An Interview with

ROBERT GOLDBERG

OH 207

Conducted by David S. Cargo

on

27 July 1990

Flagstaff, AZ

Charles Babbage Institute
Center for the History of Information Processing
University of Minnesota, Minneapolis

Robert Goldberg Interview
27 July 1990

Abstract

Goldberg describes his use of Icon in the classroom at Illinois Institute of Technology and emphasizes the strength of Icon for instructional purposes over other programming languages. He discusses his interaction with Ralph Griswold when working on the various implementations of Icon. Goldberg describes the strong documentation produced by the Icon project and his own work on conversion of Icon from a UNIX to a VAS VMS implementation. He discusses the development of a PC version of Icon and his own use of Icon in normal work.

ROBERT GOLDBERG INTERVIEW

DATE: July 27, 1990 INTERVIEWER: David S. Cargo

LOCATION: Flagstaff, AZ

CARGO: Okay, Bob, why don't you tell me about your educational background prior to the time that you got involved with Icon.

GOLDBERG: I have a bachelor's degree in mathematics and computer science from the University of Illinois at Urbana. I have a master's degree in computer science from Illinois Institute of Technology in Chicago.

CARGO: Could you give me a couple of dates for those?

GOLDBERG: Bachelor's degree - 1973; master's degree - 1977. Since that time I have taught part-time at IIT. And I think my title, at this point, is adjunct assistant professor. I typically teach one course a semester. And one of the reasons that I went to IIT is that I became a SNOBOL aficionado when I was at the University of Illinois, I ended up at IIT because of Robert Dewar, who had done SPITBOL, and I worked with him while he was at IIT. And through that I met Ralph Griswold in, I think, 1977, and through my SPITBOL work I continued to stay in contact with Ralph. In the early 1980s I visited Tucson a couple of times and found out what they were doing, and that's how I found out about Icon - just by being in the SNOBOL community, which then started to move into the Icon community.

At IIT we taught a course in programming languages, and one of the languages, of course, was SPITBOL - SNOBOL 4, actually, but we used SPITBOL implementation. And in the early 1980s, with Icon available, I thought it would be a good idea to replace SNOBOL 4, which was creaking in some ways from an educational standpoint - it's hard to deal with "go to's" and no control structures - and switched to Icon. Unfortunately, that turned out to be a little more work than I had bargained for, because there were no implementations of Icon running on VAX VMS systems at that time. And so I became more directly involved in Icon by volunteering to do a port to VAX VMS. [VAX/VMS port done at Datalogics while I was a consultant there.]

I can speak a little bit about that if you would like. It was an interesting period. I don't remember the exact year; it had to have been 1981 or 1982. It could have even been as late as 1983, but I don't think so. And I can check on that. Apparently there was a fellow in Canada who had tried to do a VAX VMS port and was unable to do it. And Ralph volunteered to send me everything that the person had done and take a look at it. And from the porting situation, this was a very curious situation, because what had turned out was that there was a basic underlying problem in the Icon implementation at that time. They hadn't really designed Icon to be a distribution programming language -- meaning "put into distribution" to be used by others outside of the computer science department at Arizona. It was research-oriented. And the graduate students who had worked on it, like many other programmers who worked on UNIX systems based on PDP 11s or VAXs, took great advantage of the fact that they knew how the UNIX compiler handled register allocations and what the stack frames looked like. And the Icon implementation at that time also took advantage of that in building the frames for generators and things like that.

Now, the problem that came about on the VAX VMS system was that the VAX VMS development people (DEC employees had come up with a very clever compiler that did a different type of optimization, let's say, than the typical UNIX compilers. In particular, they did their own register allocation and ignored whatever the programmer had specified to put in registers. And this caused a fundamental conflict that could not be resolved without doing something strange. In order to get a port on VAX VMS, either the system would have to be recoded, which was unlikely and impractical, or something different would have to be done.

And my contribution here, if it could be called that, was that I suggested that there was no way to defeat the VAX C compiler under VMS. There was no way to get rid of the register allocator or any of these things. There was no way to cheat. The only possible mechanism that might have a hope of working was if they fed me the output of the UNIX C compilers in assembly language, and then I would convert it to be VAX VMS assembler-compatible and link it with the VAX VMS C runtime libraries.

Amazingly enough, that all worked. And I used a SPITBOL program, of course, to do the conversion work and was

able to get a VAX VMS port limping along and running well enough to be distributed. Unlike the other Icon implementations at that time, that was the only Icon implementation that was not distributed with source code because of the odd manner in which it was handled. We were worried about "legality" of taking UNIX compiler output and using it on non-UNIX systems. Issue was AT&T's educational license agreement -- was not clear if this activity was permitted. [In particular there was some concern that there might be some problems in taking code generated by a UNIX C compiler and porting it over to another system and getting it to run.] And we didn't really talk at all about how we had done the port. Now in retrospect it is all history and should be fine to talk about it, but we never really did talk about how we did the port.

And the benefits of this is one, to me personally, it allowed me to start teaching Icon, which was my goal. I mean, I basically became a developer by need. I had a need to teach Icon, and the only way I was going to be able to do it was to do the port because no one else had succeeded and no one else had volunteered. And it was quite a bit of hair pulling and it took quite a bit of time, and there was a lot of feedback going back to the Icon group in Arizona about how things might be made more portable.

Some of the things that came up were, well, of course, the coding style that assumed registers were used for specific variables based on their order of appearance in function declarations, and mick [?] going in stack frames, and also things like the standard I/O library would be different from UNIX to VMS with the VMS C compiler. Sbrk, the bane of expanded regions in Icon, was a problem. Various returns codes - sometimes errors were null; sometimes they were minus one. So there were a lot of interesting periods of just going through the code.

And Ralph supported me in this and was really a very big help. I mean, I could not have done it without his support, of course. But I mean he would take my calls when I would call in the evening or on a Saturday afternoon and say, "I just ran into this problem. What do we do?" You know, we would talk about it and work out a solution. And I think that, all in all, first, it was a very enjoyable experience, and luckily I was able to extricate myself out of it after a time, because eventually the VAX VMS port went down to The University of Arizona (back to the computer science department there) when the Icon code itself became more portable. They were able to do their own port, because

doing a port at that time was very time-consuming. It took quite a bit of hours. I don't know how many at this point, but it was quite a bit of hours.

But again, the point was to be able to get a working version to teach, and we have successfully, since that time, taught Icon at IIT. And all the undergraduates or graduate students who take that programming language course use Icon still to this day. And these days, of course, we use the PC versions - not necessarily the VAX versions. But it's still going very strongly as a language resource at that institution.

A few years later I ran into the same situation again. I was still teaching the programming language courses and, because of the work I do, we were starting to build systems based on Compaq 386 computers, and then AST, and then generic 386 PCs became available from other manufacturers. And anyone who had done any serious programming in Icon on PCs had run into memory problems - running out of memory. And we at work particularly were building extended mode DOS programs. Basically, an extended mode DOS program is built from compiled 32-bit code that really takes advantage of the real 386 32 bit instruction set, and have the hooks to talk to DOS for you to read and write files and do the standard DOS things. And of course this would solve the memory problem, finally, on the PC side, at least for the 386 machines. And I had all the developer tools because we used them at work. [This 386 PC port was done when I was at Dewar Information Systems.]

