RES.TLL-004 STEM Concept Videos, Fall 2013

Transcript ï Basic Programming Techniques

There are many ways to complete a task---even a seemingly simple one like eating cereal.

When programming a computer to complete a task or solve a problem, repetitive techniques like iteration and recursion are extremely useful. In this video, we will look at these problem-solving techniques.

This video is part of the Problem Solving video series. Problem-solving skills, in combination with an understanding of the natural and human-made world, are critical to the design and optimization of systems and processes.

Hi, my name is Niaja Farve. I am a doctoral student in Electrical Engineering and Computer Science at MIT.

Before watching this video, you should be familiar with introductory programming and simple data structures.

After watching this video, you will be able to: Divide a programming problem into simpler, analogous pieces. And, solve the problem by combining solutions to the simpler pieces.

Chapter 1: Problems are combinations of simpler problems.

In computer science, we often want to solve complex problems. However, computers deal best with performing easy tasks over and over again. We utilize the computer's ability by implementing repetitive techniques to incrementally solve our complex problems.

Though eating a bowl of cereal is a fairly simple task that most of us can complete automatically, we need to think carefully about how to program a computer to do the same. Let's take a closer look at the problem and identify the fundamental steps used to frame cereal eating for repetitive computation.

In this example, let's suppose the computer understands the basic operation of eating a single bite of cereal, but does not understand how to eat an entire bowl of cereal. So how do we "teach" the computer to eat a bowl of any size greater than one bite? Pause the video here and think about it.

The first step when approaching a complex programming problem is to break the problem up into analogous, but simpler pieces that we can tell the computer how to directly solve. So what is a simpler version of eating a whole bowl of cereal?

One possibility is eating a smaller amount of cereal.

We already mentioned that a small, non-zero amount of cereal the computer can handle eating is a single bite.

Going one step up, eating two bites-worth of cereal is equivalent to eating a single bite twice. The next step when approaching a complex problem is to deduce a pattern. That is, how does a generally larger problem look in comparison to the simpler version? Here, we notice that eating any amount of cereal is equivalent to the sum of eating multiple bites-worth of cereal.

Now that we've broken up our problem and understand how the pieces will fit back together, we can put our solution into a generalized code framework:

Start with telling the computer the procedure for solving the simplest problem. Then, repeat this procedure on subsequent pieces until the desired endpoint is reached:

If the bowl contains cereal, take one bite of cereal. Repeat until there is no more cereal.
You may recognize this type of solution as an iterative approach.

Or we could also take the following alternative approach:
Start with telling the computer how to solve the simplest problem. Then, break the problem into simpler and simpler pieces until we reach the version we've already told the computer how to solve.
Does the bowl contain 1 bite of cereal? If so, take the bite. If not, divide it up into a bowl containing one bite and a bowl containing the remainder. Repeat this procedure on the resulting bowls.

In this case, we end, up with a series of bowls containing one bite, which the computer knows how to eat.

Breaking up a problem into progressively simpler, but analogous pieces in this way is known as a recursive approach. Because the solution to the most complex problem depends on solutions to the simpler pieces, recursion creates a queue of jobs waiting to be completed.

In contrast, when using iteration, there is no such dependency from one instance of the problem to the next.

So even though the computer ends up consuming n bites of cereal in both cases, the iterative and recursive approaches arrive at this answer in very different ways.

There are many different ways to successfully eat a bowl of cereal or solve any given programming problem. The key is to 1. Break the problem into analogous pieces that the computer can solve, and 2. Combine the solutions to the pieces into a solution for the more complex problem.

Chapter 2: String manipulation

In our previous cereal-eating example, breaking down the problem into simpler pieces was fairly straightforward. Now, let's look at a slightly more complex problem.

We would like to write a function, downup, that takes an input string and prints out progressively smaller and larger sub-strings of the word as so.

To help us work with strings, we have a helper function, substring, which extracts a portion of a string, beginning from the first character up though a specified end index.

Following the framework we discussed earlier, what is a simpler version of down up that we can easily handle?

How about down up of a single letter string. The desired output is achieved by simply printing the string.

Moving one step up, we see that the desired output of a two-letter string can be accomplished by printing the string, printing the string shortened by one letter, and printing the full string a second time.

And moving up one more step to a 3 letter string…

Are you starting to notice any patterns? Pause the video to think of a possibility.

Though there are many different patterns, here is one you may have come up with: With every string, we are sandwiching the _solution_ to a string one character shorter between two printings of the full string. With this mindset, we are poised for a recursive solution to this problem.

