Fall Meeting

October 26, 2005

Harriton Senior High School
Lower Merion School District
AGENDA

5:00 – 5:45 Check-In and Food

Check-in / Program pick-up
Informal discussion
Dinner
Try out Alice, Lego Mindstorms, and BlueJ

5:45 – 6:00 Welcome and Introduction

Host - Andrew Ruether, Harriton High School

6:00 – 7:00 Presentations

Alice – Steven Cooper, St. Joseph’s University
Teaching Objects First – Mickey Engle, Plymouth-Whitemarsh High School
CIS Forum - Bob Allen, Drexel University

7:00 – 7:30 Discussions

University faculty: What advice do you have for high school teachers?
High school teachers: Review of course offerings at different schools.

7:30 – 7:50 Group discussion summary

7:50 – 8:00 Conclusion

What’s next?
E-mail list
Spring meeting?
To access the Internet at Harriton High School:

- If using your own computer: set up proxy server for proxy.lmsd.org, Port 80
- When prompted enter username: webaccess, password: password (October 26th only!)

Scientists from the RAND Corporation have created this model to illustrate how a “home computer” could look like in the year 2024. However, the needed technology will not be economically feasible for the average home. Also, the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the FORTRAN language, the computer will be easy to use.
AP Computer Science Workshop Notice - Summer 2006
Villanova University

Villanova will be offering a course or workshop this summer for AP CS teachers (new or experienced). Our goal is to create the course/workshop in such away that we can share teaching techniques, strategies for topics, "cool" labs, and AP specific resources that benefit teachers who are already experienced with Java. If there are teachers that still need content knowledge (particularly when it comes to AB topics) then we could gear part of the workshop/course in that direction.

This will be an opportunity for teachers this summer to get together, share/learn Computer Science, and get 3 college credits. Last year it was a 1-day a week for 9 weeks course. This summer it may be that again or a 1 week workshop. Exactly what the course/workshop looks like is still open to feedback. It's Villanova's goal make sure AP CS teachers have the tools/knowledge to do it effectively.

Contact Brian Ellis for more information or feedback:

Brian Ellis
K-12 Mathematics Consultant
Manheim Township High School
Lancaster, PA 17601
(717) 560-3097 x3324
brian_ellis@mtwp.net
The **CIS Forum** is a Learning Community to support K-12 Computer and Information Science Education. We aim to provide a variety of high-quality resources, support for teaching and online activities for students.

The CIS Forum is just being launched. We are getting help from our friends at [Math Forum](mailto:MathForum@drexel.edu). At first we are focusing on Computer Science, but we will also be developing an Information Science Forum. Please check back to watch us grow.

---

**Problems of The Week**
**Notes and Links**
**Computer Club Central**
**Computer and Information Scientists**

---

For more information, contact [CISForum@drexel.edu](mailto:CISForum@drexel.edu)

---

E-mail Bob Allen via [CISForum@drexel.edu](mailto:CISForum@drexel.edu) if you have ideas for how an online forum for computer science could help your teacher.
Favorite Assignments

Tips and Techniques - Sneaking in Extra Material
Daniel Joyce, Villanova University

Visual Basic Game Project
Curt Minich, Wyomissing Area High School

Empirical analysis of graph algorithms
Mary-Angela Papalaskari, Villanova University

"Ninja-Sort" in Alice and Overlap of Simple 2D Graphics
John P. Dougherty, Haverford College

On - Off Widgets and Devices
Robert Beck, Villanova University

The 12 Days of Christmas
Woody Woodbury, North Penn High School

Algorithm Analysis Lab
Mickey Engel, Plymouth-Whitemarsh High School

Automated Teller Machine
Sally Moritz, Lehigh University

Basic problem solving/programming
Carol R. Sweeney, Villa Maria Academy

What is programming? and Auxiliary Functions
Megan G. Connolly, The Haverford School

Karel J. Robot Dance Party: Get the Party Started
Andrew Ruether, Harriton High School

C++ Final Exam
Katie Cooper, Lower Merion High School

Image courtesy of Cal Poly's Comm and Computer Services
1. Introduction
Did you ever wish you could sneak some extra material into a course but it just doesn’t fit with the syllabus and there’s not enough time to add another unit to an already crowded calendar? In this Tips and Techniques I described how I snuck some computing history into an introductory programming course.

