Statement

In humanities fields, it is practically a cliché to say teaching is about inspiring young minds and conveying a love for a subject. In more technical subjects like Computer Science, however, these sentiments are seldom uttered and rarely believed. It is often argued that classes must first and foremost teach our students practical skills and develop the abilities they will need to obtain industrial jobs. Some will argue that introductory courses must be boring and tedious since it is only after enough basic information is learned that the more exciting material can be introduced.

I couldn't disagree more with those positions. I became a Computer Scientist because of a genuine love of the material, and I aspire to convey that to students when I teach. This is by far my most important goal when I design a course, prepare or deliver a lecture, lead or follow a discussion, create an assignment, or interact with a student. Our institution is not a vocational school, and that means our focus should be on inspiring our students to appreciate profound ideas and use them in creative and interesting ways. These are lofty goals that can only rarely be achieved. But we do our students a disservice if we do not strive to achieve them. This has been a guiding principle for me in teaching both undergraduate and graduate courses as well as advising students individually.

Although I make plenty of room for discussion and special activities, all my courses so far have been primarily lecture based. I enjoy lecturing, and believe it is the most effective way of teaching much of the material I want to teach. I try to make sure my lectures always tell a story. They have a clear beginning that introduces characters (often abstract ones), puts them in a troubling situation (a problem or a paradox we don't understand yet) and motivates why we should care about them; a middle that looks at the situation in different ways and considers possible solutions; and an ending that resolves the problem (but often introduces a bigger problem for next time). I believe humor and entertainment value are important parts of good teaching. I'm proud to say that I have never given a lecture without including a decent attempt at humor, and I nearly always succeed in at least making many of the students laugh out loud. Students almost always remember things better if they are connected with something humorous, and I strive to make my attempts at humor serve some pedagogical purpose. I also think it is important and worthwhile to describe the history and people behind an idea. Knowing a bit about the real people behind abstract concepts helps students relate to the material better and makes it more interesting and exciting. Telling stories about them often captures the students' imaginations and regains their attentions as they wane during the course of a long lecture.

Teachers are often torn between doing what is best for student's *learning*, and what is best for fairly and accurately *evaluating* students. In University courses, I believe fostering learning is far more important than evaluation. Hence, I encourage students to work together on nearly all assignments. I find most students learn best when they work with others, and the quality of coursework improves dramatically when students work together. I have recently begun requiring students to work together (often in pairs, occasionally in larger groups) and turn in a common assignment. To avoid the risk of students freeloading, I pseudo-randomly assign students partners and make sure they work with different people on every assignment. Although a few students complained about not getting to work with their friends at first, I believe they all ended up benefiting from the experience of working with people different from those with whom they normally interact. This has been especially true the graduate course I am currently teaching, in which about half the students are international. In the end, I do have to assign grades to students individually, so occasionally it may be necessary to slightly sacrifice learning for evaluation, but I avoid doing this as much as possible. I have found that a short comprehensive individual final (possibly given orally) is enough to fairly evaluate individual students who do most of their

coursework in groups. I have never given a closed notes or multiple choice exam or assignment, and I doubt I would ever do so.

I teach Computer Science because it is the field I find most intellectually compelling and exciting. I teach, in general, because I find teaching to be a most rewarding and satisfying endeavor. In addition to teaching Computer Science classes, I have taught private piano lessons, spent a summer teaching English in the Czech Republic, and coached a youth soccer team.

Project Focus

The focus of my fellowship project will be to develop and teach an introductory Computer Science course targeted to College and other non-Engineering School students. Despite its name, Computer Science has very little to do with the beige boxes we call computers, and it is far from being a science. It has more in common with music and mathematics than it does with science or engineering. At its core, Computer Science is the study of *imperative knowledge*. Whereas mathematics is all about declarative knowledge ("what is"), Computer Science is all about "how to" knowledge.

Most of Computer Science stems from three simple ideas:

1. You can define things in terms of themselves (*recursive definitions*).
2. You can treat procedures and data as one and the same (*first class procedures*).
3. When you give something a name, it becomes more useful (*abstraction*).

