CSCI 2500 — Computer Organization
Lab 02 (document version 1.0)
Everything is just ones and zeros

- This lab is due by the end of your lab session on Wednesday, September 15, 2021.
- This lab is to be completed **individually**. Do not share your code with anyone else.
- You **must** show your code and your solutions to a TA or mentor and answer their questions to receive credit for each checkpoint.
- Labs are available on Mondays before your lab session. Plan to start each lab early and ask questions during office hours, in the Discussion Forum on Submitty, and during your lab session.

1. **Checkpoint 1:** The concept of **endianness** plays an important role on your laptop (or any computer hardware architecture). Specifically, endianness dictates whether the most significant byte (i.e., "big-end") or least significant byte (i.e., "little-end") is stored as the first byte of the value being stored at a given address. Some architectures are big-endian (e.g., Motorola 68K, older IBM processors), while most other architectures are little-endian (e.g., x86 and its descendants). See below for more info:

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

   Start with copying the code below. Attempt to understand what is happening in the given code as you copy it into your editor. The key to understanding this code is that a `long int` data type in C is likely to be stored as eight bytes on your machine while characters are a single byte.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
  /* Insert your eight bytes of ASCII for your secret message */
  /* The 0x prefix below indicates a hexadecimal number */
  long int z = 0x2167724F706D6F43;

  char* c;

  int i;
  for (c = (char*)&z, i = 0; i < sizeof z; c++, ++i) {
    printf("%c", *c);
  }
  printf("\n");

  return EXIT_SUCCESS;
}
```

Compile and run this code, then replace the hexadecimal value of variable `z` with your own secret eight-letter word. To do so, look up each letter in an ASCII table and convert them to their corresponding hexadecimal values. You can find an ASCII table at the following URL:

https://en.wikipedia.org/wiki/ASCII#ASCII_control_code_chart

Answer the following questions:

- What is the purpose of using commas in line 13 of the code above?
- How the output would change (if at all) if we replace `c++` with `++c` in line 13? What about replacing `++i` with `i++`?
- What is the purpose of `sizeof z` in line 13? Why aren't there any parenthesis around `z`?
- Based on your output, what is the endianness of your system? You'll need to know this for the subsequent checkpoints. Verify your answer by running your OS command that shows endianness (e.g., `lscpu` on Linux).

For the last two checkpoints, fill in the provided template code given in `lab02.c`. This has empty functions for Checkpoints 2 and 3. The expected output format is given below.

2. **Checkpoint 2:** Download the `lab02-data.[your-endianness].dat` file and attempt to decipher what the file contains. From the shell, you can try viewing the file in a text editor or use the `cat` and `hexdump` to learn what you can about the file. Any guesses as to what data is stored in this file?

   Assume that the data file contains an array of `long` variables. Write a C program to read this data file entirely into memory, then display the data using a loop. You'll need to open the file, determine its size in bytes, allocate an array, and then read the binary data into the array. You can easily accomplish this using the following functions:

   `calloc()`, `fopen()`, `fseek()`, `ftell()`, `fread()`, and `fclose()`.

   Note: `fread()` reads in binary data directly to some memory address. `fseek()` and `ftell()` can get you the size of the file.

   Finally, display the data as follows (hint: "%2d" might be useful):

```
Data point # 0: <long-value>
Data point # 1: <long-value>
Data point # 2: <long-value>
...
Data point #22: <long-value>
Data point #23: <long-value>
Data point #24: <long-value>
```

3. **Checkpoint 3:** Modify your solution for Checkpoint 2 by instead assuming that the given data file contains an array of `unsigned int` variables.

   Display the data as follows:

```
Data point # 0: <unsigned-int-value>
Data point # 1: <unsigned-int-value>
Data point # 2: <unsigned-int-value>
...
Data point #47: <unsigned-int-value>
Data point #48: <unsigned-int-value>
Data point #49: <unsigned-int-value>
```

This data file contains values that are known in mathematics by a proper name. Can you find out what this list of numbers is called and what its significance is?

Note that we can interpret the raw data however we like, as it really is just a bunch of ones and zeros. We can treat it as an array of `long` values, an array of `unsigned int` values, a string of printable characters, etc. But, only one interpretation turns out to be useful. Such is life.