2. Context
When I found myself teaching an introductory programming course for the first time in many years, in the Fall of 1999, I decided to try to include some exposure to computing history as part of the course. Part of the impetus for this mini-project was the fact that the first programming assignment that I gave to the class involved programs to encode and decode secret messages. I was struck by the fact that one of the earliest uses of computers, during World War 2, was related to this first assignment. So, I decided to tie the description of the assignment to computing history. Once I got started, it became a challenge to also tie later assignments to some aspect of computing history. All together, I linked 7 of my 10 programming assignments to computing history and/or some aspect of the modern use of computers. Hopefully this provided the students with a realistic context for the “toy” programs they were developing.

3. Application
My first programming assignment required the students to write two programs, one to encode a string so that it is difficult to read, and one to decode the string. The assignment description was posted on the web and began as follows:

“The early development of computers was hastened by the need for computing power during World War 2 … in England, computing pioneer/genius Alan Turing continued his work on cryptanalysis (coding and decoding messages), work which helped the Allies immensely during the war. Read about this chapter (and others) of Turing’s interesting life: Alan Turing - A short biography.”

The last five words of the above description were a web link to an interesting essay about the life of Turing. The problem introduction was, of course, followed by a description of the required program functionality along with some sample input and output.

So, the pattern was set. I would give an assignment description, write a little bit about how it ties into some aspect of computing, and provide a link to a related web site for the curious.

Here is a list of the assignments that I managed to present following this pattern:

1. Encoding and decoding secret messages were connected to Alan Turing’s life and work.
2. Creating an applet to draw “art” in its window was, with a bit of a stretch, related to Donald Knuth and his series titled “The Art of Computer Programming”. Hey, I never claimed the connections between the assignments and history would be perfect!
3. A program that holds a conversation with its user is more clearly related to Joseph Weizenbaum’s “Turing Test” program Eliza.
4. For an exercise in repetition I noted that it was the repetition inherent in engineering work that prompted pioneers like the German engineer Conrad Zuse to search for mechanical aids.
5. Implementing the dice game Pig was related to the relationship between computers and games.
6. Running multiple simulations of the Pig game and collecting and presenting information about the results was tied to the use of computers for simulations and modeling.
7. Extending the previous assignment to present the information as a histogram was tied to the use of computers for information visualization.

The assignment descriptions are available at http://www.csc.villanova.edu/~joyce/csc1051/fall99/intro.html.

4. Results
If nothing else, I had some fun making the assignments and tying them to computing history. It would be fair to ask if the students actually followed the links and read the historical information. Based on the results of a history related surprise bonus question, that I included on the first test, I would have to report that the students did not seem to absorb much of the historical information during the first third of the semester. But the good news is that they did much better on the history related “surprise” bonus questions on the remaining tests! Imagine that.
Visual Basic Game Project  
Curt Minich  
Wyomissing Area High School, Wyomissing, PA

Class: Visual Basic .NET Programming, an introductory, high school programming course taken by 9th graders. This course prepares and feeds students into our AP Java Programming course.

The assignment details are listed at:


The students have about 6 weeks in April & May to complete this project working on it in class about one 40 minute class period per week until the last week or two when about 3 days per week are devoted to the project. It's a great culminating project that puts a lot of responsibility on them to research, design, & implement practically every concept and technique that they learned in the course (logic, debugging, if & loop control structures, variable declaration & assignment, object/method property syntax, methods and call statements, and even designing & using their own VB.NET class.) I then distribute the students' executables at the end of the class as a sort of class portfolio!

The only downside to this assignment is that it really gets my students excited about GUI programming only to take my AP Java course the following year where I cover very little GUI or applet programming until after the AP exam in May.
Empirical analysis of graph algorithms
Mary-Angela Papalaskari
Dept of Computing Sciences, Villanova University

Class name: Analysis of Algorithms (for college juniors through grad students).

