Programming Methodology and the Semantics of Programs July 2016
David Gries

My part of this summer course on programming methodology and the semantics of programs is designed to introduce you to semantics for programming languages that give insight into methodologies for the practical development of programs (or algorithms).

Visit this website for course materials: http://www.cs.cornell.edu/gries/July2016/

You will learn about calculational logic.

You will see Sir Tony Hoare's axiomatic basis for computer programs, as described in a paper published in 1969. It gives a definition of a programming language in terms of how to prove programs correct, not in terms of how to execute programs.

You will then see Edsger Dijkstra's *weakest precondition* definition of a programming language, which gives us insight into how to *develop* a program and its proof hand-in-hand, with the proof ideas leading the way. These ideas were published in about 1976.

Both of these definitions will be very formal, and you will learn that programs (or algorithms) can be *formally* developed —even "calculated". Learning and practicing these ideas will change how you *think* about developing programs. You will become much more efficient and accurate programmers, because you will use a methodology when developing programs. You will, no doubt, do lots of development *informally*, but it will work because it will be based on formalisms that you will have learned and mastered.

Below, we give you several programming problems. We suggest that you try to solve them *any way you can*, relying on your own skills at the moment and not just doing what someone else shows you. The idea is to have a *benchmark*; something that shows you where your programming skills are *now*, so you can compare them to what you can do a week or two from now.

Write the algorithms in any language you want. Try to argue why they are correct, because any professional programmer must convince others that their work is good.

1. Table of cubes. Write an algorithm that, given $n \geq 0$, creates a table of cubes: $0^3, 1^3, 2^3, \ldots (n-1)^3$. More formally, store values in b[0..n-1} to truthify the following postcondition. How fast is your algorithm?

(**forall** i: $0 \leq i < n$: $b[i] = i^3$)

Restriction: Do not use exponentiation or multiplication!

2. Saddleback search. Given is a 2-dimensional integer array b[0..m-1, 0..n-1], each row and column of which is in ascending order (see example to the right). It is guaranteed that a value x is in b. Write an algorithm to find one place where x occurs (there may be more than one). A row-major search finds x in worst-case time m*n. Can we do better? How fast is your algorithm?

b	0	1	2	3	4
0	2	3	5	6	6
1	2	4	5	7	8
2	5	5	6	8	8
3	5	7	7	8	9

Example. For x = 6, there are three possible solutions for the array to the right above: position [0, 3], position [0, 4], and position [2, 3].

3. Binary search. Assume that integer array segment b[0..n-1] is sorted in ascending order (although it is not necessary, as we will show you later). Assume that virtual array element $b[-1] = -\infty$ and $b[n] = \infty$ The algorithm can't reference these virtual elements; they are there only to help us describe the algorithm. Write an algorithm to find an integer k that satisfies the following postcondition for given x. How fast is your algorithm? Is it a binary search for x?

b[k] <= x < b[k+1]

4. Function fusc. Consider function f defined below. Write an algorithm to calculate f(n), for n >= 0. How fast is your function?

f(0) = 0 f(1) = 1
f(2n) = f(n) for 0 < n
f(2n+1) = f(n) + f(n+1) for 0 < n

About David Gries

It may help you to know something about your instructor for the first part of this course. I was born and raised in New York City —the borough of Queens. I majored in mathematics at Queens College, where my father was a professor of the classics —he taught Latin, Greek, Greek Mythology, French, and German.

I took only *one* computer science course, as a senior in college, in 1959, 57 years ago. We wrote programs in the assembly language of a fake computer. We wrote subroutines to calculate sines, cosines, things like that. We couldn't run them because we the college didn't have a computer.

Upon graduation, I got a job as a mathematician-programmer, as a civilian, for the U.S. Naval Weapons Laboratory, in Dahlgren, Virginia. They taught us Fortran in one week. I spent a great deal of time programming in the assembly language of the IBM 7090 computer. Most programmers were writing programs to handle the first weather satellite, to calculate ballistics tables, things like that. All numerical. People did not think of doing data processing on the computer much at that point.

In 1962, my wife, Elaine, who I met in Dahlgren, and I went to graduate school at the University of Illinois, she in philosophy and me in math. I had a research assistantship, which was to help two Germans write one of the first Algol 60 compilers, for the IBM 7090 computer. They designed the compiler; I was the lead programmer.

After a year, we went with them to Munich, Germany, where I finished the compiler and also got my PhD, in math, in June 1966. It was in numerical analysis—theory of norms.

Elaine and David 1964

Computer Science Departments were just starting then, and my Professor, F.L. Bauer, wrote the Chair of the Department a letter (there was no email), telling him to take me as an assistant professor. So I got my first academic position without an interview! Doesn't happen these days!

My research was in compiler writing, and I ended up writing the very first text on compiler writing, *Compiler Construction for Digital Computers*, in 1971. But I also got interested in the problems of programming, inspired by the first NATO Conference on Software Engineering in 1968 in Garmisch, Germany, where I met many famous people. I got to know Edsger Dijkstra, Tony Hoare, and others, and worked with them throughout my life. So, most of my research has been in programming methodology, semantics of programs, logics —anything that could help me understand programming and how to teach it.

In 1969, I moved from Stanford to the CS Department at Cornell, where I have been ever since except for sabbaticals and leaves at places like Munich, Germany; Oxford, England; Austin, Texas; and Athens, Georgia.

When I got my PhD in 1966, I told my father that I would never be an administrator, which he was at the time. Research and teaching were too important. But as we get older, we change our views, our perspectives. Here is something to remember: *Never say never*.

I was Chair of the CS Department for 5 years in the 1980s, and from 2003 to 2011 I was associate dean for undergrad programs in Cornell's College of Engineering. I "retired" in 2011, but I still teach because I like it and because our course enrollments are so high. I also served as Chair of the Computing Research Association in the late 1980's, when it changed from a small group of computer science chairs to a real organization that served the interests of computing research. I am proud of these contributions to the CRA.

Computing wasn't my whole life. I have a wife and twins (who are now 47 years old!). Son Paul teaches CS at the University of Toronto —we wrote a book together. Daughter Susan works for the State of Oregon, dealing with environmental and efficiency issues.

I used to play sports —American softball, bowling, basketball, golf, volleyball, etc. But as one gets older, the body doesn't respond the same way and one gives up these sports. Music has always been exciting for me. I used to play the piano, and my wife and I still sing in the Ithaca Community Multicultural Chorus. You can see a video of a song I wrote when a dean retired: www.youtube.com/watch?v=R-6MX7A611k

I first went to China in 1981. I taught the *Science of Programming* at Fudan University, Shanghai, for 3 weeks. Just what I am teaching you in this summer course. I have been back to China 4 or 5 times since then.