So again, I had this need; I wanted to write some big parser generators and some things like this for a compiler course and I wanted to use my 386 so I wouldn't have to log in and, you know, use a 2400 baud dial-up line from home, and all these things. So I got involved in another port by volunteering - my second port. And by this time the code was significantly improved, and the battles in porting became more along the lines of what C compiler standard I/O library (UNIX library) did to you on this version versus that version. And we found there were various things. It was no where near as difficult an undertaking as the original VMS port was, but it was still relatively time intensive the first time around - the result being that I did get a 386 port running. I was able to run very, very large programs very quickly. I made it available to the university, to Ralph and the department so they could distribute it to other people. And I tracked a number of incremental versions, and again, I don't know that I can even remember which versions,

but somewhere in the 7 range I stopped being the implementor of the 386 extended mode DOS version of Icon and passed that baton to Mark Emmer of Catspaw, Inc. who now has taken it over, who has more time to deal with it and properly license it. [Turned over to Catspaw when an issue arose dealing with distribution of Phar-Lap's DOS extender as part of public domain software.]

What I guess is interesting for an outsider is that, again, the rationale for me to do the port was not for the love of doing the port, which itself can be fun and is fun at some level, but was the fact that I had a need. I needed the product. I needed the language to run in an environment, and the only way to get it done was to do it myself, because no one else was stepping up. And I think that's probably representative of a number of other people who have done ports over time. I really wasn't involved with the language design or anything like that, other than teaching it.

Talk about why things would look the way they did - I certainly didn't have any influence there, but I think a lot of the experiences I had with the VAX version, and certainly with the 386 version, helped contribute, as many other developers and porters did, to making the system as portable as it is now. The last time I did a rebuild of the 386 version of Icon, it got to the point where I could get it done in about an hour, which is very nice. It's very nice to be able to rebuild a new system in about an hour and not spend a period of extended man-weeks getting it to run. My involvement now is still basically as a teacher and a user of Icon. I still write programs in Icon. Depending on the particular course I teach at IIT, we still either teach Icon or use Icon as the vehicle. And I have a strong interest still remaining in non-numeric programming applications, which is why Icon is still close to my heart. What else? Where can I fill in gaps for you?

CARGO: Can you tell me more about your communications with people at the Icon Project when you were working on your port?

GOLDBERG: You know, I used to have all of this. I think I threw out a lot of it. I moved three years ago, and at that time I had a lot of the files I used to do the ports, especially on the VAX VMS, and at that time they were pretty much

trashed. I had copies of memos I sent to Ralph and things like that.

There are some anecdotes. I remember debugging the VMS version with Ralph. You know, running into a problem where the garbage collector didn't work and I called Ralph up, and, you know, we're sitting on the phone for a half hour trying to figure out what's wrong and all of a sudden he realizes that somehow I had been sent the PDP 11 garbage collector and not the VAX garbage collector [laugh]. Most of the communications I had were really to report problems that I ran into and suggested solutions for getting around them or pleas for help. All the communications were pretty much with Ralph. I sometimes would talk with Bill Mitchell as well. Bill Mitchell was very active in the Icon development at that time in the early 1980s. Thankfully, e-mail was around making some of that easier. Quite a few of them were by phone or by letter.

You know, by that time, I am sure Ralph was doing many other things, but I think he must have spent a lot of time with implementors like me trying to wade through what difficulties they were encountering during the ports. As a matter of fact, I am sure, as you heard at the conference today, there are many implementors of Icon who don't program in Icon. I mean, certainly more than you would expect. And in the beginning, when I first did the VMS port, that was true for me as well. I couldn't. I didn't write any Icon programs when I did the port. I basically did the port and checked it out by using their test suites. And of course I wrote a few simple ones, but I really didn't become an Icon programmer, in a sense, until I had completed the port and was able to use it. It was one of those things; I had to have it tested. [i.e., Icon had to be up and running before I could use it.]

And so, most of the communications with the Icon Project dealt with just getting the ports to run. I mean, if you go back to the idea that I had, which was to do the port by taking the assembly language, we spent some amount of time talking about whether it would work or not. And I know that there were reports back to me from Ralph that said there were people down at Arizona that didn't think it was possible - that it wouldn't work. And quite frankly, there was no way to guarantee that I could make it work either. I couldn't guarantee it either, but it was the only path to get it to work. I mean, that it worked made me very happy, but I didn't see any other solution and we had to try it. And that was basically the case I presented. I said, "If we are going to have this, this is the only solution I see." And no one

else had any other ideas either about how to make it or some other alternative strategies.

And so there were periods like that where I tossed around ideas for the port, because normally a port of the Icon implementation involves changing some manifest constants in a header file and go rebuild the whole system. And what we were attempting was not quite that same thing. Poor Ralph really had to do a lot of work for me. Now that I think about it, there's two sides to how much he had to support me in these endeavors. First of all, if you think about this, normally the Icon distribution was "Well, here's a UNIX tar tape. Go to it." Right? With all the C code and the header files, and what he had to do for me was that he had to basically produce all the assembly output files and send those out. On my side, since I was running on the VMS system, I had to write a program to read tar tapes before I could do anything. So there was all these background activities that had to take place.

Similarly, in the 386 port (I remember this well, and if you know ANSI C you know how they have a token pasting operator on the preprocessor)... Well, the UNIX C compiler supported token pasting. The MetaWare High C compiler that I used on the 32-bit 386 port did not support token pasting, and there was no way to fake it out. And Ralph had to preprocess all the C files for me before he could send them to me, and I am sure that had to be another agonizing thing to deal with in the middle of coordinating a number of ports and research at the same time. And so he probably ended up giving me in essence more of his time, not while directly communicating with me, but just getting me the pieces I needed to do my job, than he might have had to do with other implementors. I don't know if he sees it that way, but certainly I think there had to be much more effort put in to support me for these things than the other ones. Thankfully, MetaWare High C finally supports that, but for years he had to remember to preprocess the source files. He just couldn't send me the normal distribution. So, I mean, if you really look at the types of support we have, we're really working together as a two or three person team at some level where I was doing the real work of getting the code to run, but was getting the background support in terms of either preprocessing or compiling to assembly language from the people at Arizona. And through all the porting work the people at Arizona eventually came up with the idea to ifdef everything for all the various ports, which makes eminent sense, but it took a long time. It took a number of years before that came about. I would say that - at this point, not remembering the year, this is probably my eighth year of involvement in Icon, which is sort of hard to believe but it's really true. And my involvement with

SNOBOL goes back to about 1971 or 1972, and as a parenthetical note, I was a Macro SPITBOL implementor as well. So I have a history of doing implementation types of things, just because of my interest in having these languages run in environments where I wanted to use them. At this point, I have pretty much given up on SNOBOL and don't use it anymore. I use only Icon now, but I still have a certain fondness for the language and what it represented, what it allowed to come about with Icon. [In the summer of `91 I wrote 5,000 lines of SPITBOL for a project!! So much for the statement above.]