Although these ideas are simple, they have profound implications that it takes many years to fully appreciate. For example, defining functions in terms of themselves in interesting, but defining languages in terms of themselves opens up huge opportunities. Writing procedures that take other procedures as parameters is simple enough; writing procedures that create new procedures is incredibly powerful. Giving things like values, procedures and groups of procedures and data names is the core of modern programming languages. Giving other kinds of things names is an active research area, including efforts like aspect-oriented programming as well as my own research projects in annotation-assisted lightweight static checking and swarm programming.

The course will be designed to enable students to appreciate, use and understand these three ideas. Since we do have powerful machines for evaluating descriptions of computations, we would use computers to help understand those ideas. It is important, however, that the focus of the class is not on programming computers; computers are just a tool we can use to better understand abstract concepts.

As the main textbook for the course, I will use Harold Abelson and Gerald Jay Sussman's *Structure and Interpretation of Computer Programs* (known as the "Wizard Book"). This is not only the best Computer Science textbook yet written; it is the best textbook I am familiar with in any field. It is used in the introductory Computer Science courses at MIT, Berkeley and over one hundred schools (but not yet at UVA). Richard Feynman, Nobel laureate physicist, used a preliminary version of this text when he co-taught a course on computation.

Although it is said one should not judge a book by its cover, the cover of the Wizard Book reveals a great deal about its authors' perspective, especially in comparison to the boring covers found on other introductory Computer Science textbooks. The cover of the Wizard Book depicts a wizard and a sorceress. The wizard is holding a crystal ball with a yin-yang symbol depicting the interaction of evaluation and application in the meta-circular evaluator (an example of defining a language in terms of itself). As its cover suggests, the authors approach their subject as a magical

and wonderful thing, not the mundane practicality suggested by typical Computer Science book covers. This does not mean that the book, or the course I would teach, does not include practical ideas or teach real-world skills. Rather, it means that they are taught in a way that appreciates their intrinsic elegance and power instead of as merely means to an end.

In addition to the Wizard Book, I would use Douglas Hofstadter's *Göedel, Escher and Bach: an Eternal Golden Braid* as a source of additional material. This book was written by a Computer Science professor, but was intended for a non-technical audience (it won the Pulitzer Prize for non-fiction writing in 1980). It makes many deep Computer Science concepts interesting and accessible, and draws frequently from logic, art and music. I would use readings from this book to supplement the Wizard Book and provide background material for the course assignments.

Students would work on regular problem sets in pairs assigned differently for each problem set. Most of the problem sets would involve applying abstract concepts introduced in class to an interesting programming problem. Programming problems would be drawn from many fields such as music (higher-order procedures), art (recursive definitions), cryptography (parameter abstraction), sports (objects), and economics (tail recursion).

I had the good fortune to take 6.001 from Professor Sussman at MIT as my first Computer Science course. Although I'd done a lot of programming before coming to MIT and knew I liked programming computers, 6.001 was the course that made me appreciate what was really compelling and exciting about Computer Science and inspired me to become a Computer Scientist. Our field is new enough, that nearly all the most important and exciting ideas can be introduced in a first undergraduate course. Nearly everything I have done since, both in other courses and in my research, relates directly to something I first encountered in 6.001.

In designing my new course, I would draw heavily from what is done in 6.001, and try to adapt the best things from that course to the UVA environment and my own personal teaching style and perspective. My course would be intended for students with no previous experience in Computer Science, and I would hope to make the course accessible to all University students, not just those with strong math and science backgrounds. Although the course would be intended for students who are not Computer Science majors, I hope the course will convince many of these students to take more Computer Science courses and would prepare them well to do so. Our department is working towards offering a BA in Computer Science in the College of Arts and Sciences, and I would expect the course I develop to be the cornerstone of its curriculum.

Research and Teaching

One of the great things about being in a field as underdeveloped as Computer Science is that current research problems can be made accessible to students even in introductory courses. While most of education is about learning what is known, I believe it is often more interesting to learn about what is not known yet. I have incorporated my own research in all the courses I have taught. This is partly for selfish reasons – I hope to attract the best students to work on my research projects – but mostly because I find the research work I do exciting and interesting, and hope to convey that to the students.

One of my research projects investigates the use of annotations added to program source code to perform lightweight static checking. A small amount of effort spent on adding annotations that document programmer's assumptions, can lead to early detection of many types of program bugs and improve the quality or programs and efficiency of programmers. My research group has developed LCLint, a tool that embodies these ideas. Although LCLint was designed for industrial

use (and is actively used by thousands of industrial programmers), it has been used successfully in several external courses including introductory programming courses at RMIT University in Australia, the University of London, and Universidade Estadual Paulista in Brazil and special topics courses at MIT and Universidade Estadual Paulista. I am currently supervising an undergraduate student working on making a version of LCLint that is better suited for use in teaching.

