

Tenure Portfolio  
Steven Wolfman

September 1, 2008

## Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>4</b>  |
| <b>2</b> | <b>Teaching Accomplishments</b>   | <b>4</b>  |
| 2.1      | Motivated Learners . . . . .  | 5         |
| 2.2      | Active, Reflective, Social Learners . . . . .   | 7         |
| 2.3      | Reflective Teaching . . . . .   | 10        |
| 2.4      | Practical, Respectful, Efficient Teaching . . . . .   | 10        |
| <b>3</b> | <b>Service</b>  | <b>11</b> |
| 3.1      | Department . . . . .  | 11        |
| 3.2      | Faculty & University . . . . .  | 13        |
| 3.3      | CS Community . . . . .  | 13        |
| <b>4</b> | <b>Summary</b>  | <b>14</b> |
|          | <b>Appendices</b>   | <b>15</b> |
| <b>A</b> | <b>CPSC 101 Learning Goals Development Report</b>   | <b>15</b> |
|          | Report to the UBC CS department on the iterative development of learning goals in CPSC 101. . . . .   | 15        |
| <b>B</b> | <b>CPSC 101 2007W1 Topic-Level Learning Goals</b>   | <b>17</b> |
|          | Detailed, measurable goals for students to achieve in CPSC 101 that served as the prototype for UBC CS's Carl Wieman Science Education Initiative work on learning goals. . . . . | 17        |
| <b>C</b> | <b>CPSC 101 JavaScript Introduction Lecture</b>   | <b>22</b> |
|          | Lecture slides introducing JavaScript programming to CS0 students using a series of active learning exercises. . . . .  | 22        |
| <b>D</b> | <b>CPSC 101 "Post Mortem" Lecture Notes</b>   | <b>28</b> |
|          | Typical notes I send to myself and my TAs after many course sessions. . . .   | 28        |
| <b>E</b> | <b>CPSC 101 Notes Posted for Lecture Day One</b>  | <b>29</b> |
|          | Typical "notes of interest" that I post to the web after many class sessions. .   | 29        |
| <b>F</b> | <b>CPSC 101 Project Report Marking Scheme</b>   | <b>32</b> |
|          | Marking scheme for the primary deliverable from CPSC 101 students' term project. . . . .  | 32        |
| <b>G</b> | <b>CPSC 101 Course Archive Table of Contents</b>  | <b>39</b> |
|          | Contents listing for a typical package of materials I use to hand courses to the next instructor. . . . .   | 39        |

|          |  |           |
|----------|--|-----------|
| <b>H</b> | <b>CPSC 111 Robot Assignment</b>   | <b>40</b> |
|          | The first of three assignments from a CPSC 111 course, in which novice programmers automate tasks that use the mouse and keyboard. . . . . | 40        |
| <b>I</b> | <b>CPSC 111 Weekly Reading Questions and Answers</b>   | <b>45</b> |
|          | Typical weekly questions submitted by students with the answers I posted to the CPSC 111 newsgroup. . . . .                                | 45        |
| <b>J</b> | <b>CPSC 121 Staff Meeting Agenda</b>   | <b>51</b> |
|          | Typical agenda posted in advance of a weekly staff meeting of the 15 TAs and instructors. . . . .  | 51        |
| <b>K</b> | <b>CPSC 221 Card Trick Notes</b>   | <b>53</b> |
|          | Detailed notes from term-long exploration of a card trick using discrete mathematics tools. . . . .  | 53        |
| <b>L</b> | <b>CPSC 313 Practice Final Exam</b>  | <b>64</b> |
|          | Practice exam for CPSC 313 with learning goals attached to each question. .  | 64        |
| <b>M</b> | <b>CPSC 344/444 First Day Survey</b>   | <b>72</b> |
|          | Survey and active learning exercise that ties into ideas across CPSC 344/444.  | 72        |
|          | M.1 Survey Comments . . . . .  | 75        |
| <b>N</b> | <b>CPSC 344/444 Affect Lecture</b>   | <b>77</b> |
|          | Lecture slides for a unit of CPSC 344/444 I designed on working with affect (emotion) in design. . . . .                                   | 77        |

## 1 Introduction

This portfolio includes my CV, the artifacts in the appendices below, the artifacts included on my behalf (e.g., teaching evaluations), and this document.

I consider the artifacts and CV the key content: the former is primary evidence of my teaching, curricular development, and service accomplishments; the latter is a summary of a wider set of artifacts and those accomplishments that produce no tangible evidence.

Because of this, and because my CV already includes a section documenting individual teaching accomplishments, I see the purpose of this document as presenting a structure that lends coherence to the primary artifacts. As much as possible, then, this document is a kind of narrative annotated bibliography of its appendices.

I have separated this document into a teaching section—intertwining formal course-related teaching and mentoring, advising, curriculum development, and other teaching activities—and a second section on professional service to my department, faculty, university, and disciplinary communities.

## 2 Teaching Accomplishments

Each great teacher I have known is certain that their subject is the most exciting and engaging in the world. They respect the diversity of fields and ideas, but get them on their own subjects and their eyes widen, their pulses quicken, and their audience cannot help but ride that wave of enthusiasm, sometimes for life. Students that catch this wave transform into self-motivated learners, creating their own knowledge through avid exploration.

My primary task as a teacher is to help students find their motivation. Consistently high student evaluations of my teaching on the items “Presented material in an interesting manner” and “Stimulated students to think”<sup>1</sup> indicate that I successfully model my own enthusiasm. Section 2.1 below describes some of the ways I help students to find their own motivation through connections to research, everyday problems, and intriguing puzzles.

However, getting students hooked is the beginning, not the end, of teaching. Learning is a difficult process, and I strive to make my students better and more inquisitive learners.

At its most effective, learning is *active*, *reflective*, and *social*. That is, learners *actively* engage with the material: solving problems, discussing ideas, composing arguments, and otherwise using the skills, concepts, and knowledge they are learning. Effective learners also *reflect* on their learning process, explicitly investigating their own trajectory with an eye to what they have learned, how they learned it, and how they could learn more effectively. Finally, effective learners rely on a *social* support network of peers, mentors, and mentees to help them learn and provide the context in which their knowledge is meaningful and useful.

---

<sup>1</sup>All were over 4.2 (of 5), with a median across courses of 4.71 for “interesting manner” and 4.69 for “stimulated to think”. During 2007/8, new evaluations were phased in; my ratings on the new “inspired interest” item were 4.2 or above with a median of 4.55.

Section 2.2 below highlights the techniques I use to encourage and support students as they become active, reflective, and social learners.

It is also crucial for teachers to be reflective in their teaching. Reflective teachers investigate their own teaching looking for strengths and room for improvement. Reflective teachers also recognize that no static strategy will suit every student and class. Finally, they adjust to the changing culture and technology of teaching.

Section 2.3 details some of the ways I reflect on and improve my teaching.

Finally, beneath the lofty heights of philosophy, the teaching enterprise is an immense logistical challenge. Effective teaching also means recognizing and managing these challenges efficiently in ways that are practical and respectful to all involved.

Section 2.4 relates several of my strategies for making effective teaching efficient.

## 2.1 Motivated Learners

In motivating my students, I take three primary strategies:

- Embed the ideas in a course into a coherent, compelling, and comprehensible story.
- Connect CS concepts with research, puzzles, and students’ everyday experiences and interests.
- Help students use CS to accomplish useful tasks.

**Coherent Story:** Appendix M<sup>2</sup> is an example of how I draw a course together into a coherent story. It’s a first-day survey that I handed out in my CPSC 344 and 444 user interface design courses. I often use an early survey to learn about students, but this survey also exercises design concepts and analysis tools from throughout the course.

Question 3 is one example. Its prompt “Sketch a self-portrait” sits above an empty box. So far *all* of my students have drawn their portraits inside the box without any instruction to do so. This very simple example of communicating design intent naturally and without explicit documentation foreshadows techniques we explore throughout the course (i.e., “affordances”).

The survey includes more than a dozen other links to ideas and topics throughout the course. We spend a full class session filling out, discussing, and analyzing the survey. Later in the term, the survey is our reference point to launch new topics.

I establish such a coherent story early in each of my courses and refer back to it frequently. I compared computer architecture to plumbing, carefully defined CS as the study of “clear thought” for an introductory discrete mathematics course, introduced programming with a musical composition theme, used a card trick to thread together concepts in a data structures and intermediate discrete mathematics course (Appendix K), and demonstrated the diversity of connections between CS and other fields to a CS0 course by springboarding off of Al Gore’s visit to UBC (Appendix E). In each case, the story was a recurring motif throughout the course that tied concepts together.

---

<sup>2</sup>Appendices are clustered by course and ordered by course level (1xx, 2xx, 3xx, 4xx); so, they may not be referenced in alphabetical order in the text.

**Connecting CS:** Appendix K starts with a simple card trick: “Shuffle a deck of cards and deal five of them out to [Moe]. Moe sets one of the cards aside, puts the remaining four into a neat pile, and hands them off to [Maggie]. After investigating them for a few seconds, Maggie announces the identity of Moe’s hidden card. . .” It then probes the mechanics of the trick, its validity, and the limits to extending it farther and farther, all using a set of discrete mathematics tools.

This exposition connects CS concepts to an intricate and ever-expanding puzzle in a way that is compelling to some students. However, *performing* the trick in class drew even more students in as they tried to puzzle out the secret. (I gave groups of about four students each a nine-card deck and suggested they try a scaled-down version of the trick in which Moe sets aside one of only four cards.) This course also included examples tying to other puzzles, contemporary political issues, and research.

I connect CS concepts to puzzles, research, and students’ interests throughout my courses. Appendix I is one example of the mandatory written questions I answered from students each week in my introductory programming courses, which often tied together programming concepts with research, CS practice, and students’ personal interests. For example, Question 6 ties into the AI notions of white and black boxes and HCI concepts (at “In the CS field of human-computer interaction. . .”), Question 14 ties to the International Obfuscated C Code Contest (a favourite with students) and software engineering research (in the “P.S.”), and a student brought online gaming up in Question 16. Appendix E shows one set of the notes I posted after each day of a CS0 course. This particular notes page connects concepts of the day to Al Gore’s upcoming visit to campus, paleoclimatology research, Herman Melville literary analysis, and other topics. Appendix N is a slide set I designed for a new unit of a fourth-year interface design course on affect, which interleaves discussion of affective design and research results. For example, we discuss Clifford Nass’s surprising discovery that humans will exhibit a variety of surprising social responses to computers—responses the psychology literature has established for human-human interaction—such as revealing more personal information to a computer that first “reveals personal information” itself.

Examples such as these abound on a daily basis in my courses: philosophers’ proofs and counter-proofs about the existence of God and laser encoders for measurement equipment (tied to gray codes) in a discrete mathematics course, ski resort layout (tied to transitive relationships) and accounting fraud detection (as a programming project) in a functional programming language course, my own HCI and educational technology research in a user interface design course, etc.

When I advise student researchers, I also connect their work to their personal interests and experiences. One student under my supervision had a background in artificial intelligence and Chinese. We settled on a project for her using machine learning to design more effective Zhuyin (Taiwanese phonetic) cell-phone keypads, and she took third in the international ACM undergraduate student research competition with her work. Another student with an interest in film drew together film and CS faculty to pioneer a CS movie production course and competition (the vPortfolio project, hosted at [www.vportfolio.ca](http://www.vportfolio.ca)). A dedicated introductory CS TA worked with me to develop a platform for creative, open-ended introductory CS assignments and published at the premier CS education conference: “Poogole and the Unknown Answer Assignment: Open-Ended, Sharable CS1 Assignments”.

**Useful CS:** Appendix H is an introductory programming assignment I designed for the first four weeks of the class. In the assignment students used the `java.awt.Robot` class—which emulates mouse and keyboard input—to automate two simple tasks, drawing a star in a paint application and typing a URL into a browser. We also encourage students to experiment with automating their own work.

The assignment addressed key course learning goals but also showed an unusual way students could *do useful work* with programming. One student envisioned using the Robot to automate tasks in online games (Question 16 of Appendix I). I wrote a Robot program to post answers to “Weekly Reading Questions” (Appendix I) and discussed the code in class.

Other examples include code to automate tasks in Microsoft Office, modify the game Civilization III, and automatically check for the presence of and link to student home pages from a course homepage. In my CS0 course and in the ChicTech middle school outreach project I supervised as a member of the department’s Focus on Women in Computing Committee, students built service learning projects that used CS tools to support non-profit projects.

## 2.2 Active, Reflective, Social Learners

Effective learners engage actively with the material, reflect on their progress toward their goals, and work within a supportive social context. In this section, I illustrate some of the ways I support students in becoming active, reflective, and social learners.

**Active Learners:** The slides in Appendix C show my use of discussion questions, voting, and PRS clicker technology to encourage active learning in a CS0 course. For example, slide 4 asks “Is HTML a programming language?” Depending on context, I might treat the exercise differently, but a typical pattern is: students vote, discuss the result in groups, vote again, and discuss as a class. I also follow up on spontaneous student questions and, in programming classes like this one, use a programming environment to run experiments. My post-class notes from this lecture (Appendix D) document one exchange with a student that led to an instructive experiment: “[We discussed] variable play, including trying one ‘new experiment’ requested by a student (switching ‘ $x = y$ ’ to ‘ $y = x$ ’) that showed off both a change in semantics and ‘documentation drift’ as the displayed comments in the program fell out of date...”

In this course, I also switched from using PRS clickers for post hoc quizzes (which I and the students had found dissatisfying) to introducing material using clicker-based problems.

Slide 19 (“Predict: What Will the Code Do”) in Appendix C shows one example in which students predict the value of the expression  $num1 + num2$ , with  $num1 = 3$  and  $num2 = 4$ . The students made their predictions in small groups, with no programming background but a brief introduction of terminology. We then discussed the aggregated answers, noting that different models of program execution lead to different predicted results. In this case, students predicted either 7 or 3. A student with each prediction explained their model. Students predicting 3 saw  $num1 + num2$  as  $1 + 2$ . This introduced the idea that programming languages could be designed differently and that

different designs may be better “fits” to different people’s intuitive models. The notion that both responses are rational ways to read this code gave confidence to students encountering code for the first time; discussion of the 7 response reinforced an accurate model of execution in this programming language.

I used similar exercises to introduce loops and conditionals, to explore network routing algorithms, to probe human perception of interface designs, and so forth.

I use techniques like these in every lecture I give. Appendix N shows some other examples (with exercises on slides 1, 3, 19, 22, 23, and 30). As discussed in Section 2.1 above, Appendix M was the centerpiece of an hour-long active learning exercise.

Students also need to actively engage with the material outside of the classroom in assignments and textbook study.

CS assignments to design, build, modify, test, or evaluate systems naturally exercise classroom concepts. As mentioned in Section 2.1 and illustrated in the assignment in Appendix H, I strive to make these hands-on assignments as interesting and relevant as possible for students.

I use two strategies to encourage students to *interrogate* their texts, treating their own questions, doubts, and ideas as at least as important, and worthy of record, as the author’s words. First, I model “interrogation” by frequently referencing the questions, critiques, and other comments I scribble in my books’ margins. Second, I assign “Weekly Reading Questions”. Every week, each student must submit a question of their own based on textbook reading. I hope that this practice reinforces the habit of interrogating the text and, thus, the habit of respecting and reflecting upon their own learning process. I randomly select 10 questions to answer on the newsgroup, sometimes adding more that are widely requested or of particular interest. Appendix I shows one week’s selected questions and their answers.

**Reflective Learners:** The key scaffold an instructor can provide for reflective learning is a clear statement of what students are meant to learn. In collaboration with the Carl Wieman Science Education Initiative (CWSEI), I have taken the lead in the Faculty of Science in providing clearly articulated, measurable learning goals to students.

Appendix A is a report to the UBC CS Department on the process by which I and my colleagues Anne Condon and Holger Hoos transformed our initial vague and ambiguous learning goals for our CS0 course into clear and detailed goals. Research by colleagues at CWSEI shows that my students consciously use the goals to guide their focus in lecture and their study outside the classroom. Appendix B is a full list of learning goals from the same course.

I tie these learning goals to each element of the course, particularly lecture, TA sections, assignments, projects, and exams. Appendix L shows a practice final exam for an architecture course annotated with learning goals. The actual exam was also annotated, although those comments were displayed only to instructors and TAs. The project marking scheme for my CS0 course in Appendix F assigns one-third of the marks to learning goals developed by the students (“SATISFIES SPECIFIC GOALS OF PROPOSAL”) and myself (“SATISFIES GENERAL GOALS OF PROJECT”).

Slide 6 in Appendix C lays out the explicitly reflective process for learning programming that I taught to my CS0 students. Students “study a concept... [b]uild a



model of how it works... predict what will happen when [they use it]... [e]xperiment, by running the code... [and refine their] model to fit the results". We practiced this process in class through a series of PRS clicker exercises, and I gave students additional exercises and code samples for outside of class.

I also encourage students to think and write about their work building systems by including "Reflection Questions" on assignments and requiring proposals, reports, and often posters or in-class presentations on larger projects. The CS1 assignment in Appendix H includes three sets of "Reflection Questions", while the CS0 term project report for which Appendix F is a marking scheme was accompanied by a project proposal and an in-class presentation.

Many exercises from the previous section also encourage reflection. For example, the weekly reading questions lead to more reflective textbook use.

**Social Learners:** Effective learning is also embedded in a supportive social context. I use a variety of strategies to help students develop their social network.

In courses where I have discretion over the collaboration policy for assignments, I usually use the "Gilligan's Island" policy to encourage collaboration: students may work with anyone they like as much as they like provided they (1) cite collaborators, (2) eventually end collaboration and dispose of all records, and (3) do something mind-numbing (like watch "Gilligan's Island") between step (2) and actually composing their own, independent submission.

I also help students network to find collaborators. In-class group exercises introduce them to each other. Active newsgroups with about 10 posts per student per term (i.e., roughly 500 to 4000 posts) are an additional venue for interaction. In the first week of intro courses, I take the whole class on a "field trip" to the CS Learning Centre—a group work and help room staffed daily with TAs—so they'll be familiar with the physical spaces where they can connect.

I also work to help students connect with me. I memorize names using a screen-saver of rotating, labeled student pictures. I give a getting-to-know-you survey early in the course and frequently tell stories about myself and my family. I hold roughly two formal office hours per course per week, but I am also available before and after each class. I encourage students to contact me via e-mail, and I stress that I am free to talk anytime my door is open. I write hundreds of newsgroup posts per term (e.g., 172 and 179 in my two spring 2008 courses). I make regular "rounds" of the labs and learning centre where my students congregate, and I chat with them about their work and lives.

Beyond my own courses, I also improve the social fabric of the department. With four students' help, I led a grant that brought a total of ten of our students to their first academic conference and encouraged them to meet and network with mentors there. I act as a mentor with the department's Tri-Mentoring program. I make frequent appearances at Faculty and departmental events, often MCing: orientations, town halls, BBQs, high school outreach presentations, career panel sessions, etc. I also founded and host a weekly brown-bag lunch seminar—the "Eccentric Loup Munches", an anagram of "Computer Science Lunch"—in which current and former students (and other students, staff, and faculty) discuss CS topics varying from the application of category theory in functional languages to the CS subfields most often responsible for Armaged-

don in sci-fi movies.

## **2.3 Reflective Teaching**

Just as students learn CS material best if they reflect on their learning process, instructors improve by reflecting on their teaching. Appendix D shows a typical set of post-class notes (a “post mortem”) that I try to write up after each lecture session to reflect on successes and challenges.

