Writing a Function

When we are writing a function, we must be in a different mind set than when we are calling a function. In particular, it is not necessary to think about how someone else will USE the function you are writing during the process of the writing of the function.

Rather, once someone has given you a specification of what a function should do, with a pre and post-condition, there is no need to think about how the function will be used. Rather, you should think about how you can satisfy the requirements of those post-conditions in the situations where the pre-conditions are true. (Now, of course, often times you will be the one both thinking of the specification of the function AND writing it, but try to separate these roles as best as you can.)

Now, consider the function header we looked at last lecture:

// Precondition: The function takes in a single character as a
// parameter. The character MUST BE one of
// the following: 'A', 'B','C','D', or 'F'.
// Postcondition:The function will return a numerical value
// in between 0 and 4 corresponding to the
// parameter passed to it.
int Comp_Grade(char grade**)**

Now, let's consider how we would go about writing this function.

The function header is always the first line of the function.

Now, probably the most difficult part of writing a function is deciding what the formal parameters should be. For this example, I have already told you, so I can't go through that process for this example. But, I will do so for other examples.

The key thing to realize is that whenever someone calls your function, they have passed an actual parameter value that corresponds to each your formal parameters. Hence, you can assume that before your function has even begun, each of the formal parameters you have will already be declared and assigned to values that the function caller wanted them to be assigned to.

THIS IS WHAT YOU SHOULDN'T DO:

```
int Comp_Grade(char grade) {
        printf("Enter in the grade.\n");
        scanf("%c", &grade);
    ...
}
```

This writes OVER the value passed into the function for grade.

In this situation, we find that our only formal parameter grade is a character variable that is either 'A', 'B', 'C','D', or 'F'. In particular we would like to use this information to compute the numerical value of the grade, and then return this value to the function caller. Here is the function:

```
// Precondition: The function takes in a single character as a
//                parameter. The character MUST BE one of
//                the following: 'A', 'B','C','D', or 'F'.
// Postcondition:The function will return a numerical value
//                in between 0 and 4 corresponding to the
//                parameter passed to it. If the grade is invalid
//                 a -1 will be returned.
int Comp_Grade(char grade) {
        if (grade == 'A')
                return 4;
        else if (grade == 'B')
                return 3;
        else if (grade == 'C')
                return 2;
        else if (grade == 'D')
                return 1;
        else if (grade == 'F')
                return 0;
         else
                return -1;
}
```

The code in a function should be like code in main, in fact main is a function. You are allowed to declare extra variables inside of a function, but these need to be declared at the beginning of the function. These variables are local to the function meaning that they exist ONLY in the function and not outside of it. You must have a return line in your function of the following format:

return <Expression of the return type>;

When this line is encountered, the function will return the specified value, and then terminate. It is possible to have multiple returns stmts, but only 1 will ever get executed on a particular function call.

```
// This program conducts a shopping spree and prints the
// total spent on all of the items.
#include <stdio.h>

double Total_With_Tax(double value, double tax_rate);

int main() {

    char ans;
    double total_price = 0.0, price, tax_rate;

    printf("Is there another item to buy?\n");
    scanf("%c", &ans);
    while (ans == 'y') {

        // Read in value and tax rate of the next item.
        printf("Enter the item price and tax rate for the item.\n");
        scanf("%lf%lf", &price,  &tax_rate);

        //Adjust total price according to last item entered.
        total_price = total_price +Total_With_Tax(price, tax_rate);
        printf("Is there another item to buy?\n");
        scanf("%c", &ans);
    }
    printf("You spent %lf on your shopping spree.\n",total_price);
    return 0;
}

// Precondition: Value is a positive number and tax_rate is in
//                   between 0 and 1.
// Postcondition: This function will return the total cost of an
//                   item given the value and tax rate parameters.
double Total_With_Tax(double value, double tax_rate) {
    return value*(1+tax_rate);
}
```

Example

Write a function that takes in a number n (which is guaranteed to be an integer), and computes the sum of the first n integers, and returns this value.

```
// Precondition: n is a positive integer.
// Postcondition: The function evaluates to the sum 1+2+...+n.
int Triangle_Number(int n)

        int index=1, sum=0;

        // Return 0 if input is invalid.
         if (n < 1)
                return 0;

        // Compute sum.
        for (index = 1; index <= n; index++)
                sum += index;
        return sum;
}
```

Now, use this function in an program to print out the first 20 triangle numbers. (Note the nth triangle number is simply the sum of the first n numbers

```
#include <stdio.h>

int Triangle_Number(int n);

int main() {

        int index;

        // Print out first 20 triangle numbers.
         for (index = 1; index <= 20; index++)
                printf("%d triangle number is %d\n",index,
                        Triangle_Number(index));
}

// Precondition: n is a positive integer.
// Postcondition: The function evaluates to the sum 1+2+...+n.
int Triangle_Number(int n)

        int index=1, sum=0;

        // Return 0 if input is invalid.
         if (n < 1)
                return 0;

        // Compute sum.
        for (index = 1; index <= n; index++)
                sum += index;
        return sum;
}
```