Programming Assignment 2

Computer Organization
CSI404
Spring 2015

Finish and PUSH YOUR WORK: Due Thursday, Feb. 3, 11:59PM.

Learning Goals and Assignment

How to achieve the goals is detailed after we state them.

1. Review C functions, their definition, their calling, and their parameters.
2. Reorganize an already debugged and acceptable C program so that code like your
   `printf("Output in decimal:%d\n", inData);`
   is moved from `main()` to a separate function like
   `out_decimal(int param) { ... }`
   and the original code is replaced by a function call. This is an example of what professionals call "refactoring". You will also refactor the decimal input functionality.
3. TEST, DEBUG and PERFECT a refactored program, and then save your work into a new commit in your GIT repository. In professional incremental software development, a refactored program is always completely retested before new functionality is added. Your record of tests captured by `script` will be count in your score!
4. Learn to FIRST create an output function that will help you test a more difficult function that you plan to create later.
5. Understand binary better by programming a function named `in_bin8` that reads an 8 bit binary number in the form of sequence of eight `0`s and `1`s that you type, and converts it to a C `int` value.
6. Use the output function to TEST, DEBUG and PERFECT your newly written `in_bin8` function.
7. Practice saving into your GIT repository each new version of your program obtained by adding new functionality and actually completing of full tests.
8. Understand C's bitwise "bit-by-bit" logical or boolean operations and its shift operations better by writing, TESTING, DEBUGGING and PERFECTING a function `out_bin8(int x)` that prints the lowest 8 bits of its parameter value as a sequence of eight `0`s and `1`s printed after
   `Output in binary:`
9. After making the commit with your completed project, practice pushing your work to our remote GIT server where the course staff can retrieve and grade it.

Study sections 2.4, 2.5 and 2.6 of your Patterson and Hennessy textbook to learn what you need to know for this project. Also, recall, and learn new ways, to find the C programming details you need to apply from your C course (CSI333) textbooks, the discussion of C examples in Hennessy and Patterson, and other references you learn about.

OK, here is what to do:

(If you have NOT done Programming Project 1 yet, even if it is too late for credit, you MUST do it in order to move on to Project 2 which builds on Project 1.)

1 Version where decimal input and output code is refactored into functions

TEST your Programming Project 1 program again so that a sample run looks like this:

```
% ./DataDemo
Data demonstrator program by Sample Student
Please type an int in decimal:2015
Output in decimal:2015
Please type an int in decimal:-1234598
Output in decimal:-1234598
%
```

Fix it if necessary. If you fix it, RECOMPILE, TEST AGAIN, and, when you get it right, save it in a commit with the command: `commit -a -m "HW 1 assignment finally completed correctly."`

Of course, the first 2015 is sample input somebody typed and the program MUST output THE SAME legal C int value a person types, NOT just the fixed, example values of 2015 and -1234598.

Do the refactoring! Review and get any help you need about (1) defining C functions like
`int in_decimal( void ) { ... }`
and
`void out_decimal( int what_to_output )`
and (2) calling them.

Do the testing! COMPILE, fix any syntax errors, and TEST and repeat until your refactored program behaves EXACTLY THE SAME as your correct Project 1 solution.

The test MUST be done under the `script` command so the TAs have proof you really did the testing! (Remember to command `exit` in under the script when done.)

As usual, `git commit -a -m "`*Message*`"` your work. (If your message is not informative, you will lose points!)

2 Version with Hexadecimal Output

Since the first challenge is to make a function input a binary number one bit at a time, you should have a function to print the inputted number in hexadecimal so you can reliably tell that your input function works correctly.

Develop a function declared `void out_hex( int what_to_output )` that will print its parameter value in hexadecimal after `Output in hex:`
For example when it is called with parameter 254, it should print:
`Output in hex:0xfe`
The programming is easy after you find out that the right `printf` format code is `%#x`.

Revise your `main()` function so it demonstrates how your `out_hex` function gives hexadecimal output after your `out_decimal` function gives decimal output, of the same input number of course.

