Oral History of Brian Kernighan

Interviewed by:
John R. Mashey

Recorded April 24, 2017
Princeton, NJ

CHM Reference number: X8185.2017

Mashey: Well, hello, Brian. It's great to see you again. Been a long time. So we're here at Princeton, with Brian Kernighan, and we want to go do a whole oral history with him. So why don't we start at the beginning. As I recall, you're Canadian.

Kernighan: That is right. I was born in Toronto long ago, and spent my early years in the city of Toronto. Moved west to a small town, what was then a small town, when I was in the middle of high school, and then went to the University of Toronto for my undergraduate degree, and then came here to Princeton for graduate school.

Mashey: And what was your undergraduate work in?

Kernighan: It was in one of these catch-all courses called Engineering Physics. It was for people who were kind of interested in engineering and math and science and didn't have a clue what they wanted to actually do.

Mashey: <laughs> So how did you come to be down here?

Kernighan: I think it was kind of an accident. It was relatively unusual for people from Canada to wind up in the United States for graduate school at that point, but I thought I would try something different and so I applied to six or seven different schools in the United States, got accepted at some of them, and then it was a question of balancing things like, "Well, they promised that they would get you out in a certain number of years and they promised that they would give you money," but the question is whether, either that was true or not and I don't know. But I had a good friend here at Princeton, and so after a bit of thinking, wound up here, and that has turned out, at least in hindsight, to be a pretty good, lucky decision.

Mashey: Hm. So and then what did you study here?

Kernighan: Well, I was in Electrical Engineering at the time. There was no Computer Science. There was, in fact, only one computer that I knew of on campus. This was in 1964, so computing had not spread. I don't think there were any Computer Science departments at exactly that time. It wasn't too many years before a few early schools started to have one. So Electrical Engineering at that time was mostly where senior faculty were interested in things like information theory and solid state physics and communications in general, and then there were some upstart young faculty who had done things in and around computing, but they weren't much older than I was at that point.

Mashey: So then what did you do your dissertation on?

Kernighan: <laughs> I did my dissertation on a problem which ostensibly came from practical computing. The idea of the practical question was we have a computer, it has a limited amount of memory and we are going to necessarily have to swap parts of the program in so they can execute and swap them out to make room for other parts to execute, and so the question is, "How could you organize your program, the code of the program, onto basically page-sized blocks so that you would minimize the amount of this swapping back and forth you had to do?" So that was the practical problem that ostensibly lay at the bottom of this, but the abstract problem of that is that I have a graph and that is nodes and edges and there are weights associated with the edges, and what you want to do is to take, say, a graph of this many things and chop it down the middle, so that as few of the edges that go across from one side to the other are cut. So it's called graph partitioning, and then that maps into a variety of other problems. How do you put people in offices so that they communicate with the people who are near them? How do you lay out circuit boards so that the wires between circuit boards are few in comparison to the wires that are on the circuit boards? So this one abstract problem had a lot of different specific, in theory, at least, practical implications. The interesting thing about it, it was very hard. We couldn't figure out-- <laughs> could not figure out why it was so hard. I was doing a lot of this work with Shen Lin, who was at Bell Labs at the time, and so we came up with a heuristic that made it possible to solve problems of at that time a reasonable size, reasonably effectively, but could never do anything that would sort of guarantee the right answer. This was in basically late 1968, and this was approximately then probably 10 minutes or so before Steve Cook came up with the notion of NP-completeness and the fact that there were some problems that were intrinsically hard and that we had stumbled into one of them.

<laughter>

Kernighan: And so it wasn't purely that we weren't bright enough, but there was actually some problem in solving them. So the thesis ultimately became a-- several parts, but one of them was the heuristic for this particular-- for the general case and then there were a couple of special cases where I was actually able to prove something about the properties so that you could come up with a number, a formula, for how long it would take to do an instance of that problem. So that was the thesis.

Mashey: Oh, good. Okay. So then you went to Bell Labs. How'd that happen?

Kernighan: I had actually been at Bell Labs for two summers before that, and how I got to Bell Labs for summers was one of these lucky things. I had spent the summer before even that, 1966, if you're counting, at MIT working in Project MAC, using the CTSS, the compatible time-sharing system. I'd gotten that job, I think, entirely because a graduate student from here had gone to MIT the year before and had done a great job and they had the illusion that everybody would do a great job.

Mashey: <laughs>

Kernighan: And so I got this summer job, which was a wonderful experience, it absolutely was. I worked with some great people, Corby [Fernando Corbato], for example, and Jerry Saltzer and a variety of others at that point and I got to use a computing system, CTSS, which was just a delight compared to, you know, batch card-based stuff on big IBM computers like the 7090 [I think he meant 7090], so it was just, it was a really, really nice environment with really nice people, but as part of it I sort of got to know indirectly, not in person, but indirectly, the people from Bell Labs who were contributing. This was all part of the Multics project at that point.

Mashey: Sure, mm-hm.

Kernighan: And so I knew them at least as names, if not actual physical people, and then the next summer, again, luck of the draw and so on, I wound up working at Bell Labs in Murray Hill for Doug McIlroy, but in the group of people who were working on Multics at that point, but I didn't do much that summer except work with Doug and then the subsequent summer, and this would be the summer of '68, I worked with Shen Lin on these combinatorial problems, and that became the thesis fairly quickly, and then I went to Bell Labs permanently after that because, gee, I'd had a good time, and so I never interviewed or anything like that.

Mashey: Oh, good, good. Yes. That's good luck.

<laughter>

Kernighan: No kidding.

Mashey: Okay. So all right. So then you started at Bell Labs. Then sort of what were-- talk about the research group you were in and what its sort of missions and ways of working were and how you fit into that and sort of what was it like getting into that particular group?

Kernighan: So partly I had already been in it, so I started as a naïve summer employee and by the time I got there permanently at least I had known, I knew many of the people, and I think for the first period of time I actually worked with Shen Lin more than anyone else. He and I worked on Traveling Salesman problem and Shen was just amazing guy for insight into how combinatorial problems ought to work. I think I programmed better than he did, so it was a good match, but it was in that same group of people and so I gradually drifted, I think, into things first related to document preparation, because I was interested in that but that was also a very active area of interest for people like Doug McIlroy and Rudd Canaday and Joe Ossanna and a variety of other people like that. How do you actually use a computer to print a document for you in a convenient way? Something that today people can't <laughs> I think appreciate how novel this idea was.

Mashey: <laughs>

Kernighan: But it was actually a lot of fun, so I did that and that was writing programs, but I was hanging out with the same people who were in the, doing-- who had been on the Multics project when Multics got stuck and Bell Labs dropped away. They had nothing to do, and so they started to work on basically creating new programming environment, which ultimately became Unix. So the environment was-- I would guess there were maybe 25 or 30 people in the particular group that I was in whose number was 127 at the point, and some of them were relatively call them somewhat theoretical like perhaps Al Aho. Some of them were very much programmer types like Ken Thompson, and almost all of them were very, very comfortable with software and then there were, of course, renaissance people like Doug McIlroy, who was interested in absolutely everything across the whole spectrum and expert in a lot of these things. So it was quite a-- it was a great environment, it really was. It was just great fun. Because you got really, really good people and the management structure was very loose. The way that management was done as far as I can tell was that once a year you had to write down on one side of one piece of paper what you had done and they used that to figure out how much they would pay you the next year.