I also use a variety of techniques to elicit feedback from the students to guide my reflection. Appendix A narrates a slice of my refinement of a course based on personal reflection, TA feedback from problem sessions, results on quizzes and exams, and formative in-class exercises. Many of my techniques have already been discussed above from the students’ perspective, such as: daily notes (Appendix E), reflection questions (Appendix H), weekly reading questions (Appendix I), and surveys (Appendix M).

Several of the publications and presentations listed in my CV, particularly the “Making Lemonade” paper and various “Kinesthetic Learning Activities” workshops, have begun from my post mortem notes. I have also been involved in more formal “action research” projects on my own and through the Institute on the Scholarship of Teaching and Learning and the Carl Wieman Science Education Initiative.

Finally, I keep current with literature on education through Carl Wieman Science Education Initiative reading seminars, UBC Skylight Science Suppers, and other seminars and through publications like SIGCSE and the Tomorrow’s Professor listserv.

## **2.4 Practical, Respectful, Efficient Teaching**

I have been late to only three of the roughly 600 class sessions I have taught and, besides scheduled absences, missed none. My reasoning is that, in a 60 person class, every minute I am late wastes one student-hour of time, and I deeply respect the hours of time my students give me.

This credo of respect and efficiency is what I use to guide implementation of my teaching philosophy within the logistical nightmare of courses with hundreds of students and dozens of TAs.

To show respect for my students’ time and effort and also to make their study (and my marking) more efficient, I provide a variety of study aids on prompt schedules. Each term, I guarantee to my students that my slides will be posted online by an agreed-upon time in advance of class or else I will print copies to hand out. I distribute learning goals (like those in Appendix A) with the lecture slides to guide students’ study, and I agree to restrict examinations to those goals. I design and, when possible, distribute in advance clear marking schemes for assessments, like the CS0 project marking scheme in Appendix F. I distribute practice exams and copies of old exams to students. Before students write the exam itself, I and my TAs dry-run and edit it. Finally, the teaching reflection exercises discussed in Section 2.3 above help me identify and address other problems as they develop.

My credo of practical, respectful, and efficient teaching also applies to the professionals I work with, particularly TAs. I designate a TA to polish and distribute lab

and tutorial material well in advance, saving the other TAs effort spent preparing redundant material yet giving them time to adapt the provided material to their needs. I prepare careful agendas for my weekly staff meetings with TAs. Appendix J shows a typical example from a course with thirteen TAs and two instructors. Staff meetings in that course were scheduled for 50 minutes and always ended on time or early. My post mortem notes (e.g., Appendix D) keep TAs apprised of developments in lecture. Similarly, my TAs write post mortems of their own for me and the other TAs.

Another element of respectful interaction with my TAs is to give them room to exercise their own philosophy and strengths as teachers. I make my goals for the course clear to TAs and, within those constraints, encourage them to adapt or develop new lab and tutorial material, take on projects to improve the course and students' learning, and share their insights with me and the other TAs. Given room to work, my TAs often perform amazing services (like the TA who tracked and intervened with all students missing marks in a large introductory course), develop valuable curricular material (like the "practice questions on pipeline modification" mentioned on the last page of Appendix L), and garner excellent reviews from students. Although they won on their own merits, I am proud that all three CS department winners of the UBC graduate student teaching award in the four years I have been here won while working with me.

Making my teaching practical also involves planning for teaching innovation to be sustainable. As part of this process, I prepare course-long "post mortem" packages that include my daily post mortem notes (like those in Appendix D), end-of-term notes about larger issues, and reusable teaching materials. Appendix G is the table of contents from one such course package.

Through these packages and various publications, I disseminate my curriculum development work, including overall design work on CPSC 101 and dozens each of lectures, exercises, assignments, and exams across the other courses I have taught. Details of my curriculum development work are in my CV (at the end of Section 8a, pages 5–7). Most of the artifacts in the appendix are examples of that work, all of which are entirely or primarily my own work.

## 3 Service

### 3.1 Department

I have served the department in three main roles: Communications Committee Chair, Focus on Women in Computer Science Committee member, and in various capacities associated with the Carl Wieman Science Education Initiative.

**Communications Committee:** After a brief stint as a member, I was Chair of the department's Communications Committee from the fall of 2005 through the summer of 2007. The committee's purpose is to "identify, evaluate, and develop communications-related opportunities both within the Department and external to the Department in an effort to facilitate improved dissemination of information, consistent messaging, and an enhanced reputation for the Department. . ."

The committee was only a few weeks old when I became Chair. As chair, I collaborated closely with committee members, including our Communications Coordinator. We began by generating a broad survey of stakeholder groups of interest, including lists of their representatives, needs, and resources. Based on this list, we embarked on a multi-year plan to focus on successive underserved stakeholder communities. During my tenure, we addressed first the alumni and then industry stakeholder communities.

Our improved alumni relations included vastly expanding our database of alumni contacts (from fewer than 200 to more than 4000), surveying the alumni for their preferred modes of interaction with the department, establishing new print and electronic lines of communication to alums, and beginning a series of formal and informal alumni events. Our efforts have led to strong connections with alumni and a culture of regular alumni participation in undergraduate and graduate events.

Our industry outreach included establishing an industry point-of-contact in the department and a website providing current information to industry about our programs and initiatives; a series of colloquia bringing together industry reps, faculty members, and students around research issues; and a series of consultations with industry members around key curricular issues.

Besides our focus on industry and alumni, the committee also oversaw various advertising programs, undergraduate community building events, the vPortfolio film competition, career panels for both undergraduate and graduate students, outreach sessions for high school students and teachers, and the establishment of smooth mechanisms for students to transition into their role as alumni.

As Chair, I also established a routine of carefully specified agendas distributed in advance of meetings and posted on our committee website.

**FoWCS Committee:** I was a member of the department's Focus on Women in CS Committee from the summer of 2005 through the summer of 2007. The committee has "responsibility for initiatives aimed at recruiting and retaining women in Computer Science, and for promoting a supportive environment for women in our department."

As a FoWCS member, I assisted with a variety of projects and led work on three: ChicTech, an Institute for the Scholarship of Teaching and Learning (ISoTL) project on recruitment and retention of women in first-year CS courses, and an alumni panel series (jointly sponsored by the Communications Committee). Our ChicTech project was based on previous work at Simon Fraser University and brought teams of grade 9 and 10 girls together with UBC mentors to develop websites for non-profits. The ISoTL research project used surveys and interviews to explore women's reasons for choosing whether to study and whether to persist in CS. In the alumni panel, a gender-balanced panel of alums share stories about the challenges and rewards of careers in CS and describe their very different career paths.

**Carl Wieman Science Education Initiative:** Along with Paul Carter, Kurt Eiselt, and David Lowe, I led our department's efforts to secure funding from the Carl Wieman Science Education Initiative. More recently, I am chair of the subcommittee that leads our \$1.5 million CS Science Education Initiative. My role has included administrative responsibilities—such as two rounds of recruiting and interviews, vetting of curriculum

development proposals for third and fourth year courses, and budget planning—as well as participating in the development of learning goals for our core CS courses. Under my guidance, the initiative has begun studying new teaching techniques in our introductory courses, upper-level AI courses, and database courses and has studied affective issues such as student perceptions of learning that cut across individual courses.

## **3.2 Faculty & University**

In the 2007-2008 academic year, I was a member of the Faculty of Science’s Killam Teaching Award Selection Committee. The committee is charged with selecting approximately four award winners from among the nominees (roughly a dozen that year) based on nomination letters, teaching evaluations, and classroom observations. As a committee member, I performed eight classroom observations across a variety of Science disciplines and helped select the winning faculty. A key part of this service to me was the opportunity to learn by observation from excellent teachers across disciplines, and I have begun developing a “teaching exchange” program with Skylight and the Carl Wieman Science Education Initiative that would lead other faculty to benefit from this type of observation.

My duties as chair of the CS Science Education Initiative (described above) also involve working with the Carl Wieman Science Education Initiative (CWSEI). Most Faculty-level duties involve reporting between the two bodies; however, I also beta-tested and provided extensive feedback on CWSEI’s new archiving system for course materials.

## **3.3 CS Community**

I have been active in the international CS education community, in particular as a member of two previous program committees and two upcoming program committees for the ACM Technical Symposium on Computer Science Education (SIGCSE), the premier conference on CS education.

In 2006, I and Lisa Kaczmarczyk coordinated the student volunteer program, which provides general factotum services to ensure smooth operation of the conference and is also a crucial networking opportunity for the volunteers. We solicited and scheduled more than sixty volunteers for a conference of over one thousand attendees. To promote networking, we secured funding from a corporate supporter so that we could afford T-shirts that readily identified volunteers and could host a party to introduce the volunteers to each other.

In 2008, I coordinated the SIGCSE workshops, which disseminate well-developed teaching practices and in the process offset conference costs to keep general registration affordable. This role involved selecting and scheduling a program of 38 workshops from 75 submissions based on over 350 reviews. I then coordinated with the workshop’s organizers to prepare the workshops section of the proceedings, provide any special assistance needed to make the workshops successful, and ensure that attendees had the advance resources they needed. At the conference, I managed a contingent of volunteers to ensure smooth operation of the workshops. I introduced per-workshop

mailing lists that improved communication between organizers and attendees, streamlined organizers' reimbursement process, and experimented with changes in our seating limits. SIGCSE 2008 saw a new record of over 600 workshop registrations.

I have already begun work in my capacity as Program Chair for SIGCSE 2009 (with ultimate responsibility for all elements of the program but special focus on technical papers) and Symposium Chair for SIGCSE 2010 (with overall responsibility for the conference as a whole), both in partnership with Gary Lewandowski. For example, as Program Chairs for 2009, Gary and I have recently matched over 500 reviewers with the 302 paper submissions. I expect to use these roles to continue expanding and improving the quality of resources available to the CS education community.

Besides these roles, I have attended, reviewed for, or presented at various CS education venues including the Northwest Regional Conference of the Consortium for Computing Sciences in Colleges (CCSC-NW), Frontiers in Education (FIE), the International Computing Education Research workshop (ICER), the Journal on Educational Resources in Computing (JERIC), the Microsoft Research Faculty Summit, the ACM Technical Symposium on Computer Science Education (SIGCSE), and the Western Canadian Conference on Computing Education (WCCCE). With my colleague Kurt Eiselt, I helped convince SIGCSE to accept their first ever bid from a non-US host site (Vancouver) and to come to the Pacific Northwest for the first time in over thirty years. With Diana Cukierman from Simon Fraser University and others, I am forging tighter ties between the local and international CS education communities by securing SIGCSE "in-cooperation" status for the WCCCE conference.

## 4 Summary

Four years ago, I was excited to join the already thriving culture of learning at UBC. During my time here, I have striven to establish and strengthen that culture of effective learning in my classes, in our departmental community, and in the CS education community as a whole. I believe the key to such a culture is to motivate students and provide them with the cognitive tools and environment they need to pursue their own learning. Through my research, my reflection on teaching, and my service, I continue to seek ways to accomplish that task better: to reach and motivate more students, to help them learn to learn effectively, and to connect students, faculty, and others to form a strong and supportive community around learning.

# Appendices

## A CPSC 101 Learning Goals Development Report

I prepared this report for the department as we began establishing learning goals for our core courses. It shows both what constitutes an effective learning goal and a viable process to develop those goals in the context of the CS0 course I co-developed with Anne Condon and Holger Hoos, CPSC 101. The complete set of learning goals is in Appendix B. The goals correspond with the lecture in Appendix C and the post mortem notes in Appendix D.

(Note: this is the complete document. The comment about “elided goals” was also in the original document.)

## Tracking Changing Learning Goals (Brief Version)

Anne and Steve are building learning goals for CPSC 101. Our efforts fall *between* the proposed level for core and non-core courses and focus at the *lecture* not *course* level. Here, we discuss development of JavaScript (JS) unit goals, illustrating some efforts and benefits of the process.

**15 months ago:** 101 students were to “learn JavaScript”, but detailed goals came only from the text, sample exam questions, and the remarks: “Learning JavaScript will introduce you to programming: the expression of algorithms in a computer executable language!” and “JavaScript will illustrate to you central programming concepts such as: the importance of structuring data and key ‘control constructs’.” We worked with two examples in lecture and then held an in-class problem session. Students fared poorly on problem sessions and exams, and we were disappointed with our goals’ vagueness and tenuous connection to exams.

**10 months ago:** We articulated our implicit learning goals for the JS unit, connecting to high-level course goals. We added new goals to our existing slides, such as “You will:

- appreciate the extra power to express processes of a programming language (vs. a markup language)
- understand a few key programming “control constructs”, particularly events and conditionals [2 more goals elided]
- be able to modify existing JavaScript code to suit your purposes”

Explicitly articulating goals clarified our own intentions, suggested small changes to the lectures, and eased “hand-off” to TAs and other instructors. Students and staff could also “attribute” quiz and exam questions to explicitly stated goals. But, these goals remain vague. (What does “understand a few key programming ‘control constructs’ ” mean?)

**Now:** We are again revising goals, with more changes to teaching materials. We previously underemphasized a key skill: reading and tracing existing code. We now explicitly teach this as the “study/model/predict/experiment/refine” process. New goals emphasize this process, e.g.:

- accurately model & predict the behaviour of variables in JS programs
- accurately model & predict the flow of control in a JS program through sequential execution
- accurately model (and so predict) the evaluation of any expression, no matter how complex, as long as you have a good model of the parts
- accurately model (and so predict) the flow of control in a JS program through a function call

Lectures pose small programming problems for students to solve, *before* detailed instruction. Students discuss solutions and the models implied by each solution, experiment, and refine their models. Assessment is surprisingly straightforward, e.g., we can assess “accurately model & predict the behaviour of variables in JS programs” with a problem like:

```
x = num1;      // Line 1
y = num2;      // Line 2
x = y;         // Line 3
z = x + y;     // Line 4
z = z + 1;     // Line 5
alert("z is: " + z);
```

Sketch the state of each of the variables after each line of code executes. What does the program output?

Formative in-class assessments suggest this approach is working well.



## B CPSC 101 2007W1 Topic-Level Learning Goals

This is a full list of the topic-level learning goals from CPSC 101 (CS0). Exams were designed only to assess topic-learning goals, and this fact was repeatedly communicated to students. Also given but not listed here were learning goals for the overall course (not assessed), labs (assessed only in labs) and the term project (assessed only in the project).

### **Introduction:**

- Anticipate the structure of CPSC 101 lectures and exams based on their learning goals
- Categorize a well-described device as a computer or non-computer with clear, articulate reference to (for one pass) your own explicit definition and (for another) our course's agreed definition
- Define the term algorithm
- Relate the components of an algorithm to a new domain

### **Digital Representation (*guest lecturer designed or helped design goals*):**

- Be able to define a data representation scheme
- Understand, and start working on, course learning goals on data representation
- Describe properties of data representation schemes that can be found in many contexts of the world around you
- Critique properties of data representation schemes, from the stand-point of usability and engineering considerations, given information about the context in which the scheme is used
- Engage in design of schemes, for example by proposing modifications that address shortcomings of given data representation schemes
- Put your knowledge to practical use, for example in making decisions about representing your own data or applying the knowledge in your own field

### **Human-Computer Interaction:**

- Explain how tools augment and constrain our power to think and act
- Define the “myth of human error”
- Use the concepts of negative transfer, familiarity/consistency, good mapping and metaphors, and useful feedback to explain strengths and weaknesses of simple interfaces
- Compare the strengths and weaknesses of graphical vs. command line user interfaces

### **Networks:**

- Explain why breaking a system into layers can make it easier to understand and build systems at each layer
- Define “IP address”, “domain name”, and “URL”

- Explain the parts and relationships among IP addresses, e-mail addresses, domain names, and URLs
- Trace the process of communication from one computer to another through the layers of the internet
- Justify key elements of a point-to-point routing protocol
- Justify each step of the party protocol for broadcast networks

#### **Web pages and Search Engines:**

- Briefly describe the parts of a web search engine
- Explain how a search engine finds and indexes web pages
- Predict how and whether a page will be found by a web crawler given the link structure around the page and which pages the crawler already knows of
- Construct HTML to present the structure and visual appearance you intend for an HTML web page

#### **Implementing Algorithms in Javascript, Part 1:**

- Apply the “study/model/predict/experiment/refine” technique to learn new programming concepts
- Connect the following terms to their use in programs: function, function declaration, function call, function name, function body, parameter, parameter list, variable, value, statement, expression, orthogonality, sequential execution
- Accurately model and predict the behavior of variables in javascript programs
- Accurately model and predict the flow of control in a javascript program through sequential execution
- Employ the concept of “orthogonality” to combine your knowledge of different programming elements, including being able to interpret their combined effect

#### **Implementing Algorithms in Javascript, Part 2:**

- Accurately model (and so predict) the evaluation of any expression, no matter how complex, as long as you have a good model of the parts
- Model the construction and interpretation of expressions using numbers, strings (text snippets in quotes), arithmetic operators, and function calls
- Use the <script> tag and events to add behaviors to your web pages (assuming you have a reference listing the available events and the behaviors you want are ones that you know how to implement in javascript)

#### **Implementing Algorithms in Javascript, Part 3:**

- Apply the “study, model, predict, experiment, refine” technique to learn about new elements of a programming language
- Accurately model (and so predict) the flow of control in a javascript program through a function call
- Accurately model (and so predict) the flow of control in a javascript program through conditionals (both if and if/else statements)

**Representing Images:**

- Explain how to represent an image as a grid of pixels (“raster graphics”)
- Define the term “pixel”
- Construct or recognize colours in the “hexadecimal” form used in HTML pages, given a calculator or chart to convert between hexadecimal and decimal numbers
- Explain how to represent an image as a list of drawing commands (“vector graphics”)
- Compare and contrast the suitability of raster and vector representations for different image representation needs

**Artists and Computers:**

- Identify the styles of computer art of Molnar, Noll, Truckenbrod, and Cohen (in the periods of their art that we discuss)
- Connect key advances in computing—including display technology, price and accessibility, and software—to changes in computer art

**Algorithms (*guest lecturer designed or helped design goals*):** <sup>3</sup>

- Characterize a problem for solution by an algorithm in terms of the inputs given, the outputs produced, and the key constraints on inputs, outputs and their relationship to each other
- Faithfully simulate an algorithm that requires only actions possible for humans (or identify parts requiring clarification along with multiple plausible interpretations of those parts)
- Propose multiple alternate algorithms (expressed in sufficient details for faithful simulation by skilled humans) to solve problems for which you already know multiple intuitive solution processes (such as searching)
- Identify the key tradeoffs among two or more algorithms that solve the same problem given: 1) the requirements to execute each algorithm and 2) performance information along any relevant dimension (such as difficulty of implementation or speed of execution) in cases where information is not clear from the algorithms’ descriptions or brief simulation of the algorithms
- Accurately predict and model the behavior of array objects in javascript programs
- Accurately model (and so predict) the flow of control in a javascript program through a while loop

**Algorithms in Art, Computer Graphics:**

- Define the computer graphics technique “raytracing”
- Model the creation of a 2-D image from a 3-D scene using raytracing using your knowledge of image representation and loops

---

<sup>3</sup>At this point, students chose whether to follow the Art or Biology “track”.

**Algorithms in Biology, DNA as Information** (*guest lecturer designed or helped design goals*):

- Work on learning goals pertaining to algorithms and data representation schemes, in the context of examples relating to DNA

**Algorithms in Art and Biology, the Traveling Salesperson Problem:**

- Define the traveling Salesperson Problem clearly as an algorithmic problem
- Outline the code for “random”, “greedy”, and “random restart” approaches to solving TSP

**Algorithms in Art, Traveling Salesperson Art:**

- Trace the step-by-step process of producing “etch-a-sketch” art both as a high-level algorithm and broken down as a series of algorithmic steps. (Given our definition of algorithm, this means thinking about inputs, outputs, constraints, and steps taken to produce this art.)

