CSCI 240 Lecture Notes - Part 4B -- Function Summary Sheet

Functions are chunks of code designed to do a particular task.

They are **called** from other functions (like *main()*) to perform the task.

The calling function often passes information used by the function to accomplish its task (via **arguments**).

You may need to define additional variables (**local variables**) inside the function to accomplish its task. Local variables are invisible outside the function. They do not retain their values between calls to the function (actually, there is a way to make this happen, but we do not cover it here).

The function often **return**s an answer to the caller. In a sense, the function "becomes" the value returned.

When you design a function, from the caller's point of view, decide:

- the task of the function
- the name of the function
- the return data type of the function
- the information needed by the function (arguments)

From the point of view of writing the function, decide:

- how to accomplish the task, given the arguments provided
- what local variables may be needed

You must make the following true of the function you write and use:

- the data type of the value returned matches the function's declared return type
- the arguments supplied in a function call match those specified in the function's header in terms of number, order, and data type
- the arguments supplied in a function call have defined values before the call is made

Consider the following function:

```
int cube(int x)
  {
  return x*x*x;
  }
```

It returns an int
Its name is cube
It takes one int argument, the number to be cubed
The name of the argument inside the function is x. You can use it as if it were a declared and initialized variable in the body of the function
The value returned is a int

Calling statements for this function might be:

```
n = cube(3);    //pass a literal

n = cube(y);    //pass a variable

n = cube(i*j);  //pass an arith. expr.

cout << cube(k);
```

When a function is called, ***copies*** are made of the supplied arguments, and the flow of control is transferred to the function, which then can use the arguments as if they were local variables. Any changes to an argument are made to the copy, not to the original in the calling program.

The code in the function executes until a return statement is encountered or control passes out of the end of the function.

If a function is declared to return a value, it ***must*** execute a return statement with a value of the declared return type.

The value supplied after the return statement is the value returned. Control then returns to the calling program.

A function may have more than 1 return statement, but only one is executed per function execution (the first one executed).

A function may have no return statement if it does not return a value (i.e. declared as *void*).

Remember, too:

main() is a function. It's always the first one called when the program starts.

No function can see ***anything*** about anything (including variables) in any other function. All it knows from the "outside world" is the values of the arguments passed to it.