Acting out Algorithms – A “hands-on” approach to teaching Computer Science

Jenna Butler: jcamer7@uwo.ca
PhD Student in the Department of Computer Science
TA and Sessional Instructor
Computer Science 1026 and 1027 – Computer Science Fundamentals I and II

Introductory computer science (CS) is challenging to teach for two reasons:

1) At Western the course is usually extremely polarized – there are students who have never taken a CS course because it is not a requirement for the program, and people who have taken it all through high school
2) It is very abstract and most of the concepts exist within the computer, making it difficult to “show” students what is going on

While these issues can prove problematic, they can also be great opportunities for new teaching methods! The polarization allows for peer instruction as often there will be some students who have already mastered the concepts and can teach those around them. Secondly, the abstract nature of the course encourages the development of more creative teaching methods.

With this in mind, I decided to take a hands on approach to teaching CS, one where students would be actively involved in the learning and teaching, and would get to have a “feel” for what is happening inside the computer. To showcase the flexibility of this I have provided three examples of my hands-on approach to teaching computer science.

Learning Goals:
I carried this idea throughout the course, but the three main examples I am presenting are for teaching inheritance, recursion and linked list algorithms. They are each explained in their own sections.

Inheritance:

The idea behind inheritance in CS is that you can write code for a general group (such as a Person) then use it to develop code for specific cases of a person (for example a Student, Professor, Lecturer, etc). The code in a sub class (in this example, Student, Professor, Lecturer, etc) can override code in the parent class (Person) or can use the code they get from their parent. This hands on example took 20 minutes to do (1 hour to prepare) and requires 4 or 5 notebooks that need to be prepared ahead of time as well has 4 or 5 student volunteers. The following describes the activity:

- I created a workbook to show in front of the class that said “Person” and each page contained different bits of the code that might be necessary to model a Person
- A student volunteered to come to the front, hold the workbook, and “be” the Person model
- A different student came to the front to act like the Student model and I gave them a notebook with the word Student on the front
- They were told it inherited form Person and asked to fill in the code necessary
- I then instructed them to open the book and they found it was filled with code auto generated from Person (much to their surprise!)
- I had other students come up to be the Professor (and Instructor and so on) only to find their books filled with code as well
- Later, to show the concept of overriding a method in Person, students were able to physically rip out pages from their notebooks and put in new pages

- The example continued with other physical representations of inheritance and the students enjoyed being actively engaged in the class. I believe having a tangible model of what was happening in the computer greatly improved their understanding of the topic and allowed them to have some fun!

Recursion:

Recursion is widely considered one of the most difficult concepts to teach in computer science. An exampled entitled the "Towers of Hanoi" is often used to teach it. The idea is you have a tower (pictured at right) and you need to move each donut/disk on the tower to another tower without violating two rules: you cannot put a bigger disk on a smaller, and you can only move one disk at a time. When a computer simulates this concept it takes over a billion years to do it for only 64 disks. It is one of the known "impossible" problems computers today still cannot solve. The Towers of Hanoi example can be found on Wikipedia and is the story of a group of monks who are stuck with the task of trying to move 64 disks.

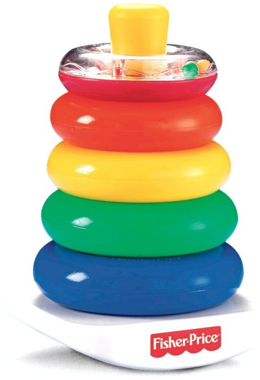

Fig 1. The "tower" the students experimented on.

In order to teach this example, I created a power point slideshow (complete with spooky music and low lighting), that took the students back to a time long ago and made the original legend of the Towers Of Hanoi come alive. I acted like the fabled story teller describing the fateful night where three monks were visited by an evil sorcerer and instructed to move the disks - but when they finished the world would end. By engaging the students in what was a silly, theatrical performance, they realized this next topic must be something interesting and different and were excited to learn how the computer approaches this problem.

Next they got the opportunity in groups to try to solve the towers on their own, then present their results in front of the class. Then the groups got to attempt to code the problem on their own (a difficult task) and actually requested more group time because they were so eager to find the solution. While I was nervous to present this difficult topic in an unusual way, I believe the students will never forget the concept of the Towers of Hanoi.

The only requirements were 3 sets of towers and it took about an hour for the whole activity.

Linked-list algorithms:

Linked-lists (shown to the right) are a computer science concept where data is stored in "nodes" that are "pointing" to the node beside them. The algorithms on these lists involve complicated changes to the "points" as you can see in the picture. One added complication is if you break a link you physically lose all the data it pointed to (resulting in a memory leak) unless you have added some safe guards.

In order to teach these algorithm, I had 8 student volunteers assist me. The only required materials are the students and it took about 30 minutes to go through the four prepared examples.

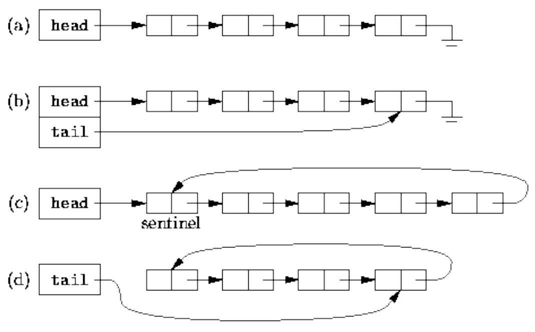

Fig 2. A series of linked lists

Each student at the front played a role. The node students would stand in a line, holding hands with the node that they would be pointing to. The "head" node pointed to the whole list, and then the other students were the "programmers". They would physically have the students break their links and point to new people. If they got stuck, students from the audience would shout out suggestions to the students. Often, one of the programmers would break a link that resulted in a memory leak and all the student nodes would be left on their own and the audience could actually see that they were physically gone from the system! This worked very well and the students got a kick out of the acting at the front of the class.