Nifty Assignments Panel

Nick Parlante (moderator)
Stanford University
nick.parlante@cs.stanford.edu

Julie Zelenski
Stanford University
zelenski@cs.stanford.edu

Mike Clancy
University of California at Berkeley
clancy@eecs.berkeley.edu

Owen Astrachan
Duke University
ola@cs.duke.edu

Stuart Reges
University of Arizona
reges@cs.arizona.edu

Introduction

The Nifty Assignments panel is a practical forum for the sharing of assignment ideas — demonstrating good assignment features and tradeoffs, and providing concrete materials for the CSE community to use.

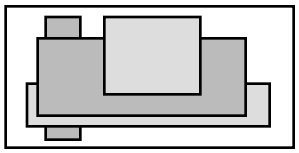
Each panelist will introduce the basic idea of their assignment, give a quick demo, and describe its niche in the curriculum — its strengths and weaknesses. The live presentation is merely an introduction to each assignment. Fortunately, the following web page has handouts, data files, etc. for each of the assignments...

http://cse.stanford.edu/nifty/

This is the second SIG-CSE with a Nifty Assignments panel. So far, the panels have been constructed by strong-arming people known to the moderator to have good assignments. This has worked well early on — high quality with very little editorial or logistical effort. However, given the level of interest, future assignment-sharing panels would benefit from drawing wider participation with a real CFP style call for people's favorite assignments.

Mike Clancy - Windows and Regions (CS1)

The Windows and Regions program involves list manipulation in the solution of a very visual problem. The problem centers on a stack of rectangular regions in a window. The regions are arranged randomly and overlap each other. One region is in front, and gradually smaller and more irregular pieces of the other regions are exposed as they go back. A click in an uncovered part of region brings it to the front.



The graphics code is provided to the students, and they concentrate on the storage and algorithmic parts of the assignment. Nifty things about this assignment:

- The problem requires an extension to the Stack data type, and thus isn't merely "reinventing the wheel".
- It is not too hard, yet it builds something visual and active that the students can relate to the real window managers they use all the time.
- The output, activity, and bugs of the program all express themselves visually.
- It is extensible in various ways. Examples: multiple implementations of the region list, one using an array, the other using a linked list; support of colored squares within regions (clicking on an uncolored area creates a square, and clicking on a colored square erases it).

Stuart Reges - Personality Test (CS1)

This assignment is based on a personality/temperament test developed by Keirsey and Bates in the pop-psych book Please Understand Me. The first part of the assignment is a straightforward application of 70 multiple choice questions. Collecting answers to these questions is a fine CS1 problem which may or may not be used as part of the assignment (if not, there are tools to do it).

Each person's survey boils down to four numbers. One number for Extroversion vs. Introversion, one for Intuition vs. Sensation, one for Thinking vs. Feeling, and one for Judging vs. Perceiving. Of course the results can be fabulously inaccurate, but they do appear to vaguely correlate with each person's self-image. Accuracy aside, the results are great to play with and start discussion.

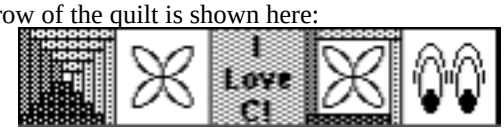
The nifty assignment is to write a program which manipulates the personality data of all the students. You can store, sort, and massage the data in the obvious flat-file ways. The most interesting application is to think of each of the four qualities as representing a dimension in space, so each person is a point of a four dimensional space. The fascinating analysis then is to pick a particular person, and compute the most-similar/most-different ordering of all people in increasing order of distance from that person. This shows you the most similar people to that person at the front of the list, and the most different at the far end of the list.

Part assignment and part ice-breaker, this assignment builds something that the students want to play with. It also leads to a great gift-exchange where you pair off all the students, each with their "most similar" person (computing the stable marriage may be part of the assignment or not — it lacks the fun factor of the "similar/different" sorting). Each person gets their dual a gift — the theory being that their intuition in gift selection will be perfect.

Julie Zelenski - Quilt (CS1)

The quilt assignment is a simple CS1 assignment that has the students write a program to draw a "sampler quilt" using a simple graphics library.

Our sampler quilt is made up of five blocks, each with its own individual theme, while still featuring common elements Each block is repeated five times and all 25 blocks are arranged into a square to form the entire quilt. One row of the quilt is shown here:



The result is an attractive pattern of interesting shapes and colors of which the student can feel proud.