Mashey: <laughs>

Kernighan: They took a very long view of this and so it was actually, I think, pretty effective, and there was no direct pressure to work on things that were relevant for AT&T communications and there was no particular pressure to publish either and so people kind of I think wandered back and forth between what we'd call doing useful things for the phone company and doing things that would be publishable computer science related work on the outside and I certainly did bits and pieces of both of those.

Mashey: So at some point in there you got into writing books. So why don't you start at the beginning of those and talk about how did you happen to write, like, "The Elements of Programming Style," for instance?

Kernighan: Yeah. So I think a recurring theme is luck and sort of being in the right place at the right time through no planning on your own part, but the first summer that I was at Bell Labs I was in an office on a corridor [off Stair 8]. I think I was sharing it with somebody. I'm not even sure now, and next door to it was somebody else. He came by my office, door's open, the very first day I was there at about eleven o'clock and he said, "Hi, my name is Dick [unintelligible]. Let's go to lunch."

Mashey: <laughs>

Kernighan: And I thought, "Well, okay. Dick, let's go to lunch," and so we went to lunch. I had no idea who this guy was, and after lunch he went off someplace and I went and snuck a look at the nametag on his door and it was Dick Hamming.

Mashey: <laughter>

Kernighan: Creator of Hamming codes.

<laughter>

Kernighan: Author of the numerical analysis text that I was using at Princeton, and I thought, "Oh."

<laughter>

Kernighan: "This is very interesting," and so I-- having Dick next door, you know, he was a curmudgeonly guy in some ways, but he was just absolutely wonderful person and he was always prodding people to do things, to do interesting things, and he came to-- and this now would be several years later, but he had always been saying that people didn't program very well and he used to-- I can still remember something along the lines of, "Kid, we give people a dictionary and a set of grammar rules and we say, 'You're a great writer. This is the way you learn to program.'" and he said, "That's wrong. There ought to be books on how to program well," and there weren't at that time any books on how to program well, and so he came in one day to my office. This was an ongoing discussion, but he came into my office one day bearing a book, a numerical analysis book, and he said, "Look at this. This thing is terrible," and I opened it up and I said, "Jesus Christ, this is terrible," and we were seeing totally different things. But what I saw was an example of terrible program, which is in fact the first example in this book, "The Elements of Programming Style." Very first one. It came from that conversation with Dick Hamming. I still have the book, and it was so bad that I thought, "Gee, there's got to be a better way to do that. Could we explain to people how to program better by something which was sort of a rip-off of the Strunk and White style? Here's bad stuff. Here's how you could do it better. How can you learn from that to make a better piece of code?" and so Bill Plauger and I-- Bill happened to be in the next office-- talk about luck. Bill Plauger and I worked on this for a while and we wrote a book on "The Elements of Programming Style." So that's how I got started writing the books. It was entirely due to Dick Hamming's prodding.

Mashey: <laughs>

Kernighan: And so, you know, some people had mentors and I had-- and Dick was a mentor in some ways, but he was also a prod, which was, I think, a very useful way to do things. Was kind of fun.

Mashey: Now, I have to admit, I'm reminded, I once visited Ravi Sethi's house, which had been Hamming's house, which Ravi got cheap because there were so many built-in bookshelves that the realtor said no one else would buy it. <laughs>

Kernighan: Yes.

Mashey: Yeah, so…

Kernighan: No. I still remember that house.

Mashey: <laughs>

Kernighan: Because Dick and Wanda were very, very kind to, you know, basically junior guy and his wife and we used to go there for social occasions from time to time and I remember the house, in the woods with raccoons coming down with a waterfall and everything like this, yeah.

Mashey: <laughs>

Kernighan: The good old days.

Mashey: Good. That was-- yeah. So you're talking about 1970-ish, right?

Kernighan: Right. Exactly.

Mashey: Yeah. Okay. So that was "The Elements of Programming Style," right? So what did you do next?

Kernighan: I think I had been writing FORTRAN programs for quite a bit. The programs that I wrote with Shen Lin for both the graph partitioning and Traveling Salesman program were FORTRAN programs. I had also written a FORTRAN program to typeset my thesis here. Princeton was sort of one of these kind of like Jerry Salzer's RUNOFF, only very, very simple. But it was written in FORTRAN because that was always available on the local machines, and so I was very good, comparatively, at writing FORTRAN but it was really a pretty awful language for the kinds of things that I wanted to do. It didn't manipulate characters very well, of course, but it also just, control flow was terrible, because this was FORTRAN 66, essentially, and so what I did in there somewhere was to write a translator that would basically put a veneer on FORTRAN, sort of C-like control structure on FORTRAN and I called it RATFOR for Rational FORTRAN, and so very simple preprocessor that-- and it actually was very nice. It made it possible to write programs in FORTRAN very effectively and easily, much more so than it had been, and it took on a life of its own for a while and eventually died, but one of the byproducts was then that Bill Plauger and I started to talk about, "Well, could you do a whole book about the sort of basic tools and ideas that were in Unix at the time?" This would be kind of the mid-'70s at the latest. "Could you do a whole book on the

tools but write them in RATFOR?" Because at that point C hadn't spread anywhere really, and so could we write them in RATFOR so that they would be something that the ordinary reader could look at and appreciate and understand because the language would be nicer? And so that's the genesis of "Software Tools," which Bill and I published I think in 1976.

Mashey: Well, that's in your stacks. Yes.

Kernighan: Ah, so it is.

Mashey: Yes.

Kernighan: And oddly enough, this is a copy Ravi Sethi left behind when he moved.

<laughter>

Kernighan: Okay.

Mashey: Okay. <laughs> Wow. Well, so the "Software Tools" sort of took on a life of its own, as I recall, right?

Kernighan: Yes, it did. Yeah, "Software Tools" was picked up, a lot of people used it inside Bell Labs. I'm not sure how far that spread but certainly it was used there and it was used extensively outside. There was a group of people that was primarily at Lawrence Livermore.

Mashey: Lawrence Livermore, yeah.

Kernighan: I think Joe Sventek and Debbie Scherrer and Dennis Hall, I think--

Mashey: Yes.

Kernighan: --I recall correctly, and they decided that they could make something much more robust and solid of the RATFOR package, and so they did a lot of work on the tools. They started something called the Software Tools Users Group. They were active as a sort of little side part of Unix conferences in various places, and they kept that going for quite a while. They did a lot of work on hard engineering, packaging, popularization, documentation. Lots and lots of things like that, and so it became surprisingly widely used. Now, eventually it died because C took over and Unix became widely available. There was

no need to write in FORTRAN in that environment and in other environments FORTRAN 77 was at least a little better than FORTRAN 66.

Mashey: <laughs>

Kernighan: Not perfect, but better. So RATFOR kind of died at that point.

Mashey: But that lasted quite a while actually. I mean, given a fairly limited scope that you had for it at the beginning.

Kernighan: Yeah.

Mashey: Yeah.

Kernighan: Right.

Mashey: Yeah.

Kernighan: I think it's still easier to use than FORTRAN XX, for modern X's although they have more capabilities.

