Lab 5

Integer Arithmetic

In this lab, we will investigate various ways to store integers in memory, interpret bit patterns in your program, and how computers carry out arithmetic on integers. Understanding the nature of computer arithmetic will help you write more reliable code.

Exercise 1. [Byte Ordering] We learned that some systems like Intel-compatible processors store least significant bytes first in memory (*little endian*), and some systems store most significant bytes first (*big endian*). In this exercise you are asked to write a C function `is_little_endian` which will return 1 when compiled and run on a little-endian machine, and will return 0 when compiled and run on a big-endian machine. This program should run on any machine, regardless of its word size. Write a simple driver (the `main` function) to test your function.

Exercise 2. [Unsigned Addition] Write a function with the following prototype:

```
int add_ok(unsigned x, unsigned y);
```

This function returns 1 if arguments `x` and `y` can be added without causing overflow; 0 otherwise. Write a driver to test your function. The header file `limits.h` defines constants for unsigned integers, including `UINT_MAX`, which defines the maximum value for a datum of type `unsigned int`.

Exercise 3. [Signed Addition] Write a function with the following prototype:

```
int tadd_ok(int x, int y);
```

This function returns 1 if arguments `x` and `y` can be added without causing overflow; 0 otherwise. Write a driver to test your function. The header file `limits.h` defines constants for unsigned integers, including `INT_MIN` and `INT_MAX`, which define the minimum and maximum values, respectively, for a datum of type `int`.

Exercise 4. [Signed Subtraction] Write a function with the following prototype:

```
int tsub_ok(int x, int y);
```

This function returns 1 if the computation arguments `x-y` does not overflow. It is tempting to write the function as the following:

```
int tsub_ok(int x, int y)
{
   return tadd_ok(x, -y);
}
```

What's wrong with this function? For what values of `x` and `y` will this function give incorrect results? Write a correct version of this function.

PART 2

Grab It With `grep`!

1. `grep` *pattern filenames...* searches the named files or the standard input and prints each line that contains an instance of the *pattern*.

2. Pattern matching using *regular expressions*

3. *Tagged* regular expressions `\(` and `\)` are used to match and save the matched strings in \1, \2, etc.

   (a) Using tagged expressions write a shell script to find all palindromes – words spelled the same backwards as fowards.

   (b) Write a C program to search for all palindromes. Compare their running time using `time` command. Which one is faster?