**Algorithms in Biology, the Fragment Assembly Problem** (*guest lecturer designed or helped design goals*):

- Adapt TDP algorithms, plus your ideas for FAP algorithms, for a javascript implementation
- We’ll look at two algorithms: random (like TSP random restart) and greedy
- We’ll start with random, which is simpler

**Bringing Together Algorithms in Art and Biology** (*guest lecturer designed or helped design goals*):<sup>4</sup>

- Explain how the algorithmic concepts that you learned in your chosen discipline (art or biology), especially the TSP, apply in the opposite discipline
- Clearly be able to frame a problem in either discipline (art or biology) as a TSP problem
- Be able to apply your knowledge of TSP to a new domain, by identifying and describing: 1) the kinds of data that you need, and how you will turn that information into the data that a TSP solver can use; 2) how you will take the output from the TSP solver and turn that into a useful solution

**Self-Similarity:**

- Define the term “self-similarity”
- Identify self-similar structures in examples from the world around you, algorithms, and other fields
- Recognize new examples of self-similar structures
- Simulate the progression of simple self-similar systems

---

<sup>4</sup>End of Art and Biology “tracks”.

**How Computers Work:**

- Describe how a program (as text) is represented digitally
- Describe how a program instructions flow through a computer's Central Processing Unit (CPU)
- Describe how simple circuit elements can be linked together to execute at least one instruction
- Explain several key concerns in designing text representations, and ASCII and Uni-code's response to those concerns
- State the parts of fetch/decode/execute cycle
- Simulate the operation of the CPU, given a diagram of its parts and a simple program

**Minds and Machines:**

- Make strong and articulate arguments for and against the intelligence of proposed systems (human, computer, combinations, or other) in your own words by appealing to the ideas laid out by Turing and Searle

**Robotics (*guest lecturer designed or helped design goals*):**

- Explain how technology can be used at home and in healthcare (give examples of some robots discussed in class)
- Distinguish between autonomous and semi-autonomous systems, along with the strengths and weaknesses of each
- Discuss the challenges in the field of assistive technology
- Discuss asimov's laws

**Music:**

- Explain how music can be represented using traditional "samples"
- Given the sampling rate, length, and bits per sample for a piece of audio, calculate the bits required to represent it
- Predict how sound quality and size of a piece of audio might change as we alter the number of bits per sample and the sampling rate used for it

**Diversity in Computer Science (*guest lecturer designed or helped design goals*):**

- Justify the need for diversity in the field of Computer Science with at least two difference, valid reasons
- Connect changes in the enrollment of women in Computer Science over the last 35 years to overall trends, trends in CS-related industries, and changes in the availability/accessibility of computers

## **C CPSC 101 JavaScript Introduction Lecture**

I developed these slides for CPSC 101 (CS0) in term 1 of 2007–2008 to introduce JavaScript to students by experimentation. The slides lay out the “study / model / predict / experiment / refine” process I encourage students to use to actively explore programming: study a concept, model how it works, predict the outcome of some interesting programs that use it, experiment to see what happens, and refine your model based on the result, iterating the whole process repeatedly.

These slides (and the process by which I developed them) correspond with the story of learning goal development in Appendix A. The “post mortem” notes in Appendix D document a class session using the slides.

To save space, I omitted the latter half of the slides (25–36), which introduce the programming language notion of “orthogonality” and how to use it to reconcile programming features with each other.

# Implementing Algorithms in JavaScript, Part I

CPSC 101 / WMST 201

1

## Learning Goals for Part I

After Part I of the “JavaScript” unit, you will be able to:

- apply the “study/model/predict/experiment/refine” technique to learn new programming concepts
- **connect the following terms to their use in programs:** function, function declaration, function call, function name, function body, parameter, parameter list, variable, value, statement, return statement, assignment statement, expression, orthogonality, sequential execution
- accurately model and predict the behaviour of variables in JavaScript programs
- accurately model and predict the flow of control in a JavaScript program through sequential execution
- employ the concept of “orthogonality” to combine your knowledge of different programming elements, including being able to interpret their combined effect

mostly  
book  
work!

2

## What is a Computer Program

Let's work from our agreed definition of a computer: “A device that receives **a list of instructions** (drawn from a well-defined set of possible instructions) and **interprets them to perform some process** in the world, such as physical activity or transformation of information.”

So a computer program is: “A list of instructions given to a computer which it interprets to perform some process.”

In other words, a program is an algorithm in a language a particular computer understands!

3

## What is HTML?

The HyperText Markup Language:  
a language for annotating a document with information about its properties and preferred display.

Is HTML a programming language?

4

## What is JavaScript?

A language for expressing processes to a web browser (or other JavaScript interpreter).

JavaScript can be embedded in HTML; so it's easy to use on web pages!

- JavaScript programs can go right into HTML in a `<script>` tag
- We can give behaviours to HTML elements (like links) by putting JavaScript snippets inside events (more on that later!)

JavaScript is a programming language; what computer runs it?  
Your computer runs the JavaScript it downloads.

5

## How to Learn a Programming Language

Study

Model/  
Refine

Predict

Experiment

- **Study** a concept
- Build a **model** of how it works in your head.
- Try using the concept, and **predict** what will happen when you run the code.
- **Experiment**, by running the code!
- **Refine** your model to fit the results... and repeat!

Computer Scientists have a  
“wet lab” on every desk!

6

## The “Boilerplate”

All of our experiments will be based on the following “boilerplate” or common code. Let’s start by very *roughly* understanding it.

```
function tryMe(num1, num2)
{
    return num1 + num2;
}

alert( tryMe(3, 4) );
```

7

## The “Boilerplate”

Parameter list (everything in the parentheses)

```
function tryMe(num1, num2)
{
    return num1 + num2;
}

alert( tryMe(3, 4) );
```

8

## The “Boilerplate”

Parameter

```
function tryMe(num1, num2)
{
    return num1 + num2;
}

alert( tryMe(3, 4) );
```

9

## The “Boilerplate”

Function name

```
function tryMe(num1, num2)
{
    return num1 + num2;
}

alert( tryMe(3, 4) );
```

10

## The “Boilerplate”

```
function tryMe(num1, num2)
{
    return num1 + num2;
}
```

Function body  
(everything in  
the curly  
braces)

```
alert( tryMe(3, 4) );
```

11

## The “Boilerplate”

```
function tryMe(num1, num2)
{
    return num1 + num2;
}
```

Return  
statement

```
alert( tryMe(3, 4) );
```

12



## The “Boilerplate”

```
function tryMe(num1, num2)
{
  return num1 + num2;
}
```

```
alert( tryMe(3, 4) );
```

Function call  
(to the alert function)

13

## The “Boilerplate”

```
function tryMe(num1, num2)
{
  return num1 + num2;
}
```

```
alert( tryMe(3, 4) );
```

Function call  
(to the tryMe function)

14

## The “Boilerplate”

```
function tryMe(num1, num2)
{
  return num1 + num2;
}
```

```
alert( tryMe(3, 4) );
```

Parameter values

15

## Reminder: Study/Model/Predict/Experiment/Refine

Study

Model/  
Refine

Predict

Experiment

16

## Study: Variables

- Variables store values called “declaring” variables
- We can create new variables by:
  - Using a statement starting with “var”
  - Putting them into a parameter list
- We can access a variable’s value just by using its name

17

## Model: What is a “Variable”

```
function tryMe(num1, num2)
{
  return num1 + num2;
}
```

```
alert( tryMe(3, 4) );
```

num1 and num2 are variables and store values.  
How do they work? Form a model!

18

## Predict:

### What Will the Code Do

```
function tryMe(num1, num2)
{
    return num1 + num2;
}

alert( tryMe(3, 4) );
```

19

## Experiment:

### What Does the Code Do

```
function tryMe(num1, num2)
{
    return num1 + num2;
}

alert( tryMe(3, 4) );
```

20

## Refine:

### Let's Build a Better Model

```
function tryMe(num1, num2)
{
    return num1 + num2;
}

alert( tryMe(3, 4) );
```

21

## Predict:

### Playing with Variables

```
function tryMe(num1, num2)
{
    var x, y, z;
    // what's in x, y, and z?
    // what should we look out for?!
    x = num1;
    y = num2;
    x = y;
    z = x + y;
    z = z + 1;
    return z;
}

alert( tryMe(3, 4) );
```

22

## Experiment:

### Playing with Variables

```
function tryMe(num1, num2)
{
    var x, y, z;
    alert("The value in x is: " + x);

    x = num1;
    alert("The value in x is: " + x);

    y = num2;
    alert("The value in x is: " + x);
    alert("The value in y is: " + y);

    x = y;
    alert("The value in x is: " + x);
    alert("The value in y is: " + y);

    z = x + y;
    alert("The value in x is: " + x);
    alert("The value in y is: " + y);
    alert("The value in z is: " + z);

    z = z + 1;
    alert("The value in x is: " + x);
    alert("The value in y is: " + y);
    alert("The value in z is: " + z);

    return z;
}

alert( tryMe(3, 4) );
```

23

## Refine:

### Playing with Variables

```
var x, y, z;
// what's in x, y, and z?
// what should we look out for?!
x = num1;
y = num2;
x = y;
z = x + y;
z = z + 1;
```

What do the lines starting with // mean?  
What happens when we assign to a variable?  
How does the computer work with a sequence of statements?

24

## What's Next?

We've started programming and we have some tools that will help us learn much *more* about programming. (The most important of these is "study, model, predict, experiment, refine"!)

Let's learn more about how a statement is built and how JavaScript fits into HTML. Then, we'll learn more about control flow.

37

## TO DO

- VERY important: Read over your third lab and do the "before the lab" section **before** you attend
  - This lab will be even more difficult and longer than the HTML lab! So, *do the prep!*
- Complete the upcoming reading
- Get ready for Quiz 1

38

## D CPSC 101 “Post Mortem” Lecture Notes

These notes from the 2007 September 24 lecture of CPSC 101 (CS0) are representative of the notes I try to write after each class session and similar to (but more extensive than) what I ask my TAs to write after their lab or tutorial sections. The lecture they document uses the slides from Appendix C.

Date: Mon, 24 Sep 2007 14:22:57 -0700 (PDT)  
From: Steve Wolfman <wolf@cs.ubc.ca>  
To: [Omitted: TAs]  
Cc: [Omitted: Guest Lecturer], Steve Wolfman <wolf@cs.ubc.ca>  
Subject: 101 lecture post-mortem: 2007/09/24

I had a bit too much excitement in today's class: the copier crapped out on me this morning; so, Mary Anne and Andrew got the copies of the quiz together for me and delivered them in the nick of time to the classroom.

We therefore dove straight into lecture and did the quiz at the end, which turned out VERY well. Surprisingly, the class was extremely engaged and interested, rather than distracted by the quiz as I'd expected. We also had the highest clicker participation (90 students) that we'd ever received.

In the end, we went through the remaining slides from the FIRST set of JS notes at a reasonable pace, spending plenty of time on discussions about the variable play, including trying one "new experiment" requested by a student (switching " $x = y$ " to " $y = x$ ") that showed off both a change in semantics and "documentation drift" as the displayed comments in the program fell out of date (the alert box still said "the value after ' $x = y$ '").

By the last statement of the program, essentially everyone in class had a clear grasp of the semantics of variable assignment, at least judging by excellent results on " $z = z + 1$ ". Hopefully, they'll hold onto that for the lab and beyond :)

Orthogonality discussion went fine and at a reasonable (though not leisurely) pace.

I did NOT get a chance to talk about expressions, the `<script>` tag, or events. MANY apologies to the Monday and Tuesday labs for that oversight! Hopefully, the students will be able to get along without them.

Students finished the quiz in plenty of time, with only about a dozen people still working when time was called.

Cheers,

Steve

## **E CPSC 101 Notes Posted for Lecture Day One**

I wrote and posted these notes as a followup to the first day of my summer 2006 CPSC 101 (CS0) class. I posted similar notes for many sessions of my CPSC 101 and 121 courses, tying concepts and interesting discussions of the day to politics, research, related disciplines, and other topics. These notes also demonstrated HTML techniques to students.

# CPSC 101 Notes: 2006 May 08

**Disclaimer: you will not be tested on these extra notes on the exam. When there is crucial content, I will preface it with a note saying "Key info"**

Along with discussing the slides themselves, our conversation also ranged across a variety of supplemental websites. I'll include links to and descriptions of those sites here. This is *also* a good web page to use as an example during your second lab, since I wrote the page by hand and it uses relatively straightforward HTML code. (Just for fun, I ran this through the [Watchfire website validator](#) and made some changes to improve the quality and accessibility of the site according to Watchfire, such as the "DOCTYPE" line at the start of the HTML code.)

However, before you worry about the links below, note that you will not be tested on these extra notes on the exam. (Look for that phrase in all your classes; it generally indicates the most exciting, valuable, and interesting material!)

## Interesting Links

- We started off talking about climate change. That story worked its way from [Al Gore's visit to UBC](#) to the [trailer for An Inconvenient Truth](#) at the [Apple trailer site](#) (a personal vice!).

From there, we used [Google](#) to [search for climate change](#) and learn more about the topic. (Notice that I linked "Google" to the Google website — Canadian version! — but I linked "search for climate change" to the search itself! Google uses a style of web-based form for its searches that tacks the information you enter in the form onto the end of the URL you use to access its search. Can you go to the climate change search link and then modify the URL directly to search for "real change" instead of "climate change"?)

A sponsored link at the top of the page led to the [National Academies Press's collection of climate change publications](#). The National Academies Press is the premier publisher of scientific reports in the US and publishes the results of inquiries requested by the US Congress and President.

The book on [Abrupt Climate Change](#) caught our attention, and we discussed several figures from the text: [Figure 1.2](#), summarizing the temperature in Greenland over the past 17000 years; [Figure 2.1](#), showing the variety of data used to understand historical climatic conditions; [Figure 2.2](#), showing blow-ups of the rapid climate changes at the boundaries of ice ages; and [Plate 1](#), showing the National Ice Core Laboratory of the US.

Use of computers and computer science permeates the entire excursion into the topic of climate change from Gore's talk (advertised to me via e-mail) to the computer-plotted climate data. Furthermore, like most sciences, paleoclimatology (what a nifty word!) has been profoundly changed by progress in computing, and fluency with information technology is a critical talent for a modern paleoclimatologist (excuse the [oxymoron](#)).

- We also discussed how computing has changed Herman Melville literary criticism (because of the scarcity of primary sources from Melville). The [Chronicle of Higher Education](#) has an article on [Melville and computing](#), which is sadly password-protected. (You can decide for yourself whether "bugmenot" is an ethical solution. I won't link to it; try Googling for it!) My wife and I **do** subscribe to

the Chronicle, however. So, I looked up the two websites they discuss: UVA's [multiresolution version of Typee](#) and Boise State's [Melville Marginalia](#).

- We saw quotes from several prominent figures, including:
  - [Vera Molnar](#), a computer artist
  - [Frank Gehry](#), an architect that builds distinctively bizarre buildings, including the [Experience Music Project](#) in Seattle and the Weisman Art Museum, pictured on the right. (I included a picture of the Weisman and not the EMP because I found a copyright-free picture of the latter on [Wikipedia](#). The photographer, Mulad, released the work.) You can learn more about Gehry and how computers revolutionized his work in the recently released [Gehry documentary](#).
  - [Nuala O'Faolain](#), an Irish author whose name I murdered in class :(, and who describes the social forces that encourage women and some groups of men to avoid technology.
  - [Don Norman](#), a prominent user interface expert whose [gallery of good designs](#) includes my measuring cup and ice scraper (the latter, at least, is not a coincidence)
- We also saw two examples of origami folding languages: the [Origami Instruction Language](#) and the [Origami Fold Language](#). Plus, we saw an [origami wild boar](#)! :)
- Finally, when talking about copyright law, we discussed the [Internet Archive](#), which stores billions of old web pages without explicit permission of owners (though they *do* allow you to "opt out" of their database, in much the same way that search engines allow you to opt out of their databases!). The Internet Archive has gotten in some trouble for storing and allowing access to other people's data. Check out this fascinating [article on a lawsuit against the Internet Archive](#)!



We also talked about [Project Gutenberg](#), which posts electronic versions of books that are explicitly no longer under copyright. I *love* their Mark Twain collection!

We didn't talk the following three fascinating sites related to online copyright law: [Creative Commons](#), which publishes a simple way to make clear how people *can* use works you create (avoiding the often insidious rules of copyright law!); the [GNU project](#), dedicated to free software and creator of the (in)famous [Gnu Public License](#), which allows free use and can "infect" projects that use GPL'd material; and [SourceForge](#), a source of open source software (software whose code is available for anyone to see and modify). The Creative Commons, especially, is something that you might want to investigate if you're considering putting your art and writings on the web but don't want them (simply) copyrighted.

---

[Steve Wolfman](#)

2006/05/09

## **F CPSC 101 Project Report Marking Scheme**

This is the marking scheme for the CPSC 101 course project in term 1 of 2007–2008. CPSC 101 is a CS0 course focused on connections between CS and other disciplines. Students create a project that applies or connects CS to a subject of interest to them. They work in teams of 2-5 over approximately five weeks of a thirteen week term. Their first milestone is an early project proposal. After the proposal is approved (usually edited collaboratively with their TA), it acts as a contract between the team and the TA shepherding their project. At the end of the term, teams submit their report and make an in-class presentation about their project.

I developed this scheme over several offerings of 101. Feedback from TAs was critical, particularly feedback after actual marking. In later terms, the scheme was released to students at the same time the project was assigned so that they could anticipate the constraints on the project. The scheme is designed to be filled in by a TA and sent to the project team members.

Student work on this project was often outstanding. One memorable project illustrated a forest sampling method by generating a random sized “forest” of four types of trees (different size and colour circles) with user-selectable distributions across the forest. It then overlaid a typical sampling pattern and reported the estimated population breakdowns the sample would yield. Their report (posted below the simulation) described both the mechanics of the simulation and the forestry technique it models.



## PROPOSED PROJECT MARKING SCHEME

Student Names and IDs:

Note: items marked PGM are "Per Group Member", meaning that different group members may get different marks on those items. However, in extreme cases, the markers may differentiate marks further among group members.

The project and report as a whole are worth 75 marks.

Overall mark:

General comments:

---

### PROJECT & REPORT ADMINISTRATIVE DETAILS [6 marks]:

You should have included each of the required report elements.

Mark breakdown:

- 1 mark: report linked from student's page (PGM)
- 1 mark: project is linked from report or student's page, if applicable (some projects may only make sense during the presentation, rather than on a website)
- 1 mark: proposal is linked from report
- 1 mark: report includes work breakdown among team members
- 2 marks for citations:
  - 2. report cites all appropriate sources of materials and ideas using a reasonable citation format (or, rarely, requires no citations)
  - 1. report cites at least some sources
  - 0. report fails to cite any sources

Mark for this section:

Comments:

---

### REPORT ORGANIZATION/STRUCTURE [6 marks]:

Your report's structure should make it easy to read and also easy to search quickly for key information.