Previous Teaching Experiences

I arrived at the University of Virginia in November 1999, and have taught here for three semesters. My first semester I taught a core graduate course on programming languages, next I taught an undergraduate elective course on security, and I am currently teaching a graduate elective course on programming languages. With each course, I have learned a great deal. I've dropped things that I found didn't work well, kept or extended things that worked effectively, and experimented with a variety of different approaches. I believe I've improved significantly each semester, and hope to be able to continue to do so. I have also found each course I have taught to be more enjoyable and rewarding for me personally than the last.

Spring 2000 – CS655: Graduate Programming Languages (23 students)
http://www.cs.virginia.edu/~evans/cs655-S00/

CS655 was a graduate course that was then part of our core curriculum required for all first year graduate students. Although the course had been taught before, I conducted a major redesign of the course, with the primary goal to integrate theory and design issues throughout the course. Previous versions of the course had focused primarily on language design, studying the history of programming languages, and left formal techniques such as operational semantics and type theory until the end of the course. I wanted students to appreciate formal semantics techniques as ways of understanding and analyzing language design issues, so I introduced some simple formal semantics techniques (e.g., operational semantics) early in the course. These were used throughout the course – for example, to show the meaning of types in a precise way, to investigate different forms of parameter passing, and to understand concurrency primitives.

Lectures frequently included group exercises where I would divide the class into small groups what would discuss and work on a problem and then present their results to the class. For example, in one class I divided the students into five groups and assigned each a programming language in which to design a Web browser. Groups presented their designs in a way that clarified the advantages and disadvantages of different programming language properties.

Of the innovative teaching methods I used, the most popular was a mock trial conducted over two class periods about two-thirds of the way into the semester. I selected a well-known language designer to put on trial, and students were given roles to play such as witnesses (impersonating authors of papers we had read) or attorneys. The remaining students acted as jurors and wrote position papers justifying a verdict and suggesting a sentence if their verdict was guilty. Students took the trial far further than I had expected, and it worked effectively as a way of leading students to look more deeply into some of the issues we had encountered in class and to get even the most shy and weakest English-speaking students actively engaged in and contributing to the class.

Although the course was a general introduction to programming languages and assumed very little background from the students (most has not taken a compilers course), I did include a large research component to the course. We read and discussed recent papers from conferences

including POPL and PLDI (the leading conferences in the field), and I strove to relate the formal techniques we learned in class to current research problems. For example, my second lecture on axiomatic semantics focused on proof-carrying code, and students did a problem set that used axiomatic semantics and type theory in a proof-carrying code illustration. Course assignments were divided between position papers (short essays) and written problem sets that involved proofs and problem solving. A major component of the course involved research projects, which students conducted in groups of four. Students were free to choose topics based on their interests, but I took an active role in advising the projects. At the end of the course, students presented their projects at a special evening class in the Rotunda's Lower West Oval room which I reserved for the occasion.

The course received very positive reviews from students (more than half the students rating it in the very top category, and the remainder in the second best category) and was described by one student as the best preparation to do research in his graduate career. (The complete reviews are found in the supporting documents.)

Fall 2000 – CS551: Security and Privacy on the Internet (46 students)
http://www.cs.virginia.edu/~evans/cs551/

Last semester, I developed and taught a new undergraduate course on security. For my PhD thesis, I had done research on code safety (restricting the allowable behavior of programs) so I was well versed in the system security area. I was not, however, very familiar with cryptology, the mathematics and art of secrets. One of the reasons I wanted to design and teach this class was so that I could learn about cryptology myself. I find that teaching something is by far the best way for me to learn it.

