Pop Quiz:

Define a C++ function printRev(S) with a string argument. It should print the string S (whatever it is) in reverse order on a line by itself. For example,

```
printRev("Hello");
```

prints

```
olleH
```

Ans:

```
void printRev (string S) {
    for (int X = S.size()-1; X >= 0; X--) {
        cout << S[X];
    }
    cout << endl;
}
```

Example: reverse(S) returns the reverse of S:

```
string reverse (string S) {
    string R = "";
    for (int X = S.size()-1; X >= 0; X--) {
        R.push_back(S[X]);
    }
    return R;
}
```

Homework (hand in, in your next lab):

Read in numbers from a file called "data.txt" until you reach an end of file. Sort the data into ascending order using the list sort() function (see LCR 4). Print out the numbers, one per line, on the console in descending order.

Data Types:

C++, like most PLs, has several different kinds of data type:
Simple (non-composite) data types: they have no parts. Examples: ints, doubles, bools, chars.
Compound data types: these have parts, which could be simple data types or compound data types.
Two primary kinds of compound data types: homogeneous (e.g., C-arrays and vectors) and heterogeneous (e.g. structs and classes).

In a heterogeneous data structure, the components do not have to be of the same type. This gives you more flexibility in what you can put into it, but less flexibility in the way you access it, because you can't compute the index of a component.

What is a struct? It's a group of instance variables, member variables, or fields that together make up one structure, class, or record.

Suppose we want to set up an employee database. Think of each record as standing for an employee. (We think of the structs as objects belonging to the type or class employee.) So in OOP we think of the computer objects as representing or simulating objects in some world. The objects belong to classes that behave in a similar way.

Parameter Passing Modes

There are several different ways to pass parameters to functions in C++. Two of them:

Pass by Value: what you normally get. A copy of the parameter is passed to the function, and discarded on function return.

Pass by Reference: what you get if you put "&" between the parameter's type and name. A reference to (or the address of) the actual parameter is passed to the function, instead of a copy. This permits the function to modify the original. It's also more efficient if the parameter is big.