Mashey: Hm, hm, hm. So okay. What was next after that then?

Kernighan: I think two things, and I don't remember the exact dates at this point. I probably should. Bill Plauger and I decided to do a version of "Software Tools" in Pascal. That was dumb.

Mashey: Oh, yeah.

Kernighan: <laughs>

Mashey: So talk about that some. And there's a famous memo that I recall of yours.

Kernighan: <laughs> Yeah. So Bill and I decided we would basically transliterate the FORTRAN programs into Pascal, and in some ways that was good because Pascal was in some ways nicer than

FORTRAN. For example, it has recursion and it actually had a sort of data structure mechanism of records, and so you could do a better job in Pascal in that respect, but it had an incredible number of really strange constraints I think imposed by compiler technology and so the order in which you presented the program to the compiler had very rigid and not very natural-- it was no better at processing characters than FORTRAN was, maybe even worse in some ways. Its input/output mechanisms were awful, and it was not extensible. So we managed to do the job on the book and got it out and the programs worked but it was really kind of an ordeal and so I wrote a memo about that called, "Why Pascal Is Not My Favorite Programming Language," basically enumerating the things where Pascal really wasn't up to the kind of task that we had used it for. I think it was a perfectly fine language for, you know, teaching beginning programming.

Mashey: It was pretty popular for teaching, right, at that point?

Kernighan: Extremely popular for introductory programming classes and I think with some minor reservations I think that would be a perfectly fine use for it. It was not suitable for system programming. Even though people would say it was, it simply wasn't, and the solution for, that was always offered for, "Well, here's a problem with it," "Oh, well, you just use this non-standard version of it," and I'm sorry. Using non-standard versions of things is a recipe for disaster because your standard's going to be different than my standard. So I think that wasn't a real thing. The reason it was such a stupid exercise in retrospect is we should've done it in C.

Mashey: <laughs>

Kernighan: It just would've been better. It would've sold probably a hundred times as many copies of the book.

<laughter>

Kernighan: But you make mistakes, and that was one. So fortunately, about, I would guess, about the same time, Dennis Ritchie and I talked about writing a book about C and that's--

Mashey: That one, yes.

Kernighan: --down there somewhere in this infinite pile. Yes.

Mashey: Yes. That one did--

<crew talk>

Mashey: That one did a bit better.

Kernighan: This one did a bit better, yeah, and I think it did better for a variety of reasons. One was it was the right place at the right time. Again, blind luck. The second one is that it came with the C reference manual, which was written by Dennis. I had not a word of contribution on that and Dennis was a spectacularly good technical writer. Ordinary writer too, but particularly for technical materials, and so the combination of a wave as Unix became popular and a very, very well-written reference manual. The only reference manual for the book at that-- for the language at that time. Made it a pretty successful book, and it's still selling. Its Amazon rank today's probably in the two or three thousands or something like that, so it's-- which for a book published originally in 1978 is doing okay.

Mashey: Yeah. <laughs>

Kernighan: And there was a second edition in 1988. So I think arguably the smartest thing, or luckiest thing I ever did, was to arm-twist Dennis into writing the book with me, so…

Mashey: I still remember learning C from the 20-page technical memo that Dennis had, which was enough and even better was looking at all the code.

Kernighan: Yeah.

Mashey: Yeah. So yeah, that one worked out better, yeah.

<laughter>

Kernighan: Definitely.

Mashey: Okay. But then you've done some other language things like AMPL. Talk about that.

Kernighan: Yeah. I guess there's two languages there. Let me talk about AMPL first, I guess. I don't remember even chronologically, but AMPL is a book for-- <inaudible>.

Mashey: Yeah.

Kernighan: AMPL is a language for setting up optimization problems. Linear programming is the classic example of an optimization problem where you have basically a set of constraints and you have to find the values and variables that will optimize some objective function given those constraints, and at that point in time, this was in 1983 or thereabouts, most optimization was done by basically-- the constraints represented with a very large, sparse matrix and it was a nightmare to fill those in by hand because it was large. It was sparse. There was no obvious pattern, and so people had created programs that would try to generate those matrices but it was not very well done. I think the best of them was a language called GAMS, for General Algebraic Modeling System, something like that, which is still around, but it was very FORTRAN oriented and we were past FORTRAN and so Bob Fourer, who was from Northwestern's Industrial Engineering Management Science Department was spending his sabbatical year at Bell Labs with, in particular, Dave Gay, who did numerical analysis, and I was talking to the two of them and Bob had very clear idea there ought to be a way to specify these optimization problems algebraically, sort of the way you would write them on the board, if you were trying to describe a problem to a colleague, and so we thought about the language and designed a rough approximation and I built the first implementation of that. It was my first C++ program. It was pretty crappy, but it was enough to prove that this was actually a viable way to do things and so we continued to develop that subsequently and at some point in there we wrote a book. I think the bulk of the text in the book comes from Bob Fourer and the AMPL program, the code itself, has been David Gay for a very long time. My contribution's essentially at this point zero to the whole thing, and this is the second edition of the AMPL book. There was a first edition that came out sort of 10 years earlier or something like that. So that's an example of a specialized language, a what might be called domain-specific or application-specific or little language or whatever. The idea is it's focused on a particular area rather than being a general purpose, "I'll take on anything," kind of language like C or FORTRAN, where-- and that was actually one of my, the things I'd been interested in for a long time, was this idea of domain-specific languages. Things where you can focus on a particular application and because you're doing that a lot of the constraints go away. You can invent syntax to taste and to make the language natural for whatever you're trying to do, and you typically don't have to worry about efficiency or anything like that. Doesn't matter how fast it runs because typically these things are going to be small, and so you can just have fun creating a language, and so RATFOR would be an example of that sort of thing. AMPL definitely was, and the other one that was done about the same time, I guess, digging into the pile here, would be AWK.

Mashey: Ah, yes.

Kernighan: Which <inaudible>.

Mashey: Yeah, talk about how that happened.

Kernighan: Yes. So AWK came about-- for historical accuracy there, the AWK, "A," is Al Aho, my colleague, and the "W," is Peter Weinberger, my colleague, and in fact, the three of us were in adjacent

offices at Bell Labs, which you do your best work with the people who are right beside you, so you want to live with people who are really good.

Mashey: Yeah.

Kernighan: I've been lucky about that too. But the genesis, there were several inspirations that came together on that. One of them was a language that was done at Piscataway by Marc Rochkind who you may remember.

Mashey: I remember him[1].

Kernighan: And he had a language. It was just basically a set of regular expressions that was compiled into code and then you ran data past it from something and if one of the regular expressions triggered, it said, "There's something wrong in the data that matches that regular expression." So it was a very specialized language but using the power of regular expressions to identify or validate the data <inaudible>. A really, really, really nice idea, and so that was one of the ideas in AWK. Peter Weinberger had been interested in basically simple data processing kinds of things for a long time and he actually, I think, understood RPG, the report generator language that was created by IBM that none of us understood really. But was very effective and powerful in certain environments, and then Al, of course, was interested in regular expression technology. He had just finished doing EGREP, and I was just kind of interested in languages that let me handle numbers and text with more or less equal ease or fluency or something like that, got me back to this word processing kinds of thing. And so the three of us with these kind of related but not the same interests and being in adjacent offices, talked a bit about what would a language look like for that kind of thing? I don't think we talked very long. I would be surprised if it was more than a week or two, and then Peter Weinberger wrote the first implementation over a weekend using the tools we had, YACC and LEX. I'm not sure he used LEX. Maybe. I just don't remember at this point, and then people started to use it, and so it's been maintained ever since. There's multiple versions of it, of course. The GNU version, GAWK, is by far the most widely used, but I still maintain one that's used by a lot of people and there's one in BusyBox, and a lot of other places have done it as well. So it's a nice example perhaps of one of the first scripting languages. There was sort of a 0^{th} generations with things like SNOBOL and so on.