Description: This was a major assignment, complementing the course's theoretical analysis of algorithms. It uses peer evaluation to teach students how to put together good reports and the importance of good documentation. There is a short paper that I presented at ITICSE in 2003 that describes the assignment further: http://www.csc.villanova.edu/~map/peereval.html

See next page for the contents of the paper.
ITiCSE 2003 Tips & Techniques
Peer Evaluation in an Algorithms Course

Mary-Angela Papalaskari, Dept of Computing Sciences, Villanova University, Villanova PA 19085, (610) 519-7333, map@villanova.edu

Students need a lot of practice in expressing themselves clearly and concisely; making their ideas understood; and presenting their work in a professional manner. For this reason, in my courses I generally try to emphasize that the programming part of the projects is just the beginning and that, in addition to a working program, I expect a clear, convincing, and complete report describing the work. I describe one such project that I gave to a graduate algorithms class at Villanova University and which was then graded by other students in the same course. I have given similar projects to undergraduate classes, but so far I have experimented with peer evaluation only in the graduate classes.

There are several good reasons for incorporating peer evaluation:

- Students are often locked into a particular way of thinking about a problem. Seeing the work of others makes them realize that there may be alternative viewpoints.
- When explaining a solution to their professor, who, they assume (rightly or wrongly!) knows all about it already, students are not really focused on actually explaining their thinking --- they are just trying to show that they know "the answer." If they are aware that someone else who is just familiar with the problem, but has not actually solved it, will be reading their report they are better disposed from the outset to write a clear explanation.
- We can talk ourselves dry with the importance of good programming style and documentation. They need to see the horrors produced by someone else, because it is hard to recognize their own.

I choose a relatively complex project that has several parts and divide the students in two groups. Students of each group work individually. Some parts of the project are tackled by students of both groups, while others are only done by one of the groups. For example, in Fall 2002 I gave a project to implement some graph algorithms. All students had to implement depth-first-search to find cycles in graphs represented using adjacency matrices and they also had to write a module that, given a probability p, generates a random graph such that the probability of an edge between two vertices is given by p. Group A also had to implement depth-first-search using adjacency linked-lists, whereas Group B had to implement breadth-first-search instead. Each group then had to experiment with their programs in order to answer specific graph-theoretic questions: - How large does p (density) have to be so that the probability of a cycle is 50%? How is this affected by graph size? (Group A) - Which algorithm finds a cycle faster: DFS or BFS? How is this affected by graph density and size? (Group B)

Some specific guidelines for experiments were included in the project description. See http://www.csc.villanova.edu/~map/8301/f02/proj2.pdf for details.

Note that both groups were working on similar, but not identical problems. This puts them in a good position to understand each other’s work, without having already done it themselves. Having done something similar, they have a clear appreciation for the difficulties involved.

This project was a big success overall and an enormously rewarding experience. The only real difficulty, for me, was the logistics of administering such a project. The students were asked to write a brief summary of what they learned after the peer evaluation phase and these were most enthusiastic, sometimes pointing out benefits that I myself had not anticipated. Several of my students already have industry jobs and they said how valuable this type of project is and how they wished they had been given such projects when they were undergraduates. So, that is my next goal.
"Ninja-Sort" in Alice

You are given a list of evil ninjas of varying sizes and asked to put them in order, initially ascending, then descending, then by "the one you like most" -- in practice you use selection sort interactively by choosing a ninja and Alice swaps it into its place in line.

The benefits of this exercise is the introduction to sorting without any programming, then can be studied for such topics as linear structure, loop vs. recursion, abstraction representation, etc...

The lab from last spring is at the link below (see optional exercise):
http://www.cs.haverford.edu/curriculum/courses/cmsc100/lab2.html

Overlap of Simple 2D Graphics

Currently taught in C++ with Python interface.

This is our first assignment in the majors course used to get students to think deeply about problem specification, solving a problem instance vs. a general problem, even pre- and post-conditions. The student is asked to develop a sequence of function definitions to determine if two graphic objects overlap in 2D. In lecture we present 1D range overlap, then 2D rectangle overlap (like many GUIs). In lab students implement similar functions for:

- 2 circles
- 1 circle and 1 rectangle
- 2 line segments (this is now extra credit)

This work is part of Lab 1, see the link below:
http://www.cs.haverford.edu/curriculum/courses/cmsc205/links.html
On - Off Widgets and Devices
Robert Beck
Villanova University

Class Name: Human-Computer Interaction (college advanced elective course)

Here's an assignment that I've used to get the human-computer interaction students thinking about design choices in an interface. It works well because it ties the familiar (turning a light on or off) with computer interface, reinforcing the role of mental models in system design. Depending on the classroom, initial examples are easy to come by. It lends itself to impromptu enhancement including the problem of figuring out which switch controls which light (a Car Talk Puzzler). Sometimes students actually take a field trip to Home Depot to check the light switch designs.

