**C programming assignment #1**
**Working with functions and more**
**Due Tuesday, October 25, 10pm**

Go to codepad.org, choose C, set your "pastes" to be private. Enter code in the edit window and submit to receive results.

Note: the reference, "the C handout", is to the "A Quick Introduction to the C Programming Language" handout.

1. Enter the following program from section 5 of the C handout. (Like Java, indenting is for the human reader.)

```
/* C program #1 */
/* Put your name here! */
#include <stdio.h>
main ( )
{
     int s;
     s = square(2);
     printf("The square of 2 is %d  \n", s);
     return (0);
}
int square(int x)
{
     return(x * x);
}
```

**Submit**. Run with different values for parameter and change it in the output line as well, convincing yourself that the program works properly. Type the link (see near top of left on codepad) to your "Paste" below, so that we can see it.

2. Create a new paste  based on the one above (there is a link to do this near the bottom of the paste for program 1) and change the program number to 2.

   a. Modify the code so that the square method returns a "double".  Also, change the declaration of variable s to be of type double. **Submit.** What result do you get?

Copy the link to the paste here:

**b.** In the error message above, notice the reference to the "implicit declaration of 'square'". As mentioned in section 5 of the C handout, when a call to a previously undeclared function is made, C assumes that that function will return an `int`. This allows compilation to continue "in a single pass." Create a new paste based on this one and add the following declaration of square to your C code, right after the #include line: `double square( );` and **Submit.** What is the result now?

Fix the "printf" line to display a real number by changing "%d" to "%f". **Submit**. Better? (Why or why not?)

Copy the link to the paste here:

Note that the prototype, added above in #3b, tells the compiler that function square will return a double. This tells the compiler all it must know if order to be able to translate the call to square in the `main`.

c. C is weakly typed, but maybe we can get it to do more for us, if we put more information in the program. Create a new paste from the previous one and try the following prototype instead of the one already there and indicate below each what the results are after **Submit**:

```
double square(y);
```

Before moving on, change the function header (not the prototype, which should remain unchanged) and the call to include two arguments (`square (2, 3)`) and parameters (`double square(int x, int y)` ), respectively, and **Submit**. Any problems detected related to numbers of parameters? ___________________

What does this tell you about C's type checking as far as verifying parameter counts goes?

Let's see if it checks calls to functions? Change the call to include only a single argument (as in `square(2)`), while leaving the header with two parameters. What are the results?

What do you <u>think</u> might be happening to generate the result above?

Let's check out the value stored in y in the body of the function square. Add the line `printf("y = %d \n", y);` right before the "return" statement. **Submit**.

What do you <u>think</u> might be happening to generate the result above?

Copy the link to the paste here:

Generate a new paste from the one linked above and change the prototype to the following:

```
double square(int y);
```

,so that now the prototype, the call and the function definition agree
in numbers of parameters and the prototype and function agree in type of parameter/arugment. NOW, change the parameter in the call to a double, such as 3.5, and change it in the output line as well – so your output continues to make sense. Do you expect to get an error here? ______________________ **Submit**.

Result?

What must be happening?

Copy the link to the paste here:

The Java compiler would detect an error in sending a double literal value into an int parameter. Obviously this is neither a compile-time or run-time error in C, it is important to know this! (Remember what I said about C programmers needing to be more disciplined? If you sent that real number in by mistake, C doesn't know it, and, unlike Java, it will "narrow" the value for you; that is, it implicitly converts your real number to an int – surprise! (An implicit conversion is called a coercion.)

Would you prefer a compile-time error or a coercion? **Why?**

3. Create a new paste – we're starting on a new problem. Enter the following program.

```
# Your name goes here
#include <stdio.h>
#define MAX 5
int main()
{
    double data[MAX] = { 34.0, 27.0, 45.0, 82.5, 22.1 };
    int i;
    for (i = 0; i < MAX ; i++)
        printf("data[%d]=%5.2f\n", i, data[i]);
    return 0;
}
```

a. **Submit**. Results?
b. Try putting the declaration for i as part of the initialization of the loop control variable like we often do in Java (and C++). What happens?
c. Put the code back like it was before (in part a). Trying changing the formatting values in the printf statement and <u>tell me what happens with each change</u>:
    a. Change `"%5.2f"` to `%8.3f"`

    b. Change `%8.3f"` to `%f"`

    c. Change `%f"` to `%15f"`

    d. Change `%f"` to `%2.3f"`

    e. Change `%d"` to `%5d"`

    f. Change `%d"` to `%f"`

    g. Change the %f back to %d in part f and change `%2.3f"` to `"%d"`

    h. Change the formats back to the original and write the link to your paste below here.

4. Create a new paste based on the one above and change your array declaration to the following: `char data[MAX] = { 'a', 't', '8', '*', 'B' };` and your `printf` statement to: `printf("data[%d]=%c\n", i, data[i]);`
**Submit**. Results?

Now, change the "`%c`" format in your print statement to "`%d`". Can a char be interpreted as an integer? Where do these numbers mean?

Let's see if this would this work the other way. That is, I wonder if we could store an array of integer and them print them out as characters… Give it a try… modify the code so that data contains the integers { 32, 69, 115, 550, 551 } Can you explain what is happening here?

Now let's check how well C does on checking array bounds, at compile time or run-time. Using the same code as in the immediate previous problem, change the loop termination condition to <= MAX, so it should run off the end of the array! **Submit**.

Any problem detected during execution?

Let's see other access errors might be detected…
Can an array be indexed by something other than an integer?
Try changing `data[i]` to `data[100.0]`. **Submit**. Result?

Try changing `data[100.0]` to `data['a']`. **Submit**. Result?

Set the reference back to `data[i]`. **Submit**. Write the link to your final paste below: