PLAYBOOK NOTES: RECURSION – PART 1

What is recursion?
Recursion is a process, that computer science uses to simplify a problem

How do you code recursion?
To code recursion, you are simply having a function call itself.

How does recursion work?
Recursion works because every time a function calls itself the problem gets a little smaller.

To use recursion, you must have a base case and recursive case.

What is a base case?
A base case is when the function does NOT call itself

What is the recursive case?
The recursive case is when the function does call itself.

What happens to all the function calls?
From your main you will call your recursive function, for this example we will call the function foo(number) where number is an integer parameter.

Every time you call a function, the function call goes on the call stack.

What is stack?
Think of a stack as pile of plates at a buffet. When you add plate, you put the plate at the top of the pile. When you take a plate off the stack you take the top plate off.

Assuming you have a bunch on plates on the stack, the last plate that got put on the stack is the first plate that comes off the stack. We will learn more about stacks later in the semester.

Coding Practice:
If you have had me as a programming teacher before, you know that I advocate against using multiple return statements.

However, when it comes to recursive functions, using multiple return statements makes life easier.

Two things to keep in mind:

1) When executing a return, your function exits. You cannot have executable code below you return statement.

2) Every possible path your code takes, must return a value. If your function is supposed to return a value and you have a path with no return, you will receive an error, or a warning and we don't want either of those.

Defining the Foo function:

```
//--------------------------------------------------------------------------
// Function Name: Foo
// Description:
//   This function will demonstrate the proper way to create a recursive
//  function.
//
//--------------------------------------------------------------------------
int Foo(int number)
{
    // Check to see if number is greater than 1
    if (number > 1)
    {
        // Recursive case
        // The recursive function call will pass in a number that
        // is one smaller than the original.
        return Foo( number - 1);
    }
    else
    {
        // Base case
        return 1;
    }
}
```

NOTE: It is easy to see that this function doesn't accomplish anything. It is just an example of how to create a recursive function. The purpose of this function is to show you the flow of a recursive function. Later, we will trace recursive functions that do something.

Tracing the Function:

Before we can trace the function, we must first call the function:

```
5   int main(void)
6   {
7       int value = Foo(3);
8
9       return 0;
10  }
```

To create our trace, we will use the following table:

Function:	Parameter(s)	Next Line to Execute:

NOTE: There is no output for this function, so that column will be left off.

Every program starts in the main, so the first function on the stack is always the main function. When the program ends, this function is removed from the stack.

We start the program and let us assume there is a break point at line 7. (Program execution has paused at line 7).

Our call stack will look like this:

Function:	Parameter(s)	Next Line to Execute:
main	None	7

At line 7, there is a call to our function Foo passing in the value of 3.

Next, we step into the function and now our call stack looks like this:

First function call: The call stack looks like this:

Function:	Parameter(s)	Next Line to Execute:
Foo	number = 3	If-statement
Main	None	9

Now we will execute the next line code in the Foo function. You can see that the if-statement is true (3 > 1). We go inside the if-statement and call Foo again, this time with 2 (3-1).

Notice that later, when we return to this function that the there is no other code to execute. The else portion of the if-statement will be skipped and there is no code below that.

Next, we step into the second function call.

Second function call: The call stack looks like this:

Function:	Parameter(s)	Next Line to Execute:
Foo	number = 2	If-statement
Foo	number = 3	End of the function
Main	None	9

Now we will execute the next line code in the Foo function. You can see that the if-statement is true (2 > 1). We go inside the if-statement and call Foo again, this time with 1 (2-1).

Next, we will step into the third function call.

Third function call: The call stack looks like this:

Function:	Parameter(s)	Next Line to Execute:
Foo	number = 1	If-statement
Foo	number = 2	End of the function
Foo	number = 3	End of the function
Main	None	9

Now we will execute the next line code in the Foo function. This time the if-statement fails (1 > 1). We have now reached the base case. Since if-statement fails, we execute the else portion of the if -statement and return 1.

Since the function has finished, the third call comes off the stack returning a 1.

After the third function call: The call stack looks like this:

Function:	Parameter(s)	Next Line to Execute:
Foo	number = 2	End of the function
Foo	number = 3	End of the function
Main	None	9

Since there is no other code to execute the second function call will now finish and we will return to the first function call.

After the second function call: The call stack looks like this:

Function:	Parameter(s)	Next Line to Execute:
Foo	number = 3	End of the function
Main	None	9

Since there is no other code to execute the first function call will now finish and we will return to the main function.

Since the returning value when we reached the base is 1, the final return value is 1.

After the second function call: The call stack looks like this:

Function:	Parameter(s)	Next Line to Execute:
Main	None	9

When the program ends, the main function is removed from the stack and there nothing left.

This example is kind of pointless, but remember, the point of this function was to show you the flow of a recursive function, not to accomplish something.