Course assignments included several problem sets (typically involving applying ideas from class to new and challenging problems), an open book in-class midterm (a partially encrypted version of which was handed out the week before), and a take-home final that involved using technical ideas from the class to solve an open-ended design problem (designing a national digital signatures infrastructure). Students also worked in groups on course projects of their own design. I encouraged students to find project topics that were interesting and relevant to the course, but also would have an impact outside. For example, one group's project was to assess the requirements of on-line course evaluation systems and design and implement a secure prototype. As part of their project, they analyzed the on-line system used by the School of Engineering for all course evaluations. With permission, they attempted to discover flaws in the system and did find several serious problems. These were reported to the developers of the system, and eventually led to a meeting with the Associate and Assistant Deans of the Engineering School and others involved in course evaluations. It was extremely gratifying to me to hear my students explaining both technical concepts and non-technical principles they had learned in class to the Deans as well as I could have explained them myself at this meeting. Several concrete changes to both the technology and process used for course evaluations resulted from the students' project.

Another feature of the course was a series of challenge problems that students could solve for extra credit, but mainly for the respect and admiration of their peers (and myself). Challenge problems included decoding a message encrypted using a cipher invented by Thomas Jefferson, analyzing the validity of claims in a VeriSign advertisement, proving the correctness of the RC6 encryption and decryption algorithms, and explaining the rationale behind some strange steps in the instructions for casting a Virginia absentee ballot. All of the challenges except the Jefferson cipher were solved successfully. Students who solved challenge problems presented their answers to the class. I also challenged students in the course to break into my account and

modify the encrypted course grades file. No one succeeded completely, but two students did manage to break into my account and explained what they did to the class.

By many measures the course was quite a success. In addition to the normal SEAS course evaluation, I conducted my own evaluation containing the kind of highly specific questions I feel are necessary to obtain useful information for improving future editions of the course and my teaching in general. Results from these surveys are included in the supporting documents. One of my students won a CRA Outstanding Undergraduate Honorable Mention (one of 27 awarded nationwide) partially for work she had done for her course project. Six students from the course are now working with me on research projects or senior theses. Several students have accepted interesting (and lucrative) job offers directly related to what they learned in the course. One of my students was on the committee to organize Engineering Week, and invited me to speak in a series of talks intended for the general public (the other speakers were Astronaut (and Dean) Kathy Thornton and an award-wining brewmaster). Ethan Miller, a professor at UC Santa Cruz, came across my course materials on the web and has used them to help in developing a similar course he is teaching at UC Santa Cruz. A course based closely on what I taught this Fall has been approved by our department's curriculum committee, and will soon be brought to the SEAS faculty for approval.

Spring 2001 – CS655: Graduate Programming Languages (10 students)
http://www.cs.virginia.edu/~evans/cs655/

I am currently teaching CS655. Although it has the same name and number as the course I taught in Spring 2000, because of changes in our graduate curriculum (that had been adopted before I joined the department, but not enacted in Spring 2000) it is no longer a required course. Because of this, the class is smaller (only 10 students) and self-selected to our best graduate students. This difference, and what I have learned from two semesters of teaching, has led me to make large changes to the course, and it is quite different from what I taught in Spring 2000.

Perhaps the most important thing I have learned in my semesters of teaching so far, is how to get a better sense of how well the students are understanding a lecture. I have also gained the confidence necessary to keep trying different ways of explaining things until most (or all in a small class) of the students clearly understand the most important concepts. Whereas a year ago, I felt the need to cover everything I had intended to cover when I prepared a lecture regardless of how confused the students might be, I now believe that it is much better to have students understand a few things well and deeply than to cover all the material I intended. I am very willing to spend the time necessary to make students understand the things I think are most important, even if it means some of the planned material will have to be dropped.

One example of this is a comparison of how I covered Lambda calculus in the first and current versions of CS655. Lambda calculus is a formal system invented by Alonzo Church in the 1940s. Terms in the calculus are generated by a grammar with only four productions and no primitives, and manipulated by straightforward substitution and reduction rules. It is incredibly simple and elegant, yet can be used to represent and reason about all possible computations. Understanding Lambda calculus involves deep concepts like what do numbers really mean and how can recursive definitions be understood. The entire system can be described on two slides, yet it takes many years to fully appreciate its implications.

In the Spring 2000 version of CS655, I planned to have one lecture on Lambda calculus, and I did manage to finish the lecture in the allotted class period. I realize now, however, that only a few students understood even a fraction of what I covered and none of them developed an

appreciation for the power and elegance of the ideas. This year, I again planned only one lecture on Lambda calculus, but it ended up spreading over four class periods since I took the time to make sure everyone in the class understood the most important aspects of the material. I believe all of the class did come to understand the essential ideas, and about half the class genuinely appreciated why I find them so compelling.