Recall the general framework for a recursive solution: Tell the computer how to solve the simplest problem. Then break the problem into simpler pieces until we reach the simplest problem:
If the string is a single letter, print it.
Otherwise, print the string, solve for down up of the string one character shorter, and print the string again.

Try now, if you haven't already done so, to frame the problem in a more iterative manner. Pause the video to discuss.

We can notice that we are repeatedly printing substrings of the full string, with each step moving the end index from the original length down to 1.
This is followed by again printing substrings, but this time increasing the end index back up to the original length.

We can program this solution using two iterative loops. Recall the general iterative code framework: Tell the computer the procedure for the simplest problem. Repeat the procedure on subsequent pieces until the endpoint is reached.

In our first loop, we print the substring, decrease the index by one, and repeat the procedure until the index reaches one.

In the second loop, we move in the opposite direction. Print the substring. Increase the index by one and repeat the procedure until the index is greater than the original length.

Both approaches, while very different, are completely valid! There is no one correct way to solve a problem. Some solutions may even have both recursive and iterative elements.

Chapter 3: Towers of Hanoi

Now lets solve a third problem with an even more complex pattern.

In the famous Towers of Hanoi problem, the goal is to transfer a stack of rings from pillar A to pillar C.
We can only move a single ring at a time, and can use pillar B as “extra” workspace.
We cannot place a larger ring on top of a smaller ring.

Lets follow the framework and start by working out the simplest problem: transferring 1 ring. Here we can simply move the ring from A to C.

Okay, now let’s try to transfer a stack of two rings. First we move the ring 1 to B, then ring 2 to C, then move ring 1 from B to C.

Now, let’s try something a bit harder. Let’s try to transfer a stack of three rings. Ring 1 moves to C, ring 2 goes to B, and ring 1 goes to B. Now ring 3, which was on the bottom, is free to move to C. Then, ring 1 goes to A, ring 2 goes to C, then ring 1 finally goes to C.

Notice that after we moved the bottom ring to C, we essentially arrived at the same conformation as when trying to transfer 2 rings: We have two stacked rings and an extra empty pillar. And because the largest ring is in the desired, final position and does not impede the movements of any of the remaining smaller rings, we can treat the pillar as being empty.

This observation is crucial to the recursive implementation of the towers of Hanoi solution. We transferred n-1 rings to the extra pillar, moved the largest ring to the final position, then transferred the n-1 rings to the final position.

Can you frame a pseudocode solution to the problem? Pause the video here to work out a possible solution.

Recall again the general recursive framework: Tell the computer how to solve the simplest problem:
If we’re transferring a single ring, move it to the destination pillar.

Then, break the problem up into progressively simpler pieces:
Transfer n-1 rings from the source to the extra pillar, transfer ring n from the source to the destination, and transfer n-1 rings from the extra pillar to the destination.

Note that the source, destination, and extra pillar designations change with each function call!

It’s always a good idea to check your code with a test case. Pause the video here and check your code for the case of N equals 4. You may also wish to check our code and compare the two solution methods.

Now lets check our code and traverse through the solution for transferring 4 rings.

We’re not transferring a single ring, so lets transfer 3 rings from A to B.

We're still not transferring one ring, so lets transfer 2 rings from A to C.

We're still not transferring one ring, so lets transfer 1 ring from A to B. And we can finally move this single ring!

Now we return to the previous call, and can move ring 2 from A to C. Then we transfer our n-1 stack from B to C. This results in transferring 2 rings from A to C!

Returning one more step back, we move ring 3 to B. Now we transfer 2 rings from C to A.

Finally, we’ve returned back to our original function call, and we’ve completed transferring 3 rings from A to B. So, we can move our largest ring number 4 from A to C.

Now we need a whole other set of recursive function calls to transfer the 3 ring stack from B to C! Pause the video now to finish checking the second half of our solution.

Notice how quickly the number of function calls grew! Good thing we have a computer that is very good at following repetitive instructions!

To Review in this video, we showed you how recursion and iteration take advantage of a computer’s ability to repeat simple tasks.

To approach a complicated programming problem, first solve some simpler versions and try to identify a pattern. Then, depending on the type of pattern you found, fill in a recursive or iterative code framework. And remember, iterative, recursive and even mixed solutions to a single problem may all be correct!

RES.TLL.004 STEM Concept Videos

Fall 2013