Assignment 1

Version 1.02 (last update: Sept. 5, 17:15)
Changes highlighted in yellow
Due date: Tue, Sep 22, 11:59 PM

Summary and Purpose

For this assignment, you will be writing a small C library that manipulates pointers, bits and bytes in order to better understand the C basic data types and to refresh your C programming skills.

Deliverables

You will be submitting:

1) A file called **`bandb.h`** that contains your function prototypes (see below).
2) A file called **`bandb.c`** that contains your function definitions.
3) A **`makefile`** that contains the instructions to compile your code.

You will submit all of your work via git to the School's gitlab server. (As per instructions in the labs.)

This is an individual assignment. Any evidence of code sharing will be investigated and, if appropriate adjudicated using the University's Academic Integrity rules.

Setting up your work environment and using git:

Start the SoCS VM. While running in the VM, begin by cloning an empty directory for your assignment using the command:

```
git clone https://gitlab.socs.uoguelph.ca/2520F20/skremer/A1
```

But, use your own login ID instead of **`skremer`**.

Then, **`cd A1`**, to move to the assignment directory.

Work in the A1 directory. Use the command:

```
git add filename
```

For each file file that you create as part of your code (one .c file, one .h file, and the makefile).

Every so often (especially if you get something working), use the command:

```
git commit -a
```

to commit your changes.

And then make sure to use:

```
git push --all
```

to push everything to the server.

If you want to check out what's on the server you can rename the A1 directory to A1.day1 and then use

```
git clone https://gitlab.socs.uoguelph.ca/2520F20/skremer/A1
```

to get a copy of exactly what is currently in the repo.

Function prototypes and descriptions for your assignment

```
void getbytes( unsigned char dest[], int bytes,
               void *src, int reverse );
```

This function should copy the byte values of exactly **bytes** bytes from the given **src** address to the unsigned char array **dest**. If **reverse** is **0** (false), then the bytes can be transferred in their original order. If **reverse** is non **0** (true), then the bytes must be in reversed order after the transfer.

It is the caller's responsibility that the memory addresses ranging from **src** to **src+bytes-1** are valid and readable memory addresses. It is the caller's responsibility that the memory addresses ranging from **dest** to **dest+bytes-1** are valid and writable memory addresses. It is the function's responsibility not to access memory locations outside of these ranges.

```
void getbits( char dest[], int bytes,
              void *src,
              int start, int end );
```

This function should retrieve individual bits from the computer's memory, represent their values as the characters '0' and '1', and store them in the string dest, which must be null ('\0') terminated. The location of the first bit to be added to the string is indicated by the value of **start**, while the last bit to be added is indicated by the value of **end+1.** The value of **start** will not be smaller than the value of **end**. **start** and **end** values must be decoded by doing an integer division by the number 8 to yield both an integer **quotient** and an integer **remainder** (modulu 8). The value of **bytes-quotient-1** is the index of any bit's byte in the **src** array. The specific bit within the byte is given by the **remainder** with **7** begin the most significant bit, and **0** the least.

Example: suppose that **`bytes`** is **`4`**, **`start`** is **`30`** and **`end`** is **`22`**, you would be retrieving 8 bits and writing a 9 character string (8 plus null terminator) as follows:

src	[0]								[1]								[2]								[3]							
data	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
bit Index (start, end)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		start								end																						
quotient (byte)	3								2								1								0							
remainer (bit)	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	**7**	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
bits retreived		1	1	1	1	1	0	0	0																							

Adapted from:
https://en.wikipedia.org/wiki/Single-precision_floating-point_format#IEEE_754_single-precision_binary_floating-point_format:_binary32

It is the caller's responsibility to ensure that **`src`** has **`bytes`** readable bytes, and that **`dest`** has **`start-end`** writable characters. It is also the caller's responsibility to ensure that **`start>=end`**, that **`end`** is not-negative and **`start`** is less than 8 times bytes. It is the function's responsibility to read and write the correct bytes only for the correct locations.

`unsigned long long bits2ull( char *bits );`

This function should read characters from the string **`bits`**, and process up to, but non including, the terminating null character. The characters processed with be either the character **`'0'`** or the character **`'1'`**. The function should return an **`unsigned long long`** that is equal to the binary number that it read, character by character, from the highest valued column, all the way to the 1s column.

It is the caller's responsibility to ensure that **`bits`** is a null terminated string consisting of only the characters **`'0'`** and **`'1'`** and the terminating null character **`'\0'`**. It is the function's responsibility to not access any other memory.

`long long bits2ll( char *bits );`

This function should read characters from the string **`bits`**, and process up to, but non including, the terminating null character. The characters processed with be either the character **`'0'`** or the character **`'1'`**. The function should return a **`long long`** that is equal to the *two's complement* binary number that it read, character by character, from the highest valued column, all the way to the 1s column. The value of a two's complement number can be calculated by multiplying each binary digit by its column value, just like a binary number, except that the highest column value is negative.

E.g.

column value	-2^7=-128	2^6=64	2^5=32	2^4=16	2^3=8	2^2=4	2^1=2	2^0=1
-8	1	1	1	1	1	0	0	0
120	0	1	1	1	1	0	0	0
-128	1	0	0	0	0	0	0	0
127	0	1	1	1	1	1	1	1

The Last 20%

The above, constitutes 80% of the assignment. If you complete it, you can get a grade up to 80% (Good). The rest of the assignment is more challenging and will allow you to get a grade of 80-90% (Excellent) or 90-100% (Outstanding). Make sure you complete the first part well, before proceeding to the following additional part.