On-Off Widgets and Devices

Introduction
One of the fundamental actions connected with artifacts that use electricity is turning something on or off. The devices that are used to accomplish this action (switches) come in many designs. In this experiment, you are to catalog these designs and to propose reasons why there are so many different ones.

Step 1:
By exploring your living space, the lamp department at the local department store, and the electrical aisle at the hardware store, discover at least 15 different devices used for turning a light on or off. Record your answers in a table that looks like:

<table>
<thead>
<tr>
<th>Device Name</th>
<th>User Action</th>
<th>State Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard wall toggle switch</td>
<td>Move toggle up or down</td>
<td>ON is up, ON label shows</td>
</tr>
</tbody>
</table>

Discuss why there are so many different switch designs.

Step 2:
Find at least 8 types of widgets that are used within a WIMP interface to turn a system ON or OFF. Record your answers in a table similar to the one above.

Discuss whether the reasons you proposed in Step 1 for the multitude of choices for light switch design carry over to the widgets you described above. Reflect on whether having several choices for widget design should be allowed given the call for consistency within the interface.
The 12 Days of Christmas
Woody Woodbury
North Penn High School
woodywoodbury@comcast.net

Class: AP Comp. Sci (although I use it on other programming classes as well)

I have my students write a program where the user enters a day number from 1 to 12 inclusive. The program computes and displays the total gifts and cost for that one day, the program then computes the cumulative number of gifts and cumulative cost of all the gifts from day 1 up to and including the requested day.

This is a list of the 12 gifts and their unit prices. The prices are several years old but each December, PNC bank reports the up to date unit prices on their web site.

a Partridge in a Pear Tree; $113.00;
2 Turtle Doves; $25.00;
3 French Hens; $5.00;
4 Calling Birds; $70.00;
5 Golden Rings; $50.00;
6 Geese a Laying; $25.00;
7 Swans a Swimming; $500.00;
8 Maids a milking; $5.15;
9 Ladies Dancing; $436.70;
10 Lords a Leaping; $353.70;
11 Pipers Piping; $134.02;
12 Drummers Drumming; $133.09;

Keep in mind that each day, “My true love” gives not only the gift of the day; they also gives one of each of the previous gifts as well.

On day 4 for example, the recipient receives not only 4 calling birds but also 3 more French hens, 2 more turtle doves and another partridge in a pear tree. That means by the end of day 4, the person has been given a total of 4 calling birds, 6 French hens, 6 turtle doves and 4 partridges in their respective pear trees.

In java, each day is an object. The concepts of arrays, array lists, recursion and GUIs can nicely be incorporated into the assignment.
Algorithm Analysis Lab
Mickey Engel
Plymouth-Whitemarsh High School

Taught AP Computer Science AB
Ideal for all AP and College classes as an introduction to Big-O notation.

This assignment is a good introduction to Big-O notation and as a first assignment in a Data Structures course. The assignment combines class design and implementation with some graphics components and a review of the sorting algorithms. The result are graphs that clearly show that some algorithms operate faster than others. With these graphs in view, the conversation regarding Big-O can begin with an analysis of the graphs for linear, quadratic, and logarithmic properties. Last year, it took over 5 days in the block to implement the sorting/searching algorithms and generate the graphs in this lab. For that reason, I have assigned the sorting/searching algorithms as a summer assignment. We will begin the class on the first day this year with this lab.

Continued on the following pages.
In this lab we will be looking at and comparing the efficiency of sorting algorithms.

1. Create a class called timer that has the following methods: `start()`, `stop()`, `elapsedTime()`, and `reset()`. Use the `System` object for obtaining your time, not the `Calendar` object.
2. Import the Sorting and Searching classes provided in the last lab (without iterators) into the project.
3. Create a tester program to test the each of the sorting algorithms 10 times for each number of elements and find the average. Once an algorithm reached 10 seconds to process, stop. Fill in the chart below with the information.

