Nifty Assignments

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

Owen Astrachan
Duke University
ola@cs.duke.edu

Brent Heeringa
Williams College
heeringa@cs.williams.edu

Thomas P. Murtagh
Williams College
tom@cs.williams.edu

David Reed
Creighton University
davereed@creighton.edu

Mehran Sahami
Stanford University
sahami@cs.stanford.edu

Christopher A. Stone
Harvey Mudd College
stone@cs.hmc.edu

Karen Reid
University of Toronto
reid@cs.toronto.edu

Categories and Subject Descriptors
K.3.0 [**Computers and Education**]: General.

General Terms
Algorithms, Design, Languages.

Keywords
Education, assignment, homework, project, repository, library, nifty, object oriented programming.

Introduction
Assignments determine much of what students actually take away from a course. Sadly, creating successful assignments is difficult and error prone. With that in mind, the Nifty Assignments session is about promoting and sharing successful assignment ideas, and more importantly, making the assignment materials available for others to adopt.

Each presenter will introduce their assignment, give a quick demo, and describe its niche in the curriculum and its strengths and weaknesses. The presentations (and the descriptions below) just introduce each assignment. The Nifty Assignments home page, **http://nifty.stanford.edu**, organizes and provides the handouts, data files, starter code, etc., for each assignment freely on the web. Nick's running ACM editorial: I look forward to the day when, in line with its mission to promote CS, the ACM organizes and distributes all of its materials freely in this way.

If you have an assignment that works well and would be of interest to the CSE community, please consider applying to present at Nifty Assignments at **http://nifty.stanford.edu**.

Star Charts and Constellations (CS1) Karen Reid
Who can walk outside on a clear night in the countryside and not look up at the stars? Identifying constellations is one of the first lessons in astronomy and ties together science and mythology. Numerous applications present star maps to help stargazers identify the stars and constellations they look at. In this assignment, students use data from published star catalogs to draw a star map complete with constellations. The shapes of the constellations come from the well-known book by H. A. Rey, The Stars: A New Way to See Them.

The assignment works in an attractive domain and benefits from the use of real-world data. The graphical nature of this assignment is appealing, yet the underlying data structures and algorithms involved are simple enough for CS1 students to grasp.

This assignment gives students practice reading files and building data structures. It could be expanded to teach students about combining data from multiple sources and the noise or errors present in real data sets.

Perhaps most importantly, the result of this assignment is a picture that students can show to their friends and family to explain what they are doing in their CS class.

Hiding in Plain Sight: Steganographic Images (CS1) - Brent Heeringa and Thomas Murtagh
Steganography is the art of hiding one message "in plain sight" within another. Like cryptography, steganography helps ensure privacy in communication. Unlike cryptography, steganography does not prevent eavesdroppers from examining the contents of a sensitive message. Rather, it conceals the fact that a sensitive message is even being exchanged by hiding its contents within another, non-sensitive message.

In this assignment, students implement algorithms to steganographically hide messages within images and then to extract hidden messages from images. The messages to be hidden can either be other images or text. In both cases, the hidden message is encoded by changing the least significant bits of the numbers encoding the brightness of the pixels in the image used to camouflage the secret information. The modified images are indistinguishable from the originals, but can hide millions of bits of information.

This assignment provides appropriate ways to exercise student skills with array manipulation in a CS1 course. The pixels of the images being processed are presented as two dimensional arrays. Text messages to be encoded are first converted into one dimensional arrays holding the bit values corresponding to their ASCII encoding and then distributed systematically over the two dimensional arrays of pixel values. A simple GUI interface is implemented to select the images and/or text to be processed.

SIGCSE'09, March 3–7, 2009, Chattanooga, Tennessee, USA.
ACM 978-1-60558-183-5/09/03.

Random Art (CS1-PLs)
Christopher A. Stone

Random expressions produce Random Art! Build an expression in x and y by nesting simple primitives (e.g., sin(π×_), cos(π×_), project, and average), and consider the 2-by-2 square around (0,0). We can interpret the value at each point as a grayscale level, or use three expressions to get RGB values.

As Andrej Bauer (http://www.random-art.org) has observed, when an expression is sufficiently complicated (say, 8-12 levels deep) there is a surprisingly good chance of getting an interesting picture. Thus, one can generate completely random expressions, see how they look, and keep the best pictures.