Mashey: Yeah, hm.

Kernighan: And the shells and then AWK was the, in some sense, the next one where it was a language where the various components were more or less of equal stature rather than the sort of mix and match stuff that you get in the shells. Yeah.

[1] [Editor's note] Marc was in the office adjacent to the interviewer at PY.

Mashey: Yes. Well, certainly I ran a project that used it rather strongly, right?

Kernighan: Mm-hm.

Mashey: And I remember, I think I left you a printout once upon a time that was a surprise. <laughs>

Kernighan: There were very large AWK programs.

<laughter>

Kernighan: Embarrassingly large AWK-- it was never-- <laughs> the language was meant for one or two-line programs, something where, you know, what you're trying to do should only take a couple of lines in a decent language, and then people, perverse people, came up with things that were noticeably bigger. I think the largest I ever saw it was something like 20,000 lines, which is just absurd.

Mashey: Well, we had close to that, yes.

<laughter>

Mashey: Ah, and it was extraordinarily useful, so that was a good thing. So let's try some of the other efforts in there. Let's go back to the text processing, yeah, with, DITROFF and related things. How did that all happen and…

Kernighan: So as I said, I was interested in text processing. I wrote this program here for doing my thesis. I had seen RUNOFF at MIT in the summer of '66. When I came to doing my own thesis I was too cheap to pay somebody to type it because I knew it would be done over and over again and so I wrote this program that would create mine. Got to Bell Labs for real and somewhere in there discovered that there were other people who were very interested in it as well and particularly Doug McIlroy, and so Doug had done something called ROFF and Joe Ossanna did something called NROFF, which was New ROFF, which was more programmable. It didn't have everything sort of built in, issues like, "How do you do pages?" or, "How do you do multiple columns?" and so on, but rather it was programmable. It was Turing complete in a formal sense, and so that meant that you specified things like, "When do you want page breaks? How do you do--" lots of things, by writing code in a truly awful language.

Mashey: Yes.

Kernighan: Truly awful, but manageable language, and then that produced output on typewriter-like devices, so it was very limited in that sense, and at some point Joe Ossanna and colleagues convinced management to buy them a typesetter that is a machine that would actually produce proportional typeset of various sizes, of high quality, and then he wrote a program called TROFF, which was basically NROFF but with all the extra things you'd need to do multiple fonts, multiple sizes, and proportional spacing. But with the same horrible language for actually <laughs> controlling what went on the page, and so Joe wrote this program and then unfortunately he died in I guess very, very either late '76 or early '77.

Mashey: Yeah.

Kernighan: Something like that. Leaving this program, which was pretty inscrutable because it had been transliterated from assembly language, and so it was really kind of a mess inside, and it lay fallow for a while and then I thought, "Okay. There's things I would like to do. TROFF can't do them. I'm using it to produce books and I can't make it do some of the things I need to do." So I, with a bit of fear and trembling, started very carefully to modify it to make it independent of the particular output device. It had been tied very strongly to one specific typesetter and now there were different typesetters available and so I-- it was limited by 16-bit architecture. Now we had some 32-bit architecture, and so I gradually modified it slightly to make, and particularly the typesetter a parameter, read-in tabular description of what the properties of the typesetter were, what characters were available and things like that, and then was able to in that way come, produce what I called, Device Independent TROFF, and that's sort of the one that then was used for a very long time for all kinds of things. And I've used that for essentially every book I've ever written. It's the devil you know. Not because it's better but because I know it.

Mashey: Sure.

Kernighan: <laughs> So it was-- I haven't touched it and don't even have the source code for it in any useful form at this point, but somewhere it was picked up by James Clark and he did this version of it called GROFF and then related things for the mathematical typesetting and tables and so one and it's a wonderful job and it's kept TROFF alive and very useful because he got rid of an awful lot of the funny limitations but maintained compatibility with the past, so that's really what I've used for newer books is GROFF.

Mashey: Hm, hm. Yes. Well, having been a user of those things, despite their horrific programming language, it was pretty useful. Yeah. So… <laughs> Okay. So that was, yeah, in some sense yet another little language, one way or another.

Kernighan: Yeah. I guess one of-- probably worth mentioning. Inside the typesetting aspect of things, the typesetting language, TROFF in particular, made it possible to do ordinary text. You know, the kind of text that would show up in a detective story or a romance novel or a newspaper, right, just words on the

page. But a lot of the documents that were written at Bell Labs were scientific papers and they tended to have a lot of mathematics in them, and so mathematics was very difficult to do. You could sort of fake it in NROFF with half-line up and down motions and it was just appallingly bad, and so Lorinda Cherry and I worked on a language called EQN, which made it possible to describe mathematical expressions in a form that's almost oral so that you would say things like X sub I. If I say X sub I to you, you have a perfectly exact picture what that means. There's an italic X and a little down from it is an italic I, things like that, and so we wrote a program that would translate that stuff, X sub I equals pi over 2 or something, into TROFF commands that would actually cause it to print properly on the typesetter, and so that worked out extremely well. It was done as a separate program because TROFF was already too big. I mean, was as big as you could have a program in the PDP-11's that we were running, so it had to be a separate program. It took advantage of the newly developed pipe mechanism so that you could take the output of EQN, pipe it into the input of TROFF, and it was created with the newly invented YACC parser generator, compiler-compiler that Steve Johnson and Al Aho had created. So it was all these things coming together in a way that would've been impossible if any of the pieces hadn't been there, and so EQN in some sense after RATFOR would be the next example of a special purpose language, but not a programming language. A language for describing mathematical expressions. That worked out, I think, really, really well. That's the impetus for the mathematical component of TeX, which everybody uses today. That math mode in TeX derives from EQN, then uniformized and made much more general and, of course, very, very well done by Don Knuth. This was also the inspiration in, loosely, for Mike Lesk's TBL program that did tables. Another example of a specialized area so you can get a notation that's really natural for that particular area, and so Mike created this program that made it possible to specify tables, and it worked, so you could have mathematics inside tables, and then "Gee, that's fun," and so I did one, as we got a better typesetter, I did one for doing pictures. That is line drawing kinds of pictures, flow charts or organization charts and that kind of thing. Called PIC and so a specialized language and again, not the same syntax as basic document preparation or mathematics or tables, but for pictures. So that is another instance, again, of a special purpose language.

Mashey: And kind of when did PIC get done?

Kernighan: Ah, I will guess the mid-'80s.

Mashey: The mid-'80s, yeah.

Kernighan: Can't remember. No. It must've--

Mashey: It's early '80s. Early '80s first made.