CARGO: When you embarked on your VMS port, what kind of documentation did the Icon Project send you?

GOLDBERG: I remember this distinctly. They sent me the porting document that they had at the time for version five point something and I remember reading this document in bed, and it was describing how they knew about the stack frame allocation and how their code accessed the stack frame. And I was getting really mad because I knew it was torpedoing my port, and I don't think until I told them they couldn't do this that they really understood they couldn't do it. I really don't think they were aware of what they had done. It's not that they had done anything wrong; I just don't think they realized that the VAX VMS C people had done something entirely different with their C compiler than the UNIX people.

And so the porting document really set off the chain of events to come up with the idea for porting. The documentation was very good. The documentation basically gave you internal organization details. I don't remember at that time if they had everything organized for manifest constants for the configuration files. I don't believe they did. Again, I just threw all those documents out. I am very sorry. I had all the stuff from the days of doing these ports. One of the things that is very interesting about the work that is done on the Icon Project is that you're involved in acquiring code from other sources that you didn't control, you didn't develop, or wasn't developed by someone in the same organization. As a developer or a porter you have a lot of problems because there are things they don't tell you or they did something weird and you discover it the hard way. Along the lines of "no doc" [no documentation] or the famous, "Here's a tape. Go figure out how to make it run."

One of the things about the Icon Project has been that they have always produced documentation. And the documentation for doing the ports has evolved and it's gotten better and better and better. And at this point someone with a certain amount of experience can really get a port up very fast based on the documentation. I really felt that the documentation was very good. And I remember they sent a stack of documents; you know, about the language and all the others in the porting doc, and it had to be a half inch thick. And again, it was the porting document, or the description of the internals that began the process of me understanding that it wasn't going to work the normal way. When I did the 386 ports, by that time the code had been organized with conditional compilation, preprocessor directives, and they had all been organized and documented, so it was very clear when you went through the source you could find where you had to make modifications to run in your system, and where to look and what the manifest constants meant and what to do about them. And it became a significantly easier process, and the doc continued to improve. I mean, this is again so astounding, because so often you will just get stuff, and it just sort of falls in the UPS in box and you get it and now what do you do with it? You have to dig through it and wade through it and figure out what you have to do. And that can be interesting and fun, but often you don't have the time to do it. I get the impression Ralph has run in it; I don't know. But the Icon Project has produced very, very good porting documents to help developers. And certainly at this point in the evolution the porting process, other than if you run into C compiler peculiarities, or machine architecture peculiarities, like 16-bit machines, the port should be relatively straightforward - or word address machines. I had wanted to do a Data General port at one point, because I was working on both VAXs and Data General, but there wasn't really the time. It would have been a very big challenge at that point to have done it, I think. I don't think it would have been successful at that time, but perhaps. I think someone finally did one for that, but I haven't really worked with that machine for a number of years.

CARGO: Which model of Data General was that?

GOLDBERG: It was an MV 8000 - the 32-bit version that ran, I guess AOS was the operating system and they had a C compiler. I should give some credit to Steve Brown, president of Datalogics, Inc. In the early 1980s I was consulting to Datalogics. I had been an employee of theirs in the 1970s. Steve Brown was a graduate of IIT [?] as well in the 1960s and he knew Dewar and was a big SPITBOL fan. And Datalogics is a company that produces text processing

systems -- typesetting systems, and they sell them to publishers and insurance companies and whatnot. And they always do a lot of conversions, and so SPITBOL has always been a very popular language within the organization for doing certain types of conversions. And Steve Brown was always very supportive of SPITBOL work, and he basically allowed me to do the porting work on Icon for VMS on his system. And that was very nice, and where I work now at Dewar Information Systems [Robert Dewar developed SPITBOL at IIT. Dewar Information Systems is owned by Robert's brother, Stewart. Robert has no connection with Dewar Information Systems.] I use the machines there to do the porting work for the 386 versions. And again, those are very pleasant that I have been able to have people - with PCs, of course, you just put it in your car and you take it home over the weekend. But in the VAX days, I was able to use Steve's VAX at Datalogics for all the porting work on weekends. And he does deserve thanks for that, and he received recognition when the VAX VMS port was announced through the Icon Newsletter. Back to you.

TAPE 1/SIDE 2

CARGO: ... had teaching about Icon and IIT, both in terms of the kind of preparation you felt that you need to do in order to teach a course and the kind of course you teach... or courses you have taught.

GOLDBERG: Right. There have basically been two courses that I have taught that have used Icon one way or another. The first of the two courses is CS440, whose title is "Programming Languages and Translators 1." The second of the courses is CS540, which is "Syntactic Analysis of Programming Languages." In the CS540 course, I can talk about that very briefly. Basically, we don't teach Icon in that course. We expect the students to use it as an implementation tool for the various programming projects we use. And I can return to that later and talk about that. But in the 440 course, the course is a cross between a comparative programming language course and a course about getting people to think about implementation issues in programming languages, syntactic and semantic... the whole idea of what does it mean... what does a programming language mean? What types of programs can you easily write in it? What type of expressive power do you have in various languages?

The impetus for Icon, again, came from the fact that we were trying to teach at various times, if you can imagine this, PASCAL, SNOBOL 4, LISP, and even ALGOL 60 to a certain extent. I mean, the flavor of the course might change depending on who was teaching it and what books were used, but basically, people would write programs in various languages. It was just very hard to teach SNOBOL 4 with the absence of reasonable control structures. It just became that you really wanted to talk about issues in SNOBOL 4 like pattern matching, like the run-time typing, which I refer to as weak typing [weak typing -- is wrong term -- SNOBOL4 is an untyped language], and the fact of innovative data structures. But it was because of the whole control structures of being "go to's," it just became hard to deal with the student body as to what the real issues were. They couldn't see the forest for the trees in this case. They became hung up on the "go to" issue and we couldn't teach them the other issues of the power of the language, and so Icon seemed like a reasonable alternative - the same developers, in essence (Ralph), same emphasis on run-time typing, same carry-over of many of the features that made SNOBOL so interesting. So preparation involved reading the book and writing various Icon programs.

Now, because of the types of projects and the course that we would handle and the types of projects we might have people program would be simple macro-processors, simple interpreters, like for LISP or SCHEME - again not fully implementations. We had people do things like analyze a block-structured program and tell us what the variables would be and which activation records at a given instant in execution. We actually in some cases had them build interpreters, things like this that would allow them to use the facilities of Icon to do fast prototyping - in essence, be able to get a project rolling quickly.

The teaching of Icon itself was very interesting for me because it's very much fun to balance and compare and contrast Icon with languages like PASCAL or MODULA 2, where certain types of programming are painful, almost. It's just too much code to get something simple done, it seems. And in a language like Icon, it becomes a very elegant small program, and these are the types of programs you like to use when teaching a course to illustrate it. The problems that come about in teaching Icon have been... First of all, there are always very good students who catch on, and they probably would catch on even if you weren't there. They would just read the book and understand it and become great Icon programmers and understand the idioms of Icon.

