Lab 7: `read()` and `write()`

For this lab, create a new directory named `lab7` under your `cs449` directory and create your program there:

```
mkdir lab7
cd lab7
```

For this lab, you can get the starter file (`lab7_readWrite.c`) to your directory using the following command:

```
cp /afs/cs.pitt.edu/usr0/tkosiyat/public/cs0449/lab7_readWrite.c .
```

For those who works on Ubuntu on your own computer, the starter file is also available on the CourseWeb.

Short Introduction

In UNIX systems, everything is a file. For example, if you want to put a string (array of characters) onto the console screen (display a string), you write the string to a file. Similarly, if you want to read the keyboard input, you read from a file. In the past, when you want to read from or write to file, you interacted with **file pointers** (e.g., `FILE *fp`). File pointers are used by standard C library such as `fread()` and `fwrite()`. For this lab, we are going to perform read and write file using system calls, `read()` and `write()`. These system calls need **file descriptors** which can be imagine as a raw format of `file pointers`. A file descriptor in UNIX system is simply an integer number. In UNIX systems, when a program is executing, three default file descriptors are created, 0, 1, and 2. File descriptors 0 is for standard input, 1 is for standard output, and 2 is for standard error. These three values are defined in `unistd.h` which can be accessed by variable named, `STDIN_FILENO`, `STDOUT_FILENO`, and `STDERR_FILENO`, respectively.

System Call `write()`

The system call `write()` is used to write data to a file descriptor. To use this system call, you need to include `unistd.h` in your C file. The signature of the system call `write()` is as follows:

```
ssize_t write(int fd, const void *buf, size_t count);
```

The system call `write()` writes up to `count` bytes from the buffer pointed `buf` to the file referred to by the file descriptor `fd`. The following is the hello world program that uses `write()` instead of `printf()`:

```
#include <unistd.h>

int main(void)
```

```
{
        write(STDOUT_FILENO, "Hello World!!!\n", 15);

        return 0;
}
```

In standard C library, `printf()` uses the system call `write()` to put characters onto the console screen. `printf()` simplifies the way we display information on the console screen using formatting string and formatting characters. Without `printf()` we have to convert numbers to characters manually and display it using the system call `write()`.

System Call `read()`

The system call `read()` is used to read data from a file descriptor. The signature of the system call `read()` is as follows:

```
ssize_t read(int fd, void *buf, size_t count);
```

The system call `read()` attempts to read up to `count` bytes from file descriptor `fd` into the buffer starting at `buf`. The following is an example of a program that read a string from the keyboard using the system call `write()` and display the string using the system call `read()`:

```
#include <unistd.h>

int main(void)
{
        char buffer[100];
        int numCharRead;

        numCharRead = read(STDIN_FILENO, buffer, 100);

        write(STDOUT_FILENO, buffer, numCharRead);

        return 0;
}
```

Note that the system call `read()` will stop when user press `Enter`. Its returns the number of characters read including the newline (`\n`) character. At this point, you might be able to guess that `scanf()` uses this system call to read data from your keyboard. Similar to the `printf()` function, `scanf()` function simplifies the way we enter data into our program. It formats a series of characters into a data of your choice according to the formatting string and formatting characters. Without `scanf()`, you have to do this process manually.

What to Do?

For this lab, you are going to implement various functions to replace `printf()` and `scanf()` functions. Functions that you must implement and their signatures are as follows:

- `int readInteger()`: This function uses the system call `read()` to read a string representing an integer number. You job is to convert the input string into an integer number and return it. Assuming that user will only enter a valid integer number smaller than 10 digits.
- `void printString(char *str)`: This function uses the system call `write()` to print a null-terminated string (`str`) on the console screen. Note that since the system call `write()` needs to know the length of the string to be printed.
- `void printInteger(int n)`: This function uses the system call `write()` to print an integer `n` on the console screen. Since the system call `write()` can only print a string, you must convert the integer `n` into a string before making the system call. Assuming that the integer `n` will be smaller than 10 digits.
- `void printFloat(float f)`: This function is similar to the function `printInteger()`. You must convert the floating-point number `f` into a string. For this function, we will always print 6 significant digits. For example, 3.25 should be printed as 3.250000. Assuming that there will be less than 10 digits before on the left of decimal point.

These function can be found in the starter file (`lab7_readWrite.c`). They are marked `TO DO`. Note that the starter file only contains `#include <unistd.h>`. You are **not allowed** to include any other standard C library. In other words, in your final submission, you are **not allowed** to use any functions provided by C library other than `read()` and `write()`. However, while you are implementing/testing your program, you are allowed to use `printf()` or `scanf()`. But do not forget to remove them before submission. The following is an example of the output associated with the provided `main()` function:

```
Please enter an integer: 5
Please enter another integer: 9
5 + 9 = 14.
5 - 9 = -4.
5 * 9 = 45.
5 / 9 = 0.555555.
```

What to Hand In

First, let us go back up to our cs449 directory:

```
cd ..
```

Now, let us first make the archive. Type your username for the USERNAME part of the filename:

```
tar cvf USERNAME_lab7.tar lab7
```

And then we can compress it:

```
gzip USERNAME_lab7.tar
```

Which will produce a USERNAME_lab7.tar.gz file.

If you work on cs449.cs.pitt.edu (thoth) you can skip to the next section. **If you use your own machine, you need to transfer the file to cs449.cs.pitt.edu first**. This can simply be done by a command line. For example, assume that your username is abc123 and you are in the same directory as the file abc123_lab7.tar.gz. To transfer the file to cs449.cs.pitt.edu use the following command:

```
scp abc123_lab7.tar.gz abc123@cs449.cs.pitt.edu:.
```

The above command will copy the file to your home directory in cs449.cs.pitt.edu. If you want to copy it to your private directory, use the following command:

```
scp abc123_lab7.tar.gz abc123@cs449.cs.pitt.edu:./private/.
```

Copy File to Submission Directory

We will then submit that file to the submission directory:

```
cp USERNAME_lab7.tar.gz /afs/cs.pitt.edu/public/incoming/CS0449/tkosiyat/sec1
```

Once a file is copied into that directory, you cannot change it, rename it, or delete it. If you make a mistake, resubmit a new file with slightly different name, being sure to include your username. For example USERNAME_lab7_2.tar.gz. **Check the due date of this lab in our CourseWeb under Labs/Recitations**.