represent the color as the bits

```
00000000 RRRRRRRR GGGGGGGG BBBBBBBB
```

a. Write a function, `int rgb (int r, int g, int b)`, that combines the three components into a single integer.

b. Write a function, `int red (int rgb)`, that extracts the red component.

c. Write a function, `int green (int rgb)`, that extracts the green component.

d. Write a function, `int blue (int rgb)`, that extracts the blue component.

e. Write a function, `void print_rgb (int rgb)`, that prints out the rgb color as `(R,G,B)`.

Hint: Bit shifting and bit masking will be useful.

Submitting Your Homework

Using `script`, build logs of sample runs of your programs.

Put those logs, your source code, and any other files you deem appropriate in a directory called *`usernames`*`.hw4`.

Make a tarball of that directory.

Submit the tarball on Pioneerweb under the Assignment 4 link.
