Goals

Understand how to build simple C language functions

Background

Functions

- Functions provide a mechanism for partitioning a large, complex program into a collection of small, easily understood components.
- Each function should be designed to solve ONE aspect of the problem at hand.

Function definitions

A C function is comprised of 4 components. The first three are collectively referred to as the function header.

- the type of value returned by the function
- the name of the function
- parenthesized declaration of function parameters (values passed to the function by its caller)
- a basic block containing local variable declarations and executable code

```c
int sum(int a, int b) {
    int total;
    total = a + b;
    return(total);
}
```

Without the comments in the middle of the function header, the above function would look like this:

```c
int sum(int a, int b) {
    int total;
    total = a + b;
    return(total);
}
```
A C function is called or invoked by just using its name anywhere a variable or constant might be used in an expression:

```c
int main()
{
    int num = 0;
    num = sum(4, 5);
    num = num + sum(num, 6);
}
```

When an expression is evaluated, the value returned by the function replaces its invocation in the expression. So here, the first call to `sum()` will return a 9 and 9 will be assigned to `num`. Then the second call will return a 15 and so the final value of `num` will be 24.

NOTES:

- Local variable name spaces of different functions are completely disjoint. I can declare a local variable called `total` in `main()` and it will be completely independent of the `total` in `sum`.

- The correspondence between the formal parameters (a, and b) in `sum()` with the actual arguments (4, and 5) in `main()` is positional. When this program runs, if we were to include `fprintf(stdout, "%d %d\n", a,b);` in `sum()`, we would see `a` holds the value 4 and `b` holds 5 during the first call and `a` holds the value 9 and `b` holds the value 6 during the second call.

- Therefore, the number and types of the actual arguments used when invoking a function should exactly match the formal parameters of the function definition.

**Function prototypes:**

A function prototype is the function header followed by a semi-colon.

```c
int sum(int a, int b);
```

If the actual function definition comes after the function from which it is called or appears in a difference .c source file, then a prototype is required to ensure that the actual arguments passed to the function match the formal parameters in the function header/prototype. If the function definition precedes its invocation, then the proper information can be picked up from the function itself. Nevertheless, including a prototype is never an error.
Assignment:

A program called lab6.c is located here: www.cs.clemson.edu/course/cpsc101/labs/lab6S15/lab6.c. Complete the bodies of the three missing functions. You may assume that the input character will always be a legitimate lower case or upper case alphabetic character. You may avoid having to do bit operations by adding or subtracting the expression ('a' - 'A') as required.

Turn In Work

Show your TA that you completed the assignment. Then turn in your lab6.c program using the handin page: http://handin.cs.clemson.edu