Lab 1: Manipulating Bits Using C

Overview

The purpose of this assignment is to become more familiar with data at the bit-level representation. You'll do this by solving a series of programming "puzzles". Many of these puzzles may seem artificial, but in fact bit manipulations are very useful in cryptography, data encoding, implementing file formats (e.g., MP3), etc. By working your way through these problems, you will get very familiar with bit representations and hopefully will have some fun. You will also be doing some very basic pointer manipulations and arithmetic. Again, the purpose is to get you familiar with data representations and pointers.

Instructions

Perform an update to your course-materials directory on the by running the `update-course` command from a terminal window. On the script's success, you should find the provided code for lab1 in your course-materials directory. As a convenience, here is an archive of the course-materials directory as of this lab assignment: lab1_2.tar.gz.

The lab1 folder contains a number of tools, described later, a bits.c file, and a pointer.c file. Both files contain skeletons for the programming puzzles, along with a comment block that describes exactly what the function must do and what restrictions there are on its implementation. Your assignment is to complete each function skeleton using:

- only straightline code (i.e., no loops or conditionals);
- a limited number of C arithmetic and logical operators; and
- no constants larger than 8 bits (i.e., 0 - 255 inclusive).

The intent of the restrictions is to require you to think about the data as bits - because of the restrictions, your solutions won't be the most efficient way to accomplish the function's goal, but the process of working out the solution should make the notion of data as bits completely clear.

Similarly, you will start working with basic pointers and use them to compute the size of different data items in memory and to modify the contents of an array

The Bit Puzzles

This section describes the puzzles that you will be solving in bits.c. More complete (and definitive, should there be any inconsistencies) documentation is found in the bits.c file itself.

Bit Manipulations

The table below describes a set of functions that manipulate and test sets of bits. The Rating column gives the difficulty rating (the number of points) for each puzzle and the Description column states the desired output for each puzzle along with the constraints. See the comments in bits.c for more details on the desired behavior of the functions. You may also refer to the test functions in tests.c. These are used as reference functions to express the correct behavior of your functions, although they don't satisfy the coding rules for your functions.

Rating	Function Name	Description
1	bitAnd	x & y using only ~ and |
1	bitXor	x ^ y using only ~ and &
1	thirdBits	return word with every third bit (starting from the least significant bit) set to 1
2	getByte	Extract byte n from word x
3	logicalShift	shift x to the right by n, using a logical shift
4	bang	Compute !x without using !
Extra Credit:		
3	conditional	x ? y : z

Two's Complement Arithmetic

The following table describes a set of functions that make use of the two's complement representation of integers. Again, refer to the comments in bits.c and the reference versions in tests.c for more information.

Rating	Function Name	Description
2	fitsBits	returns 1 if x can be represented as an n-bit, two's complement integer
2	sign	return 1 if positive, 0 if zero, and -1 if negative
3	addOK	Determine if x+y can be computed without overflow
Extra Credit:		
4	isPower2	returns 1 if x is a power of 2, and 0 otherwise

Checking Your Work

We have included two tools to help you check the correctness of your work.

`dlc` is a modified version of an ANSI C compiler from the MIT CILK group that you can use to check for compliance with the coding rules for each puzzle. The typical usage is:

```
$ ./dlc bits.c
```

The program runs silently unless it detects a problem, such as an illegal operator, too many operators,

or non-straightline code in the integer puzzles. Running with the -e switch:

```
$ ./dlc -e bits.c
```

causes `dlc` to print counts of the number of operators used by each function. Type `./dlc -help` for a list of command line options.

`btest` is a program that checks the functional correctness of the code in bits.c. To build and use it, type the following two commands:

```
$ make
$ ./btest
```

Notice that you must rebuild `btest` each time you modify your bits.c file. (You rebuild it by typing `make`.) You'll find it helpful to work through the functions one at a time, testing each one as you go. You can use the `-f` flag to instruct `btest` to test only a single function:

```
$ ./btest -f bitXor
```

You can feed it specific function arguments using the option flags `-1`, `-2`, and `-3`:

```
$ ./btest -f bitXor -1 7 -2 0xf
```

Check the file README for documentation on running the `btest` program.

Advice

Start early on bits.c, if you get stuck on one problem move on. You may find you suddenly realize the solution the next day. Puzzle over the problems yourself, it is much more rewarding to find the solution yourself than stumble upon someone else's solution. If you do not quite meet the operator limit don't worry there will be partial credit, but often times working with a suboptimal solution will allow you to see how to improve it.