We use this assignment fairly early on in CS1. After a few weeks of syntax, variables, and control structures, we move on to functions and decomposition. A graphical drawing assignment seems to work marvelously for teaching students the benefits of designing reusable functions and encouraging thoughtful decomposition. The elements of the picture that are visually repeated help point the student in the correct direction of unifying the underlying code. The graphics library we use in CS1/2 offers just the basic primitives: lines, arcs, text, and color. By deliberately not including often-used routines such as rectangles, circles, etc. these become the first building blocks that the students construct for the quilt. Within the quilt design, we deliberately include repeated elements within and among the blocks (the filled-and-framed circle, centered text, etc.) that give further opportunity for code unification given appropriately parameterized utility routines.

The substance of this assignment is about gaining experience with functions, parameters, and decomposition, and as a side effect they become familiar with the graphics library. There are some simple calculations involved in arranging the blocks and drawing the lines and arcs, but nothing too off-putting for the non-mathematically-inclined student. I see the content matter as welcoming to all students, and perhaps particularly appealing to the creative, right-brain type who might be turned off by a more mathematical/engineering project.

The assignment seems to work well. The first more substantive project in CS1 usually causes frustration for students who haven't yet developed strong debugging skills, yet this has not been as much a problem for quilt given the very direct correlation between bugs and their visual result. The students are excited by a program with a tangible result and it often inspires them to embellish beyond the basic requirements; we have many beautiful and extraordinary versions submitted by the more advanced students.

Owen Astrachan - Word Ladder (CS2)

The Word Ladder program is similar to a "6 degrees of separation" or "Oracle of (Kevin) Bacon" game, but using words rather than people or actors. Simply, from a given starting word, find the shortest "ladder" of single letter changes which leads to some final word, where each intermediate state is also a word.

For example: clash, flash, flask, flack, flock, clock, crock, crook, croon, crown, clown.

This program is nifty on several levels. It can be used early in a CS2 course when queues are discussed, and brought up again later when/if graph algorithms are covered. A naive algorithm of checking every word as the potential next word in a chain runs quickly, but precomputing all edges in the graph of words makes finding ladders instantaneous at the expense of a significant up-front cost to compute the graph.

We have used this assignment early in a CS2 course after covering linked structures (to connect a word with its predecessor), stacks, and queues. We have also used it at the end of a CS2 course as an example of breadth-first search in a graph. We have had students animate the process of putting words on the queue to understand that a shortest path is guaranteed to be found. For extra credit we have given the problem of finding the longest word ladder which is, in general, an intractable problem, though solvable in the case of real five-letter words.

We are in the process of incorporating the Oracle of Bacon in addition to words into the assignment. Certainly the next step is to connect nifty CS educators.

Nick Parlante - Tetris Brain (CS2)

The first part of the Tetris Brain assignment implements Tetris in a modular way — dividing the solution into Piece and Board classes. We've given this assignment with procedural ADTs in Pascal and C, and later switched to OOP in C++ and Java. This first part of the assignment stresses classic OOP modularity: build modules that encapsulate significant complexity and have clean abstractions, test the modules separately, and then fit the modules together to solve the overall problem.

The second part of the assignment adds a robot brain module that plays the game by itself — an algorithm that given a piece and a board, finds a good rotation and column to place the piece. The code to build a decent brain is surprisingly simple, since the Piece and Board classes hide most of the implementation complexity.

Once the brain is working, the program becomes sort of mesmerizing. The students can spend hours watching their brain play and tuning its heuristic code. A neat extension of the brain is a Tetris "adversary". Normally, the Tetris game chooses the next piece for the player randomly. In contrast, the adversary looks at all the possible pieces that could come next, finds the piece that gives the worst best move according to the brain, and gives the player that piece. Spring it on the roommate. The adversary is not that hard to code, again, because it builds on the abstractions of the other modules.

The Tetris assignment is fairly difficult— requiring perhaps 2 weeks in CS2. The Board class in particular is algorithmically complex and has many little boundary cases. The strength of the assignment is that it shows off OOP modularity well — taming the complexity by dividing the program up. The brain and adversary features build on the basic modules nicely and make the thing fun to watch and play with.