- 6. Organization of report is extremely clear and professional. Key information is easy to locate quickly. Formatting and typesetting make organisation clear and aesthetically pleasing. Note: this does NOT require fancy but rather effective formatting and typesetting.
- 5. Organization of report is clear. Key information is easy to locate quickly. Formatting and typesetting supports organisation.
- 4. Report structure is readable. After a complete pass through, key information can be teased out. Formatting and typesetting do not hinder organisation substantially.
- 2. Report structure is difficult to read. Finding key information is a challenge, possibly requiring multiple reads and manual collection of information. Formatting and typesetting may hinder

organisation.

0. Report does not coherently organise key information, or formatting and typesetting makes organisation impossible to discern or understand without extraordinary measures, such as reading HTML source.

Mark for this section:  
Comments:

-----  
REPORT CLARITY/USE OF LANGUAGE, FIGURES, AND TABLES [6 marks]:

Your report's language should communicate your ideas clearly and succinctly without interfering with their expression.

6. Language is crisp, clear, and professional. English is flawless. Word choice, figures, and tables consistently support the document's goals.
5. Language is clear. English is good. Word choice, figures, and tables generally support the document's goals.
3. Language is sometimes difficult to understand due to poor English or word choice, or figures and tables may be difficult to understand or lacking at key points.
1. Language is a severe impediment to understanding the document.
0. The document is unreadable due to poor English.

Mark for this section:  
Comments:

-----  
ORIGINAL CONTRIBUTION IN PROJECT [6 marks]:

Your project should represent your own creation, such as an artifact (like a work of art) or a synthesis of ideas (like a report on an area or a tutorial).

6. Project (supplemented by the report) represents an interesting, compelling, and substantial new contribution that clearly goes well beyond material and discussions from class.
5. Project (supplemented by the report) represents an interesting new contribution.
3. Project (supplemented by the report) includes at least some original ideas or work.
0. Project and report include no original contribution from the team members.

Mark for this section:  
Comments:

-----  
SATISFIES SPECIFIC GOALS OF PROPOSAL (PGM) [12 marks]:

You and your TA agreed to a "contract" of what you would accomplish for the proposal.

At the TA's discretion, group members may be marked on individual commitments to and progress on the project rather than the group's.

- 12. Project meets and substantially exceeds the minimal goals agreed on.
- 10. Project meets the minimal goals agreed on.
- 6. Project almost meets the minimal goals agreed on.
- 2. Project made progress toward the minimal goals agreed on (from the proposal).
- 0. Project bears little relation to the minimal goals agreed on.

Mark for this section:

Comments:

-----  
SATISFIES GENERAL GOALS OF PROJECT (PGM) [12 marks]:

The three goals of the project were: (1) to deepen your understanding of some aspect of Computer Science or its connections to other disciplines, (2) to hone your skills in HTML, JavaScript, graphical software, or other technical tools, and (3) to explore a topic of interest to you.

Goal 3 can be marked per group member at the TA's discretion. In extreme cases, goals 1 and 2 may also be marked per group member at the TA's discretion.

Goal 1 [5 marks]:

- 5. Project and report illustrate interesting and important insights into CS or its connections to other disciplines, which go well beyond the material covered in the course.
- 4. Project and report illustrate interesting and important insights into CS or its connections to other disciplines, beyond just the material covered in the course.
- 3. Project and report illustrate some insights into CS or its connections to other disciplines, although these may just reiterate existing lessons from the course.
- 1. Project and report relate to CS or its connections to other disciplines.
- 0. Project and report bear little relation to CS or its connections to other disciplines.

Goal 2 [5 marks]:

5. Project and report demanded substantial new technical skills for their completion and clearly illustrate the technical skills learned.
4. Project and report demanded incremental improvements in technical skills or extensive use of existing skills learned in the course for their completion and clearly illustrate the technical skills practised.
3. Project and report noticeably demanded incremental improvements in technical skills or extensive use of existing skills learned in the course for their completion, but do not make clear what those technical skills are.
1. Project and report included some technical component beyond bare-bones HTML.
0. Project and report included no substantial technical content.

Goal 3 [2 marks]:

2. Project (and report) show evidence of team's interest in a plausibly interesting topic.
0. Topic is not plausibly interesting or project and report show team's disdain for the topic.

Mark for this section:

Comments:

-----  
INDIVIDUAL CONTRIBUTION AND TEAM ORGANISATION [12 marks]:

Each individual on the team should have contributed substantially to the project. Also, the team as a whole should have operated smoothly as a collaborative unit.

Note that team organisation will NOT be marked per group member but will be the same mark for all team members.

Team organisation [5 marks]:

5. Each group member contributed substantially, the project scope is appropriate to the size of the team, and the project and report feel like a coherent, unified contribution (rather than disjoint slices contributed by separate members).
4. The project scope is appropriate to the size of the team, and the project and report feel reasonably coherent.
3. The project scope is appropriate to the size of the team, and the individual members' contributions fit together in the report, although the report may not be a coherent unit.
1. The project scope is too small for the size of the team, or the

individual members' contributions fit poorly together.

0. Team member contributions are missing altogether or there is little connection among members' contributions.

Individual contribution (PGM) [7 marks]:

7. The individual's documented contribution well exceeds the expectations set in the proposal.
6. The individual's documented contribution meets the expectations set in the proposal.
4. The individual's documented contributions are significant but fall short of the expectations set in the proposal.
2. The individual's documented contributions are minimal, or individual contributions are not sufficiently documented to understand (and yet the project is significant enough that the marker expects the individual contributed at least minimally).
0. The individual did not contribute to the project according to available evidence.

Mark for this section:

Comments:

-----  
PEER EVALUATION [15 marks]:

Each team member is required to briefly answer the following question for each person on the team (including themselves). You should e-mail your responses to the TA that is shepherding your project. Be sure to include your name and student ID and, for each rating, the name and student ID of the team member you are rating. Your mark and your team members' marks will be calculated according to the evaluation you submit. Late evaluations will NOT be accepted!

Question: Did this team member's contribution to the project fulfill the team's expectations as expressed in the project proposal?

4. The team member fulfilled and exceeded the team's expectations.
3. The team member fulfilled the team's expectations.
2. The team member fell somewhat short of the team's expectations.
1. The team member fell substantially short of the team's expectations, possibly including impeding other team members' work.
0. The team member's lack of contribution or behaviour completely blocked the entire team's progress on the project.

Your answer should also include a brief description of why you chose each rating. Ratings of 0, 1, 2, or 4 should be accompanied by more substantial explanations than a rating of 3. ALL RATINGS ARE CONFIDENTIAL and will be seen only by the CPSC 101/WMST 201

instructional staff, NOT by your team members.

Marking breaks down as:

Team completion of evaluations [5 marks]:

5. All team members completed evaluations on time and fully.
4. All team members completed evaluations on time.
3. At least one team member's evaluation was late or missing.
2. More than one team member's evaluation was late or missing.
0. For groups of 3 or more only: no evaluations were provided.

Team member's peer evaluation (PGM) [10 marks]:

Note: any team member that fails to turn in a complete set of evaluations (one for EVERY member of the team including themselves with ratings and explanations for ratings) on time will receive a 0 on this section.

10. Team member was consistently rated as exceeding expectations, descriptions back these ratings up, and the TA agrees that the member's effort exceeded expectations.
8. Team member was consistently rated as at least meeting expectations, and the TA agrees that the member's effort met expectations, OR the team member is not consistently rated as at least meeting expectations but did meet them in the TA's judgment. (Note that a team member cannot exceed this rating unless all members rated that person.)
6. Team member was rated as falling short of expectations by at least one team member and did not meet expectations in the TA's judgment.
3. Team member was consistently rated as falling short of expectations, descriptions back these ratings up, and the TA agrees that the member's effort fell short of expectations.
0. Team member was consistently rated as blocking progress on the project, descriptions back these ratings up, and the TA agrees that the member's behaviour seriously impeded project progress.

Mark for this section:

Comments:

## G CPSC 101 Course Archive Table of Contents

This is the contents listing from the package of materials I handed off after teaching CPSC 101 (CS0) in the summer of 2006. To make my curriculum development sustainable, I try to hand off clean and comprehensive archives of course material to instructors who succeed me on a course, including my post mortem notes from individual lectures (like those in Appendix D).

2006S1 CPSC 101 Post Mortem ReadMe File

This package contains materials summarising the 2006S1 offering of CPSC 101. In the root directory, you'll find post mortem notes by the instructor (Steve Wolfman) and the TAs. In particular, the overall post mortem summarises the class's strengths and weaknesses. The lecture post mortems are individual reviews of the lectures written just after (almost) every lecture. The lab post mortems similarly summarise labs from the TAs' perspectives. Similar material (that was distributed to the students) is in extra-materials/lecture-notes.

Directory-by-directory listing:

exams:

copies of the midterm and final exam, supporting materials for the exams, solutions and marking schemes (where available), and review questions (where available; specifically for the final exam).

extra-materials:

random extra goodies; my bookmarks for 101; Jake Wires's page on checking UBC mail and newsgroups; random materials from some lectures (perhaps should go with the lectures folder!); the publicly posted post-class notes from each lecture; a macro to help build the list of links to student web pages in emacs

labs:

extra lab material BEYOND what is available on the web; for now, only Ori Livneh's rewriting of the Ultimate Paint lab to use GIMP

lectures:

slides and supporting materials for lectures; note that key-files-101-cs-music-lectures.zip contains general notes on how to use the files within the zip file; sand-rocks-water.ppt was never presented in odp format

project:

marking materials for the project and sample marked projects

quizzes:

copies of the three quizzes, supporting materials, solutions and marking schemes (where available)

## **H CPSC 111 Robot Assignment**

This is the first of three programming assignments given to my CPSC 111 (CS1) class in term 1 of 2006–2007. The assignment asks students to use the `java.awt.Robot` class to automate two simple tasks on their computer and prepares them to automate more complex tasks in the future. The assignment writeup also requires a series of “Reflection Questions” that I introduced to encourage students to be reflective learners.

To save space, I have included only the portion of the assignment I designed (parts two and three) and omitted the submission details.



# CPSC 111 2006/7 Winter Term 1, Assignment 1: Taxes and Robots

---

Contents: [Details](#) | [Part 1](#) | [Part 2](#) | [Part 3](#) | [Deliverables](#)

---

## Details

**Out:** Friday, 2006-September-15

**Due:** Friday, 2006-September-29, 4:00pm, both hard copy and electronic handin

**Value:** 4% of final grade (subject to instructor discretion)

### Objectives:

- to debug a program that contains syntax and logic errors
- to write complete programs that use variables, operators, and method calls to accomplish an interesting goal (controlling another application program)
- to use good programming style
- to learn how to use an API
- to devise a set of tests to ensure that the specified problem has been solved correctly
- to reflect on what you've learned (with satisfaction!)

**Note:** Each assignment (and usually each part of each assignment) will include reflection questions. You should include your answers to the reflection questions in a **separate** text file from your code named **README.txt**. Please do not submit your questions in any format other than plain text. (Note that neither Microsoft Word nor PDF is plain text! Whatever editor you use to write your source code should be usable to write a plain text **README.txt** file.) You should answer the reflection questions for all parts of an assignment in the same **README.txt** file.

---

## Part 1: Debugging [Bananania Tax Computation](#) Code

OMITTED

---

## Part 2: Controlling Your Computer with the Java Robot Class

Java includes a class called `java.awt.Robot` that allows a Java program to take control of your computer's input devices, like the mouse and keyboard. This means you can write Java programs that run other applications, or that pretend to be you to your computer! The main practical purpose of the `Robot` class is to allow automated testing of programs, but you could also use it to automate all sorts of tasks — like navigating an annoying series of menus in Microsoft Word or selecting a complex list of options quickly in your favorite game. (If you really want to do this, ask in the WebCT bulletin board, and Steve will tell you how to run a Java program from a single keypress in Windows!)

### Specification:

We are providing a really simple Java drawing program [SimpleDraw.class](#), which you can download and play with. It's a class file, so you can download it, and then run it as usual using the `java` command. (If you're curious, you can look at the [Java source code](#) for this program, but it's not needed for this assignment.)

In this part of the assignment, you will complete a Java program, [RobotStar.java](#), that uses the Java Robot class to draw a simple five-pointed star in the drawing program's window. You should have the drawing program running, with the window starting from the top-left of your screen and covering a large part of your screen. Then, you run your RobotStar Java program. The program will ask the user for the location on the screen for the top point of the star, and how tall and wide to make the star. The program will then move and click the mouse in an attempt to draw the star (so hopefully it'll be on the drawing program's window).

The drawing program draws line segments when you click and drag the mouse, so your program will need to compute locations for the five points of the star and then click and drag the mouse to those points.

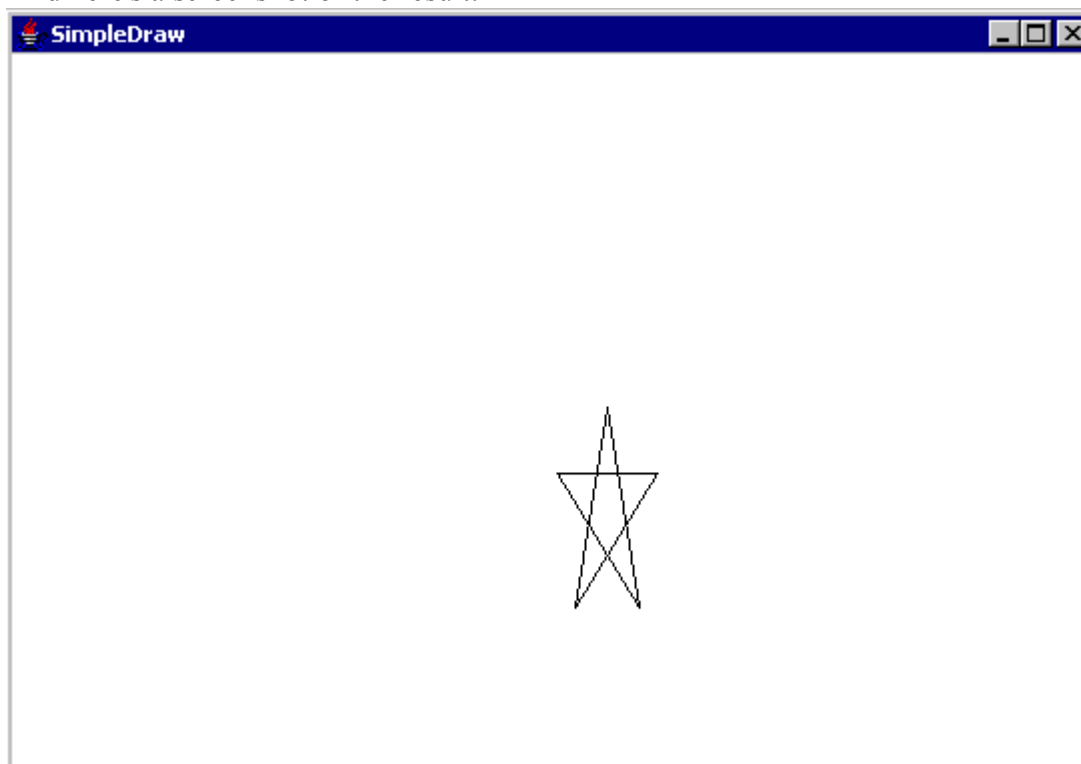
**WARNING:** The Java Robot really does take over your mouse and keyboard, exactly as if you were operating your computer! This means that if you're unlucky, your program might cause you to delete files, or send an email, or anything else your computer can do. It's like letting a baby play with your mouse and keyboard. When you start writing and debugging your program, don't have your program do any mousePress or keyPress events until you're confident you've got the cursor moving to the correct places on the screen.

Here's an example session with the program:

```
Enter x coordinate of top of star: 300
Enter y coordinate of top of star: 200
Enter width of star: 50
Enter height of star: 100
```

(Note that the user's input is in **green** for clarity. You do **not** have to make the user's input green in your program!)

And here's a screenshot of the result:



The star you draw doesn't have to be perfect, but it must have the height and width (plus or minus one or two is OK, since you'll be doing arithmetic with integers) that the user asked for, and it must have all five points.

For the sanity of the markers, please make all your prompts and output look as similar as possible to the example above.

## Reflection Questions:

Be sure to answer these reflection questions about Part 2 after you finish the part. There are no right answers to these questions, and we don't expect more than a couple of sentences for each question. However, we will grade on your effort to answer thoughtfully.

1. Did you end up implementing and testing the assignment piece-by-piece or all-at-once? (You won't be penalized for either answer!) In either case, what were the biggest frustrations you encountered with your approach? Which approach (or combination) do you think you'll use next time?
2. What's one task you perform on the computer that would be handy to automate with a `java.awt.Robot` program? Would it be difficult to write? Why or why not?
3. **OPTIONAL:** What was especially interesting or frustrating about this part of the assignment? What could we have done to make it better for you?

## Hints:

Do not try to write the whole program in one go. Instead, try writing and testing it bit by bit. For example, you might follow these steps, ensuring at each step that you have a working program:

1. Start with the empty program shell we provided, [RobotStar.java](#). Read it. (There are additional hints in the comments in the code.) Then, make sure you can compile and run it.
2. Add the code to prompt the user for input and read in the user's responses. You will probably find the Java object [Scanner](#) very useful. Make sure you are familiar with its method `nextInt`. **Test your code** by printing or logging the values you read in.
3. Work out on paper the math for computing the locations of the five points of the star. Note that Java has an (x,y) coordinate system for locations on the screen, like in math class. However, in Java the y coordinate increases as you go **down** the screen. So, (0,0) is the top-left corner of your screen, and (10,20) is ten pixels to the right and twenty pixels down. (A pixel is the smallest sized dot that your screen displays.)
4. Add the code to compute the locations of the five points of the star. **Test your code** by printing out the locations you computed.
5. Add the code to move the cursor to the points you've computed, in the pattern needed to draw the star. **Test your code!** You will want to familiarize yourself with the Java [Robot](#) class, especially the `mouseMove`, `mousePress`, and `mouseRelease` methods.
6. When you're sure you can correctly control the mouse movement, then you can add the `mousePress` and `mouseRelease` method calls to actually do the drawing. **Test your code!**
7. Test the full program to make sure that you've implemented the complete specification.
8. Make sure you're not redoing work unnecessarily. If you find yourself doing the same operation more than once consider creating temporary variables.
9. Make sure your code is fully commented, and then **test your code!** We're not kidding. It's easy to accidentally mess something up when you think all you're doing is adding comments. (True, comments have no effect on the function of your code, but what about that accidental extra character you typed into an identifier name without noticing it?)
10. Declare victory!

When you have difficulty making a piece work, write as small a program as possible that uses just that piece. That way, you can isolate the problems you're having in a small, simple piece of code.

---

## Part 3: Controlling A Real Application with the Java Robot Class

For the last part of the assignment, you're going to write a complete Java program `RobotTypeURL.java`, that

also uses the `java.awt.Robot` class. (Note, this previously said to name the file `JavaTypeURL.java`. If you named your file `JavaTypeURL.java`, that's fine!) This program should move the mouse to the URL (web address) entry box of a web browser of your choosing, and enter the address of your favorite website (or any other website you choose). You may position your web browser at a convenient location on the screen (like the top left corner), if that makes this easier. (No, your code will not necessarily work to type a URL on our computer's browser. Don't worry about that; just get your program to work on your computer. We'll most likely mark your code by reading it rather than running it. So, comment well!)

### Reflection Questions:

Be sure to answer these reflection questions about Part 3 after you finish the part. There are no right answers to these questions, and we don't expect more than a couple of sentences for each question. However, we will grade on your effort to answer thoughtfully.

1. How hard was this part, after you completed Part 2? Why?
2. Did you learn anything more in this part, beyond what you learned in Part 2? If so, what? If not, why not?
3. **OPTIONAL:** What was especially interesting or frustrating about this part of the assignment? What could we have done to make it better for you?