Read the Wikipedia pages on:

- "Single-precision floating-point format" (https://en.wikipedia.org/wiki/Single-precision_floating-point_format)

and,

- "Double-precision floating-point format" (https://en.wikipedia.org/wiki/Double-precision_floating-point_format)

Write the following functions:

```
void spff( char *sign, char *exponent, char *significand, float *src );
```

This function should retrieve individual groups of bits from the floating point number at the address given by **`src`**. It should retrieve the sign bit (1), the exponent bits (8) and the significand bits (23).

It is the caller's responsibility to ensure that **`sign`**, **`exponent`**, and **`significand`** have 1+1, 8+1 and 23+1 writable characters. It is the function's responsibility to write the correct characters ('0's and '1's) into those location and terminate each string with a null character ('\0').

```
void dpff( char *sign, char *exponent, char *significand, double *src )
```

This function should retrieve individual groups of bits from the floating point number at the address given by **`src`**. It should retrieve the sign bit (1), the exponent bits (11) and the significand bits (52).

It is the caller's responsibility to ensure that **`sign`**, **`exponent`**, and **`significand`** have 1+1, 1+11 and 52+1 writable characters. It is the function's responsibility to write the correct characters (‘0’s and ‘1’s) into those location and terminate each string with a null character (‘\0’).

You can write additional helper functions as necessary to make sure your code is modular, readable, and easy to modify.

Testing

You are responsible for testing your code to make sure that it works as required. The CourseLink web-site contains 6 test programs to get you started. However, we will use a different set of test programs to grade your code, so you need to make sure that your code performs according to the instructions above by writing more test code.

Your assignment will be tested on the standard SoCS Virtualbox VM (http://socs.uoguelph.ca/SoCSVM.zip) which will be run using the Oracle Virtualbox software (https://www.virtualbox.org/wiki/Downloads). If you are developing in a different environment, you will need to allow yourself enough time to test and debug your code on the target machine. We will NOT test your code on YOUR machine/environment.

Full instructions for using the SoCS Virtualbox VM can be found at: https://wiki.socs.uoguelph.ca/students/socsvm.

Makefile

You will create a makefile that supports the following targets:

`all:` this target should generate 6 test programs called **`test_getbytes`**, **`test_getbits`**, **`test_bits2ull`**, **`test_bits2ll`**, **`test_spff`**, and **`test_dpff`**.

All programs and .o files must be compiled with the **`-c99 -Wall -pedantic`** options and compile without any errors or warning.

`clean:` this target should delete all **`.o`** and executable files.

`test_getbytes:` this target should create the executable, **`test_getbytes`**, by linking the appropriate **`.o`** files, but not compiling any **`.c`** code.

`test_getbits:` this target should create the executable, **`test_getbits`**, by linking the appropriate **`.o`** files, but not compiling any **`.c`** code.

`test_bits2ull:` this target should create the executable, **`test_bits2ull`**, by linking the appropriate **`.o`** files, but not compiling any **`.c`** code.

`test_bits2ll:` this target should create the executable, **`test_bits2ll`**, by linking the appropriate **`.o`** files, but not compiling any **`.c`** code.

`test_spff:` this target should create the executable, **`test_spff`**, by linking the appropriate **`.o`** files, but not compiling any **`.c`** code.

`test_dpff:` this target should create the executable, **`test_dpff`**, by linking the appropriate **`.o`** files, but not compiling any **`.c`** code.

`test_getbytes.o:` this target should create the object file, **`test_getbytes.o`**, by compiling the appropriate **`.c`** file.

`test_getbits.o:` this target should create the object file, **`test_getbits.o`**, by compiling the appropriate **`.c`** file.

`test_bits2ull.o:` this target should create the object file, **`test_bits2ull.o`**, by compiling the appropriate **`.c`** file.

`test_bits2ll.o:` this target should create the object file, **`test_bits2ll.o`**, by compiling the appropriate **`.c`** file.

`test_spff.o:` this target should create the object file, **`test_spff.o`**, by compiling the appropriate **`.c`** file.

`test_dpff.o:` this target should create the object file, **`test_dpff.o`**, by compiling the appropriate **`.c`** file.

`bandb.o:` this target should create the object file, **`bandb.o`**, by compiling the appropriate **`.c`** file.

All compilations and linking must be done with the **`-Wall -pedantic -std=c99`** flags and compile and link **without any warnings or errors**.

Git

You must submit your .c, .h and makefile using git to the School's git server. Only code submitted to the server will be graded. Do ***not*** e-mail your assignment to the instructor. We will only grade one submission; we will only grade the last submission that you make to the server and apply any late penalty based on the last submission. So once your code is complete and you have tested it and you are ready to have it graded make sure to commit and push all of your changes to the server, and then do not make any more changes to the A1 files on the server.

Academic Integrity

Throughout the entire time that you are working on this assignment. You must not look at another student's code, now allow your code to be accessible to any other student. You can share additional test cases (beyond those supplied by the instructor) or discuss what the correct outputs of the test programs should be, but do not share ANY code with your classmates.

Also, do your own work, do not hire someone to do the work for you.

Grading Rubric

get_bytes	3
get_bits	3
bits2ull	3
bits2ll	3
style	2
makefile	2
spff	2
dpff	2
Total	20

Ask Questions

The instructions above are intended to be as complete and clear as possible. However, it is YOUR responsibility to resolve any ambiguities or confusion about the instructions by asking questions in class, via the discussion forums, or by e-mailing the course e-mail.