TEST, PERFECT, TEST under `script`, `exit` and COMMIT as usual. At least 6 of the tests MUST be on "round" hexadecimal numbers or their neghbors, such as 15, 16, -1, 256, 1023, etc.

3 Version to Input 8 bits and Convert Them to a C integer

Add to your program a function named `in_bin8()` that behaves similarly to `in_decimal` except that after it prints
`Type 8 bits of binary:`
it (1) inputs a binary number in the form of eight `0`s or `1`, (2) converts that binary number into a C int value, and (3) returns the converted value. AND OF COURSE program your `main` to test and demonstrate this by reprinting the inputted binary number in hexadecimal and decimal. The hexadecimal output makes testing of the conversion very easy.

How can you do the input? The first idea is use the operation
`scanf("%1d", &input_int_variable );` in a loop. The field width qualifier `1` (one) in the format code makes it read just one digit.

Second, you may have to consult Patterson and Hennessys' section 2.4 "Signed and Unsigned Numbers" to learn the mathematics of binary that you need to know. The loop will process the bits one at a time in a calculation that updates the value that the `in_bin8` function returns after the loop is finished.

Probably the most efficient way to DIRECTLY apply the first formula of section 2.4 is to use the C bitwise "Bit-by-bit Logical Operations" explained in section 2.6. How?

1. Use the left shift `<<` to compute each power of 2. For example, `1 << 0` evaluates to 1 which is 2^0, `1 << 1` evaluates to 2 which is 2^1, `1 << 2` evaluates to 2^2 equals 4, and `1 << 3` evaluates to 8 (why?). In general, you can program the computation of 2^i within the body of a loop with C code `2 << i`.

2. You can use either plain addition (`+`) or bitwise OR (`|`) to make the loop add the current term, either 0×2^i or 1×2^i, into the `int` variable for accumulating the sum. It's important to know that addition and bitwise OR both work because developers use them.

   Beware of the COMMON MISTAKE to confuse bitwise OR and AND programmed with `|` and `&`, with the OR and AND, programmed with `||` and `&&` used in the logic coded in `if( )` and `while()...` The value of an `&` or `|` expression is ALWAYS zero (`0x0`) or one (`0x1`); the value of an `&&` or `||` expression is typically has many of its bits on and many bits off.

   You will learn during CSI404 why computing by add and by bitwise OR are equally fast in today's CPUs.

Test, debug, perfect, capture tests with `script`, and COMMIT as usual! (Proof of testing and informative messages will always count!)

4 Version to Output 8 Bits of a C int Value

C integer values are stored in binary, so of course the shift and bitwise logical operations of C provide the most efficient way to generate the `0`'s and `1`'s to display a C value in binary. Suppose the value whose bits you want to print is stored in the C int variable named `value`. Here are the tricks:

1. Suppose i is a bit index (as in section 2.4) that ranges from 7 down to 0. Then `(value & ( 1 << i ))` evaluates to 2^i when the bit at index i is 1, and it evaluates to 0 with that same bit is instead 0.

2. You don't want to print 2^i (which looks like 1, 2, 4, 8, 16, 32, etc.) when you should print one bit (which looks like 0 or 1). The calculation programmed by `x >> i` (right-shift) produces 0 if `x` equals 0 (good) and produces 1 if `x` equals 2^i (also good).

What to do:

1. Use these tricks within a loop to develop your own C function named `out_bin8` that prints the lower 8 bits of its integer parameter value. Its printout should HAVE THE SAME STYLE as our other `out_` functions.

2. Make a function for TESTING AND DEMONSTRATING `out_bin8` along with `in_bin8` for input and outputs also from both `out_decimal` and `out_hex`.

3. Program your `main()` function to print the program greeting and then allow you to run the TEST AND DEMONSTRATION of inputting numbers in 8 bit binary and outputting them in 8-bit binary, hexadecimal and decimal.