### Hints:

1. The `keyPress` and `keyRelease` methods of the [Robot](#) class are useful for this part of the assignment.
2. You may want to pick a short URL (web address) for your program to type. It's OK (in fact, it's expected) that you have this URL hard-wired into your program, i.e., you do **not** have to prompt the user for the URL.
3. It can be tricky to get enter the colon character `:`. You can do this by having the robot do a `keyPress` of a shift key, and then the semi-colon key. Or, if your browser lets you enter URLs without typing the `http://` part, you can avoid this altogether.

---

## Deliverables

OMITTED

---

## I CPSC 111 Weekly Reading Questions and Answers

In my term 1 2006–2007 CPSC 111 (CS1) course, students were required to submit a question about their reading on paper, in class, every week. The questions were marked on a binary scale, with only obviously bogus questions (i.e., “Is this worth a mark?”) getting a 0. Students who earned 1s roughly 75% or more of the time received full credit, worth the same as one of the three assignments.

I chose this week (the third set of questions due) to line up with the assignment included in Appendix H. Questions 4, 5, and 16 all refer to that assignment.

Each week, I would post about 10-20 questions and answers on the newsgroup. I used a Java program similar to the one the students wrote in the contemporaneous assignment to transfer the questions from my text editor to individual newsgroup postings.

To save space, I have omitted questions 1–3, 7–10, 12, and 15, but I have attempted to retain a representative sample.

=====QUESTION 4=====

Q: Is there a way that I could dynamically create an automated program that, for example, enters a URL that the user requests? (Say, the user types in "google", and I open a browser and enter "http://www.google.com/" into the address bar.)

A: Yes! I'll post my code that does just this under the "Random Goodies" icon in the Section 101 house on WebCT. It's called WeeklyReadingQuestionPoster.java.

Note that this code uses LOTS of stuff we haven't seen before. It's also REALLY long and repetitive. Guess what? I wrote a couple of little programs (simple text-editing programs called "macros", that you can write in many editors) to handle the repetitive parts for me. I got the base code from inside the source code for the KeyEvent class. That really long repetitive method is the method RobotWrapper.typeText(char). (In other words, the typeText method inside the RobotWrapper class that takes a single parameter of type char.)

=====QUESTION 5=====

Q: How can I input a : using keyPress and keyRelease?

A: You have to use the shift key. Imagine you were typing a colon (or just go and do it!), you press shift and then press and release the semicolon key. Then, you release the shift key.

Can you write Robot code that does a shift keyPress, then a semicolon keyPress, then a semicolon keyRelease, and finally a shift keyRelease?

P.S. The typeText method I mentioned in my previous answer does some clever stuff to make this happen. Of course, it also uses syntax that we haven't learned yet.. but don't let that scare you off! Just read it and get what you can, and don't sweat the parts you don't understand.

=====QUESTION 6=====

Q: Don't "black boxes" take away from the power of programming? If a computer scientist understood these so-called "black boxes", wouldn't they be able to do more when coding for programs, etc.?

A: The stock answer to this is: definitely not! After, all, didn't treating Robot as a black box let you do some pretty cool stuff in Assignment #1?

(Not sure what a black box is? Yes you are.. you treated Robot as one in Assignment #1! What does that mean? Well, you read a bit about what the methods in Robot do and then used them to accomplish your own tasks, but you never tried to or had to understand HOW those methods ACTUALLY work.

The term black box is best understood as being the opposite of a transparent box. Ever seen a computer or monitor with a transparent case? Some of the Macs are built that way: you can actually see the pieces inside and how they're put together. Conceptually, a transparent box (usually called "white box") is any piece of machinery (whether it's gears and levers or code) you work with where you not only know how to use the machine, you know how it works inside and out.

So, what's a black box? It's any machine you know how to use, but you don't know how it works inside.)

The more considered answer is definitely "it depends". Sometimes, it's INCREDIBLY useful to know how a machine works inside because it lets you get more out of the machine. (My mom's car once stalled and wouldn't start again on a busy road. The guy behind us turned out to be a professional racecar driver. He listened to the sounds the engine was making and then told her to pump the pedal several times and do some other magical mojo, and the car started right up. He could do that because the car was a white box to him.)

Other times, the immense amount of effort to understand how a machine works doesn't pay off.. because someone else could have done the same tasks as you WITHOUT understanding it.

In the CS field of human-computer interaction, we often talk about helping users of computer programs develop reasonable "mental models" of how the program works. Sometimes, you want to help the user build a much simplified or even inaccurate mental model (like a "fake" transparent box) because it's easier to express to the user and still leads them to the right conclusions.

You should understand that, too. Why? Well.. do reference variables REALLY store arrows in the computer's memory? If not, then why would I ever teach you that they do?

=====QUESTION 11=====

Q: Can I look at the source code for System.out.println?

A: Definitely! If you carefully follow the Eclipse setup steps in lab

#3, then you should be able to see the source code for `println`. Just do the following in Eclipse:

Write some (legal!) code that uses `println`.

Right-click on `println`.

Choose "Open Declaration" from the menu.

There's the source code! Depending on what type of argument you gave to `println`, you might get a different version of the `println` method, but it should say something like:

```
public void println(String x) {
    synchronized (this) {
        print(x);
        newLine();
    }
}
```

The "synchronized (this)" part is in case your program is actually doing several things at once. (For the person who asked this in a weekly reading question, yes, Java CAN do that. Post and ask me how if you'd like to know how.) Let's ignore it.

The next two lines just call other methods of the `PrintStream` class:

```
print(x);
```

is calling the `System.out.print` method. Notice that because of the IMPLICIT PARAMETER of the method, we don't have to say `System.out.print`. Instead, this method knows which object it has been called on automatically (THAT'S the implicit parameter: the object the method was called on, or `System.out` in this case!). It just calls `print` on that object.

(By the way, we wouldn't WANT to say `System.out.print` here. Why not? See if you can figure it out. Hint: do you think there can be other `PrintStream` objects BESIDES `System.out`?)

So, that line just prints out the text you passed in using `System.out.print`. The next line calls the `newLine` method of the `PrintStream` class. Try to find that in the Java API for `PrintStream`: <http://java.sun.com/j2se/1.5.0/docs/api/java/io/PrintStream.html>

You can't find it. Guess why not..

It's private!

`newLine` is a method that the people who designed `PrintStream` wrote to help them write other parts of the class.. but they didn't intend it for outside use. So, they made it private.

Ain't it neat how all these weekly reading questions come together? :)

=====QUESTION 13=====

Q: What is the maximum length of a String?

Q: If I read an int with something like `scanner.nextInt()`, but the user types "15 20", what would happen? An error? I get 15? I get 20?

Q: There's this thing that I'm curious about that involves writing some code. How can I find out about it?

A: Write some code!

Don't forget what I said in week #1: experiment experiment experiment!!!

So, how would you test the maximum length of a String? Try writing successive programs that look like this:

Program #1:

```
String longStr = "1234567890";
longStr = longStr + longStr;
System.out.println(longStr.length());
```

If that works, try program #2:

```
String longStr = "1234567890";
longStr = longStr + longStr;
longStr = longStr + longStr;
System.out.println(longStr.length());
```

If that works, try program #3:

```
String longStr = "1234567890";
longStr = longStr + longStr;
longStr = longStr + longStr;
longStr = longStr + longStr;
System.out.println(longStr.length());
```

And so on. Don't think you'll ever get the answer this way?

Well, I'll tell you what.. program #30 has a String that's roughly a BILLION letters long. Do you think your computer can store a billion letters?

If you do, what do you think will happen with program #40? That's a trillion letters.

=====QUESTION 14=====

Q: Why does the textbook keep going on about how important comments are? I thought experienced programmers didn't need comments!

A: Ack, no! Experienced programmers definitely DO need comments. It's true that the more experienced you get, the easier it will get to understand code, but code can quickly get far too complicated to understand, even for experienced programmers.



Don't believe me? Check out the international obfuscated C code contest:

<http://www.ioccc.org/>

By the way, here's a program written in C that generates mazes. Need some comments to understand it? I do.

```
char*M,A,Z,E=40,J[40],T[40];main(C){for(*J=A=scanf(M="%d",&C);
--      E;      J[      E]      =T
[E  ]=  E)  printf("._");  for(;(A-=Z=!Z)  ||  (printf("\n|"
)  ,  A  =  39  ,C  --
)  ;  Z  ||  printf  (M  ))M[Z]=Z[A-(E  =A[J-Z])&&!C
&  A  ==      T[      A]
|6<<27<rand())||!C&!Z?J[T[E]=T[A]]=E,J[T[A]=A-Z]=A, "_." : " |";}
```

Even in serious programs, it's incredibly hard (and usually impossible) to understand everything from the code alone. For example, what if a variable's value is always supposed to be between 2 and 7. A comment can state this quickly and easily.. otherwise, I have to deduce from the way the code is written what the variable's value is!

Even if you COULD deduce everything from code, you wouldn't want to! Remember the discussion of black boxes and white boxes from a previous answer this week? If there were no comments, EVERYTHING would be a white box, and we'd have to understand all the code someone wrote to use their code. That would be terrible!

P.S. There are some VERY cool projects to allow us to express things like "this variable's value is always between 2 and 7" in documentation that the compiler can enforce (the way it does for private and final). ESC/Java is one example:  
<http://research.compaq.com/SRC/esc/>

=====QUESTION 16=====

Q: Is using a program based on Robot legal for playing online games?

A: Nifty idea :)

I doubt it's EVER illegal unless you sign a legally binding contract saying you won't do it. It's not even unethical or against the rules in most cases. (However, one of the students in my other class who wants to write a robot to play World of Warcraft mentioned that WoW specifically tells its users NOT to write robots for the game unless they use the approved API that WoW distributes.)

=====QUESTION 17=====

Q: items + 1 is like items++. Is items + 2 like items+++?

A: No, on two counts.

First, items + 1 and items++ are VERY different.

Let's say this is memory:            items

```

---
| 5 |
---

```

After you execute the code:

```
int x = items + 1;
```

memory looks like:

```

items  x
---  ---
| 5 |  | 6 |
---  ---

```

If you INSTEAD executed the code:

```
int x = items++;
```

Memory would look like this:

```

items  x
---  ---
| 6 |  | 5 |
---  ---

```

Look carefully and see how VERY different that is.

items++ actually CHANGES the value of the variable items. It happens to evaluate to the old value, but I prefer NEVER to take advantage of that. So, I'd only use items++ on a line of its own like:

```
items++;
```

On the other hand, there's no point in putting items + 1 on a line of its own:

```
items + 1;
```

It doesn't DO anything. It just calculates what a value one bigger than items is!

items+++ isn't legal at all. (items++)++ turns out not to be legal either. (Why? Because you can only apply ++ to a variable, and (items++) isn't a variable.)

If you wanted to add 2 to items, you could use either of the following two equivalent statements:

```
items = items + 2;
```

```
items += 2;
```

P.S. I said ++ only applies to variables. That's true enough, and if you're happy with that, read no further.

For those still reading, ++ actually applies to "lvalues": things that can go on the left side of an assignment statement. The only lvalue we've learned about so far is a variable. Subscripted arrays are also legal lvalues. I think that may be it in Java, but I'm not sure. (Want to be sure? Check the Java language specification!)

## J CPSC 121 Staff Meeting Agenda

This is a typical staff meeting agenda from my term 2 2007–2008 CPSC 121 course, an introductory discrete mathematics course with roughly 200 students, 13 TAs, and two instructors. These agendas were normally posted to a private staff newsgroup at least a day in advance to give TAs some time to add their own items.

The fifty-minute meetings ended on time or early every week, a testament to the TAs (showing up promptly to a painfully early-morning meeting!) and to organization. Nonetheless, we frequently spent significant time on the key item “State of the Course Report”, in which instructors and TAs discussed pressing issues (like recurring student misconceptions) that the rest of the staff could benefit from knowing.

Message no. 74

Posted by Steven Wolfman (...) on Wednesday, January 16, 2008 8:11pm

Subject: Agenda, 2008/01/18 staff meeting

Proposed agenda below. Please propose additional items if needed.

Friday 2008/01/18 staff meeting agenda:

- Attendance:
  - Steve, Meghan
  - Massih
  - Tracy
  - Lyon
  - Jeff
  - Billy
  - Dave
  - Jan
  - Jerry
  - Jill
  - Jessica
  - Deirdra
  - Matthew
  - Su
- Rough hours report
  - \* We'll have a sign-up sheet;
    - please ARRIVE with an (unofficial) estimate in mind for hours spent on 121 since the start of last week's meeting
- Reminder of URL for student account management
  - \* <https://www.cs.ubc.ca/local/views/ugradTeacher/>
  - \* Handle things like resetting student passwords, temporary print quotas, and managing lingering/zombie logins
- State of the Course Report
  - \* Instructors
  - \* TAs describe particular problems/successes from the previous week
- Mark management update [led by Tracy]
  - \* How have things gone this week?
  - \* What marks are still out-standing this week?
  - \* What columns on WebCT can be released?
  - \* CSLC participation point scheme

- Labs
  - \* Introducing yourself reminder
  - \* Lab 1 redux (if needed)
  - \* Issues with upcoming Lab 2
    - \* Marking scheme discussion
  - \* Any difficulty attending next week?
- Tutorials
  - \* Introducing yourself reminder
  - \* Issues with upcoming Tutorial 2?
    - [[only if needed, can be just Dave and Jeff afterward]]
- CSLC
  - \* Sign/introductions check
  - \* Reminder to proactively (though not constantly) offer help
    - \* Every 10 minutes or so if no other activity,
      - "drop in" on students in room
  - \* Reportback for the week
  - \* Schedule OK?
- Assignments
  - \* For NEXT week: A#1 pickup (by Jessica?)
    - o Drop box key handoff?
    - o Plan for marking UTAs (Jill, Jessica, Deirdra?)
- Quizzes and exams [Reminders]
  - \* Midterm exam: March 6 @ 6:30PM in Hebb Theatre
    - o Invigilators: Tracy, Dave, Steve, Meghan
    - o EVERYONE, please clear as much of Mar 7 as possible for marking
    - o Marking MAY stop to either Sat or Mon; we'll work out which
    - o Anyone interested in offering review session/sessions?
  - \* Final exam: to be determined; do NOT schedule travel before May 2!
  - \* Quizzes
    - o Quiz 1 upcoming in TWO weeks; more next week
- Course development
  - \* Regular Expression lab
    - o Dave, Steve & Meghan to discuss content (via email or a meeting?)
  - \* update of Lab 4 (Matthew)
- In-class participation point marking
  - \* Results from first week's exercises?
  - \* Second week's exercises handoff
- Hourly TA timesheets, if needed

## **K CPSC 221 Card Trick Notes**

These are my posted notes from exploration of a magic trick in my summer 2005 CPSC 221 course, an intermediate discrete mathematics course. I demonstrated the trick—and invited students to figure it out for themselves—early in the term, after which we investigated successively more detail. Near the end of the course, we worked through the bipartite matching proof that shows that the trick works at its theoretical maximum deck size.

# A Discrete Mathematics Magic Trick

Shuffle a deck of cards and deal five of them out to MOE (Magician OnE). Moe sets one of the cards aside, puts the remaining four into a neat pile, and hands them off to MAgGle (MAGIcian two). After investigating them for a few seconds, Maggie announces the identity of Moe's hidden card, to the audience's eternal delight. (The audience, being computer scientists, are easily amused.)

How did Maggie and Moe do it? Does their trick always work? How big a deck does their trick work for? Is there some other strategy that works for bigger decks? What's the largest deck that any strategy can work for?

This page is designed to walk you through questions like these using the concepts of discrete mathematics. The underlying goal is to show you how a simple problem can become a delightful playground for theoretical investigation, and how the discrete mathematics concepts you've been learning can help support your curiosity as you explore these problems.

## A Note to Students and Instructors

This page describes a detailed progression in the exploration of a magic trick using discrete mathematics concepts, including permutations, combinations, the pigeonhole principle, functions, bijections, and bipartite graphs.

Although not all of these concepts will be mentioned directly, and a student might understand this page without having learned all of these concepts, the student who has seen each concept will be better prepared for the exploration below, and the exploration will in turn reinforce these concepts for the student.

As you read, play with the concepts and ideas expressed. Try to answer the questions before reading our answers. Relish differences in approach or results that you discover!

## The Basic Trick

### Setting up the problem

Moe gets five cards at random and must find a way to select a secret card and order the remaining four cards so that Maggie can always identify the secret card. To figure out the trick, let's start by figuring out some of the constraints on the trick.

How many different cards could be the secret card?

Maggie sees the four cards Moe hands her; so, the secret card cannot be any of them. There are 52 cards in the deck. That leaves  $52 - 4 = 48$  cards that could be the secret card.

Moe can provide information to Maggie just by ordering the cards. How many different ways can Moe order the four cards?

To start with, let's pick some way to order cards. How about we assume suits are ordered alphabetically so that Spades > Hearts > Diamonds > Clubs. Then, one card can be greater than another card if the first card's rank is greater or if the ranks are the same but the first card's suit is greater. We'll let Aces be low cards; so, the smallest cards in order are: AC, AD, AH, AS, 2C, 2D, 2H, 2S, 3C, 3D, ...

Now, Maggie can think of the four cards she sees as the "lowest, 2nd lowest, 3rd lowest, and 4th lowest" cards (or "1, 2, 3, and 4"), regardless of what the actual values of the cards are. Our previous question reduces to "How many different ways can Moe order the numbers 1, 2, 3, and 4?"

That's just a question of permutations. How many different permutations are there of four objects? Well, there are 4 objects that could be chosen first, and then 3 left to be chosen second, 2 left to be chosen third, and only 1 to be chosen last:  $4 * 3 * 2 * 1 = 4! = 24$ .

Finally, Moe chooses the secret card from among five possibilities. Clearly, he can pass some information by this choice, but how that works is not clear.. yet.

## How can Maggie and Moe use orderings to pass information?

Using just the ordering of the 4 cards Moe passes to Maggie, Maggie can distinguish among 24 options. To make the trick work, however, both participants need a clear way to translate between orderings and numbers between 1 and 24. That amounts to is picking an ordering of the arrangements of the numbers 1, 2, 3, and 4 (just like we picked an ordering of the 52 cards above). One natural ordering is lexicographic order. For example, 2134 comes before 2314 (since the second character of the first arrangement '1' comes before the second character of the second arrangement '3').

How can Maggie and Moe perform the translation? Well, say Moe wants to pick an arrangement representing 17. He first decides which of the four cards to arrange in the first slot. There are four possibilities, and for each possibility there are  $3! = 6$  arrangement of the remaining cards. So, the arrangements starting with card #1 are arrangements #1–6. The arrangements starting with #2 are arrangements #7–12, those starting with #3 are #13–18, and those starting with #4 are #19–24.

Moe wants to express 17; so, he'll pick card #3 (the third lowest card) to be first in the arrangement. There are three possible cards to be next in the arrangement; for each one, there are two arrangements of the remaining two cards. So, card #1 is the next card in arrangements #13 and 14, card #2 is for arrangements #15 and 16, and card #4 is for arrangements #17 and 18. Moe picks card #4 as the second card. Finally, putting the last cards in order finishes arrangement #17, while reversing them finishes arrangement #18. So, Moe puts card #1 next and card #2 last.

Moe expresses 17 using the arrangement 3412. As a side note, notice that once Moe picks the first card, arranging the rest of the cards is actually a recursive step: arrange three cards (which can be renumbered 1, 2, and 3) to represent a number between 1 and 6.

