Nifty Assignments Panel

Nick Parlante — Stanford University — nick.parlante@cs.stanford.edu (Moderator)
Owen Astrachan — Duke University — ola@cs.duke.edu
Mike Clancy — University of California at Berkeley — clancy@eecs.berkeley.edu
Richard E. Pattis — Carnegie Mellon University — pattis@cs.cmu.edu
Julie Zelenski — Stanford University — zelenski@cs.stanford.edu
Stuart Reges — University of Arizona — reges@cs.arizona.edu

Introduction

This panel is a forum for sharing a few favorite CS1 and CS2 assignments from the SIG-CSE community. Given our limited time, each panelist will concentrate on the broader niche, strengths, and weaknesses of their assignment, rather than its specific implementation. Fortunately, for each assignment there is a web page with all sorts of practical details and materials for anyone interested in using or studying the assignment.

More information about the panel and its assignments is available at: http://cse.stanford.edu/assignments/. Although obvious, it bears mentioning that you should cite the original author if you decide to use one of these assignments. Citation promotes the atmosphere of sharing.

On its face, this panel presents practical information to help you adopt or get ideas from some successful assignments, and promotes a discussion of the techniques and lessons of successful assignments generally. In the larger picture, this panel is an experiment in communicating very applied, practical material within the SIG-CSE community. This idea may have future directions in SIG-CSE publications and electronic repository efforts.

Mike Clancy - Cat And Mouse (CS1)

My favorite CS1 assignment simulates an urban park, where a cat chases a mouse around the circular base of a statue of (say) Niklaus Wirth. If the cat sees the mouse, it moves toward the statue. If it doesn't see the mouse, it circles around the statue. If the cat and the mouse are both at the base of the statue and the cat would move past the mouse, it eats the mouse as it passes. If the cat spends too much time chasing the mouse, it gets bored and wanders off.

I had three goals for the assignment when I designed it: a solution shouldn't need arrays, it should need around 75 to 100 lines of code, and it should involve some sort of mathematical computation. (I was teaching the "intro programming for engineers" at the time.) This problem met those goals nicely. The cat and mouse positions can be represented in two radius/direction pairs; a few other variables are needed for bookkeeping. I have a commented Pascal solution that's 85 lines long. The trigonometry necessary to derive the formula for the "sees" function is nontrivial.

I later noticed other nifty features. For instance, the problem's complexity may be varied by reversing the direction of the cat with respect to the mouse: when both are circling in the same direction, the possibility of an infinite loop arises. In either case, control flow is sufficiently complex that thorough testing is a significant task. Where graphic display facilities are available, the chase can be graphically depicted. Also, the assignment adapts well to an object-oriented programming environment, with cat and mouse objects. For more information, please see:
http://cse.stanford.edu/assignments/catandmouse/

Stuart Reges - Bagels/Jotto (CS1)

Bagels is a number guessing game similar to MasterMind. The user repeatedly guesses a number and is given clues as to how many digits in the guess are correct and in the correct place and how many digits are correct but in the wrong place. I assign this program about halfway through CS1. It involves loops, interaction with the user and array or string manipulation. I have given the assignment in several different forms and in several languages (Pascal, C++, Java).

The assignment has two major benefits. First, the students enjoy it. I have observed many students showing their bagels program to friends and relatives and using the program even after the course is over. The second benefit is that the problem involves a nontrivial bit of algorithmic thinking. Writing code that provides the correct clues takes a while to get just right, mostly because a digit cannot match more than once.

Probably the biggest weakness of the assignment is that it takes a while to explain the rules of the game. I also suspect that most instructors will find the program too difficult for a one-week assignment and not rich enough to justify making it a two-week assignment. To address this problem, I use a variation of Bagels known as Jotto. It involves guessing 5-letter words with other 5 letter words. Because of its difficulty, I'm happy to use Jotto as a 2-week or even 3-week assignment towards the end of CS1. For more information, please see:
http://www.cs.arizona.edu/people/reges/bagels/

Richard E. Pattis - Efficiently Determining DNA Sequences (CS1)

A molecular biologist wants to determine the bases in a gene (a strand of DNA, consisting of a sequence of tens of thousands of bases: A, C, G, or T). The technique is to clone the gene, randomly break the clones into fragments small enough to be sequenced, and then figure out the original gene by looking at the fragments. This works because fragments from one clone will often overlap with nearby fragments from another clone, allowing the construction of larger and larger fragments.

The program stores a vector of fragments (where each fragment is an object that contains a string label and a vector of its bases). Then, it finds two fragments that overlap, removes them, merges them, and puts the merged fragment back into the vector of fragments, until there is only one fragment (success) or no fragments overlap (failure).