### SELECTION SORT

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### BUBBLE SORT

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### INSERTION SORT

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>Avg</td>
</tr>
<tr>
<td>-------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Rewrite the code in your tester class so that the program generates a random list and calculates the time to process each, and finally calculates and displays the average time for processing a list of that size. Remember, time only the algorithm, not the creation of lists or output.

5. Find the break point of each algorithm, in other words, at what n is the processing of the list too big for the computer to handle or too costly in terms of time. In other words, I don’t want you to sit for 5 minutes while an algorithm processes a list. For this exercise, our break point will be the n (number of elements) that requires @ 30 seconds to process.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BreakPoint(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection Sort</td>
<td></td>
</tr>
<tr>
<td>Bubble Sort</td>
<td></td>
</tr>
<tr>
<td>Insertion Sort</td>
<td></td>
</tr>
<tr>
<td>Merge Sort</td>
<td></td>
</tr>
<tr>
<td>Quick Sort</td>
<td></td>
</tr>
</tbody>
</table>

6. Next, set a loop that will create a random list of n numbers. Process this list with each of the sorting algorithms. Store it as a Point in the form (n, time). ‘n’ starts at 0 and increases by 500 each time for Selection, Bubble, and Insertion and 1000 each time for Merge and Quick. Be sure to have conditions that stop processing algorithms before they reach their break point. We will come out of this program with 5 PointLists, one for each algorithm.

7. Take the PointLists created in the last exercise and plot them on a graph, you will have to determine scale and each graph should be in a different color. When you are finished, print the graph for you instructor. You will need to determine a scale to convert your points so that they will appear properly on an 800x600 canvas. You may want to adjust your loop conditions to only calculate values that will appear on the graph. Use a max n of 100,000.
Automated Teller Machine
Sally Moritz
PhD student at Lehigh University

Class: High school Java

I teach some elements of UML (specifically, use cases and class diagrams) in my Java class. We spend quite a bit of time on design, focusing on defining the requirements of a problem and planning the solution before jumping into code. My students (11th and 12th grade) are currently developing a simple ATM (automated teller machine) program. I have them write out use cases first - before any thought about program design or code. They did very well with the problem. I think it's because all of them are familiar with how ATM machines look and work. Even so, they still had a lot of good questions on the specific functionality of the ATM. It showed them how much goes into making sure all the requirements of a system are identified.

Problem Description:

Implement a simple ATM (automated teller machine). It should work much like most common ATMs work. A customer walks up to the machine and enters his or her card. The machine reads the card number, then asks the customer to enter a PIN (4-digit password). The ATM looks up the customer's card number in the bank's central database and verifies that the PIN is correct. If it is, the ATM presents 3 options to the customer: Check account balance, Deposit cash or check, and Withdraw cash. Check account balance simply displays the current balance. Deposit asks the customer to enter the amount, then updates the account balance. Withdraw cash asks the customer how much to withdraw; if there is enough cash in the customer's account, the money is output and the account balance is updated. A receipt is printed for both deposits and withdrawals.

When the ATM is turned on at the beginning of the day, money to be given out for withdrawals is placed inside the machine. That money is decremented each time a withdrawal is made. If a customer asks for a withdrawal and there is not enough cash left, a message telling the customer that the transaction cannot be completed must be displayed. A running total of all the cash and checks deposited throughout the day is also kept.

Test cases listed on following page.
ATM Machine

Actors: Customer, Bank Database

Use Cases:

1. Verify card number/PIN
   1. Customer enters card number
   2. Machine asks customer to enter PIN
   3. Customer enters PIN
   4. Machine asks Bank Database to verify card number and PIN
   5. Machine displays list of options to customer

Alternative: Wrong PIN entered
At step 4, if Bank Database determines card number/PIN invalid
   1. Machine displays message "PIN is invalid"
   2. If this is the third attempt, end transaction, else go to step 2

2. Check balance
   1. Customer asks to see current balance
   2. Machine gets account balance from Bank Database
   3. Machine displays balance

3. Deposit money
   1. Customer enters deposit option
   2. Machine asks customer to enter amount
   3. Customer enters amount of deposit
   4. Machine adds amount to current balance
   5. Machine updates current balance in Bank Database
   7. Machine adds amount to total deposit money in machine