Do not include the `<stdio.h>` header file in your bits.c file, as it confuses dlc and results in some non-intuitive error messages. You will still be able to use printf in your bits.c file for debugging without including the `<stdio.h>` header, although `gcc` will print a warning that you can ignore.

You should be able to use the debugger on your code. For example:

```
$ make
gcc -O -Wall -m32 -g -lm -o btest bits.c btest.c decl.c tests.c
gcc -O -Wall -m32 -g -o fshow fshow.c
gcc -O -Wall -m32 -g -o ishow ishow.c
$ gdb ./btest
GNU gdb (GDB) Fedora (7.1-34.fc13)
Copyright (C) 2010 Free Software Foundation, Inc.
```

```
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Reading symbols from /homes/iws/dvhc/cse351/lab1/src/btest...done.
(gdb) b bitXor
Breakpoint 1 at 0x8048717: file bits.c, line 144.
(gdb) r
Starting program: /homes/iws/dvhc/cse351/lab1/src/btest
ScoreRatingErrorsFunction

Breakpoint 1, bitXor (x=-2147483648, y=-2147483648) at bits.c:144
144}
(gdb) p x
$1 = -2147483648
(gdb) p/x x
$2 = 0x80000000
(gdb) q
A debugging session is active.

Inferior 1 [process 12728] will be killed.

Quit anyway? (y or n) y
```

The `dlc` program enforces a stricter form of C declarations than is the case for C++ or that is enforced by `gcc`. In particular, in a block (what you enclose in curly braces) all your variable declarations must appear before any statement that is not a declaration. For example, `dlc` will complain about the following code:

```
int foo(int x)
{
    int a = x;
    a *= 3;     /* Statement that is not a declaration */
    int b = a;  /* ERROR: Declaration not allowed here */
}
```

Instead, you must declare all your variables first, like this:

```
int foo(int x)
{
    int a = x;
    int b;
```

```
    a *= 3;
    b = a;
}
```

Using Pointers

This section describes the four functions you will be completing in pointer.c that is also in the lab1 folder you downloaded. Refer to the file pointer.c itself for more complete details.

Pointer Arithmetic

The first three functions in pointer.c ask you to compute the size (in bytes) of various data elements (ints, doubles, and pointers). You will accomplish this by noting that arrays of these data elements allocate contiguous space in memory so that one element follows the next. **You are permitted to use casts for these functions.**

Manipulating Data Using Pointers

The fourth function in pointer.c asks you to change the value of an element of an array using only the starting address of the array. You will add the appropriate value to the pointer to create a new pointer to the data element to be modified.

The next two functions deal with bounds checking of pointers and the final function deals with more bit manipulation.

Checking your work

For pointer.c, we have included a simple test harness: ptest.c. You can test your solutions like this:

```
$ make ptest
$ ./ptest
```

This test harness only checks if your solutions return the expected result, not if they meet the specific criteria of each problem. We will review your solutions to ensure they meet the restrictions of the assignment.

Submitting Your Work

You will be able to submit your assignment with the `submit-hw` script that is bundled with the lab1 course-materials update. Using the script should be straight-forward, but it does expect you to not move/rename any files from the "course-materials" directory. Open a terminal and type the following commands:

```
  submit-hw lab1.bits
  submit-hw lab1.pointer
```

This will submit both bits.c and pointer.c (one file per command). However, you may just want to submit all parts for Lab1 together. You may use the following command to do so. It will save you the trouble of calling both commands, but it will still count as a submission for each part.

```
submit-hw lab1
```

After calling the `submit-hw` script, you will be prompted for your Coursera username and then your submission password. **Your submission password is NOT the same as your regular Coursera account password!!!!** You may locate your submission password at the top of the assignments list page.

After providing your credentials, the `submit-hw` script will run some basic checks on your code. For bits.c, it will run the `driver.pl` script and warn you if you get below a score of 37 (full-credit without the extra-credit). For pointer.c, it will simply check to see if `ptest` reports any failures to the tests.

Once confirming that you wish to submit, with a working Internet connection, the script should submit your code properly. You can go to the assignments list page to double-check that it has been submitted and check your score as well as any feedback the auto-grader gives you. You may also download your submission code from this page, if you wish.