In short, to represent a number, find which group of 6 it falls in to select the first card. Then, use the group of 2 within those 6 the number falls in to select the second card. Then, use which of the remaining 2 is the number to select the order of the remaining two cards.

Given an arrangement, Maggie does the reverse of Moe's steps to find the number it represents. So, given the arrangement 2431, Maggie notes that the arrangement is in the second group of 6: 7–12. She then notes that the arrangement is in the last group of 2 within those 6: 11 and 12. Finally, the last two cards are out of order; so, the arrangement is the larger of the two: #12.

Of course, this is just one of many possible ways to translate between arrangements and numbers. In fact, any *bijection* (one-to-one, invertible function) between arrangements and the numbers 24 works. However, with some practice, humans can calculate this particular bijection in their heads without too much trouble.

## Where do Maggie and Moe get their extra bit?

With a total of 48 cards that can be the secret card and 24 orderings of the 4 non-secret cards, Maggie's only a factor of 2 away from a successful trick. How will we narrow down the options that one bit more?

Take some time to think about how to solve this. (Avoid the temptation to encode that extra bit in the physical placement of the cards! "If those first two cards are overlapping, that's one set of 24; if they don't touch, that's a different set of 24.") In the process, you might discover some interesting properties of the selection of cards Moe and Maggie can see. For example, can all of Moe's cards be the same rank?

There are actually many strategies that solve this problem. The variety of strategies is an interesting point that we'll return to later. For now, let's develop one particular strategy that is fairly "clean", meaning a human can do the trick quickly without a cheat sheet!

The extra information Moe passes to Maggie *must* come from Moe's choice of the secret card. Moe gets to choose among five possibilities when he selects the secret card. (That's  $5C1$  or equivalently  $5C4$  possibilities in combinatoric terms.) So far, we've ignored that choice. Let's consider it now. How can Moe's choice of the secret card pass sufficient information to Maggie for her to complete the trick?

It's tempting, with just one bit to go, to divide the cards in half into red cards (hearts and diamonds) and black cards (spades and clubs). Unfortunately, this doesn't work out well. Can Maggie always assume that the secret card is black? No, because all five original cards could have been red. Can Maggie assume that the colour of the hidden card will be a kind of parity bit for the other colours: red if an even number of the other cards are black and black if an odd number of the other cards are black? No, because all the original cards could be black.

As a side note, the division of cards into red and black cards is just one of many ways to partition the cards evenly into two sets. As with our ordering of the cards and our bijection between the numbers 1–24 and arrangements of four cards above, any such division is arbitrary from an algorithmic perspective. From a human perspective, however, it's easier to consider the set of red cards and black cards rather than sets like "all red cards except the 7H, 4D, and AH but including the 8C, 9C, and KC". Sometimes we want to devise strategies that make sense to humans, and sometimes we want to prove the existence or absence of any strategy, regardless of its complexity.

Back to Moe's problem. How should he choose the secret card? Well, let's pick a simple but broken strategy and then try to repair it. Moe can pick the highest card of the five he sees as the secret card. Then, he can take the "number" of the highest card (its place in the ordering of cards), subtract the number of the second highest card, and encode that as an ordering of the remaining cards.

For example, say Moe sees the 3C, 5H, 5S, 6D, and 9S. We can translate these cards into numbers between 1 and 52 (recalling that Aces are low, and the suits are ordered alphabetically: Clubs, Diamonds, Hearts, and then Spades). Then, these cards are:  $3C = 9$ ,  $5H = 19$ ,  $5S = 20$ ,  $6D = 22$  and  $9S = 36$ . The 9S is the high card, and  $9S - 6D = 14$ . So, Moe would set 9S aside as the secret card and arrange the other cards to represent 14. That's arrangement 3142 or  $\langle 5S, 3C, 6D, 5H \rangle$ .

When Maggie sees the arrangement  $\langle 5S, 3C, 6D, 5H \rangle$ , she immediately knows to start counting from the highest (remaining) card: 6D. From the arrangement 3142, she knows to count 14 cards up from 6D. That's 3 ranks (12) plus 2 suits (2):  $6D \rightarrow 7D \rightarrow 8D \rightarrow 9D \rightarrow 9H \rightarrow 9S$ . Maggie announces that the secret card is the 9S.

Great!

Now, what does Moe do if his original 5 cards are instead  $\{3C, 5H, 5S, 6D, KC\}$ ?  $KC = 49$  and  $6D = 22$ ; so,  $KC - 6D = 27$ . That's too big to represent with our strategy so far! So, Moe cannot set aside KC and



complete the trick. Does he have some alternative?

Take a look at the 3C and KC.  $3C = 9$ , and  $KC = 49$ . Imagine counting up from KC and "wrapping around":  
 $49 \rightarrow 50 \rightarrow 51 \rightarrow 52 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$ . That's 12 steps.

Moe could set aside the 3C as the secret card and arrange the other cards to indicate 12: <5S, KC, 6D, 5H>. Maggie can then count from the highest card she sees, just like in the last example, but wrap around when she goes above 52. (In mathematical terms, she's essentially counting modulo 52, except for the minor detail that we're using the numbers 1–52 rather than 0–51.)

That's it! Moe's strategy is to select the highest card as the secret unless it's more than 24 places away from the second highest card, in which case he selects the lowest card. Either way he finds the (wraparound) distance between the secret card and the highest remaining card and encodes that value in the arrangement of the remaining four cards.

Maggie's strategy is to find the highest card of the four she's given, find the number encoded in the order of those cards, and count up (wrapping around) from the highest card to the secret card.

Try the trick out for a few times to confirm that it works. Write a program to execute the trick. *Play* with the trick. For example, keep statistics in your program on how often Moe must choose the low card as the secret card rather than the high card.

## Proving the Basic Trick Works

Do you believe that Maggie and Moe's trick always works? Why?

How could you *prove* their trick works? Could you enumerate every five card hand and show that Moe can find a four card ordering to give Maggie? Would that be enough? How many five card hands are there?

I'll just answer the last question here. There are  $52C5$  five card hands. That's 2598960 hands. You could probably write a program to work through each of those hands, but you certainly couldn't do it.. ahem.. by hand. Let's try a different approach.

We know that our translation between arrangements and numbers 1–24 is a bijection; so, Moe can always create an arrangement given a number 1–24, and Maggie can always find the number Moe intended given an arrangement. Similarly, we know our translation from cards to numbers 1–52 is a bijection.

Put these facts together with our algorithm, and we know that *as long as the secret card is 24 or fewer (wraparound) places away from the highest remaining card*, Maggie and Moe's strategies work. Can we prove that Moe never chooses a secret card more than 24 places away from the highest remaining card?

Let's assume that Moe picks a secret card that's 25 or more places away from the highest remaining card. If the highest card is  $\leq 24$  places away from the second highest card, Moe would select the highest card as the secret card and the secret card would be within 25 places of the highest remaining card. So, the highest card must be  $\geq 25$  places away from the second highest card, and Moe must have selected the low card as the secret card.

Now, if we count from a card all the way back around to itself, we must go exactly 52 places. That's because there are 52 cards in the deck. So, let's count from the highest card all the way back to itself. There are  $\geq 25$  places from the high card to the secret card. There are  $\geq 1$  places from that card to the second card,  $\geq 1$  places from *that* card to the third card, and  $\geq 1$  places from to the fourth card. So far, we've gone  $\geq 28$  places. From the discussion above, we know that there are  $\geq 25$  places between the second highest and highest cards. So, to

get all the way back to the highest card takes  $\geq 53$  places.

Wait! There are only 52 places total. It *cannot* take 53 or more places to get from a card back around to itself. To reach such a contradiction, we must have assumed something impossible. Our only assumption was that Moe picks a secret card that's 25 or more places away from the highest remaining card. Therefore, Moe *never* picks a secret card that's 25 or more places away from the highest remaining card; the secret card is always 24 or fewer places away from the highest remaining card.

Along with our discussion above, that proves that Moe and Maggie's trick works! (This is a pigeonhole proof: we develop a contradiction by showing that Moe can count more cards than there are spots for.)

## Dealing with a Joker

What happens if Moe starts with a deck that contains a Joker? Does the trick still work? Can you prove whether it works or not?

If Moe starts with a deck of 53 cards, our proof above no longer works. We can still show that there are  $\geq 53$  places to get from the high card back around to itself. Unfortunately, that fact no longer causes a contradiction since it *should* take 53 places to get back around to the high card. Does that prove that the trick doesn't work?

Actually, that does *not* prove that the trick doesn't work. It's incredibly important to understand why not.

(Invalidating a proof of a concept X does not prove that X is false. It just shows that a particular proof of X doesn't work. For example, here's a proof that  $35 > 21$ : "The sum of the digits in 35 is 8. The sum of the digits in 21 is 3. 8 is greater than 3. Therefore,  $35 > 21$ ." That proof is completely bogus. Nonetheless, 35 really is greater than 21!)

That leaves us with the same question: does Moe and Maggie's strategy work on a deck with a Joker? Let's assume *without loss of generality* that Maggie and Moe number the Joker 53. (The "without loss of generality" part essentially means that this assumption doesn't count against us. Numbering the Joker as 53 doesn't affect the proof. Maggie and Moe must number the Joker as something, and whatever they choose, the rest of the proof would still work, although perhaps with minor but obvious changes.)

What happens if Moe faces the five card hand {AC, AD, AH, AS, 8C}?

$8C - AS = 29 - 4 = 25$ , and  $AC - 8C = 1 - 29$  which wraps around to 25. In other words, regardless of whether Moe chooses the 8C or the AC as the secret card, Moe's strategy will fail. Does this prove that Moe and Maggie's strategy is not guaranteed to succeed for a 53 card deck?

Sure! We've shown an example for which the strategy fails; so, it's not guaranteed to succeed. (By the way, on how many five card hands does their strategy fail? What's the probability of the strategy failing on a random hand? How big would the deck have to get before Maggie and Moe would even worry about their strategy failing?)

Does this prove that no strategy can possibly work for a 53 card deck?

## Proving that No Strategy Can Work

Our work so far just shows that Maggie and Moe's strategy doesn't work for a 53 card deck. We haven't proven that no strategy exists for a 53 card deck. Let's see how big we have to make the deck before we

know how to put a proof together that no strategy will work. (Note: this section gets a bit more technical than the previous sections.)

A strategy for Moe must map any possible five card hand to a four card ordering. Moreover, each of those four card orderings need to be distinct. Otherwise, Maggie could get a four card ordering and not know which of two (or more) possible hands led Moe to construct that ordering.

So, Moe's strategy is a one-to-one, total function from five card hands to four card orderings. (As a side note, there's no particular reason that Moe's strategy needs to be "onto", i.e., that every four card ordering must be the image of a five card hand under Moe's strategy. If such a four card ordering exists and Maggie gets it, she'll have no idea what five card hand it came from. However, Moe would never construct that ordering; so, Maggie will never see it anyway.)

What's an example condition under which no strategy exists for Moe?

Let's say, arbitrarily, there were 1,000,000,000 five card hands and 999,999,999 four card orderings. In that case, Moe couldn't possibly construct a one-to-one, total function from five card hands to four card orderings. By the pigeonhole principle, some pair of those billion five card hands must map to the same four card ordering. (If that doesn't make sense, try it with 4 five card hands and 3 four card orderings. The first 3 five card hands use up all the available four card orderings. The last five card hand has nowhere to go except an already used ordering.)

Therefore, no strategy exists for Moe if there are more five card hands than there are four card orderings.

How many five card hands and four card orderings are there for a deck of size  $n$ ?

This is a combinatorial problem. There are  $nC5$  five card hands. There are  $nC4$  four card hands and  $4!$  orderings of each such hand; so, there are  $4! * nC4$  four card orderings.

For what deck sizes are there more five card hands than four card orderings?

Well, first let's figure out which one grows faster.  $nC5 = n! / ((n-5)!5!) = (n * (n-1) * (n-2) * (n-3) * (n-4)) / 120$ .  $4! * nC4 = 4! * n! / ((n-4)!4!) = n! / (n-4)! = n * (n-1) * (n-2) * (n-3)$ . In asymptotic terms,  $nC5$  grows as  $\Theta(n^5)$  while  $4! * nC4$  grows as  $\Theta(n^4)$  (i.e.,  $nC5$  has one more  $n$ ). So,  $nC5$  asymptotically dominates  $4! * nC4$ .

That means that there's some deck size after which the number of five card hands is always bigger than the number of four card orderings. (Note: I should actually prove that once these cross, the first is always bigger than the second!!!)

At what deck size is the number of five card hands first larger than the number of four card orderings?

We can find that by finding the  $n$  for which the number of five card hands is equal to the number of four card orderings. For all larger decks, the number of five card hands will be bigger:

$$\begin{aligned} nC5 &= 4! * nC4 \\ (n * (n-1) * (n-2) * (n-3) * (n-4)) / 120 &= n * (n-1) * (n-2) * (n-3) \\ (n-4) / 120 &= 1 \\ n-4 &= 120 \\ n &= 124 \end{aligned}$$

Therefore, any deck with 125 cards or more has more five card hands than four card orderings, and we know for sure that no strategy will guarantee a successful trick.

## Narrowing the Bounds

For decks of size 52 or fewer, Maggie and Moe's strategy will work. Based on this "witness" of the feasibility of a 52 card deck, we know that some strategy is guaranteed to work. We can easily show that the same strategy scales to smaller decks down to five cards. For decks of size 125 or more, we know that no strategy is guaranteed to work. What about decks of size 53 to 124?

There's a few strategies we could take to narrow these bounds. Here are three: we could try to prove that no strategy exists for a smaller deck size than 125; we could try to find a strategy for a larger deck size than 52; or we could try to prove that a strategy exists for a larger deck size than 52 (without actually supplying the strategy). Think about which of these seems the most promising. Try them out.

Based on a couple of clues, we'll work on proving that a strategy does exist for a deck larger than 52 cards. One clue is that, for the deck of 124 cards, the number of five card hands is *exactly* equal to the number of four card orderings. That equivalence suggests that, perhaps, there's a strategy that "just fits" for a 124 card deck.

The second clue is that Maggie and Moe's strategy only very rarely requires Moe to select the lowest card as the secret card. (If you haven't already worked out the probability of Moe selecting the lowest card as the secret card, now's a good time!) That asymmetry suggests that Maggie and Moe's strategy is somehow "overkill" for the problem. That is, their strategy has extra information potential that they throw away on just a few special cases. We *know* their strategy doesn't work as it stands on 53 cards, but maybe if it did a better job of utilizing the available information passing mechanisms, it could.

Both clues suggest that the trick could work on decks larger than 52 cards. Two natural sizes to try at this point would be 53 card decks and 124 card decks. The two deck sizes suggest very different approaches. Give them both a shot!

With the benefit of hindsight, we'll focus only on the 124 card deck. Is there a successful strategy for a 124 card deck? What does it mean for such a strategy to exist?

As above, we'll consider a strategy to be a function mapping five card hands to four card orderings. The function must be one-to-one and total. Because a 124 card deck has exactly the same number of five card hands and four card orderings, the function must also be onto, meaning that it is a bijection.

Let's consider the space of possible functions. Given a five card hand, Moe can't just pick *any* four card ordering. He *must* pick an ordering whose cards are drawn from the five card hand he's given. How many four card orderings can be drawn from a given five card hand?

How many four card hands can be drawn from a five card hand?  $5C4 = 5$ . How many four card orderings can be created from each of these four card hands?  $4! = 24$ . So, the number of four card orderings drawn from a given five card hand is  $5C4 * 4! = 5 * 4! = 5! = 120$ . Every five card hand can map to one of 120 different four card orderings.

Maggie, in turn, guesses a secret card given a four card ordering. Another way of thinking of Maggie's task is to *reconstruct* the five card hand Moe was given by adding the secret card back in. Given a four card ordering, then, she might guess any five card hand that contains all four cards in the ordering. How many five card hands contain a given set of four cards?

With a deck of 124 and a given set of 4 cards, there are 120 cards remaining that could be the fifth card in the hand. So, each four card ordering could come from 120 different five card hands.

Now we know the constraints on the possible 124 card strategies. How can we picture the set of strategies? One convenient representation is a bipartite graph. On the left are  $124C5$  nodes, each representing one five card hand. On the right are  $124C4$  nodes, each representing a four card ordering. Each node on the left has a degree of 120; that is, 120 edges depart each node leading to the four card orderings Moe could choose from that node on the right. Each node on the right has a degree of 120; that is, 120 edges depart each node to the five card hands that Maggie could reconstruct from a given four card ordering. (Of course, these two sets are the same edges considered from opposite sides!)

We can rephrase our central question in terms of this bipartite graph. Our original question was: Does a strategy exist for Moe and Maggie to successfully complete the magic trick with a 124 card deck? Our new question is: Is there a way to select 124 edges in this bipartite graph such that each node in the graph is incident to exactly one of the selected edges?

If such a set exists, then Moe's strategy given a five card hand is to find the node for that hand, follow its edge across to the matching four card ordering on the right, and construct that ordering for Maggie. Maggie finds the four card ordering she's given on the right, follows its edge across to a five card hand, and identifies the secret card as the extra card in that hand.

(As a side note, such a set is called a perfect matching. The marriage theorem establishes the conditions under which a perfect matching exists, and we could proceed from this point using principles proven for matchings. You should give it a shot! However, in this discussion, we will prove our result directly using techniques that can also be used in proving the marriage theorem.)

Our bipartite graph of the space of possible strategies is immense and fiercely complex. So, it's not immediately clear whether a perfect matching exists. We'll try to prove that a matching exists by imagining a (not very good) way to select a strategy and then proving that we can always improve that strategy until it's perfect. Step 1 is to pick an easy way to select an initial strategy. How could we do that?

We'll use a greedy approach. Start by marking every node as uncovered. Now, starting from the top node on the left, work your way down the graph. At each node, find the edge incident on that node that leads to the highest uncovered node on the right. If such an edge exists, add it to our strategy and mark both nodes as covered. If no such edge exists, just continue on to the next node down.

If this yields a perfect matching, we're done. If not, there's  $k$  nodes on the left and  $k$  nodes on the right that are uncovered. Let's say we were to prove that, for any imperfect matching, we can find a way to alter (or "augment") the matching to cover one extra pair of nodes. In that case, we could just keep augmenting our imperfect matching, one step after another, until there were no more uncovered pairs. That would be a perfect strategy! So, if we can prove that these augmentations always exist for imperfect matchings, then we've proven that a perfect matching exists.

What would an augmentation of an imperfect strategy look like?

Well, we can say for sure that there won't be any trivial fixes. That is, there won't be any uncovered pair with an edge between them. If there were, then our greedy algorithm would have chosen that edge and covered the pair.

Let's imagine an uncovered node on the left. It has 120 edges leading to nodes on the right, but every one of those nodes is itself covered. In fact, if we wanted to "turn on" one of those 120 edges to cover one of the 120 right-hand neighbours, we'd have to "turn off" the edge that was currently being used to cover that neighbour. Doing that would, in turn, uncover the node the right-hand neighbour used to be paired with. So, our original uncovered node is now covered; its right-hand neighbour is now covered but was covered before the change as well; and a new node on the left is now uncovered. It looks like we haven't made any progress,

right?

Well.. no, but we might *now* be able to make progress. Unlike the original uncovered node on the left, the newly uncovered node on the left might be connected directly by an edge to an uncovered node on the right. Nothing in our original algorithm rules that possibility out.