Kernighan: Yeah, it's got to be early '80s because it was done at the time we got the Linotron 202 typesetter, because that device had enough resolution and enough speed that you could actually do the

kind of almost plotting by dots that was necessary to produce the output on, at that time, photographic paper and then later on laser printers. But it predates laser printers and it predates things like Postscript, let alone PDF.

Mashey: So let's see. During the mid- to late-'70s, you mentioned 16-bit and 32-bit. There was a big portability effort going on in that lab, right? Could you talk about that some?

Kernighan: I don't remember a lot about the portability. There definitely was. One of the components of it, that C from the beginning did not specify very much, if anything, about the sizes of the operands. The idea was that "char" would be a byte and "int" would probably be two bytes, but could've been four, and "long" probably were four but might've been bigger and it sort of depended-- it totally was unspecified in the language, and that meant portability was, in fact, a problem. I think the big portability effort that I remember was mostly Steve Johnson writing the Portable C Compiler. And there the issue was not so much the portability of the language of portability, of the compiler and because the compiler itself was portable then it meant you could also in portability the operating system. So there was a decent chance that by retargeting your C compiler you could get your operating system to run on a different computer. The sizes of integers and the endianness and all of those kinds of things played into that. But there were a lot of different moving parts to do that kind of portability and I wasn't really involved I think in any of them.

Mashey: But think at least from watching it, it seemed like that was some of the solution to the earlier problems of software tools not having an environment around them became easy to move UNIX than it was to--

Kernighqn: Yes.

Mashey: -- put things on top of a bad environment.

Mashey: Yeah.

Kernighan: Yeah, definitely. 'Cause a lot of the commercial operating systems at the time were not very good and they were very different from each other. And so as UNIX became more widely available then that meant it you could-- and became-- because it was portable at that point, then you could take all of your accumulated software and drop it on UNIX on a new piece of hardware. And I think that helped a great deal with the minicomputer marketplace. And, you know, Sun was an obvious beneficiary, MIPS was an obvious beneficiary--

Mashey: Yeah.

Kernighan: -- and quite a few others. So that was one of the waves that kept all of this stuff going.

Mashey: So- so then in somewhere in here you got to be addressing this sort of overall programming environment issue. So why don't you talk about what you were saying there. There's a book, right? <laughs>

Kernighan: Was there a book? Oh, yes, there is.

Mashey: There's a book.

<laughter>

Kernighan: I-- that's right. Yes. I'm trying to remember. More advertising.

Mashey: Yeah, you do get to advertise here, so go ahead.

Kernighan: Okay. Glad to advertise. Yeah, Rob Pike and I had been thinking about, I mean you and I had been thinking about how you write programs and how you-- how is this environment productive and why is it more productive than some of the other environments in trying to explain things like programs that could be connected to each other with pipes? All of which were kind of novel ideas from the commercial side of things, but were just the way you did things on UNIX. And so at some point Rob and I decided we would try and write some of it down in an orderly fashion for what we knew about UNIX. So you could argue that in some ways rather than saying software tools, go build the tools, instead say here are the tools, how do you use them well in this particular environment. So how do things like pipes work? How do you use the shell to glue things together? How do you organize the data in the file system? And build a lot of small tools, but they're all pretty small, none of them are very giant. The biggest one in there is probably the calculator program at the end, which is an effort to show how you build something with YACC, the parser generator, that is a reasonably serious but simple language design. So it was kind of a manual of how to think about programming in this particular environment, how to do it particularly well. And I think it's still useful, although at this point the world has changed a lot. This was done before networking barely came along and so networking changes a lot. It was also I would say sort of done before screen terminals really were everybody's desk. I don't remember the exact dates, but again, there were lots of things that became different after around this time. So it's not clear at all how one would, for example, update that book.

Mashey: So that sort of gets us to early '80s. Anything, anything missed, is particularly important from the '70s and early '80s?

Kernighan: You know, the big thing that happened somewhere in there, I got sandbagged for a while by being in management. I'm sure you remember days like that too.

Mashey: <laughs>

Kernighan: And but, again in a sort of lucky thing, one of the peep-- I was moved into management in sort of probably early 1981, something like that. I don't remember. And the management at Bell Labs was not management the way it is in most places. It was, you didn't tell people what to do. It was more like trying to understand what they were doing so that you could relay it to others. And in some ways-- and it was partly the umbrella theory of management, it keeps stuff from coming down on their heads. But one of the people in my first group of people was Bjarne Stroustrup and he had joined Bell Labs probably late '79, early '80, something like that, and wanted to do something that would be an improved version of C. Something that took advantage of object oriented programming, because he had some things he wanted to do with it that would be hard to do otherwise. And so Bjarne was in my department for essentially all of the development of C++, starting in all of early '81 when it was just starting to exist as C with Classes and then became C++ in roughly 1984. And then when AT&T split in '95 or something like that Bjarne went off to AT&T Labs in Florham Park. And, but I was sort of his management chain at that time for a long time. So I got to see—I had nothing to do with the creation, but I was present at the creation for C++.

Mashey: <laughs>

Kernighan: So I would say that's an important aspect of something that happened in the early '80s primarily and it went on for quite a while thereafter.

Mashey: Okay. So then talk about the rest of the '80s what-- where you are in management.

Kernighan: Ha. <laughs> You know, <laughs> I have no desire to be in management in some sense. I don't remember a huge amount of what was going on at that point. If I look at this is in some sense this is parts of what was going on in that period of time. But I think the combination of management and there not being so many really brand-new interesting things going on. That it wasn't quite as much, wow, that we can do something new every-- that the low-hanging fruit have gotten somewhat further off the ground in a sense. But I went through a phase where I was very interested in user interfaces. So I worked with Tcl/Tk and was part of a group that built a system for designing indoor wireless systems for AT&T. It never-- I don't think it ever went public, but did a lot of that. Shen Lin and I did some work designing systems that were for optimizing AT-- networks for AT&T's customers. Again, that never went public. That was used by parts in the company but didn't go anywhere outside. So not the same sort of thing. Part of this, something I mentioned earlier this back and forth between stuff that's visible on the outside and things that are actually mostly for the inside.

Mashey: Okay. So let's see, so then you are at Bell Labs until what, 1999?

Kernighan: Yeah, roughly that.

Mashey: Yeah. Okay. So talk a little bit about how the Bell System rearrangements affected things.

Kernighan: <laughs>

Mashey: If it's not too painful.