And I am one of those people who is not convinced you can teach somebody how to program. I don't know if it's innate or something else, but I think you can show people by example how they should program. Whether they learn from that, I don't know. Now this is a very, very personal feeling, and so the way that I like to teach Icon and prepare it is to talk about an issue - let's say, like data structures (you know, tables and sets) - and talk about the semantics of them, talk about perhaps even how they might be implemented behind the scenes, which is not really necessary in all cases, but it's in the spirit of the course that we teach, and then have programming assignments that use them. And of course, since the programming assignments usually deal with programming language translation issues at one level or another, many of the things in Icon are immediately useful - tables as an example, string scanning fall right out. The problems that came out of teaching Icon, and there were some problems... [If you go back to the green book for SNOBOL 4, the book was organized so that there was an introductory chapter - relatively short - that sort of gave an overview of the language. And then Chapter 2 was the killer. It talked about pattern matching and it went on for 50 pages. And it was the student killer. You know, they couldn't read it, they couldn't get through it, and you had to somehow organize your lectures in SNOBOL 4 to not bury students in pattern matching. You had to be able to show them other things...]

The problems we ran into, or I ran into, in teaching Icon were that you could spend a phenomenal amount of your time just talking about issues that are important, but they are not the real issues in the language, like this is how you do string subscripting and this is, you know, you deal with integers or whatever. And the book was organized so that you didn't really get to some of the interesting parts of Icon until, I think, Chapter 7 Expression evaluation and there's a chapter on generators and string scanning... were buried all the way in the back. And so, as an instructor, you had to have some way to bring out the elements of Icon in a sequence that didn't necessarily match the linear reading of the book from front to end. You had to mix and match sections of the book.

The things that we would concentrate on in Icon specifically would be, first of all, the whole way that Icon dealt with strings. For students who have been inundated with the C style or PASCAL or MODULA 2 style of strings, this is just a fundamental change for them, to get used to this, that a string is in essence an object unto itself. It's not an

array of characters. It certainly requires a different teaching process, but that wasn't the problem for me coming from a SNOBOL background, but it really requires for them to get the sense of the difference, the real semantic difference. Quite frankly, it's not always possible to convey it. I must say that there have been many discussions in classes where students just didn't understand why you just didn't write it in C, and that you have string copy or cat[enation] and all these other string operations, why would you want to use Icon? I think there are many very good reasons you would want to use Icon and try to explain them.

But a lot of this appeared to me to be indoctrination by various of the companies they worked for - one of them primarily being the AT&T people. We taught a lot of AT&T people and many of them were very good, but those were usually the people who objected. There was always some small group of typically AT&T students who really didn't get it for whatever reason. The next step would be to talk about (not necessarily the sequence would be the same every semester) the innovative data structures in Icon beyond strings - records and the tables and eventually the sets came about. Well, records, of course, were available in other languages, but the fact that the fields and the records can take on any type in Icon is a bit unusual compared to the statically typed languages. Tables - a wonderful data type. The whole idea of having a built-in symbol table and "Let's not worry about how you actually do the look ups and insertions. Just do it. Get it for me." This is a beautiful concept for a programming language course. Sets as well - real sets, as opposed to what they have in PASCAL or MODULA. Wonderful things to discuss. Expression evaluation with the goal-directed evaluation and how expressions are evaluated. Another very important topic to talk about, and again one where the semantics are so much different than the traditional languages that it's a bit of a shock to the students when they hear about it - string scanning and generators.

And those would be the topics that we would concentrate on, and we would build sample programs in class during the lectures to do various things, and then we might talk about how these programs could or could not be handled in alternative languages, and then reinforce this at some level by having programming projects again where hopefully they would use these features. The students seemed to like the book and would agree that the book was easy to read and that the sample programs in the book, some of which were very clever, were very nice. Many of the problems, of course, were getting them to read the book and then play with the programming of it. I had made for

myself sets of notes from the book of the topics I wanted to talk about and would revise them from semester to semester, although I don't think anyone else could ever have really used them.

Another thing that made the teaching very nice was that when eventually Cheyenne Wills got the PC version of Icon going (we were doing all of our work on the VAX VMS system) and once the PC version was available then students could start to work at home on their own and not have any access with the computer center. And eventually, our computer center (we have a lab of PCs now where people run these things) became a very attractive part of Icon - the fact that the implementation was basically free so that people could run it on a personal computer at home and do their class work. Since the courses I teach are at night, and they are not all full-time students, but many of the students are part-time students who hold full-time jobs or part-time jobs, the ability to do the programming at home instead of having to go the computer center was a big advantage in using Icon. Just in that aspect of it was a very big win.

We typically didn't talk about co-expressions. It's very hard to come up with interesting examples of how to use co-expressions. I'm sure that others have interesting examples. You know, the producer consumer problem, well, that's interesting, I guess, at some level, but about the only useful co-expression that everyone could ever agree on in any of the classes was the label generator - that here you are generating code and here you can use a co-expression to generate the next unique label name. I would say that I probably only talked about co-expressions 30% of the time out of the semesters I taught, that either we ran out of time or it just became less important than something else during the process.

In the CS540 course that I taught, the compiler course (I have taught the course twice; I have taught CS 440 many more times than that - I can't even count) I have traditionally handed out assignments to do things like grammar analysis - feed a program a grammar and let's find out what the first and the follow sets are and see which non-terminals are left recursive or right recursive and so on. And there is an ideal program for Icon. Again, the working assumption at that point was that the students would know what Icon was and how to program in it. Since this was a graduate level course, if they didn't know how to program in Icon they could get the book and learn relatively

quickly. You only need to read 70 pages of the old Icon book and you can be programming in Icon, so we are not talking a substantial effort for someone who has some level of experience to get in Icon, at least from my perspective. And of course, Icon provides an ideal vehicle. You have string scanning, so you can "rip" the input grammars apart. You have wonderful data structures to build the internal representations. The biggest problems that we would run int are computation speed and memory available. If you start to "grind" on large grammars we would run out of room or you would run out of patience on a PC waiting for it to compute these things.

And after the grammar analysis program I typically would assign a parser generator of some sort, it being LL or some LR parser generator. And I would provide various tools and programs in Icon to possibly fit in, either take the generated tables from the parser generator and use them to do a parse, or a scan or whatever. And the next phase would be to take the output from the parser generator and write a compiler in Icon to generate code for some abstract machine or real machine of student's choice. And interestingly enough about this, I personally didn't care what programming language the students used. They could use anything that was available to them. I strongly recommended, for instance, in the grammar analysis program, that they could write that program in C, for instance, if they were more comfortable in C than Icon, but that they should switch when we started to get to the later projects. And a number of students would not switch and stick with C. And none of the people who used C as a programming language for the final projects ever completed them. Only the people who used Icon were able to complete them, and I think that owes a lot to the fact that Icon is so expressive that you could do things much more easily and debug them - certain much more easily. Interestingly enough, on the first program, the grammar analysis program, the program size in terms of lines is not necessarily that much different in C than in Icon - the actual number of lines of codes. Now perhaps that's because most of the people programmed in Icon, including myself, in more of a C style. But as you got deeper into the other programming problems, Icon definitely saved a lot of code and a lot of time. And in general, even the C people admitted at the end of the semester that they probably had made a mistake and they just had decided that they couldn't bail out soon enough. They had just gotten too deeply into the C side to bail out and start over in Icon.

