Week 7 - Introductory Programming Lab 3

Exercise 4: Functions

Background

You are familiar with functions in mathematics -- you've handled functions like g(x) = x^2 + 2x - 3. In this case, the function *returns* a value. The returned value is computed based on the input. You have also seen functions with multiple inputs: h(x,y) = 5x - 6x + x*y.

We use functions like this in programming. In C, we need to specify the type of variable being returned -- `int` or `float`. We also need to specify the type of input variables.

We also sometimes use functions which do not return any values -- these are useful if we want to repeat a few commands, or simply for program organization. These functions are called **`void`** functions.

Functions must be declared before ("higher in the .c file") they are used.

Note that each function has its own *scope* of variables. A function cannot refer to a variable which was declared in the `int main()` function. In fact, you can re-use the same variable name in different functions!

This is the time we have seen variable scope, but we can illustrate the problem even without functions. Consider the following example -- it will not compile. The `int x` is defined inside an extra code block { . . . }. The scope of variable **`x`** only lasts for that code block; referring to this variable outside of that scope will produce a compiler error.

```
int main() {
     {
          int x = 0;
     }
     x = 1;
}
```

This is one reason why indentation is useful in source code.

Technical details

Simple function:

```
#include <stdio.h>

float g(float x) {
     return x*x + 2*x - 3;
}

int main() {
     float a = 3.2;
     float b = g(a);
```

```
    printf("g(%f) = %f\n", a, b);
    printf("g(5) = %f\n", g(5));


    // wait for a keypress
    getchar();
}
```

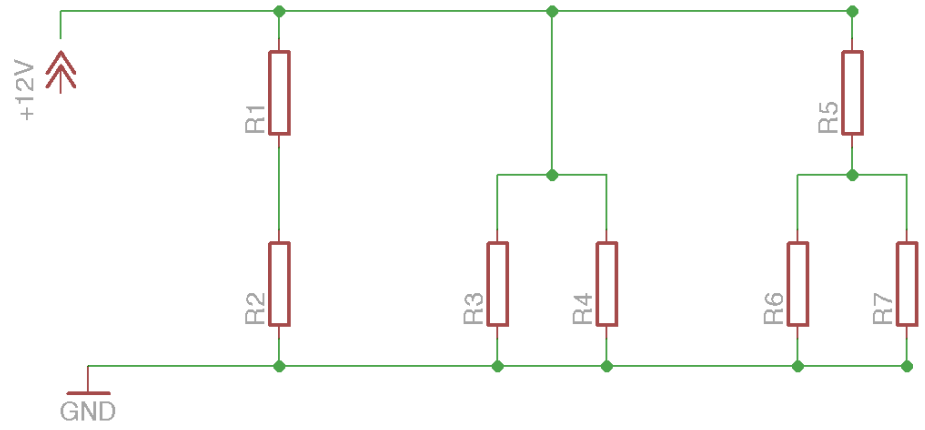
Note the different variables names in the above example!

Various functions:

```
#include <stdio.h>

int catYears(int human_years) {
    float y = human_years / 7.0;
    // this would be a good place for a comment
    // why did I write the next line?
    int rounded = y+0.5;
    return rounded;
}

float sumOfSquares(float a, float b) {
    return a*a + b*b;
}

void p(int years, float c, char animal[100]) {
    printf("If I were a %s, I'd be ", animal);
    printf("%i years old.\n\n", years);
    printf("The sum of squares ");
    printf("was %f units.\n", c);
}

int main() {
    int my_age = 18;
    float x = 3.3;
    char animal[100] = "cat";

    float years_a = catYears(my_age);
    float sum = sumOfSquares(x,7.4);

    p(years_a, sum, animal);

    // wait for a keypress
    getchar();
}
```

Your task...

As electrical engineering students, you are familiar with Ohm's Law (*V = IR*) and calculating the value of resistors in series (*R = R1+R2*) and parallel (*1/R = 1/R1 + 1/R2*). Consider the following circuit:



Write a program to calculate the overall current in this circuit.

- You *must* use 3 functions: `ohm_law(...)`, `series(...)`, and `parallel(...)`. A 4th function `parallel_three(...)` is optional. Do not write functions like `sum_r1_r2(...)`.
- Calculate the current for:

  ```
  int r1=100, r2=200, r3=300, r4=400, r5=500, r6=600, r7=700;
  ```

  (you may copy&paste this into your .c file)

  The answer should be 124.6 mA.
- Calculate the current for:

  ```
  int r1=123, r2=234, r3=345, r4=456, r5=567, r6=678, r7=789;
  ```

  The answer should be 107.6 mA.
- You may find it helpful to use intermediate variables, and to print debugging information.

(optional: you can call functions as inputs to other functions, i.e.

```
z = f( g(x,y), h(x,y) );
```

Using this, calculate the current from voltage and Rx values in a single line.)

Show your work to a demonstrator/GTA.

Exercise 5: Random numbers

Background

Unless you are reading a research paper in engineering or computer science, when people write "random number" with regards to computers, they really mean "pseudorandom number". The basic problem is that if you start from a single value (known as the *random seed*), and do a series of mathematical operations on it, you will not get an actual random number -- every time you start from the same random seed, you will get the same number!

To quote one of the giants in computer science,

> "Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin."
> - John von Neumann

However, pseudorandom numbers are good enough for this course.

Technical details

Generating a random number:

`getRand()` requires extra `#include` files!

```
#include <stdio.h>
#include <stdlib.h>  // extra includes!
#include <time.h>

/* Get a random number from 0 to 0.9999999
    (you don't need to understand this function)
    ***** DON'T MODIFY THIS FUNCTION *****
 */
float getRand() {
     return rand() / (RAND_MAX+1.0);
}

int main() {
     srand( time(NULL) ); // init random
     getRand(); // kick-start the random numbers

     float number = getRand();
     printf("Random number: %f\n", number);
     getchar();
}
```

Your task...

Write a "dice rolling" game. You are probably familiar with 6-sided dice, but some games use dice with 4, 6, 8, 10, 20, and 100 sides!

- Make the computer pick a random number for a 6-sided die and a 20-sided die.
- Use the `getRand()` function, but *do not modify it*.
- Write an `int rollDie(...)` function to get a random value.
  - have one integer argument for the value of the die (`int number_of_sides`),
  - call the `getRand()` function.
  - take the value it returns (a float between 0 and 0.999...) and do some math to transform that into an `int` between 1 and `number_of_sides` inclusive).
  - return the number.
- Your `int main()` must contain only:

```
int main() {
     srand( time(NULL) ); // init random
     getRand(); // kick-start the random numbers

     int value = 0;
     value = rollDie(6);
     printf("6-sided die: %i\n", value);

     value = rollDie(20);
     printf("20-sided die: %i\n", value);

     getchar();
}
```

Show your work to a demonstrator/GTA.