Kernighan: Parts of it are definitely I think painful. It was clear at some point; many of the dates are kind of fuzzy at this point. One I remember fairly clearly, it must've been in very early 1995. Maybe it was actually announced in 1994. My wife and I were in-- on vacation and we were driving through Pigeon Forge, Tennessee, the home of Dollywood. And on the radio it said AT&T has decided it's going to split into three pieces. And this was a bit of a shock. And so unlike most of our vacations, we raced immediately to a large town to see if we could find a paper that would talk about this particular story. <laughs> I think AT&T had become sort of different after divestiture originally. They split up in 1984, had a big effect and that took away colleagues who went off to Bellcore. And then the next split in '95 or '96, whenever it was, had another big effect and it took what was the small-- a small piece and split it even further and destroyed a lot of working relationships. And I think the nature of the whole business had changed quite a bit at that point. Bell Labs used to be funded by a tax in effect on telephone calls in the United States. AT&T was a very large company; it was a regulated public monopoly. It had a guaranteed rate of return on its investment and the quid pro quo was that it would try to continuously improve the telephone service in the country. And Bell Labs was a piece of that, an important piece, of improving the telephone service. And the research part of Bell Labs, which is where I was, was again, a small piece of that. But it meant we had stable funding and a very stable environment, and an environment that encouraged people to explore things without pushing them too hard. It was not on the basis what have you done for us this quarter, but rather more what have you done over the last five or 10 years, something like that. At least that's how it seemed. But all of that had changed a lot by the time we got to let's say, the mid '90s. And it was a great deal more of thinking about business and business units and how the-- what you were doing relate to the profitability of the company or something like that. And I think most of the folks in research weren't very interested in that kind of thing, not very motivated by it and often fairly hostile or at least skeptical about a lot of the business aspect. Not that they aren't important but they didn't seem to be well aimed at the time we were there. So I think a lot of people were kind of restless and unhappy. And then the actual real splits where people went to totally different companies then and then couldn't talk to each other or things like that. I think all of those contributed to shrinking of the research environment a great deal and just making it a lot less fun place to be in many respects. And so I think somewhere in the mid-90s people started to look for other places. There was a-- and over a decade there was a diaspora. people went a lot of places. I spent one semester teaching at Harvard. Bell Labs was very generous about that. You want to go somewhere in effect on a sabbatical. I-- so I spent a semester

at Harvard teaching a very big intro computer science course and that was a lot of fun. And I'm not sure that what I learned there was of any value to Bell Labs, but they were absolutely kind about letting us do that. And, but that was useful for me because what I learned from teaching a very, very large course like that was that I could handle all the administrative stuff. All of-- all of the issues like personnel and setting exams and who knows in an academic setting. And therefore, when Princeton sort of suggested I might want to spend a year as a visitor or something like that and then come permanently, it was a lot easier. 'Cause I thought yeah, I've done some of that already. It's not a leap into the total unknown.

Mashey: Okay. So then you came down to Princeton. And what did you start teaching?

Kernighan: So the course I had taught at Harvard was called CS 50. It's still around.

Mashey: Yeah.

Kernighan: And run by David Malan. It's an amazing course. David has done wonderful things with it. It's the largest course at Harvard. So one of the things I noticed when I was there, I had 450 odd students in it, so it's a big class. David's is much bigger. And what that class included everything from freshman kids who've been programming basically in utero. I mean they just did through to terrified senior English majors who had to satisfy some or other requirement.

Mashey: <laughs>

Kernighan: And I decided that the senior English major end of the spectrum was poorly served. And so when I got to Princeton I said in the fall well, I'd like to try a course for very non-technical people, sort of the things that I would have tried to do it at Harvard course, but spread out over a semester and aimed at a population who were not technical. And so that's what I did. Matter of fact, I've taught that course for a very long time ever since, essentially _____.

Mashey: And that's what led to the most recent book? Is that right?

Kernighan: Yeah. Yeah. Yeah, the "Understanding the Digital World" book. Yes.

Mashey: Yeah.

Kernighan: Yeah. Yeah, I taught the course for quite a long time. I never found a book that I liked for it. Most of the books were these, you know, very brightly colored things that would be suitable for, you know, elementary school are not so wonderful. And so I just decided that I ultimately rather than using my note--

own notes over and over again, I would try and write a book about it. So I took a sabbatical year and wrote the first draft of that book, which I self published. And as I said earlier, that's a recipe for total invisibility. When you publish it yourself it's gone.

Mashey: <laughs>

Kernighan: But then after five or six years the world has changed quite a bit and a lot of new things worth talking about. And so I took another sabbatical and updated the book and this time it's published by Princeton University Press, which is, you know, a fine university press and they know how to do things like production and publicity in a way that the amateurs just don't know how to do. And so yes, that's the genesis of that book.

Mashey: So talk about what you cover in that.

Kernighan: So--

Mashey: And therefore, like teaching in the course. Yeah.

Kernighan: Yeah. So the-- what's in the book, I mean I take the general field that we're in and say let's talk about hardware, software, communications. and the overall premise is that computing and communications are just everywhere, an integral part of our lives and getting to be more so over the-- as time goes on, much more so than they were certainly when I started teaching this course in '99 or something like that. So, hardware, software, communications. Hardware is the tangible stuff, you can put your hands on it and if you dropped your computer on your foot you'll notice it. And that's the place to talk about things like the remarkable advances in technology, the physical technology, Moore's Law and things like that. Why is it that we can make these amazing devices that keep getting smaller and cheaper and faster sort of exponentially? And that's a really interesting story. It's a chance to hand out gadgetry in the class so that most people never seen the inside of a computer let alone the inside of some of the pieces in the computer, and some of those are still actually sort of interesting. So there's that. And then you can talk about how computers actually represent information, how do they store it and things. And this is where you can talk about bits and bytes and binary. And you can talk a little bit about how computers work at the low-level, the idea of an instruction set. Some part of the computer that's taking one instruction at a time and doing something and then going back and getting the next one. And so the, you know, the fetch, decode, execute kind of cycle. Very simple, not abstract but not very deep technical kind of content, sort of that's how computers work. So then the software side you can talk about a couple of things there. One is the sort of more theoretical end or let’s call it the idea of algorithms of how do you actually specify precisely how to get something done? You know, I've got a deck of cards, how do I sort it? Or I want to find out who's the tallest person in the room and that kind of algorithmic stuff. So we can talk about that and in an abstract sense. And this is the kind of, you know, the theory of computation or

algorithms or something that you might do, but very very simple level because the students in this class are very non-technical. And then also talk about programming. How do you actually get a computer to do this, where you have to specify it precisely, 'cause this thing is a sorcerer's apprentice. Right? You tell it the wrong thing, <laughs> it's going to do the wrong thing. So talk about programming and then talk about the various software systems that we see in our lives: operating systems, browsers, apps on your phone, all of these kinds of things. And then the final part of the course is basically communications. And that's where things get interesting because now you've got all of these devices programmed with errors and so on talking to each other around the world with people we don't know anything about. And so you talk about how does networking work, the idea of local area networks, how does the Internet work, how do you put things on top of it, like the Web, and how do you protect information is, what are the implications of having everything about you known by somebody else more than you know, all of those things. How do you talk about-- how do you think about those kinds of things? How do you defend yourself to some extent against people who would like to know about you than you wish to tell them? And so those are the three pieces of the book. Something--

Mashey: Do you assure them that their microwave is not spying on them? <laughs>

Kernighan: No, I don't assure them of that, 'cause I don't know whether it's true or not.

Mashey: <laughs> Yeah. Okay.

Kernighan: Their thermostat certainly is.

Mashey: Yes, that's true. Yes. <laughs>

Kernighan: And your TV. And a microwave is no different than any of those. So no, I'm-- there's a spectrum of paranoia that--

Mashey: Yeah.

Kernighan: -- and most people are way over on this end where it's, ah, that's fine. And I'm over on this end where no, it's not fine. And so I probably run slightly I'm sure I come across as weird to some of the students in that respect, but I'm closer to right than they are.