4. Of course, FULLY TEST YOUR WORK under `script`, and then do the COMMIT that must have an informative message.

Version with a 32 bit Binary Output Function

You will need this for future 404 projects where you will make C programs simulate the processing of the digital 0 and 1 electronic signals that CPU hardware does. Add the function named `print_bin32` that prints (this time, WITHOUT prefix like "Output in ..:") all 32 bits of its C int parameter value, in the readable form that looks like:
`0101 1011 1011 0000 1101 1111 0000 1011`
Make it end (as you did before) a new line. Tip: Use a doubly-nested loop! It will earn fewer points if it's done tediously without a loop.

Incorporate it into the data demonstrations, then `script` a few tests, and commit (with an informative message) your work as usual.

Getting Your Credit Points!

Do what might be your final review of your project AND VERY IMPORTANT your final tests before turning in your work for grading.

WRITE IN YOUR NOTEBOOK: THE COMMAND TO GET CREDIT:

`git push git404@git404.cs.albany.edu:submissions/`*YourNetID*`/COProjects master`

Of course, type in your own NetID where it says *YourNetID*! Otherwise, type the rest of the command literally.

What does this `git push` do? It it transmits (the first time) or updates (each time after the first) OUR COPY of your local repository to make OUR COPY (residing on the Internet at `git404.cs.albany.edu`) contain exactly the same things as your local repository.

Here is a sample of the what you should see:

```
$ git push git404@git404.cs.albany.edu:submissions/student/COProjects master
Counting objects: 36, done.
Compressing objects: 100% (18/18), done.
Writing objects: 100% (36/36), 3.42 KiB | 0 bytes/s, done.
Total 36 (delta 7), reused 4 (delta 0)
To git404@git404.cs.albany.edu:submissions/student/COProjects
 * [new branch]      master -> master
$
```

BUT ALWAYS DO what you hope is the FINAL REVIEW AND TEST of your project BEFORE you submit it the last time! However, GIT allows you to add changes and new commits, and then push your local repository again and again, if you realize you can improve your work after you upload it!

Use the command `git log` It runs a PAGER so you need to remember these single letter commands, especially `q`:

SPACE See the next page.

b Go back and see the previous page.

q Quit.

Final Version Sample

Many students like to see sample inputs and outputs for the final version of a project. Here we go:

```
% ./DataDemo
Data demonstrator program by Sample Student
Type 8 bits of binary:00000001
0000 0000 0000 0000 0000 0000 0000 0001
Output in binary:00000001
Output in hex:0x1
Output in decimal:1
% ./DataDemo
Type 8 bits of binary:11111111
0000 0000 0000 0000 0000 0000 1111 1111
Output in binary:11111111
Output in hex:0xff
Output in decimal:255
% ./DataDemo
Type 8 bits of binary:10101101
0000 0000 0000 0000 0000 0000 1011 1101
Output in binary:10101101
Output in hex:0xad
Output in decimal:173
% ./DataDemo
Type 8 bits of binary:00000000
0000 0000 0000 0000 0000 0000 0000 0000
Output in binary:00000000
Output in hex:0
Output in decimal:0
%
```

Some Frequently Asked Questions

4.1 How can I see what I turned in?

That's easy. What I would do and recommend is to (1) go to a temporary directory (or folder), (2) `git clone` a local copy of the repository holding your submission from `git404.cs.albany.edu`. (3) Examine it including its log to see what you turned in and we have. (4) DELETE IT SO you don't confuse this temporary copy with the stuff in your WORKING COProjects directory and your local repository you are using for real. Here are instruction steps on `itsunix.albany.edu`

1. Command: `cd /tmp`
2. Command: `mkdir` *YourNetID*
3. Command: `cd` *YourNetID* The above two commands will keep other student's stuff out of your way because the `/tmp` directory is shared.
4. Command:
   `git clone git404@git404.cs.albany.edu:submissions/`*YourNetID*`/COProjects`
5. Command: `cd COProjects`
6. Look around.
7. Use the command `cd ~/COProjects` to go back to the place where you are working for real.
   Use the command `/tmp/`*YourNetID*`/COProjects` to go back to looking at what you submitted.
8. When you done, VERY CAREFULLY remove all the temporary stuff with:
   `rm -rf /tmp/`*YourNetID*
9. Command `cd ~/COProjects` to go back to where you're working for real.