4. Withdraw money
   1. Customer enters withdraw option
   2. Machine asks customer to enter amount
   3. Customer enters amount to withdraw
   4. Machine gets current balance from Bank Database
   5. Machine gives customer money
   6. Machine subtracts amount from current balance
   7. Machine updates current balance in Bank Database
   8. Machine prints receipt
   9. Machine subtracts amount from total cash in machine

Alternative: Insufficient funds in account
At step 4, current balance is less than withdrawal amount
   1. Machine prints message "Not enough money in account"
   2. Go to step 2

Alternative: Not enough cash in machine
At step 3, amount is less than total cash in machine
   1. Machine prints message "Sorry, can't complete transaction"

5. End transaction
   1. Customer enters end option
   2. Machine returns card
Class: High school C++ class

Slides read left to right, then top to bottom.
The first lesson is a "first day activity" that I have done for a few years. The second lesson is one I used this year for the first time - teaching auxiliary functions in Scheme.

---

**Lesson Plan Title:** Day 1 – What is programming?

**Concept / Topic To Teach:** Writing an Algorithm

**General Goal(s):** To have the students understand the process of writing an algorithm. This is a “first day of programming class” activity.

**Step-By-Step Procedures:**

1. Have each student make a paper airplane. When complete, have students show their planes to the rest of the class. Be sure to point out the uniqueness of each plane.

2. Fly the paper airplanes, and record the distance each plane travels.

3. Have each student write step-by-step instructions for folding their paper airplane and exchange their instructions with another student. Following their peer’s instructions, have each student fold another paper airplane. When complete, have students show their planes to the rest of the class. Compare these planes to those created in step 1.

4. Fly the paper airplanes, and record the distance each plane travels. Compare these distances with those of the first run.

5. Review the difficulties in writing and interpreting an algorithm.
(This lesson is part of the “How to Design Programs” curriculum that uses the Scheme programming language.)

**Lesson Plan Title:** Auxiliary Functions

**Concept / Topic To Teach:** developing auxiliary functions

**General Goal(s):** To have the students understand the use of auxiliary functions.

**Step-By-Step Procedures:**

1. Have students consider the following problem:

   The owner of a movie theater has complete freedom in setting ticket prices. However, he knows that the more he charges, the fewer the number of people who can afford tickets. In a recent experiment the owner determined a precise relationship between the price of a ticket and average attendance. At a price of $5.00 per ticket, 120 people attend a performance. Decreasing the price by a dime ($.10) increases attendance by 15. Unfortunately, the increased attendance also comes at an increased cost. Every performance costs the owner $180. Each attendee costs another four cents ($0.04). The owner would like to know the exact relationship between profit and ticket price so that he can determine the price at which he can make the highest profit.

   While the task is clear (determine the best ticket price), how to solve it is not. All we can say at this point is that several quantities depend on each other. As a class, discuss and determine the various dependencies described in this problem, one at a time.

   - **Profit** is the difference between revenue and costs.

   - The **revenue** is exclusively generated by the sale of tickets. It is the product of ticket price and number of attendees.

   - The **costs** consist of two parts: a fixed part ($180) and a variable part that depends on the number of attendees.

   - The **number of attendees** depends on the ticket price.

To begin to solve this rather formidable problem, let’s describe a function for each of these dependencies. Being true to our design recipe, start with contracts, headers, and purpose statements. Here is the one for profit:

```scheme
;; profit : number -> number
;; to compute the profit as the difference between revenue and costs
;; at some given ticket-price
(define (profit ticket-price) ...)
```

It depends on the ticket price because both revenue and cost depend on the ticket price.

Allow the class to work in small groups to write the contract, purpose and header for the remaining three dependencies listed above (revenue, costs, and number of attendees).
Karel J. Robot Dance Party: Get the Party Started
Andrew Ruether
Harriton High School
ruethea@lmsd.org

Class Name: High school AP Computer Science A (Java)

I use Karel J. Robot to teach polymorphism, inheritance, interfaces, and abstract classes. After students are familiar with the basics of Karel and object-oriented concepts, I assign this lab to check for comprehension.