This simple idea inspired a popular assignment in our Programming Languages course. Implementing Random Art is an excellent "second" exercise in functional programming, being both more challenging and more interesting than reverse, append, and similar recursive code. Our assignment further exercises higher-order and first-class functions.

But there's no intrinsic need for functional programming. The core idea, representing and evaluating expressions, is simple and could be implemented in any language or paradigm. By providing skeleton code with carefully chosen holes, one could choose whether students concentrate on expression evaluation, representations for symbolic data, recursion, loops, or graphics. The assignment is nifty because it uses classic algorithmic and recursive material to build appealing graphical output.

FacePamphlet (CS1-CS2) - Mehran Sahami

Social networking is incredibly popular among students. Giving them the opportunity to implement a social network both motivates them and helps demystify the internals of a type of application that they may already be quite engaged with as users. Not to mention, it makes for a fun demo to impress their friends ("Look, I made Facebook").

The FacePamphlet assignment (i.e., a lightweight version of Facebook) manages the information in a basic social network comprised of "user profiles", where each profile has a name, as well as an image, current status, and list of friends associated with it. Profile contents are defined specifically to match features found in real social networks, with the ability to display images being an important feature to have students engage more personally with the application. The program stresses classic data management (arrays/lists and maps) as well as a solid understanding of control flow and program state. The assignment lends itself easily to extensions (e.g., file processing to save/load networks, support for communities in the network, network graph analysis, etc.).

FacePamphlet has been given successfully as a final assignment in a CS1 course, with many students noting it as their favorite assignment in the class. Interestingly, the social theme was more popular on a relative basis with female students (more so than some game assignments given in the class).

Encryption and the Enigma Machine (CS1-CS2) David Reed

Cryptography has played an important role in the history of computing, from motivating the development of the first electronic computer to enabling secure Web-based communication and commerce. Substitution ciphers, such as the Caesar cipher, are simple to understand yet form the basis of many modern encryption tools, such as the Enigma machine used in World War II.

This session will present three related assignments involving substitution ciphers. The first assignment could be given early in a CS1 course, after String methods have been introduced. Students are given a class for encoding/decoding text using the Caesar cipher, which they must then modify to make it more robust and powerful by allowing different substitution keys and subsequently rotating the key after each encoding. The second and third assignments extend the idea of a rotating substitution cipher, requiring students to design and implement classes for modeling an Enigma machine. The first of these involves a simplified model of an Enigma machine, using multiple interconnected, rotating substitution keys. The second is a more complex but historically accurate model of an Enigma. To help students visualize the workings of the machine, they first build a working model out of paper using the Do-It-Yourself Enigma Machine.

These assignments are nifty in that they combine class design, String manipulation, and GUI implementation with a broader historical context. In addition, building a working Enigma model with scissors and tape is a fun hands-on activity for the classroom.

DNA Splicing (CS2) Owen Astrachan

This assignment is based on modeling a process from computational biology/genomics related to restriction enzymes and polymerase chain reaction (aka PCR). The discovery and explanation of these processes in Biology generated two Nobel prizes, and the modeling in this computational simulation is definitely nifty. We use the assignment in our data structures course to emphasize performance trade-offs that are facilitated by structuring data intelligently. In building assignments we strive to anchor them to the real world and to show that simple, but well-designed data structures and algorithms can make a huge difference in the size of the problem that can be modeled and in what experiments can be run computationally. In the case of this simulation we think we have achieved both goals.

We provide students a simple implementation of constructing a recombinant DNA strand that takes a string representing DNA, breaks it at every occurrence of a restriction enzyme, e.g., CGAATC, and splices in a new string at every break/splice point. This implementation is a single loop using standard string search methods and concatenation. We analyze its complexity and performance in terms of N, the length of the original string; B the number of places the string is broken; S the size of the string being spliced into the strand to create the recombinant strand. We then ask students to model the DNA using a linked list rather than a string. The new implementation creates a string of size N + SB characters in N+B in both time and space, in contrast to the original implementation which requires time and space of N + SB (or worse if a Java String is used rather than StringBuilder). When S is large this is a huge difference, and we ask students to find the value of S that breaks the original implementation and to compare that to the new version which is extremely efficient.

This assignment is nifty because of the huge performance gains with a simple implementation, because it is a linked list example that makes a difference, because students get to experiment, and because it connects the slightly mundane CS ideas of strings and lists with a vivid real world example.