Mashey: <laughs> So, okay, I know also you did a book on Go.

Kernighan: Yes. Yes.

Mashey: Talk about that.

Kernighan: Yes.

Mashey: And maybe philosophize about, you know, sort of in the history of languages where does it fit? Where do C and C++ fit in the specialized language and sort of pull that together.

Kernighan: Yeah. So the book, which I wrote with Alan Donovan at Google, I spent-- I spend often summers at Google in New York--

Mashey: Mm-hm.

Kernighan: -- and hang out and worked on a variety of things, kind of like an intern in many respect. And one summer, I happened to be sitting in the group that is part of the Go group at New York and Alan is there. I hadn't known him before that and, in effect, I was his intern for the summer and I learned a modest amount of Go. And during that summer we thought wouldn't it be nice to have a book on Go. We weren't very satisfied with the ones that were already out there. And so we worked on that for the next, call it a year or something like that. Alan is an absolutely amazing programmer and he knows all languages inside out, but in particular he knows Go inside out and so the work there is at least 90 percent his. He's just- it was really pretty remarkable. So where Go, there's a sequence of languages and historically that you go back to Christopher Strachey's CPL, and then you go Martin Richards's BCPL, and then you go through the B and Bon and stuff at Bell Labs that were Ken Thompson. And then you go to C which is Dennis Ritchie and then, you know, that's sort of the-- but then following that is C++ and following that is, let's call it, Java. And many of these are reactions to the language that went before.

Mashey: Mm-hm.

Kernighan: There's something which that the current language doesn't do as well as you think it could, we could make it better by having a new language and that maybe be- more feasible because hardware has moved on, we've got more resources of processing speed and memory, and our understanding of what it is we're trying to do and how to do it has improved. So the languages in many respects continue but then sometimes they get big, so Java is a reaction to the size of C++ in part but it also has 10 years of Moore's law, so you could do an interpreter instead of a compiler and things like that. And you could argue then that- that, well, C# is a marketing response to Java and JavaScript is let's take something that sort of serves as syntax like C hands-on, but it's got to run in a browser so a different set of constraints there. And there, Go is explicitly a reaction I think to the complexity of C++. It was developed originally at Google by Robert Griesemer, Rob Pike, and Ken Thompson. And their guiding force was they wanted to do not C++ and help make it solve some of the problems that Google has writing code at scale. Very, very large programs, compilation times that were taking too long, portability issues, and dependency hell

where inconsistent versions of libraries might cause trouble, so they wanted to deal with all of those things. And so that's kind of what Go is all about. It was started in 2007 and was open-sourced a couple of years later and is a totally open-source project at this point. And it has, I think, achieved a lot of those things. Go compilation is very, very fast. There's some technical reasons for that. It is static binaries so that means that the dependency hell isn't there.

Mashey: Mm-hm.

Kernighan: Gotten rid of that. And it has a very nice model of concurrency, it's based on communicating sequential processes, Tony Hoare's idea from the '70s, but very, very well and efficiently implemented. And I think that's the part of Go that I understand better. Their-- the- a different model for how do you think about objects. It's not really object oriented program, but that's the part. I'm less comfortable with. The concurrency seems to me to be very clean and straightforward and, from the look of it, more efficient than the thread models that you find in Python or Java or probably C and C++, as well. Empirically, that is, with me doing a handful of small experiments, it's a third faster in Go to do equivalent code. Things like crawling a net or walking a distributed file system or something like that. Some people say that Go is kind of like C for the 21st century. It's enough like C on a surface syntax that if you could believe that. It has that same taste for minimal mechanisms that kind of stripped down. At the same time, it is the opposite of C in some ways. C is pretty laissez-faire, as we mentioned earlier, you know, the sizes of things who would care. In Go those are specified so this is how big various operators are. And some of the trouble spots in C+ -- in C and in C++, like order of the evaluation, complicated nested things where you can put assignment statements inside or other side effecting things inside expressions and so on, Go doesn't have any of those things. So and it takes a rigorous view of formatting. There's a standard formatting program. So all Go programs look identical and on their, you know, where the braces are and things like that, so religious wars of that kind gone away.

Mashey: <laughs>

Kernighan: And all of that means that it's actually pretty nice language for programming. I wish I had written more than I had but I find it pretty comfortable. There's odd places where I think it wouldn't be like that in C and it bothers me. But I, you know, once you get into the groove then you don't think about it anymore, and that's true of any language I think. So- so Go, it's going to be it will never probably replace C++ at Google or something like that. It's also sometimes used as a replacement for Python because Python, with dynamic typing, if you make a mistake you may not learn about it until it's much too late. With Go, if you make a type error you'll find about out of it at compile time, not at runtime most of the time and so I think that's a big advantage. So in one way it's chipping away at C++. In another way it's chipping away at Python and it's certainly got its own niche. If you're going to do a web crawler or a distributed service or something like that, Go is a really good choice.

Mashey: Hm. Hm. Hm. So how about what do you see either on the horizon or going on, I- I-- with the number of multi-core chips and graphics processors and other specialized processors, where parallelism is increasingly important? What do you see in the language world that's helping that or not?

Kernighan: I'm not enough of an expert to know. I think the coroutines in Go are an example of something where that helps, so that you have a way to express the fact that a computation is happening in multiple processors at the same time, and with some chance of controlling the flow of information. The essence of the coroutines is that if I have something, I own some piece of data or something like that, then I can give it to you by sending it over a channel. It's a typed channel so I can't send arbitrary stuff to you, but I can send something where you know what it is. And now you've got it and I don't have it anymore. Underneath the operating system or the runtime library or whatever, it has to be paddling to make sure that that's consistent. But at the level where I'm thinking about it and you're thinking about it, we don't worry about it, I've got it or you've got it and there's no in-between state. And so that means issues of, you know, shared-- inadvertently shared data structures don't happen at that level. That's not like this is free. As I say underneath, somebody has got to make it happen. And in addition, Go provides the traditional, you know, locks and mutexes and things like that but you don't need them nearly as much. So that's one approach to distributive computing especially if it's sort of all right in the same executable. More broadly, I don't know. I don't do that stuff.

Mashey: <laughs>

Kernighan: And- and it is unfortunately very important because that's the way the world is going. And more and more at every level distributed, you got multiple things going on in a single chip and then you've got multiple chips, and then you've got multiple things connected by networks of various sizes more or less all the way. And so all those problems that were hard when there were only two things, they're harder now.

Mashey: Hm. Well, speaking of things that are hard, talk about the sort of current state of computer science education from your viewpoint and what's hard, and challenges and what- what sort of things you and other people are doing about it.

Kernighan: So I think that one of the challenges, and certainly the one that is the most obvious, is that enrollments in computer science are very large and this is one of these be careful what you wish for situations.

