Week 10, Lecture 2

Today's lecture is about two related topics: (See pdf for Chapter 9, Edition 2, Top down-design in Simulation Design for the Racquetball problem)

1. Top-Down Design Method (for problem-solving)

Solve a problem by breaking it up into subproblems.

The big problem is at the "top". It's like a "grandparent".
Say there is 1 grandparent.

You break up this big problem into a few subproblems; they are one level down.
Think of these as "parents". Say there are 3 parents.

Each of these subproblems may be broken up into their own subproblems.
Now you are looking at smaller problems 2 levels down from the top.
Think of these as "children".
Say each parent problem has 2 child subproblems.

By piecing together the solutions to the subproblems 2 levels down (children), you solve a parent problem for each pair of children.

By piecing together the solutions to the subproblems 1

level down (parents), you solve the grandparent problem.

Since the grandparent problem is the original problem, this "divide-and-conquer" method has helped you to solve the big problem.

Information (parameters) are passed from levels above to levels below so levels below can work on the subproblem.

Information (results) are passed from levels below to levels above, in putting solutions together.

Basic idea:

When you start you only have a rough idea of what needs to be done.

You construct the general form of the solution, writing functions, often leaving placeholder functions for pieces you know how to solve and so will fill in later.

You focus on breaking up the parts you don't know, and go down levels to add detail.

Because you start at the "top" and work your way down, this is called the "Top-down Design" method of problem solving.

2. Modular Programming

The easiest way to understand this is to think of your solution to each of the parts of the problem above as a function.

More generally, it is a module or sub-program. But we'll think of these part solutions as functions.

So each "child" and each "parent" will be a function. Naturally the grandparent is also a function, or the main program.

Because of the structure of the relationships, you can guess that the grandparent function calls the parent functions, and the parent functions
Call the children functions.

While calls proceed downwards, the results are returned upwards.

So the grandparent receives the final result.

Basic Idea: You have to be VERY CLEAR on the purpose of each function, its limitations, its inputs and its outputs.

If you are not clear about this, you will not be able to write the functions.