The lab requires students to create a party with three different dancer types based on an abstract dancer class. Each type of dancer does a unique, student-created Karel robot dance. Two of the dancer types also know how to talk on their cell phones (which is an interface).

I provide a template for the GetThePartyStarted, Party, and Dancer classes. I also provide the CellPhone interface requires the getCellNumber, talkOnPhone, and checkNumberOfMessages methods. The students fill in the code for those classes and creat 3 Dancer subclasses, each of which has a Karel robot do a different type of dance.

The GetThePartyStarted class creates a Party, tells the Party to have its dancers do a dance, and tells the dancers to talk on their cell phones.

The Party class controls dancers. It creates three dancers and contains methods to ask the dancers to act.

The Dancer abstract class contains an abstract method to make the dancer dance, and abstract method to print out a message, and a method that returns the name of the dancer.

This lab lets me see if the students understand the object-oriented concepts we have discussed in class and the students have fun creating their own funny Karel dances.

In this example, SquareDancer, BreakDancer, and SlamDancer are implementations of the abstract Dancer class. In addition, the BreakDancer and SlamDancer implement the CellPhone interface (the SquareDancer is old-fashioned!). The dancers are instantiated in the Party class.

I would be happy to provide code examples. See my e-mail at the top of the page.
Write a program that will function as a point-of-sale system at a sports arena snack bar. The snack bar sells only six different items: a sandwich, chips, hot dog, brownie, regular drink, and a large drink. All items are subject to a 6% sales tax. Set prices for the products.

The program should repeatedly display the menu until the sale is totaled. The program should keep a running total (subtotal without tax) of the amount of the sale based on costs that you place in constants for each of the food items. The program should also keep a running total of the number of items that are ordered. The subtotal and the number of items ordered should be displayed somewhere on the screen each time the menu is displayed.

S – Sandwich (price)
C – Chips (price)
H – Hot dog (price)
B – Brownie (price)
R – Regular Drink (price)
L – Large Drink (price)

X – Cancel and start over
T – Total the sale

# Items ordered =
Subtotal =

If the sale is canceled, clear the subtotal and the number of items ordered and display the menu again. When the sale is totaled, calculate the sales tax and print the number of items ordered, subtotal, amount of sales tax, and final total due on the screen.

# Items ordered =
Subtotal =
6% Sales Tax =
Total =

After the 4 lines above are displayed on the screen, the program should print “Thank You” either vertically or horizontally for each item that was ordered.

If the user orders 5 items, the output should look like

# Items ordered =  5
Subtotal =  $TBD.00
6% Sales Tax = $TBD.00
Total = $TBD.00

Thank You, Thank You, Thank You, Thank You, Thank You
You must use 2 functions in this program.
The subtotal, sales tax, and total must be displayed with two decimal places of accuracy.
When you have completed this program, email your program as an attachment to ________________.

The subject line should be (your last name only)-FINAL.

**REMEMBER** The top of your program should have the following:
//Your full name
//FINAL
//What the program does

GRADING  (Each item below is worth 4 points. The final is worth 100 points.)

- Cost of 6 items are declared and initialized as constants
- Menu is repeated each time
- Subtotal is repeated each time
- Subtotal is calculated correctly
- Number of items is repeated each time
- Number of items is calculated correctly
- User can enter upper or lower case letters when choosing an item
- Program deals with an unexpected choice (If the user enters a W what should be done?)
- Sale can be canceled (#items ordered and subtotal should be zero)
- After a sale is canceled the user is asked to pick from the menu again
- Sales tax is calculated correctly
- Total is calculated correctly
- The final output displays the number of items ordered, subtotal, 6% sales tax, and the total
- The subtotal, sales tax, and total are displayed with two decimal places of accuracy
- “Thank You” is printed the correct number of times
- Function 1 is used correctly and efficiently
- Function 2 is used correctly and efficiently
- for loop(s) operate correctly
- else and if...else loop(s) operate correctly
- while and do...while loop(s) operate correctly
- Program runs
- Program is correctly indented
- Program contains descriptive comments
- Program was emailed as an attachment with the proper subject line
- A hard copy of the program was printed

Modified version of Project 6-2
Activities Workbook for Fundamentals of C++ second edition, page 77