Furthermore, even if our newly uncovered node isn't directly connected to another uncovered node, we can keep adding to this sequence of edges to "toggle". Each time we add another pair of edges leading to the right and coming back, we turn one edge on and turn another off, and each time we "move" the uncovered node on the left. If we eventually uncover a node connected directly to an uncovered node on the right, we can add the edge connecting them as the last in the sequence of edges, and we'll have covered an extra pair of nodes.

Can we always find a sequence of edges of this sort (which we will call an "augmenting path") that leads from an uncovered node on the left to an uncovered node on the right?

Well, let's assume that there's at least one uncovered pair but we cannot find an augmenting path. Then, if we can derive a contradiction, we'll know that we can always find an augmenting path.

Imagine building a set of nodes and edges on both sides of the graph. Start by adding an arbitrary uncovered node on the left to the set. Then, add all 120 nodes on the right that are adjacent to that node to the set. Either every one of those 120 nodes is covered or else we've found an augmenting path. Since we assumed there is no augmenting path, they must all be covered. So, add the 120 nodes on the left that are paired with those 120 nodes on the right. Now, continue adding nodes in this fashion — add all nodes adjacent to a node on the left and add the node paired with each covered node on the right — until no new nodes are added to the set.

Again, by our assumption that there's no augmenting path, this set must reach a fixed point (i.e., stop growing) before any uncovered node is added on the right. Furthermore, every node we added on the left except the first was added *because* it was paired with a node on the right. Therefore, every node on the left but the first is covered, every node on the right is covered, and there is one more node on the left than on the right.

Now, imagine dragging that whole set down to the bottom of the bipartite graph and drawing a line separating those nodes from the rest of the graph. We'll call that line the "separator". Can any edge from a node on the left cross the separator?

No, no edge from the left can possibly cross the separator. If it did, it would be an edge from a node on the left that's in the set to a node on the right that's not in the set. But, if such an edge existed, we would have added the node on the right to the set back when we were building the set up!

(As a side note, edges from the right *can* cross the separator. That's because only edges incident on nodes on the right *that are part of the matching* drag nodes on the left into the set. One other unused edge must have been the one that caused the node on the right to be added to the set. The other 118 edges incident on each node on the right can potentially cross the separator.)

How many edges depart from the left side of the set?

Every node on the left has degree 120. So, if there are  $k + 1$  nodes on the left, then  $(k + 1) * 120$  edges depart from the left side of the set.

How many of those edges are incident on nodes on the right side of the set?

Since we already know that no edge from the left can cross the separator, every one of the  $(k + 1) * 120$  edges must lead to a node on the right side of the set.

How many edges depart from the right side of the set?

Like the left-hand nodes, every right-hand node has a degree of 120. If there are  $k + 1$  nodes on the left, there are  $k$  nodes on the right (i.e., one fewer as mentioned above). So, there are exactly  $k * 120$  edges incident on nodes in the right side of the set.

**Wait!!** We've just shown that there are exactly  $(k + 1) * 120$  edges incident on the right-hand side of the set, but we have *also* shown that there are exactly  $k * 120$  edges incident on the right-hand side of the set.  $(k + 1) * 120$  is not equal to  $k * 120$ ; so, that's a contradiction!

(Intuitively, that says that there are just too many edges coming from the left to fit into the set on the right. That will be true unless there are as many nodes on the right as on the left, but the right side can't "catch up" in size to the left unless we add a node to the right that's uncovered and therefore unpaired with a node on the left.)

Therefore, our assumption was incorrect, and any time there is at least one uncovered pair, we can always find an augmenting path. Once we find the augmenting path, we can "flip" the state of each edge on the path: turning on edges that weren't part of the matching before and turning off edges that were part of the matching before. The new matching has one fewer pairs of uncovered nodes. If we keep using these augmenting paths, we'll eventually have a perfect matching!

We have *proven* that a strategy exists to do the magic trick on a 120 card deck. In a rather nifty twist, we did it without ever actually finding a particular working strategy! We just showed that such a strategy must exist. As it happens, you could actually implement the construction in our proof to generate a strategy (and I encourage you to do so!). Sometimes, however, we prove the existence of solutions in CS without even being *able* to construct the solution.

## Wrap-Up

That's as far as I've explored this particular magic trick. You can go farther in several ways. You could read up on some related topics (like the marriage theorem and maximum matching). You could explore other variants of the problem (like a generalized version of the trick where Moe is given a  $k$  card hand and selects  $m$  secret cards). You could explore other facets of the existing problems (like how big a deck can be solved with a human-usable strategy and, for that matter, what about a strategy makes it "human usable"). You could explore programs, visualizations, and descriptions that do a better job than this text at explaining the existing problems.

Finally, you can find other interesting problems or puzzles and push their boundaries using your own ingenuity and knowledge of computer science.

Have fun :)

---

Steve Wolfman  
UBC CS

## **L CPSC 313 Practice Final Exam**

My CPSC 313 TAs and I designed this practice exam to guide students as they studied for the actual final exam in my term 2 2007–2008 CPSC 313 computer architecture course. The exam is annotated both with connections to the course learning goals (to help students understand how learning goals might be assessed) and with notes on similarities to the final exam.

To save space, I have omitted some questions that were primarily based on previous work and compressed the blank answer space on the remaining questions.



CPSC 313 BOOTLEG Final Exam  
2007/8 Winter Term 2  
April 17, 2008

**COVER PAGE OMITTED**

**PROBLEM 1 [? marks]**

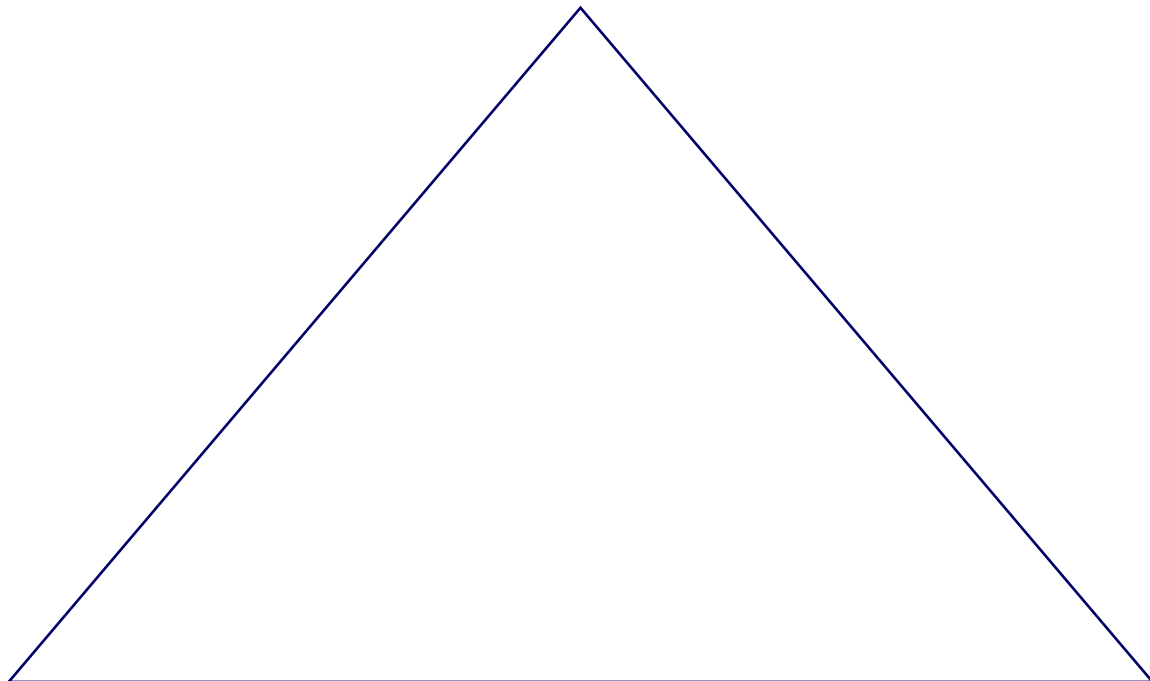
LGs:

- Illustrate with examples how different memory technologies provide different ratios of cost to speed.
- Explain why, despite the fact that VM uses main memory as a cache for disk, VM tends to have significantly different organization and mechanisms from caches.

**[An exam problem based on this would probably be a bit more focused but hit similar points.]**

Consider the following table of memory technologies. Build a sensible memory hierarchy from it by filling in the following pyramid and labelling the levels L0, L1, L2 ...

| Type      | Speed (1X = 1 cycle) | Cost  | Location              |
|-----------|----------------------|-------|-----------------------|
| XFRAM     | 1X                   | 500X  | (on chip or off chip) |
| Registers | < 1X                 | 1000X | (in CPU)              |
| DMRAM     | 10X                  | 100X  | (off chip)            |
| KENRAM    | 5X                   | 500X  | (on chip or off chip) |
| SUPERDISK | 1000X                | 5X    | (off board)           |
| SWRAM     | 100X                 | 10X   | (off chip)            |
| DISK      | 100000X              | 1X    | (off board)           |
| DRUM      | 100000000X           | 5X    | (next door)           |



(b) Discuss how each “cache” in the levels of the memory hierarchy might be configured in terms of its size, replacement policy, write policies, and mapping/associativity. Assume the bottom level has 1TB of storage.

**PROBLEM 2 OMITTED**  
**(Textbook problem)**

**PROBLEM 3 [? marks]**

LGs:

- Define and give examples of spatial and temporal locality.
- Analyze sources of and impediments to spatial and temporal locality in common coding idioms (such as loops, function calls, variable use, etc.).
- Simulate the execution of a fully specified cache on a simple piece of C code or a list of memory requests (read and write), including indicating which requests result in cache hits and misses and the types of the misses (cold, compulsory, or conflict).

**[This is somewhat too intricate for an exam question. Instead, the exam would likely focus on how changes to common algorithms (such as bubble sort) impact spatial and temporal locality (and whether we can tell how they impact these.)]**

Imagine an L1 cache with 4 8-byte blocks. The cache is fully associative and uses LRU replacement. ints are 4 bytes long. Note: LRU is **very** unusual in an L1 cache, but it’s convenient so that our by-hand problems work out consistently and sensibly, unlike with random or round-robin replacement.

Here is code for binary search:

```
int findIndexOf(int * array, int length, int target)
{
    int low = 0, high = length - 1;
    while (low < high) {
        int mid = (low + high) / 2;
        int value = array[mid];
        if (value == target)
            return mid;
        else if (value > target)
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1;
}
```

Consider a binary search over a cache-block-aligned sorted array of 15 integers. Assume that only the array access “array[mid]” causes a memory reference. For each part below, assume we start with a cold cache.

(a) If we search for an element that is **not** in this array, are we guaranteed to be able to fit all the `array[mid]` elements we need for the search into our cache?

(b) If we search for the same element that is not in the array twice, what percentage of the `array[mid]` elements accessed in the second search will be hits?

(c) Imagine we searched, in order, for the 1<sup>st</sup>, then 3<sup>rd</sup>, then 5<sup>th</sup>, then 7<sup>th</sup> ... elements of the array. In the following table, fill out which `array[mid]` accesses are hits and which are misses.

| Index sought | 1 <sup>st</sup> <code>array[mid]</code> access | 2 <sup>nd</sup> <code>array[mid]</code> access | 3 <sup>rd</sup> <code>array[mid]</code> access | 4 <sup>th</sup> <code>array[mid]</code> access |
|--------------|--|--|--|--|
| 0            |  |  |  |  |
| 2            |  |  |  |  |
| 4            |  |  |  |  |
| 6            |  |  |  |  |
| 8            |  |  |  |  |
| 10           |  |  |  |  |
| 12           |  |  |  |  |
| 14           |  |  |  |  |

(d) Imagine we searched, in order, for the 1<sup>st</sup>, then 9<sup>th</sup>, then 3<sup>rd</sup>, then 11<sup>th</sup>, then 5<sup>th</sup>, then 13<sup>th</sup> ... elements of the array. In the following table, fill out which `array[mid]` accesses are hits and which are misses:

| Index sought | 1 <sup>st</sup> <code>array[mid]</code> access | 2 <sup>nd</sup> <code>array[mid]</code> access | 3 <sup>rd</sup> <code>array[mid]</code> access | 4 <sup>th</sup> <code>array[mid]</code> access |
|--------------|--|--|--|--|
| 0            |  |  |  |  |
| 8            |  |  |  |  |
| 2            |  |  |  |  |
| 10           |  |  |  |  |
| 4            |  |  |  |  |
| 12           |  |  |  |  |
| 6            |  |  |  |  |
| 14           |  |  |  |  |

(e) Justify the following statement based on what you saw above: “Binary search often has poor spatial locality, but it has good temporal locality for a small set of elements in the array.” What is the “small set of elements”?

#### PROBLEM 4 [? marks]

LGs:

- Describe several approaches for improving the performance of CPU architectures, including pipelining and branch prediction.
- Analyze and manage the challenges that arise when adding these performance improving approaches to CPUs.
- Contrast hardware "algorithm" design with software design, particularly around the issue of parallelism.
- Identify control and data flow hazards in a proposed pipelining scheme.
- Propose and compare (in terms of correctness, efficiency, and complexity) solutions for hazards including data forwarding, branch prediction, stalls, and bubbles.
- Describe the key steps in implementing data forwarding, branch prediction, stalls, and bubbles in our pipelined y86 processor, including changes to data paths and register operation.

[The exam might have a question very similar in format but focused on somewhat different performance issues.]

(a) What properties of hardware and our sequential processor implementation make it so that a pipelined implementation is more efficient than the sequential implementation?

(b) Explain why forwarding (sending results from one instruction direct to the input of another instruction rather than communicating between the instructions through the register file) is unnecessary in the sequential processor implementation.

(c) Would it be **possible** to implement the pipelined processor without forwarding? If so, explain the strategy you would use to correctly execute the sequence of instructions:

```
addl %ecx, %eax
addl %edx, %eax
```

If not, explain what about this sequence of instructions makes forwarding necessary.

**PROBLEM 5 OMITTED**  
(Modified textbook problem)

**PROBLEM 6 OMITTED**  
(Modified problem from old exam)

## PROBLEM 7 [? marks]

LGs: various throughout the caching, pipelining, and x86 portions of the course.

**[The exam will not have a similar question, but several of the concepts touched on here will make appearances in questions on the final.]**

Debunk each of the following misconceptions.

(a) If you had plenty of money and wanted to build an ultra-fast computer with tons of memory, you wouldn't bother with caching.

(b) Branch prediction is actually a bad policy. If we just waited two cycles until we knew whether the branch was going to be taken, we could avoid any wasted work!

(c) The more memory a program uses, the worse its locality is.

(d) If you specify the way a processor functions in its ISA, that's the way you have to implement it.  
[Careful! This isn't entirely a misconception.]

(e) Recursion is inefficient because it causes the stack to grow in proportion to the depth of the recursion.

## PROBLEM 8 [? marks]

LGs:

- Translate from Y86 to bit-level encodings.
- Explain the “program prep” code in a Y86 program. (Note: prep code includes both the code at the top and the bottom of a typical full Y86 program. Bear in mind that `ncopy.y8` is actually only PART of a Y86 program! It needs a main (and this prep code) to run.)
- Translate from *very* simple Y86 code snippets to C.
- Translate back & forth between Y86 and X86 snippets.

**[An exam question might actually have slightly harder code, as this is *very very* simple, although we wouldn't ask you to produce the prep code out of thin air. (We might ask you to explain it in some detail, however!) Finally, on an exam, this would be a great chance to bang on some x86 issues again, like whether you can clearly model how the ISA works in practice.]**

Consider the following y86 code:

```
mystery:
    pushl    %ebp
    rrmovl   %esp, %ebp
    mrmovl   8(%ebp), %edx
    mrmovl   12(%ebp), %eax
    andl     %edx, %edx
    jle      L5
    popl     %ebp
    ret
L5:
    mrmovl   16(%ebp), %eax
    popl     %ebp
    ret
```

- Translate this code into C.
- Translate this code into x86. (Few changes are needed here. You may want to practice with something more challenging!)
- One instruction used here is legal and correct in both x86 and y86, but a different, semantically equivalent instruction would probably have been produced in x86. What is it?
- Add the necessary “program prep” code to actually run the function on the arguments 0, 1, and 2, including stack setup! Comment each line of the code.
- The “tear down” code for the function is repeated twice. Rewrite the code (in Y86) to use only one copy of the tear down code. Which version is longer as bit-encoded object code and by how many bytes?

## **ADDITIONALLY:**

For pre-midterm goals (assessed but not as heavily as post-midterm goals):

- The midterm and bootleg midterm.
- Your fellow students' midterm makeup questions.

For post-midterm goals:

- Dutch's practice questions on pipeline modification. We **will** have a similar question on the exam, which will also likely require modifying (though not writing) small pieces of HCL code.

And, of course, your problem sets, the challenges to the extent that we discussed them in class, your tutorials, the slides, and your textbook readings!

As a reminder, we are only looking at 10.1-10.6 in Chapter 10, and there will be exactly two questions focused on Chapter 10, one targeted at each learning goal (one of which is “spoiled” on this bootleg).

This bootleg is provided without any guarantees or warranties. It may explode if handled improperly.

WARNING: bootleg exams are known to be addictive and can cause low birth weight if used before born.

SERIOUS WARNING: not everything you need to know is covered on this bootleg, but it is a good starting point. Look over the learning goals, especially the ones from Chapter 4 on, but to a lesser extent the Chapter 3 goals as well!

Oh, and post solutions to WebCT. What a pointless thing it would be to have a blatantly wrong answer to one of these questions and not give your instructor or fellow students the chance to correct it!

## **M CPSC 344/444 First Day Survey**

I designed this survey to draw together concepts from across my CPSC 444 User Interface Design course in term 2 of 2004–2005. (CPSC 444 was later split into a two course series: CPSC 344 and 444.) The short survey is filled with features, problems, and ideas that connect to key concepts later in the course.

I have included the un-annotated survey below. Separately below that, I have included the annotations from the document that explain its use to future instructors. The annotated version prints poorly because of small font size in Microsoft Word's comment "balloons".



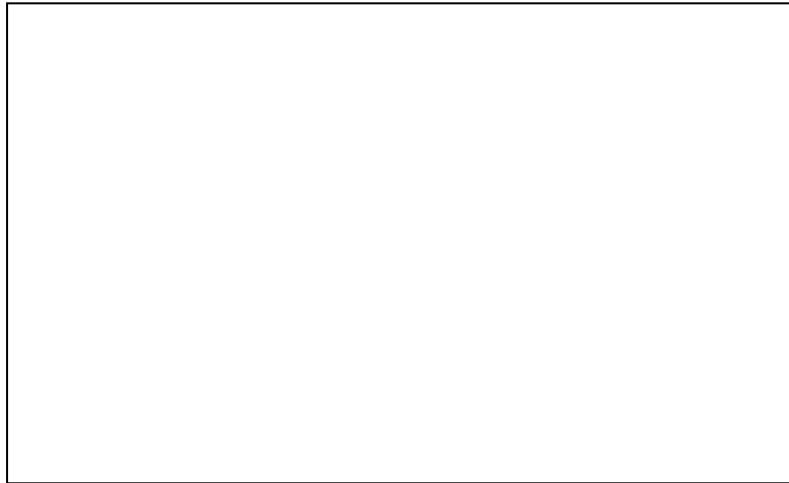
CPSC 444 2004/5  
Winter Term 2  
Wolfman

Full Name: \_\_\_\_\_  
Student ID: \_\_\_\_\_

## First Day Survey/Quiz

(to be completed individually)

1. Preferred name: \_\_\_\_\_
2. E-mail address that you read: \_\_\_\_\_
3. Sketch a self-portrait



4. Background experience:  

|   |   |
|---|---|
| <input type="checkbox"/> Co-op          | <input type="checkbox"/> Engineering          |
| <input type="checkbox"/> Other industry | <input type="checkbox"/> Design               |
| <input type="checkbox"/> Art            | <input type="checkbox"/> Software development |
| <input type="checkbox"/> Statistics     | <input type="checkbox"/> Other: _____         |

5. Working preferences (circle the number that best represents your preferences):

Finish as soon as possible    1    2    3    4    5    Wait for last minute

                                 All-nighters    1    2    3    4    5    9-5 only

                                 Open schedule    1    2    3    4    5    Many commitments

Answer the following multiple choice questions by circling the letters corresponding to the responses that you think correctly answer the questions. Please note that there may be multiple answers to any given question.