The experiences of most of the students who have taken these courses is, I tend to feel that, many of the students

who left became strong Icon aficionados. I heard today or yesterday that one of my ex-students is a big Icon person at AT&T Bell Labs in Naperville now. That's not surprising; he was a very good student. One of the pleasures that I get, though I don't necessarily grade all the programs, is that when I look at some of the students programs I am always constantly amazed that they use Icon in ways that I never imagined. You know, if you look at someone's C style as an example, I am sure there are a few people who have radically different styles. But in general, you might not care for the way they place their brackets or whatever, but you can follow the flow. A lot of students going through the courses really became true Icon programmers. I mean, you would look at their programs and they really used generators in the spirit of Icon. And it is always interesting to look at some of the programs from the better students and see those types of things.

And again, I don't think that was because they were taught that. I think it's because they did it on their own and became enamored of the language. I must say though that not everyone in the programming language course has always walked out loving Icon, but then some people just don't enjoy programming and they have to take programming courses. And I think that when people look at programming as a burden rather than a joyful experience that's one of the results. And many times I feel Icon can turn people around, because I think programming a language like Icon is a joy. And sometimes you can salvage that with some people but not always. And there are certainly a number of people I have taught who I am sure really don't like Icon, but I think it's more because they just don't like programming rather than specifically the language.

CARGO: When your were preparing your course materials, both the survey programming languages and the one that you just used Icon as an implementation vehicle, did you ever try to communicate with other educators who were in approximately the same position trying to figure out how to teach Icon, or have other educators contacted you because you had already done that sort of thing?

GOLDBERG: Well, when I taught the programming language course, there was usually another section, and I worked with a professor at IIT - well, various professors, depending on who's teaching it - but at the end, over the last few years it was Professor Thom Grace. And he and I would exchange ideas about how to present Icon. But I would say

pretty much it was an independent activity.

Funny that you should ask that - I recently got a call this summer from a graduate student of the University of Chicago who is teaching part-time at IIT and has accepted a position somewhere in the south. And he called me up because of the compiler course that I taught twice to get the material from me that I had used. And one of the things that I have to do when I go home is gather that to give to him. He is basically interested in the programming assignments and the homework assignments and whatever programs I would have built in Icon to support the class. And I am going to gather that stuff up for him. But that's really the first time that's happened. Honestly, there are a number of universities in the Chicago area - in some sense, there must be a competitive situation there, you know, competing for students - and I don't really know if the other universities even teach Icon. I don't really know their curricula well enough to answer that question. Other than Thom Grace and I working together on Icon with the comparative programming language course, so to speak, and this fellow now asking about the compiler course, I have not talked with other educators about teaching Icon.

I never did (I probably should have) ask Ralph. It would have been probably smart of me to have done that; it just never occurred to me. Next time I teach it I think I will ask him. I did use the newsletter though, and if something interesting came out in the newsletter I would feed it back through the class. And there were also areas where the book wasn't clear on the pointer semantics aspect of Icon that I think is now fixed in the new book, but in the old book people had a hard time understanding what happens when you assign a structure - that it doesn't really copy the whole structure, and, yes, Icon is called by value, but, by the way, when you pass in a structure that doesn't mean you can't necessarily modify it. And so these were issues that we would talk about in class and I would share information like this with Thom Grace. His background was more in graphics at that time than in programming languages.

CARGO: That brings up an interesting point, and that is, what parts of Icon, either as presented in the book or for whatever other reason, were major conceptual hurdles that you had to help boost your students over?

GOLDBERG: Actually, I don't think people have too much trouble with generators. It's something that, while it's different, there aren't very many other languages that have generators. There's Clu and I think a few others, but that really wasn't a problem area, surprisingly enough. They acted as if strings were a problem - the idea of strings of fundamental data type. I think really the biggest problems came in with expression evaluation, with goal-directed evaluation. I think that was something, even though the rules, the semantic rules, are very clear and very clearly stated in the book, and you can talk about them, I think people had trouble with that, because again, it's so foreign. I mean, the model of "I have an expression and I understand what type of assembly language I would generate to compute that "type of programming model" to one that has this idea that "there's some other control structure going on in there that's backing up and trying again" is not natural for many students and if they don't hear about something like that early enough, or if they don't see backtracking early enough, it might become a problem. It did become a problem.

Another area was string scanning. Quite frankly, I am not completely comfortable with string scanning, and I think that, like a lot of old SNOBOL programmers there's a, I won't say mental block, but it has a different spirit about it, and I can program reasonably simple and to medium complexity strength scanning, but I don't push it. And I think that's an area where it would be a lot of fun to take a couple of weeks out of a course and really talk about it and do it more in depth than we were able to do it. But I think that was an area that we had problems.

Co-expressions were definitely something where many students did not have any concept of what was going on. And again, depending on how the semester went, I said before that 30% of the time I taught the course I would probably talk about co-expressions. Some of the times I probably didn't talk about co-expressions were due to the sense of what percentage of the class might understand and accept what was going on. So I would pick on those areas as being the difficult ones needing help.

Pointer semantics again: since we were talking about programming languages used, we would always talk about parameter passing in the context of the course, and talk about storage allocation strategies - what's going on when your program runs, and those types of issues we would always discuss. And it didn't help that the book wasn't clear

on that. But then the SNOBOL books weren't clear on that either when they dealt with that issue, and I think that has definitely been rectified in the new book. I do think I may have mentioned something along that line to Ralph once or twice. I don't remember at this time saying that, "I wish that he might fix that in future documentation to help out in the teaching" [referring to discussion of pointer semantics in Icon book.]

CARGO: When you were instructing people about Icon and it came time to talk about debugging facilities, which of course anybody who has to develop Icon programs really needs to use, did you get the feeling that the Icon debugging facilities as they are now, or as they were a version ago, were adequate for the kind of purposes that people needed - basically, tracing and display and things like that?

GOLDBERG: Yes, I always did talk about how to debug programs in Icon. I mean, I don't remember everything, but your question brought that out. Yes, we did; I would always talk about tracing. I would always talk about the various built-in functions like image and things like that to help get debugging data out. Clearly, they're not as good as some of the more modern programming environments like TURBO C or PASCAL where you can set breakpoints. I mean, that would be phenomenal if you could have a real breakpoint. If you look at the tools that you have with just being able to do function tracing and see what's passed in and what's returned, that's a very valuable tool in and of itself that can often solve many people's problems. The idea simply that you just put a right expression in and it can print almost anything out without doing a whole lot of effort. Right? You don't have to worry about a printf format string, or use fifteen function calls in MODULA 2 to get a line of output out is a very powerful tool. So I think actually in the environment that we were in, which was teaching people about programming languages and having them write these programs, the debugging facilities were a big help, and certainly people would probably compare to some of the alternatives, like if we were using MODULA 2 at the time there really wasn't a debugger [laugh]. It was very useful - very, very useful. Icon could only improve in the sense if you could get a real debugger to allow you to do breakpointing and inspection of variables. And that would be I think the next logical step. Unfortunately, it's not a small step; it's a relatively large step. But yes, the debugging facility is extremely important.

