Program 1 - Fun With Bits (100 points)

COMPSCI 253: Intro to Systems Programming

1 Objectives

- Familiarize yourself with C
- Practice using Eclipse with C
- Understand how integer data types are stored in memory
- Use bitwise and bitshift operators

2 Introduction

You can think of memory as one large array of addresses. Typically, each address in the array is represented as a single byte. Consequently, when we store a 32-bit integer in memory, it will be split into 4 bytes and span 4 memory addresses. There are two common methods of storing bytes in memory - **Big Endian** and **Little Endian**.

The *big endian* representation will store the most significant byte (MSB) in the smallest address. For example, suppose we have an `unsigned int x = 168496141`, which is represented as `0x0A0B0C0D` in hexadecimal. The memory addresses using *big endian* would look like

Address	0000	0001	0002	0003
Value	0A	0B	0C	0D

In contrast, *little endian* will store the least significant byte (LSB) in the smallest address. The `little endian` memory addresses would look like

Address	0000	0001	0002	0003
Value	0D	0C	0B	0A

Check out the following sites for a more complete definition of endianness.

- `http://en.wikipedia.org/wiki/Endianness`

3 Starter files

`http://cs.boisestate.edu/~marissa/handouts/cs253/p1/p1.zip`

- Download the zip file that contains all the files for this assignment. It contains the following:
  - Makefile
  - test.sh
  - testdata (directory containing test input and expected results files)

4 Tasks

Write a C program that swaps the bytes of a given integer to convert from big-endian to little-endian.

- Copy the files you downloaded from p1.zip into your working directory and create a new source file called `endian.c`. Implement a `main()` function and make sure that you can build your program using the provided Makefile. Your `main()` function can just print `‘‘Hello World’’` for now.

  ```
  $ make
  ```

  After running this command, you should have an executable called `endian`.

  If you decide to modify the Makefile for any reason, make sure that it still produces the `endian` executable. This is what we will use to grade your program.

- **Print bits.** To make the next tasks easier to debug, start by implementing a function called `printBits()` that calculates the value of each bit in a given *unsigned* integer and prints it as a string of `1`'s and `0`'s. Don't forget to include the leading `0`'s. The function should print all 32 bits.

  ```
  void printBits(unsigned int x);

  input: 123456789
  output: 00000111010110111100110100010101
  ```

  Hint: You may use the `isBitSet` function we discussed in class.

- **Command-Line Input.** Your program will need to read an integer value from the command line. You will need to print a usage message if no integer is provided.

```
[marissa@onyx p1]$ ./endian
[marissa@onyx p1]$ Usage: ./endian <uint>
[marissa@onyx p1]$ ./endian 123456789
Decimal         Binary
------------------------------------------------
x  123456789    00000111010110111100110100010101
```

- **Endian Conversion.** Implement a function called `swapBytes` that will execute the endian conversion algorithm on the unsigned integer you read from the command-line. The function will take the unsigned integer as an argument and return the value of the converted integer. Here is the function prototype.

```
unsigned int swapBytes(unsigned int x);
```

  Hint: Check out the `getBits` function in the `K&R C Book`.

  The output should have the following format. Where `x` is the original integer and `x'` is the result of `swapBytes`. You will find it useful to include the hexadecimal format to quickly validate your results.

```
[marissa@onyx p1]$ ./endian 123456789
Decimal         Binary                                  Hexadecimal
-------------------------------------------------------------------
x  123456789    00000111010110111100110100010101    0x075bcd15
x' 365779719    00010101110011010101101100000111    0x15cd5b07
```

- **Extra Credit [10 points]: Binary to Unsigned Integer Conversion**. Write a function, `btoi()`, to convert a given string of `1`'s and `0`'s to an `unsigned int`. You will also need to modify your main function to read in another command line argument that will specify whether the user is passing in an integer or string representing a binary number. If the argument is `-i` you should use the following argument as an integer. If the argument is `-b` you will need to use your `btoi` function to get the integer value. (Let's assume the user will not pass in more than 32 bits). Function prototype will be

```
unsigned int btoi(char []);
```

  And some sample output.

```
[marissa@onyx p1]$ ./endian 123456789
Usage: ./endian -[i|b] <x>

[marissa@onyx p1]$ ./endian -i 123456789
Decimal         Binary                                  Hexadecimal
-------------------------------------------------------------------
x  123456789    00000111010110111100110100010101    0x075bcd15
x' 365779719    00010101110011010101101100000111    0x15cd5b07
```

```
[marissa@onyx p1]$ ./endian -b 0101
Decimal         Binary                                  Hexadecimal
-----------------------------------------------------------------
x  5            00000000000000000000000000000101    0x00000005
x' 83886080     00000101000000000000000000000000    0x05000000
```

5 Testing

Develop and hand test your program initially. Then use the provided shell script to process several integer values (and binary values if you did the extra credit) and compare your results with the provided solutions.

You can execute the test script directly, by executing

```
$ ./test.sh
```

or you can execute it using the provided Makefile.

```
$ make run
```

If you implemented the extra credit portion of the assignment, you will need to modify the `test.sh` script by changing the `EXTRACREDIT` flag to `1`.

6 Documentation

- Use comments in your program to document the program.
- You should include a file called README with your submission. I should be able to type: cat README to see the contents of your README file.
- Use the README template to structure your README file
- `http://cs.boisestate.edu/~cs121/ref/README.overview`

7 Submitting -Required files

- Before you submit, clean out the directory using: make clean
- All the files must be appropriately commented. Please submit only the following files:
  - endian.c
  - Makefile
  - test.sh
  - README
- Put all the required files for the assignment in a single directory in the lab, change to that directory and submit using the following command appropriate for your section.

section	**submit command**
1	submit spanter cs253 p1
2	submit marissa cs253-2 p1
3	submit amit cs253 p1
4	submit marissa cs253-4 p1