6. The majority of people with colour-blindness have difficulty perceiving:
  - a. Red
  - b. Blue
  - c. Yellow
  - d. Green
  
7. Someone with a score that is “two standard deviations from the mean” on an exam might have received a mark that is:
  - a. Lower than most others’ marks
  - b. Lower than a few others’ marks
  - c. Exactly the average mark
  - d. Higher than most others’ marks
  
8. What is one learning goal you have for this course?

---

---

---

---

## M.1 Survey Comments

1. *High-level comments:* Besides the individual ideas (and flaws) discussed below, the survey also serves as a starting point for discussion of surveys and other user evaluation techniques, the importance of sketching and lo-fidelity prototyping methods, the breadth of background that bears on HCI.  
With time, a possible follow-up exercise is to have students spread themselves out in two dimensions throughout the class according to their answers to Q5 a and c. Good chance to introduce themselves to others whose schedules might match theirs (for project groups).
2. *On the box in Q3:* The box affords containment: most survey users will draw their sketch inside the box even though nothing instructs them to do so. Furthermore, this communication of design intent from the designer to the user is natural and effortless. (Users don't think "what is the box for? should I sketch in the box?"; they simply do it.)
3. *On the "Other" blank in Q4:* Initially, the survey lacked a blank for "Other". Leaving it out may exclude part of the user population and definitely reduces the value of the survey. (There's no opportunity to discover new backgrounds of interest!)  
Unfortunately, I didn't prototype the survey (and discover the need for "other") until I had printed 70 copies. Those went in the recycling bin. Prototype early!  
For future reference, "psychology" and "technical or creative writing" would be good categories to add.
4. *On a response row in Q5:* Each of these 1-5 scales is a "Likert" scale. More on that later.
5. *At the beginning of page two:* Despite the lack of notice that there is a back, most people who take the survey in a class of others taking it will discover the second page. Why is that so when they are explicitly "working individually"? Many more discover it than the presence of multiple answers (below). It's because of the social nature of test-taking, even when working alone. We notice others flipping pages; we notice the general shape of text on others' papers. We respond by exploring our own papers.
6. *On the instruction "...may be multiple answers..." near the top of page two:* Some people miss this instruction, and many who read it do so only after reading the questions below (and noticing multiple apparent responses!). Is this "user failure"? No! This is a common response; we should know that our users will not read the instructions and accommodate it in our design. Furthermore, why expect multiple answers from a format that users have been trained has only one answer (negative transfer!). Also, why use an inconsistent format from the multiple answer question (about background) unless different behaviour is expected from the user? (Intentional inconsistency should communicate intended differences.)  
Data from this question can be used to test the "requirement" that the MC design

would cause 95% of users to give multiple responses to at least one question. The experiment is in the associated Excel file with artificial data. *Note: Excel file demonstrated the use of an appropriate statistical test for a proportion exceeding a target that is near 100%.*

Finally, performing this experiment brings up ethical issues. I did not get official permission from UBC for the “experiment” (probably not necessary in this instance). Worse, I did not obtain consent for the experiment (done to make a point; don’t do it again: ethical violation!). Worst, I intentionally deceive my users by “hiding” the MC instructions without disclosing that deception might occur (still an important point; still an ethical violation).

All of these “ethical quandaries” are minor compared to what students will run up against in their own course projects, which are probably, in turn, minor compared to those they’ll encounter at work.

7. *On the word “perceiving” in Q6:* Ducky Sherwood points out that this should really be trouble “distinguishing” rather than trouble “perceiving”. Practically, the change can’t be made (or it would telegraph the presence of multiple answers), but it can be discussed after the fact.

8. *On Q8:* Once designers “see the light” of user-centered design, it’s tempting to conclude that we just need to find our users and ask them what their requirements are.

Just as it was hard for many users of the survey to express their learning goals, your users will often not know or not be able to articulate their requirements! We need to find better ways to elicit users’ requirements.

## **N CPSC 344/444 Affect Lecture**

I developed these slides for my CPSC 444 User Interface Design course in the 2004–2005 academic year to introduce models of affect from Human-Computer Interaction research that could be used to inform interface design. My colleague Karon MacLean used and slightly modified the slides in the 2005–2006 academic year, and I used them again in 2006–2007. The lecture includes several stories and active learning exercises, which are annotated in the notes.

I also post links to Clifford Nass's, Don Norman's, and Rosalind Picard's research near the time that I present this lecture.

To save space, I have omitted all but a few key pages of the private slide notes. Many other slides were annotated with commentary, suggested discussion points, and quotes from and references to the source material cited on the slide itself.

cpSC 344: introduction to HCI methods

how affect affects us

class 19

Note: SW is Steve Wolfman, KM is Karon MacLean

Start by yelling at the students:

FIRST OF ALL, PLEASE BE QUIET AND PAY ATTENTION [then glare and pause for a moment]

Then, ask them to spend 30 secs writing about their response to that (emotionally, physically, cognitively); EMPHASIZE private, ungraded, unsubmitted.

KM, 2006: I tried this last year and wasn't able to pull it off. the only person I convinced was my TA, who happened to be sitting in – everyone else just snickered. \*i\* got pretty worked up, tho.

I wish I could think of something else I could try here which I feel more able to control.

today:  
affect, interfaces, and design

- we are emotional creatures
- we are emotional/social even when we think we aren't
- emotion and affect are critical to our performance
- we can model how emotion/affect work (at least a bit)
- we can consider these aspects in design

many insights in this lecture gleaned from Don Norman's excellent book "Emotional Design: Why We Love (Or Hate) Everyday Things", 2003

2

make the argument: if emotion is happening, and it happens even when we try not to do it, and it's important: then, we should consider it in design!

we are emotional creatures: the plank

fear inhibits action (and it's pre-conscious)

3

plank on ground, 3 meters up, 100 meters up example: fear inhibits action (and it's pre-conscious)

It's easy to simulate this exercise for students vicariously in class. Just "walk the plank" yourself on the ground. Then, move it onto the front row desk. Then, picture it as vividly as you can yourself and ask students to picture it 100 meters up.

(BTW, 3 meters is probably too high.. 1-1.5 metres is more appropriate. 3 is already terrifying.)



## aesthetics & a pair of ATMs

Kurosu & Kashimura developed two versions of the same ATM:

- one was attractive and the other unattractive.
- both had same functions, number of buttons, etc.

the attractive one was easier to use.

4

## gifts and brainstorming tasks

Alice Isen studied user performance on difficult tasks requiring creative thought (e.g., brainstorming).

users performed better if they were given a small gift before performing the task.

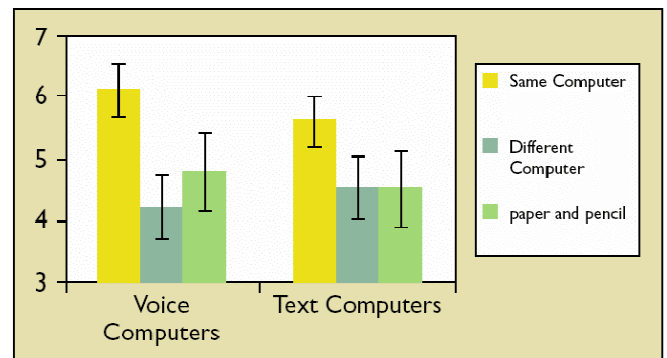
5

## we are emotional/social even when we think we are not

- “panic hardware” for emergency exits:  
in an emergency, people push on the door; if it doesn’t open, they push again, *harder*
- credibility and trust:  
people *say* they judge websites’ credibility on the basis of factors like identity of operator, presence of privacy policy, correction of false information, ...  
  
people *actually* judge websites’ credibility on the basis of superficial factors like appearance

6

## but we don’t treat computers that way... right?



positive ratings of the computer we’re talking to [Nass]

7

## but I don’t treat computers that way... right?

From the Nass study:

“We selected expert computer users to ensure the participants didn’t believe it was reasonable to be polite to a computer.”

8

## more examples of Nass’s “Computers Are Social Actors” paradigm

users will reveal more personal information to a computer that reveals personal information to them

US users help a computer that helps them more than a different computer

Japanese users help a different computer *of the same brand* more than a different computer of a different brand

users are more attracted to computers that “match their personalities” than to those that differ (e.g., extrovert vs. introvert)

users are more attracted to computers that exhibit a consistent personality (visual / audio features) than to those that are inconsistent

9

## the “Media Equation”: Nass’s research formula

1. Identify what is known about how the brain evolved to produce and understand speech.
2. Draw conclusions about how individuals respond to other people because of 1.
3. Have experimental participants interact with a speech system, rather than a person, and determine whether they exhibit the same attitudes and behaviors described in 2.
4. Draw design implications; highlight open questions.

10

## an apparently caring agent elicits more caring responses

Bickmore & Picard studied two sets of users of an exercise coaching system. **One set’s agent exhibited caring behaviours; the other’s did not.**

Subjects using the caring agents were **more likely to say the agent cared about them**, they trusted the agent, and they wanted to continue working with the agent.

At the end of their month of use, the caring subjects **were also more likely to say “Take care Laura, I’ll miss you”** rather than “Bye”

11

## emotion and affect are critical to our performance

Neuroscientist Antonio Damasio studied people with brain injuries that impaired their emotional systems.

They could describe exactly how they should have been functioning in the world, but they were unable to make decisions and function effectively: deciding where to live, what to eat, and what products to buy and use.

12

## attractive things work better

remember the two ATMs:

“happy people are more effective in finding alternative solutions and, as a result, are tolerant of minor difficulties.”

[Norman]

13

## why do we rely on emotion and affect?

Nass:

“Humans evolved in a world in which their most significant opportunities and problems, from food to shelter to physical harm, all revolved around other people. In this environment, there would be a significant evolutionary advantage to the rule: If there’s even a low probability that it’s human-like, assume it’s human-like.”

Norman:

“Human beings have evolved over millions of years to function effectively in the rich and complex environment of the world. ... Affect, emotion, and cognition have also evolved to interact with and complement one another. ... Affect, which includes emotion, is a system of judgment: good or bad, safe or dangerous. It makes value judgments, the better to survive.”

14

## we can model how affect and emotion work

15

## affect vs. emotion [Norman]

The affective system makes judgments and quickly helps you determine which things in the environment are dangerous or safe, good or bad:

The queasy feeling you experience, without knowing why.

Emotion is the conscious experience of affect, complete with attribution of its cause and identification of its object.

Anger at Harry, the used-car salesman, who overcharged you for an unsatisfactory vehicle.

16

## effects of affect

- muscles
  - tensing/relaxing
  - posture
  - facial expressions
- digestive system
  - “queasy” feeling
  - vomit response (eww!!)
- brain
  - release of neurochemicals
  - “tunnel vision”

17

## negative vs. positive affect & the brain

negative emotions  
“depth-first”:

- narrow thought processes
- high focus
- ruts and “tunnel vision”

positive emotions

“breadth-first”:

- broadened thought processes
- creativity & imagination
- leisurely
- sociable

18

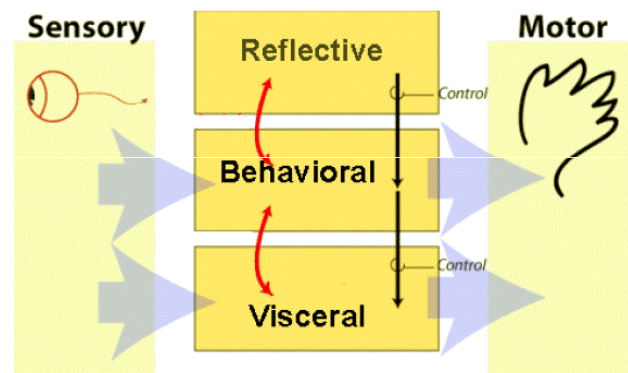
## industrial plant design example

design affect for a nuclear power plant

- target affective state under **normal** conditions?
- reinforcing elements?
- target affective state under **emergency** conditions?
- reinforcing elements?

19

## Norman's three-level affect model



Modified from Norman, Ortony, & Russell, 2003

20

## Norman's three-level affect model



**Reflective:** conscious, meta-cognitive, “sense/motor-free”, cultural, experiential

**Behavioural:** skills, non-conscious cognition, learned/practiced

**Visceral:** rapid, “thought-free”, evolved, universal (?)

21

### three levels in design: which is which?



think about it, then look at [http://www.jnd.org/dn.mss/CH00\\_Prolog.pdf](http://www.jnd.org/dn.mss/CH00_Prolog.pdf) 22

Have students match the teapots to the terms and justify the match. (May need to describe how teapot at right works.)

Reflective, visceral, behavioural

(left) reflective: An object of pride, reflection, and interest, linked to Norman's POET book.

(middle) visceral: A teapot that Norman found so beautiful, he couldn't resist it.

(right) behavioral: A teapot of exemplary design.

figure 0.3 The Ronnefeldt “tilting” teapot. Put leaves on the internal shelf (not visible, but just above and parallel to the ridge that can be seen running around the body of the teapot), fill with hot water, and lay the teapot on its back. As the tea darkens, tilt the pot. Finally, when the tea is done, stand the teapot vertically, so the water no longer touches the leaves and the brew does not become bitter.

The teapots also illustrate three different aspects of design: visceral, behavioral, and reflective. *Visceral* design concerns itself with appearances. Here is where the Nanna teapot excels—I so enjoy its appearance, especially when filled with the amber hues of tea, lit from beneath by the flame of its warming candle. *Behavioral* design has to do with the pleasure and effectiveness of use. Here both the tilting teapot and my little metal ball are winners. Finally, *reflective* design considers the rationalization and intellectualization of a product. Can I tell a story about it? Does it appeal to my self-image, to my pride? I love to show people how the tilting teapot works, explaining how the position of the pot signals the state of the tea. And, of course, the “teapot for masochists” is entirely reflective. It isn’t particularly beautiful,

Don Norman Last revised, March 23, 2003 00 Prolog: Three Teapots 4

## three levels in design

**reflective:** self-image, self-worth

**behavioural:** usability, task efficacy

**visceral:** user's "gut" reaction, user's affective state

what are some examples that work well?

23

## we can consider affect in design: "affective computing"

- cognizance of human affect
- measurement of human affect
- manipulating human social perceptions
- modeling emotion in computers
- projecting affective state

24

## cognizance of human affect

know what we respond to and design for them (remember the industrial plant example!)

Examples [Norman]:

- **positive:** bright, highly saturated hues; "soothing" sounds and simple melodies and rhythms; smiling faces; rhythmic beats; "attractive" people; symmetrical objects; rounded, smooth objects
- **negative:** heights; "looming" objects; darkness; empty, flat terrain; crowded dense terrain; sharp objects; harsh, abrupt sounds; grating and discordant sounds

25

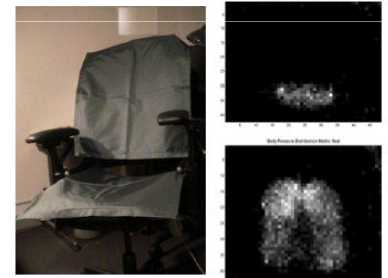
## measurement of human affect

with audio, visual, and other sensors, can gauge humans' affective state



from MIT affective computing lab:

- galvactivator glove
- posture-sensing chair



<http://affect.media.mit.edu/areas.php?id=sensing>

27

## manipulating human social perceptions

Nass's triggers of human-like responses:

- Language use
- Voice (including synthetic voices);
- Face
- Emotion manifestation
- Interactivity
- Perceived engagement with/attention to user
- Autonomy/unpredictability
- Filling of roles traditionally filled by humans

*note: we're working on how touch works into this, here at UBC*

28



## specific affective design tools

aesthetic design – e.g. colors, shapes, fonts, “feel”

agents – e.g. clippie

reassurance – information, e.g. that the transaction went through

useful error messages – give user a way to fix problem

anthropomorphism – display human-like emotions like happiness, empathy

engendering trust (how?)

29

## expressive tools: ways users can convey *their* emotion

text messages:

emoticons ;- ) - users co-opted

fonts and colors in messages – designers provided

language (word choice)

voice and video links:

tone of voice

facial expression

other examples?

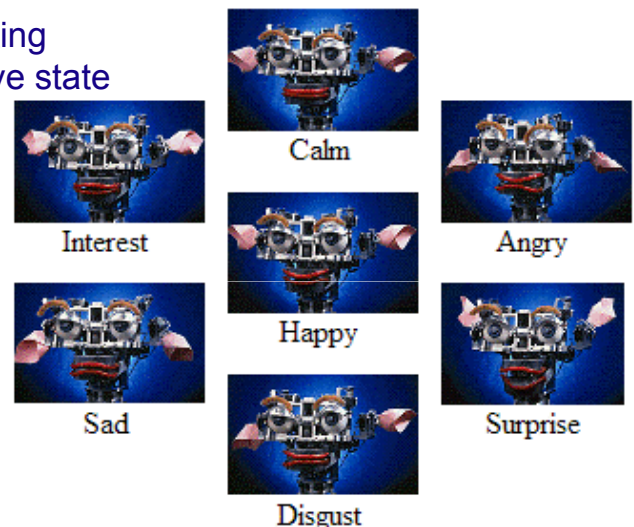
30

## modelling emotion in computers

| Prototype     | Function of the Associated Behavior           | Emotion Associated | Activation Conditions for Kismet                       |
|---------------|---|--------------------|--|
| Incorporation | Accept environmental stimulus                 | acceptance, calm   | Acceptance of a desired stimulus                       |
| Rejection     | Get rid of something harmful already accepted | disgust            | Attend to a salient but <i>undesired</i> stimulus      |
| Protection    | Avoid being destroyed                         | fear, distress     | Appearance of a threatening, overwhelming stimulus     |
| Deprivation   | React against important loss                  | sorrow             | Loss of a desired stimulus                             |
| Orientation   | React to a new or strange object              | interest           | Appearance of new or <i>salient</i> stimulus           |
| Exploration   | Explore environment                           | boredom            | Need of a desired yet <i>absent</i> stimulus           |
| Reward        | Reinforce beneficial behavior                 | joy                | Success in achieving goal of active behavior           |
| Destruction   | Remove barrier to achieve some need           | anger, frustration | Delay, difficulty in achieving goal of active behavior |
| Alert         | Startle Response                              | surprise           | Sudden, unexpected stimulus                            |

Kismet project: <http://www.ai.mit.edu/projects/sociable/emotions.html> 31

## projecting affective state



<http://www.ai.mit.edu/projects/humanoid-robotics-group/kismet/kismet.html> 32

## summary

- **affect and emotion play a key role** in our interaction with each other and computers... **even if we don't think so**
- affect and emotion are **critical to functioning effectively** as human beings
- we can **model how affect/emotion work**, including guidelines for what will trigger human-like responses and positive/negative affective responses
- we can **incorporate affect into design** by being aware of its impact on users or measuring, manipulating, modeling, and projecting affective states

33