Another big issue, at least as an old SNOBOL programmer, was the fact that all variables were initialized to the null

object, and that you really can't do very much with the null object without causing an error is in itself a very big, in my view, debugging feature, because it causes programs that have done something wrong to bomb out right away with a reasonable message. About the only time that it has been difficult to debug a program, and I know this has happened to me, is where I have had a conflict between string scanning and goal-directed evaluation, where I sometimes would write too complicated a string scanning clause and I didn't get the results I expected, and I really had no easy way to find out what was going on.

TAPE 2/SIDE 1

GOLDBERG: If you are working with commonly available debuggers now they work at a statement level, or of course at the assembly code level. Now, a debugger for Icon that would allow you to get into the goal-directed evaluation would be very nice so you could see when it was backtracking and what was going on, and that would have been a big help. But of course, it wasn't a make or break situation. It was one particular string scanning expression that all I had to do was recode it and I could get around it, but had I the tools it would have made my life easier at the time. And something I should probably mention to Ralph, although I am sure he has already thought of this - how to deal with that.

CARGO: Did your students find the facilities adequate? I mean, mainly, you would know if you got lots of complaints.

GOLDBERG: Yes, I was going to say, really, there was an absence of complaints about Icon. Basically it was very easy. The last few times I taught the course we would use MODULA 2 in the beginning because we wanted to use a block structured, more statically-typed language. And then once we would get past that we would get into Icon, and it was always very pleasant to play Icon off against MODULA 2, and because of that, most of the students really had no problems with Icon at all. Certainly, a few people would come up with questions to me, but in general, people caught on.

CARGO: I was asking apparently specifically about the debugging facilities.

GOLDBERG: No, none of the students ever inquired about the availability of the debugger. I assume they just thought it wasn't there; if it had been, they would know about it. We would have told them about it. There were certainly no complaints. You did have to talk about the debugging tools. It was apparent that many of the students would not have independently read enough of the book to figure out what they could have used to help them out. And sometimes you would explain how some of these facilities could be used to help them debug, and it was clear that someone in the class had suffered through this and wished they had known it a week before, or two weeks before. So, no, I think the reception was that the tools that were there were a help. But no one did ever ask specifically for a debugger; that never came up.

CARGO: You mentioned using MODULA 2 earlier in the semester. I was wondering if you actually had problems where you would first give them the problem in the MODULA 2 section and then later give them the same problem in the Icon section so that they had a more direct comparison of what...

GOLDBERG: We didn't do it quite like that, but as an example of the type of problem we might hand out in the MODULA 2 section, the problem is it's very painful to write any sort of program in MODULA 2 that does any sort of string analysis - lexically. I mean, it's just too painful. It's not that you can't do it; it just takes too much code, too much time. So we learned after some experiences with this that what we would end up doing is we would concoct a problem like, read in a line containing a tree represented by parentheses and single letters and delimiters, and then dynamically allocate the storage to build the tree out of the modular heap, and then do various things to it - traverse it or add it up, or whatever it represented. And of course we would simplify the string. We would simplify the lexical aspects by using single letters. And we would force them to use the dynamic storage allocation facilities that were explicitly available through MODULA. And then later on when we would use Icon, and we would have them perhaps interpret a real programming language, all of a sudden you could talk about the fact that, "Boy, isn't it a lot easier to rip apart input lines and get tokens out of them in Icon than it was in MODULA? Think about all the work." And then, "Well, think about storage management now." You know, you just say, "I want a new node. And there it is,

and you don't have to worry about freeing it later. It just goes away if it's no longer needed."

And so we were able to play it up, but we didn't explicitly say, "Write the same program again." We did it by having them write far more complicated programs using some of the same basic fundamental units. And you would say to them in a class, "Well, would you have wanted to write that program in MODULA?" [laugh]. And of course, the response from the class was, "Absolutely not," meaning that they saw the differences between the languages. And we would try to present that it's not a question of bad and good, black and white, that MODULA is terrible and Icon is wonderful, but the fact that certain tools are more appropriate, certain programming languages may be more appropriate based on what project you are working on and how it is going to be used, and how often it might be used, and how fast it might have to run, because certainly, as we are well aware, Icon doesn't always execute as fast as comparably coded C programs to do similar things. But we were able to make those points because the projects got subsequently more complicated with more details that just couldn't have been managed with other languages. I think that would come through very clearly. By about two-thirds of the way through the semester you would be able to draw those parallels very easily and have the students nodding their heads in agreement, or cheering in the aisles in some cases, because of the Icon spirit and what it allowed them to do. It worked very nicely.

CARGO: Out of curiosity, going back to one of the things you said earlier in the CS540 course, where you did have people who did the first part of a project in C, did you notice, in fact, that performance differences between C solutions...

GOLDBERG: Oh, yes; very much so. These grammar analysis programs, you know - they would go compute transitive of closure [laugh]. And you would have this matrix of binary values, that "Icon is not happy and I guess I'll have to go along with your suggestion for matrices now that I said I wouldn't go along with it yesterday because I suffered through lists of lists to deal with that." [Reference to enhancement for support of matrices in Icon proposed by David Cargo during workshop.] I hadn't thought about that. The performance problems on those programs that did grammar analysis, or generated parsers, really could bring a machine to a grinding halt. I remember that I would pump a hundred production grammar through my versions of the code. And I would walk out of the room and come

back maybe 15 minutes later, or 20 minutes later, and this was running a 386 - 20 megahertz. Why did I do it in Icon? Because, again, the programmer time and the debugging time was more important to me. That was really the issue. I certainly could have coded it in C, but my time is valuable and I wanted to get something up easily and quickly, and Icon was the way to go, which of course was what I tried to convey to the students. But yes, it really was this case where in order to use the tools, you had to limit the input data to some rational size if you expected to see results. The compiler itself could run very quickly, but anything involving the grammar analysis part that really required computations of first and follow sets and all the other things were very, very CPU intensive. And the C programs ran in a few seconds and the Icon programs ran for a long time, but I don't regret coding them in Icon, because of all the standard reasons I gave before. It was the right decision for me at the time, and all I need to do is keep finding faster machines [laugh].

CARGO: That raises an interesting question that is, how much variation in performance did you see in the homework returned by your different students?