I held one of the lectures on Lambda calculus outside in Darden court. I was pretty nervous about doing this, since I have developed a dependence on projecting slides from giving all previous lectures that way. I found, however, that getting outside, and removing the laptop projector and whiteboard, produced a far more engaged and focused class than normal. Despite my fears, I found that I was able to successfully get abstract and complex concepts across just by talking. Unfortunately, the recent winter weather has prevented me from teaching class outside since then, but hopefully I will be able to again soon. I have, however, tried to adapt some of this to my teaching even when I have to teach indoors. I still believe it is worthwhile to create lecture slides, and I always make them available to students on the Web, but I spend a lot more time now with the projector off, just talking with the students or working things out on the whiteboard.

One of the gratifying things that has happened this semester, is that one of the students who took (and did reasonably well in) the Spring 2000 version of CS655 has been coming to meetings of the new course regularly. He tells me he comes both because of interest in the material, much of which is new this year, but mainly because he says my classes are the "liveliest" he has known. I take this as a high compliment.

Fellowship Goals

I hope a University Teaching Fellowship would enable me to do two things:

1. Develop and teach the new Introduction to Computer Science course. Although I plan to teach a course like the one I described regardless of whether or not I win a Teaching Fellowship, the Fellowship would help me teach it better. In particular, I would be able to spend time in the summer preparing the sequence of problem sets for the course. Preparing good problem sets takes a great deal of time, but often makes a huge difference in how much students get out of a course, as well as whether or not they develop a love for the material. The funds provided by the University Teaching Fellowship would enable me to devote the time necessary to prepare good problem sets during the summer. Being able to prepare more material before the course begins would allow me to increase the enrollment limit on the course.

2. Learn from a mentor who is one of the best teachers at UVA. I would be especially excited about having a mentor from outside the Engineering school, preferably in a field like Music, Art, Creative Writing, Drama, Linguistics or Politics. While I have been exposed to many of the teaching methods common in technical subjects from the courses I have taken myself from some of the best teachers in the field, as well as from mentorship from my colleagues in the Computer Science department, I am less familiar with the teaching methods used in other disciplines. I believe many of these methods could be adapted effectively to improve teaching in the subjects I want to teach. I also believe there is a strong connection between the core material of Computer Science, and that of many other fields. Computer Science courses can be made more interesting and more educational by drawing on examples and problems from other fields.

David Evans – Curriculum Vitae

University of Virginia
Department of Computer Science
151 Engineer's Way, P.O. Box 400740
Charlottesville, VA 22904-4740

evans@cs.virginia.edu
http://www.cs.virginia.edu/~evans
(804) 982-2218

Title
Assistant Professor (since November 1999), Department of Computer Science
University of Virginia, School of Engineering and Applied Science

Education

Massachusetts Institute of Technology, 1989 – 1999

SB and SM degrees in Computer Science, May 1994.
Ph.D. in Computer Science, February 2000.
Dissertation: *Policy-Directed Code Safety*, Advisor: Professor John Guttag

Courses

Teacher, University of Virginia, *Programming Languages* (CS 655), Spring 2001.
Teacher, University of Virginia, *Security and Privacy on the Internet* (CS 551), Fall 2000.
Teacher, University of Virginia, *Programming Languages* (CS 655), Spring 2000.
Head Teaching Assistant, MIT, *Computer Language Engineering* (6.035), Fall 1995.
Head Teaching Assistant, MIT, *Software Engineering Laboratory* (6.170), Spring 1995.
Teaching Assistant, MIT, *Computer Language Engineering* (6.035), Fall 1993.
Teaching Assistant, MIT, *Software Engineering Laboratory* (6.170), Spring 1993.

Research
My current research activities focus on the challenge of building and reasoning about scalable, secure and reliable software systems. Active research projects include:

Programming the Swarm. I am studying the problem of programming large-scale networks of devices with dynamic and unpredictable capabilities. Because this environment is so different from traditional ones, it requires a new programming paradigm that is substantially different from existing paradigms. Programming a swarm involves describing a range of desirable emergent behavior at the system level from which device-level programs can be derived. We are investigating the primitives for swarm programming, and how they can be combined. We are developing techniques for specifying the functional and non-functional behavior of swarm programs and reasoning about the behavior of programs constructed using combination mechanisms.

