Computer Science 62
Lab 10

Wednesday, November 11, 2015

Our goal today is to get you started with C, particularly its input and output functions. The stubs of four methods are provided for you in the starter code, as well as commented out function calls from the main method.

Before lab, take a look at the C references posted on the course "Documentation and Handouts" page, in particular explore `http://en.cppreference.com/w/c`, which documents all of the features of the C language. If you would prefer a tutorial style website, take a look at `http://www.geeksforgeeks.org/c/`. Spend some time on these websites familiarizing yourself with the different C language features. You may not understand everything but that's okay.

Getting started

Begin by creating a working directory for your C programs, inside your `cs062` directory, I would suggest the name `dangerzone`, but anything is fine. Do not use your eclipse `workspace` directory for C programs. Now grab the starter code from the usual place and copy it into the `dangerzone`.

Open the makefile in your favorite text editor (such as AquaMacs) and read the comments at the top of makefile and fill out your information. Take a look at the rest of the makefile, most of it won't make any sense to you—yet. This makefile will automatically run the commands to compile our C programs (using the clang compiler), like a mini eclipse for C. While programming in C you will leave your terminal in the directory with the makefile and type in commands to compile and run your program.

Function 1: Printing to the terminal

The first part of our exercise is to familiarize you with terminal output using the C I/O model. To print a string text to the terminal we will be using `stdout` "output stream", which is analogous to Java's `System.out`. To C `stdout` looks like an ordinary file, but when we write to this "file" what we write prints to the screen.

To write a string to `stdout` we can use the `fputs` command like so:

```
fputs("Hello world!\n", stdout);
```

Now technically speaking, `fputs` returns a code (an `int`) letting us know if there was an error while writing to `stdout`, which we should be checking. Remember there are no exceptions in C! What does the `\n` do?

Fill in the body of the function `part_one()` so that when called it prints out at least two lines. Then uncomment the call to the function in `main` and try running the program.

To compile the program we type the following commands in the terminal window (make sure you are still in the directory with the makefile):

```
make
```

Thats it! If you did not get any errors or warnings, then you are ready to run the program with

```
make run
```

If the program ran without error, then it is time to move on to the next function.

Function 2: Reading from the terminal

Before you start comment out the call to `part_one` and uncomment the call to `part_two`.

For this function we will use the "input stream" `stdin`, which is analogous to `System.in`. Again, to C `stdin` looks like an ordinary file, but when we read the "file" it sees what you have typed in to the terminal. We can use the `fgets` command to read a line from `stdin` into a `buffer` with:

```
size_t buffer_size = 1000;
char buffer[buffer_size];
fgets(buffer, buffer_size, stdin)
```

The C language does not have a "true" string object like Java so we need to use character arrays for our strings. Further more since the size of an array is fixed we need to set some reasonable upper bound on the length of a line, such as 1000 characters. What happens if the line is longer than 1000? Can we check for this?

Fill in the body of `part_two` with code that echoes lines typed on the keyboard.

Your program will contain a `while` loop that reads in what is typed on the keyboard and then prints it back out for the user to see. You can let the program know you are done by sending an end-of-file signal by pressing ctrl+D. You can test for end-of-input in your program by testing the value of `feof(stdin)`, which is true when you are at the end of a file.

Function 3: Reading from and writing to files

Before you start comment out the call to `part_two` *and uncomment the call to* `part_three`*.*

The next step is to learn how to work with files on the disk. In this part of the lab, we will copy the contents of one file to another file. This will take only a minor modification to your solution in Part 2. Before you begin create a text file named `input.txt` in the `data` directory and fill it with some text.

In C you open a file for reading with

```
FILE* inputf = fopen("data/input.txt", "r");
```

and you open a file for reading with

```
FILE* outputf = fopen("data/output.txt", "w");
```

Once the files are opened you can work with `inputf` and `outputf` just like you did with `stdin` and `stdout`. In the body of `part_three`, modify your code from Part 2 to copy the contents of `data/input.txt` to `data/output.txt`.

When you are done working with the files you need to close the files:

```
fclose(inputf);
fclose(outputf);
```

Be sure that the output file is *identical* to the input file. Use the `diff` command to detect any differences. You can also use the `less` command to open up and look at a file. To exit from `less`, simply type `q`.

Function 4: An adding machine

So far we have only been reading and writing strings. In this last part we will use C to read and write `int`s from `stdin` and to `stdout`. To print `int`s we use `fprintf`, which is analogous to Java's `System.out.format`. The following will print the number 10 followed by a new line:

```
fprintf(stdout, "%d\n", 10);
```

Make sure you look up how to work with `fprintf` before lab. Conversely, we can read an integer from `stdin` with:

```
int n;
fscanf(stdin, "%d", \&n);
```

The third argument above, `&n`, is the address of `n`. We need to pass the address of the variable into the function so that we can overwrite its value. Here we are ignoring the return code of `fscanf` which tells us if the user did in fact enter an integer.

With this information in hand fill out the body of `function_4` so that when run it will allow the user to keep entering integers until the end-of-file is reached at which point it prints the sum of the integers the user entered. What happens to your program if you enter a letter? How can you fix this?