GOLDBERG: The TAs graded that. I don't really know. We never really looked at that. One of the things that was clear was that the people who had 8088 PCs, of course, had a very difficult time of it, because the machines were substantially slower. And the feedback we would get from that was that they were just painfully slow. But then again, all things considered, it was better to do it at home and have it slow than spend all the time in the basement of the computer center. So, I mean, on balance, they were still probably happy. But we really never considered the performance aspects of the students' programs. Basically, for the 540 class in particular, what we expected them to do when they would hand in a programming project is prove to us that it worked by running various sample data. And we would make some available, but I liked to have students generate some of their own test data, because otherwise you have students tailor their programs to your data. And then, of course, we would run some of our test data through it as part of the grading process. And certainly, it was somewhat time consuming for the TAs. You know, a grammar analysis program on a 15 production grammar runs fast enough in Icon that it's not painful. It's when you get the explosion in the size of the matrices, because you have introduced so many new terminal and non-terminal symbols and you start to build 70 by 70, 80 by 80 matrices, this is when Icon wasn't designed to handle that, and I

would be the first to admit it.

We did give some thought, or I should say that I gave some thought, to recoding them to using the new bit operators that became available later on, but I haven't had the time to convert them. And the hope there would be maybe I can get the performance to be rational and use it a little more than I have used it before. At one point I did move the grammar programs to the ENCORE running Icon, but it was just too inconvenient for me. It just was much easier for me to drag a PC home at night and run it at home. And so I continued to run the grammar analysis programs on the 386. Again, tying in, why did I want 386 Icon? [laugh]. Because I wanted to run these big programs that I couldn't get to run fast enough or handle large enough grammars on the standard MS-DOS version.

CARGO: And using the VAX was also too painful?

GOLDBERG: Yes, again, it's very hard to go back to 2400 baud dialup access if you could have your own PC. From my perspective, that's rather what I would be doing, and then when I have the files generated that I want, I'll upload them to the VAX or the ENCORE and let the students get to them from there. I have been privileged enough, or I should say lucky enough, to have the ability to take a 386 home from work, so it spoils you.

CARGO: I had a couple of questions involving your students' homework. One was whether you did things like save "a best solution" thus far to a particular assignment and hand that out to show people what other people had done by way of a solution. And a related question is about whether any of those had ever been submitted to the Icon program library.

GOLDBERG: The answer to both those questions is no. They are both interesting ideas. Yes, A, first of all, I think that some of the grammar analysis programs or the parser generators could have been submitted or should be submitted to the Icon program library. Whether they would really be useful to someone else, that's hard for me to judge. But on their own, they stand as good programs that do something useful. And secondly, no, we didn't typically save assignments. Now part of this is because, depending on the number of students and the TA grading

process, I didn't necessarily see every solution. I have personally saved some solutions to certain programs, problems we assigned where I thought the solution was unique. I have never handed those out. Perhaps I should change some of my ideas about teaching based on this, because I think that's actually an interesting idea.

Certainly, like in the 540 class, we would talk about how you represent things in Icon. And I know that if you are doing an LR parser generator and you have to store all the dot positions and the productions, that becomes an interesting problem and how do you solve it? And the traditional record approaches are not necessarily the best, especially when you have a small memory machine, like an MS-DOS machine. And one of the solutions that I came up with is to encode them as two integers and play the old game of stuffing two bytes in a 16-bit word by shifting, and that is what I did in my program. And I talked about that with the students. And I don't necessarily say that's a good programming technique, but it's a desperation programming technique to have something work within the constraints.

So we would talk about things like that, especially more so, I think, in the 540 class that in than the 440 class, mostly because in the 440 class the issues were not necessarily how you did it in Icon, but how you solved the problem. Again, if you were writing, as one of the projects was, an interpreter for a subset of SCHEME, the issues that we would talk about in class were more along the lines of, "Well, how do you read in that expression? How do you traverse it? How do you hook up the linkages between your symbol tables so you know that this is..." You know, all the standard things. And so, we didn't necessarily get to Icon specifics in all cases.

I must say, I haven't taught that course in over a year and a half, and there may be things I am forgetting there, but I know positively that we never handed out student solutions. One of the problems that we faced (and this is just an aside) is that there has been quite a bit of cheating on programming assignments. And the problem in handing out an assignment is not that you are going to reuse an assignment, because we don't reuse the assignments in that course because of that problem, but the fact that since many of the students are part-time, sometimes because of business, they don't necessarily attend every class, or they don't necessarily hand the programs in on time. So you have this conflict between going to hand something out and having people still not having handed in their solutions yet.

CARGO: I can certainly see much as college math books. What are solved problems in some might be half solved or simply assigned. I was thinking more in terms of having somebody else's solution to a problem that you might mention in class but not have as homework would still give people additional perspectives. And another thing that I was sort of wondering is whether you had students recommend problems based on, for example, their work that said, "You know, I have to solve problem X" and make that one of the problems that you have done in class.

GOLDBERG: No, that hasn't happened. There's a few times where problems that have come up in my work I have translated one way or another into problems for a class. But the vast majority of the students don't volunteer. If you ask a question directly you may get a response, but they don't necessarily volunteer more than what is required. Some did, but most don't.

CARGO: I have found occasionally that when people have a particular problem, so they have more of a vested interest in getting a solution, they will be a bit more vocal about it.

GOLDBERG: No, that hasn't come up that I can remember. I know that one student the last time I taught 540 was a Macintosh user, of course, which puts him in an odd situation at that time. There was no Macintosh Icon and he would really have liked it. Macintosh ProIcon came out just after the end of the semester, so he perhaps has purchased it. And this was a case of a student who became very enamored of Icon and planned on getting the Macintosh version and using it where he worked. And he already knew of problems he could attack at work using Icon that he didn't want to attack via other mechanisms. But other than that student recently and the compiler course, no, I don't recall any doing that.

CARGO: You had mentioned, you know, thank goodness for e-mail in terms of talking with Ralph about the porting problem, and I was wondering whether you still had access to e-mail and whether you got Icon Group mail?

GOLDBERG: Yes, I actually have two UNIX log-ins, though one is on a public access UNIX, which is very strange.

You have to pay a fee, but they don't tell you when you have to pay it; they just cut you off. So my primary e-mail access is through IIT's UNIX system on the ENCORE, and I read the Icon newsgroup on news.

CARGO: Comp.lang.icon?

GOLDBERG: Yes, all the time. And I also correspond with Ralph regularly - not necessarily always about Icon at this point. But certainly in the past when doing ports I responded. If there was something that was really hot, you know, if I was sitting at home at 9:00 at night and I ran into a problem, I didn't wait for e-mail; I picked up the phone. But if it was something that could wait I would always use e-mail because it was much more cost effective. I wouldn't have a long distance bill. And sometimes we might even e-mail code back and forth. But generally we didn't do that. First of all, the volume of code was too big. You could only have short chunks of code that you could effectively e-mail back and forth conveniently. I have been using e-mail in one form or another since sometime around 1977 when there was a Planet system on... Who was paying for it? Arpanet. And we were using that to talk about various issues, including SPITBOL and other things. And I find e-mail invaluable.