Annotation-Assisted Lightweight Static Checking. My research seeks to bridge the gap between standard industrial practices and formal techniques by developing tools and techniques that can be realistically integrated into industrial development environments. We have developed LCLint, a tool for statically checking C programs that provides a first step towards the adoption of formal techniques and mechanical analysis. If minimal effort is invested adding annotations to programs, LCLint can perform stronger checks than can be done by any compiler or standard lint. LCLint checking is designed to provide a clear and cost-effective payoff for any effort spent adding annotations.

Selected Publications

David Larochelle and David Evans. *Statically Detecting Likely Buffer Overflow Vulnerabilities*. Submitted to USENIX Security Workshop, 2001.

David Evans. *Annotation-Assisted Lightweight Static Checking* . The First International Workshop on Automated Program Analysis, Testing and Verification. June 2000.

David Evans and Andrew Twyman. *Flexible Policy-Directed Code Safety*. IEEE Symposium on Security and Privacy. Oakland, California, May 1999.

David Evans. *Static Detection of Dynamic Memory Errors.* SIGPLAN Conference on Programming Language Design and Implementation (PLDI '96), May 1996.

David Evans, John Guttag, Jim Horning and Yang Meng Tan. *LCLint: A Tool for Using Specifications to Check Code*. SIGSOFT Symposium on the Foundations of Software Engineering, Dec 1994.

Selected Invited Talks

Why You Should Be Paranoid About What Comes Into and Out Of Your Computer. University of Virginia, Engineering Week Talk Series, February 21, 2001.

Let's Stop Beating Dead Horses and Start Beating Trojan Horses. Panel Presentation, Infosec Research Council, Malicious Code Study Group, San Antonio, January 2000.

Systems for Safety and Dependability. Reliable Software Technologies, December 1999.

Grants

PI, NSF Career Award: *Programming the Swarm*, 2001-2005 ($285,000).
Co-PI (with John Knight), NASA: *Practical Use of Formal Techniques*, 2000-2002 ($608,422).

Students

David Larochelle (1999-), UVa PhD Candidate, MS expected May 200: *Methods for Statically Detecting Buffer Overflow Vulnerabilities.*

Andrew Twyman (1998-1999), MIT Meng 1999 (co-advised with John Guttag): *Flexible Code Safety for Win32* (winner of MIT Masterworks Prize).

Joel Winstead (2001-), UVa PhD Student, MS expected Summer 2001 (abstractions for concurrent programming).

Weilin Zhong (2000-), UVa PhD Student, MS expected May 2002 (security issues in swarm programs).

Undergraduate Thesis Advisees: Chris Barker (determining and enforcing checkable constraints for Unix/NT porting), Felipe Huice (designing and implementing a database backed web site), Jennifer Kahng (user interfaces for security), Mike Lanouette (adapting LCLint for educational uses), John David Loizeaux (modeling and predicting MEMS capabilities), Ryan Persaud (swarm programming primitives), Dan Rubin (security and politics of electronic voting), Adam Trost (applying swarm programming principles to soccer), Phil Varner (key distribution for electronic voting), Julie Vogelman (framework for web design experiments).

Supporting Documents

1. Departmental Endorsement from CS Department Chariman, John A. Stankovic.
2. SEAS Course Evaluation for CS655 Spring 2000.
3. SEAS Course Evaluation for CS551 Fall 2000.
4. Results from my course evaluation survey for CS551 Fall 2000.

Course Web Sites

All of my courses have comprehensive web sites. They include:

- A manifest for each meeting that describes the assignments and readings, and includes a list of questions that students should be able to answer after that day's class. In addition, they usually include a picture or quote relevant to the day's topic.
- Complete lecture slides (PowerPoint). Most of my 75-minute lectures use 30 to 40 slides.
- All the assignments, exams and solutions.
- The course syllabus and calendar.
- Links to relevant news articles (especially in the security course where there are frequently items related to class topics in the mainstream press).

The course web sites are:

Spring 2000 - CS655: Programming Languages
http://www.cs.virginia.edu/~evans/cs655-S00/

Fall 2000 - CS551: Security and Privacy on the Internet
http://www.cs.virginia.edu/~evans/cs551/

Spring 2001 - CS655: Programming Languages
http://www.cs.virginia.edu/~evans/cs655/