Mashey: <laughs>

Kernighan: Our enrollments here have gone up by a factor of let’s call it three or four in, let's call it, five or six years or something like that and that obviously puts a strain. We are not unique. I think if you went

to any school that offers computer science you would find some analogous sets of numbers. And the problem is the mechanisms can't react that quickly. You can't just get more faculty and put them in front of classes. You can't get more space for people to live, like offices and things like that. You can't hire instructors. It's just there's too much going on all at once. And the university mechanisms here and everywhere else are pretty conservative about growing things, because they worry about what happens when the bubble bursts or fashions change, and they have to get rid of that. So we have a lot of majors and a lot of people who are taking computer courses who are not majors. I don't know what today's number is but I would say probably, as a guess, 75 percent of every student at Princeton takes at least one computer science course. Our intro computer science course, which is given both semesters, is in total the biggest course on campus with probably, I would guess, seven or 800 kids taking that. So all of these are problems, just lots and lots of people and how to maintain the things that, at least here, make for a good education is hard. One of things that Princeton has done well is to require students to do independent work, one-on-one with faculty members. So every student has to do some of that too. Details vary but, you know, on balance most students spend four semesters working one-on-one with faculty members. If the number of students in a particular department goes up by a large amount it's hard to sustain that. So we are, in computer science, we've started seminars for some of that independent work. Not all of it but some of it so that we'll have a dozen students around the table with a single faculty member in charge, and they will be working on sort of similar things. And they will be learning from each other as well as from whoever the faculty member is. So that's a partial answer to some parts of it. A few of my colleagues are experimenting with flipped classrooms where they make video segments, relatively short video segments, on the various components that are interesting or important. Students watch those in the privacy of their own whatever. And then the class is more question and answer, discussion and groups of students working together on things, faculty member kind of orchestrating and keep an eye on that but not standing in the front and saying the stuff that's on the videos. I don't have enough experience with that to know how well that worked. The colleagues I know who do it say that it- they like it, the students like it, so I think it's got something going for it. I kind of miss having an audience when I talk so but perhaps this is a way to do it. It's also kind a labor intensive to make those videos upfront. So the other thing, I suppose there's lots of people who want to do computing and not all of them are tremendously expert at it. And some of them don't actually have perhaps an aptitude for it but they have to take courses to find that out. And so the spectrum of people's abilities and backgrounds and everything else, that one finds in the classes, is broader and that means that there's in some sense more administrative or personnel issues to deal with in classes, as well.

Mashey: So, I'm curious at the other end, what do you find in terms of kids coming in from high school that are on the sort of more expert and how good are they? What have they done?

Kernighan: <laughs>

Mashey: Are they terrifying or are they good, just good?

Kernighan: Terrifying is actually quite an accurate, yeah, perception sometimes. Some of these students who come in from high school are just amazing. They have, you know, they come in, they've started two or three companies already.

Mashey: <laughs>

Kernighan: They have paid their way through some part of their youth by selling software or services or things like that. They have apps running in the app store. They have taken four or five or six courses at the local university. And so all of these things, they know a tremendous amount. And part of the issue sometimes is to slow them down a bit and say there's more about the world than computing. There's lots and lots and lots of other things that people do that are interesting and important, and maybe it would be good for you to explore some of those, too, rather than digging in super deep when you're still a freshman.

Mashey: <laughs>

Kernighan: And so we have a few of those every year and others who are, you know, not quite that but still very, very advanced. And so that Princeton in particular is a liberal arts college, it stresses breadth over depth in a lot of ways. Not that you can't go deep but, you know, you have to do some of the breadth as well. And so, my job as an advisor for students like that is to try and get them to sample some of the other things that are going on. 'Cause people overall, mankind, do an amazing collection of different things that are interesting. And I think being exposed to those, exploring some of them is part of the reason you're supposed to be at a university.

Mashey: Hm.

Kernighan: Yeah. They don't always listen to me but--

Mashey: <laughs>

Kernighan: -- that's okay.

Mashey: Well, I think that's all I had. Now, did we miss anything in particular, any things that were particularly fun or lucky or-- well, you already mentioned some of those, right?

Kernighan: <laughs> Yeah.

Mashey: Any- any other last words?

Kernighan: Yeah, I mean last words. As- I think luck is an enormous amount. It was, if I recall correctly, Pasteur who said, "Chance favors the prepared mind."

Mashey: Hm.

Kernighan: So what you should be doing is to enhance your luck by being open to all kinds of different things, learning whatever you can, hanging out with the best people you can. All of these kinds of things mean that when something comes along that would be called luck, you're more likely to be able to take advantage of it. And I think that's hard for people to realize, but I think that's really, really important. So there's lots of ways you can do that, keep that in the back of your mind, be open to all kinds of stuff, play with the good people.

Mashey: <laughs> Cool. So actually, Brian, since I sort of missed this. Can we sort of go back early in your- your family background? Did you have particular inspirations? I mean how did you get interested in technology and- and all this computing stuff in the first place.

Kernighan: I- I-- it's hard to say exactly. But, as they say, "I was born of poor but humble parents. Poor but honest," that's the phrase. And they remained honest and they remained poor <laughs>.

Mashey: <laughs>

Kernighan: But no. My father was a chemical engineer.

Mashey: Okay <laughs>.

Kernighan: And he had grown up in a farm community and so many of the other members of his family were farmers, and therefore close to the land. But they were-- they had also gone to universities and things like, you know, agricultural college kinds of places. And so there was, I think, real respect for engineering there and they knew how to do things. They actually knew how to build things and make mechanical systems work. And that was I think very useful.

Mashey: Yeah.

Kernighan: So my father had actually gone to the University of Toronto. He graduated at the height of the Depression in 1931. And he spent some of his formative time working in soup kitchens in Toronto and that changed his view of the world a great deal.

Mashey: Hm.

Kernighan: So and I think some of that probably propagated to me. My mother was working in Toronto when she met him. They got married. I have a brother and a sister, both younger. I think it was always thought that I would go to university. And I don't remember ever thinking about anything other than going to University of Toronto just because, among other things, it was very close and my father had gone there, and it was at that time probably the strongest university.

Mashey: That's true.

Kernighan: And the economics are such that, at least in Ontario, we didn't have to pay very much money. So it was all very convenient. The ques-- question earlier that you mentioned of formative influences, I think probably a couple of people in high school. I don't remember very many people from elementary school. But in high school there were a couple of teachers who got me interested in things like history, which is sort of an interesting thing as an area. And I think if I weren't in computing, I probably be a frustrated historian instead 'cause I think that's interesting. But there was finally a high school math teacher, and this would be in my-- in grade 13. At that time, Ontario had five years of high school. In grade 13, my math teacher was a youngish guy who had worked at- in the Canadian aviation industry, at AV Roe. And some military plane got cancelled. He was laid off. He had an emergency job as a teacher. And I don't remember much about his teaching but he was one of these guys who had sort of good insights into things. And he said, "I was thinking about university. What course do I go to?" And he said, "You should try this engineering physics course. It would be right for you." And that was the right thing to do 'cause it put me on a path. If I had gone to any of the other engineering fields I wouldn't be where we are at this point, because I would have been much more focused, much more kind of nuts and bolts. And also, might well have missed the computing thing.

Mashey: Hm.

Kernighan: So- so that was probably the single most and sometimes lucky thing I got from a teacher up through the college level. And I, a few years ago, I tried to track him down. He wouldn't be that enormously older than I and I can't find him anywhere unfortunately. It would be nice to be able to say thank you <laughs>.

Mashey: Well, I think that's all I've got. Thanks a lot, Brian.

Kernighan: Okay. Well, thank you. Thank you, both.

Mashey: Great. Yup.

END OF THE INTERVIEW

