Project 1 - Fun with Bits

Objectives

- Familiarize yourself with C and low level programming
- Practice using VSCode and make
- Practice using control flow semantics in C (if, while, for and maybe others)
- Practice thinking about data as computer memory

Introduction

Memory can be thought of 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

You can find a more complete description of endianness in the wikipedia page:

https://en.wikipedia.org/wiki/Endianness

Starter files

These starter files can be found in our class repository, under the directory: projects/p1

- testdata/
  - test.in
  - test.out
- Makefile
- test.sh
- p1-rubric.txt
- README.md

Tasks

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

1. Copy the files from "Starter Files" into the p1 directory of your **portfolio repository**. Add the files to git, commit and push them. 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
   gcc -Wall -g  -c endian.c
   gcc -Wall -g  endian.o -o endian
   ```

   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 will be used to grade your program.

2. **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. The function does not print a trailing newline.

```
void printBits(unsigned int x);

Input:  123456789
Output: 00000111010110111100110100010101


Input:  0xA
Output: 00000000000000000000000000001010
```

Hint: Use the `isBitSet` function we discussed in class.

3. **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.

```
% ./endian
Usage: ./endian <int value>
```

Reference our class notes for the recommended prototype of main, which will expose the `argc` and `argv` variables. How do you convert the `argv[1]` to an integer?

```
% man 3 atoi
```

4. **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 following function from the K&R C Book:

```
/* getbits: get n bits from position p */
unsigned int getbits(unsigned int x, int p, int n) {
   return (x >> (p+1-n)) & ~(~0U << n);
}
```

We assume that bit position 0 is at the right end and that n and p are sensible positive values. For example, `getbits(x,4,3)` returns the three bits in positions 4, 3 and 2, right-adjusted.

5. **Output**. 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.

Example 1:

```
Decimal         Binary                                  Hexadecimal
-------------------------------------------------------------------
x   0           00000000000000000000000000000000    0x00000000
x' 0            00000000000000000000000000000000    0x00000000
```

Example 2:

```
Decimal         Binary                                  Hexadecimal
-------------------------------------------------------------------
x   2147483648  10000000000000000000000000000000    0x80000000
x' 128          00000000000000000000000010000000    0x00000080
```

You can use the following print statements to correctly format your results:

```
printf("Decimal         Binary                                  Hexadecimal\n");
printf("-------------------------------------------------------------------\n");
printf("x   %-9u    ", x);
printBits(x);
printf("   0x%08x\n", x);
printf("x' %-9u    ", swapBytes(x));
printBits(swapBytes(x));
printf("   0x%08x\n", swapBytes(x));
```

Testing

Develop and hand test your program initially. Then, use the provided shell script to process several integer values 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 get an error about permissions, you must set the "execute" permissions for the script in the shell.

```
% ./test.sh
bash: permission denied: ./test.sh
% chmod +x test.sh
% ./test.sh
Executing test using input file testdata/test.in.
```

Documentation

- Use comments in your program to document it.
- Complete the provided README file.
- Ensure your project adheres to the CS253-style-guide

Submission

Submission for Project 1 requires the following:

1. The files are correctly set up inside your portfolio repository.
2. A completed program named endian.c that compiles to an executable called endian.
    a. Programs with compilation errors **will not be graded**.

    b. Please see the rubric for information on how compilation warnings will affect your score. If you cannot get rid of a warning, please ask for help!
3. All project files are committed to the portfolio and pushed to the remote.