I use this CS1 assignment when students first start manipulating data structures built by composing vector and class objects. It illustrates a real-world, combinatorial problem; but the true focus of this assignment is on the separation of concerns of program correctness and efficiency. Initially students solve the problem over 2 weeks, concentrating on writing clean/simple and correct code. Then, they use a program profiler in a lab to locate the hot spots in their code, and use this information to make small changes that affect the whole program's performance dramatically. An hour's work with the profiler improved my original program by a factor of four — with the biggest improvement coming in an unexpected place. Another program that fits well with this assignment is a TCP/IP-like message reassembly task. For more information, please see: http://cse.stanford.edu/assignments/dna/

Owen Astrachan - Huffman Coding (CS2)

A favorite assignment in our CS2/Data Structures course is the implementation of a pair of programs for data compression using Huffman coding. This is roughly a two-week project that combines several of the different data structures we have studied during the semester. Implementing the programs requires care in debugging and testing since the output is not text. Most students build test and debugging methods as part of the program or eventually wish that they had.

Instructors of the course like the assignment because the implementation requires the use of several data structures: vectors to count character frequencies, maps/tables that map characters to a coding pair (bit sequence and number of bits), binary trees or tries for determining the coding pairs during compression and for determining characters during decompression, priority queues for building the coding tree/trie. We typically assign the program near the end of the course when we have covered all these topics.

Students like the assignment because they build a demonstrably useful program from scratch. There is room for distinction in the program as well since students are free to develop more sophisticated methods for representing the encoding information at the beginning of a compressed file. A complete description of the project, instructor guidelines, and possible student starter materials in C++ and Java are available at: http://www.cs.duke.edu/csed/poop/

Julie Zelenski - The Random Sentence Generator (CS2)

The “Random Sentence Generator” (or RSG as it is affectionately known) is a fun and versatile assignment we have given in many forms and languages in our CS2 and later courses (Pascal, Ada, LISP, C++, C, Java). The basic idea is to read in a context-free grammar, and then use it to generate random sentences. The grammar is a collection of definitions, where each definition gives the collection of possible expansions for a non-terminal, and each expansion is a sequence of terminals (plain words) and non-terminals. Representing the grammar in memory is a nice, reference-intensive data structure. Once the grammar is read in, producing random sentences is a pretty straightforward but satisfying recursion problem. Over the years, the students have supplied us with an entertaining collection of grammar files: Star Trek episodes, James Bond movies, Dear John letters, extension requests, haiku, and so on.

My favorite use of the RSG is to teach collection abstractions by using them to store the several layers of the grammar. The needed collection abstractions may be pre-provided things such as the java.util or C++ STL objects. Better yet, have the students implement a collection ADT (week-1), and then use the RSG as a follow-on client (week-2). This works well because the grammar's natural three layers of “collection” make it a demanding client. If the week-1 collection implementation also happens to be complex (say, a chunked linked list), then the student gets a real appreciation of the wall of abstraction from both sides.

A useful feature of the RSG is that it doesn't need any platform-specific features (i.e. no graphics, sounds, events, threads, etc.) so it has few portability obstacles. It's text-only assignment that's still very entertaining to run. For lots of RSG materials, including grammars, and a demo Java applet, please see:
http://www-cs-faculty.stanford.edu/~zelenski/rsg/

Nick Parlante - Darwin's World (CS2)

Darwin's World is my favorite CS2 assignment for stressing modularity — it's large and has several natural sub-components. It also happens to be fun to play with. Its individual data structure are modest: 1-d and 2-d arrays, structures, possibly some simple pointers. However, the whole thing layered together is pretty complex. Its size and difficulty are its main weakness. A CS2 student will need two or more weeks to complete it.

Externally, Darwin's World looks like a chess-board type world inhabited by creatures of a few species. The creatures hop around the world and compete with each other (Darwin does need simple graphics). Internally, the program has three significant parts: A textual species programming language, creatures that inhabit the world and run (interpret) their species program, and a world that moderates things. Darwin's good points are...

-It reaches that critical mass of complexity where decomposition and modularity matter. The component parts are individually complex, but they fit together nicely.

-The running version of the program is fun to play with.

-Through the Species programming language, the program gives students a sense of how interpreted languages work. Also, species/creature is a nice analog for class/object.

-With a working program (theirs or otherwise) students get drawn into working on their own species — devising more and more intelligence into them so they do well against other species. It's easy to have a species tournament.

The assignment is difficult, but worthwhile. It stresses the right skills, and it results in something that students like playing with. For more information, please see:
http://cse.stanford.edu/assignments/darwin/