I find the Icon newsgroup interesting because people ask various questions and I see the answers. Some of the questions, of course, are not so interesting, but some really are very interesting, and the answers are very interesting. And also sometimes the code that's posted there by various contributors is interesting as well. Jerry Nowlin, who is at this workshop as well, has posted some code over the years that I have found very interesting. He definitely writes programs in the spirit that Icon intended them to be written in. I would feel very alone and isolated if I didn't have e-mail and access to a UNIX news system, quite frankly. I typically log in, at least once every two days, and usually once a day. But it does sometimes go a couple days before I get a chance. But it's a very important link.

CARGO: I was wondering whether you had made any connections then with any other Icon instructors. It sounds like you hadn't bumped into any and then found out that Steve was one up here.

GOLDBERG: Yes, I actually knew Steve was one up here. I just haven't corresponded with him. A lot of these things

are like I didn't think of them, and, "Boy, that's a good idea and I ought to do that." I should get on the phone tomorrow morning before I leave and call Steve and say, "Steve, what material do you have in teaching Icon that might be transferable to me? You know, what types of programming assignments have you given? What types of sample programs have you shown people that you came up with?" And I think that I could learn a lot and maybe even give better presentations in class based on that.

First of all, that type of information has not been flowing over the network because no one has posed the questions. And certainly not in the Icon news group. Most of the questions there are about, "How can I get flavor X?" or "How do I port to flavor Y?" or "Here's something I ran into a problem with and how do I solve it?" I haven't really remembered seeing anything about the teaching. Also, in general, I am just relatively silent on news. I am more of a reader than a contributor. I have no problem talking one on one, or even groups or anything like that, but I have stayed away from contributing to the news stuff. No rational reason; I just haven't been active as a contributor.

Based on this workshop, and I really felt this today and yesterday in talking about the program library, I had already decided that I would really like to contribute some programs to the Icon program library. I hadn't thought of contributing student solutions based on their agreement, but I think that's a very good idea. I was thinking more along the lines of programs that I would have written in the past or problems that I might want to write, or functions, procedures I would want to write that I would contribute. But I think some of the problems we have tackled in the courses other people might find interesting to look at. Just the idea that we had students write a SCHEME interpreter. Not that it's so hard to write a SCHEME interpreter, but maybe there is someone out there who is interested in LISP, SCHEME and Icon, and here is a way for them to fool around and play, learn something extended, make it into a better tool perhaps.

The grammar analysis program - definitely useful; no question about it - for people working in compiler area or syntactic analysis, just the front ends. No question it should be contributed. Part of the problem, of course, is that the student programs are not always commented, much less industrial strength. You know, even my own contributions would probably have to be cleaned up before I would be willing to submit them. But in general, on

valid input they work. The question is what they do in the event of illegal input, what happens?

CARGO: I guess that raises an interesting question about do your students programs get graded on stylistic things like are they commented and are they easy to understand? And when you think about trying to prepare them for going out in industry, does it make sense to grade them for writing programs that other people can understand?

GOLDBERG: Boy, you opened a rat's nest there! You've got a hard-liner here. I have taught programming one way or another for a long time, and I remember the days we used to give whatever the breakdown was for style and documentation, whatever. And you would end up giving somebody so many points, and they didn't have to have a program that did anything - meaning didn't do a thing. Didn't work, and in some cases couldn't even compile. And after years of seeing the system abused, we have come around to the idea that your program has to work. Granted, maybe it's not a hundred percent working, but it has to do something. It has to do something rational. It has to have some comments. We have to be able to see what's going on. And after that, we don't do much, because at this point we are not trying to teach introductory programmers how to program. They are supposed to know something about programming when they walk into the comparative programming languages course. And therefore, the things that we are trying to impart to the students in the class are not the same as the lower-level programming classes. So we feel at this level the programs have to do something. And that's just the requirement.

And I can tell you that this really came about as a result of years of suffering through students who had no idea of how to program whatsoever and were trying to fake it by making it look nice but it effectively did nothing. We just can't allow that. It's not fair to the student, and granted, many times they just want to get the grade and get out, but it's not really fair to them. It diminishes the meaning of taking the course and getting a grade and getting a degree at the institution. And it's also not fair to the students who sweated through getting a program to run. So we have really come to that conclusion after seeing the system abused for so many years. And we don't have any complaints. I mean, most of the time, students who are not doing a good job know it. And they are going to push to see how much they can get away with. That's fine; that's part of the game. But if you are fair about it and you make it clear - your program has to run. It doesn't have to be 100%, but it's got to run. You can't cheat; don't copy someone else's

program - don't steal it. Get the ground rules out in front. We don't usually have too many problems.

So when the TAs grade the programs or I grade the programs, we are really looking for, what does the program do, and can I understand what it is doing? And that's how they get graded, based on that. In general, I will say that the programs are usually not commented to industry standards. But then again, there's plenty of industry programs I have seen that haven't been commented either [laugh]. So I think that's a generic problem. I mean, Bill Mitchell was talking tonight about how at one point when he was affiliated with the Icon Project, Ralph mentioned, "You know, there's not too many comments in the code." And Bill took out, you know, I don't know how many days of sitting down and going through the code and inserting comments. That's somewhat of an interesting experience to document the program after it's finished rather than while it's being developed, but I don't think it's a totally unreasonable way to do it.

I will say that one other aspect of grading that has been applied that really has nothing to do with Icon is that we will tell people that they have to use a certain method of implementation. As an example, if they are going to read in a line and build a tree out of it, we will them they have to use dynamic storage allocation. And some people persist in not doing that. They don't get full credit. In the case of a compiler course that I taught, I said that people had to build an internal tree of the expression and then walk the tree to generate code. Some people persisted in generating the code on the fly as they parsed the expression. And these are things of course where you have to read the code to find out what they are doing, and the comments do help if they are there. Often they are not. But we don't require them, for an example, to use a generator or to use co-expressions or to use string scanning. If the course were oriented in a different way I think we could do that, where we were writing a number of Icon programs to explore features of Icon for their sake, then I think we could use that type of grading scheme, or use that as part of the grading scheme.

CARGO: I think I have run out of questions unless you have thought about some other things.

TAPE 2/SIDE 2

GOLDBERG: I am looking forward in the future to teaching the comparative programming language course again so that I can use the new second edition of the book - the Icon book - and see whether that allows me to make a better presentation of the language and the features - my feeling is that it will - that it allows us to get into the heart of the matter right away. So that should be fun - from my perspective at least. I think the students will like it as well.

CARGO: I was just realizing the area that I hadn't had an opportunity to ask any questions about. And that is you said that you were doing your teaching in the evenings, implied that you had a job during the day - and whether or not you use Icon in your job.

GOLDBERG: Actually I do. I personally use Icon to write utility programs - to do conversions or do analysis of. I basically manage a department of programmers, and I have to make decisions about what special projects are doable. And I wrote an Icon program for my own benefit recently to analyze the sequence of commands that a front-end typesetting system was sending to a typesetter to see whether we could intercept it, make sense out of it and convert it to something we can use. And I did that in Icon. Frankly, if the project is now a go, we will probably write the final version of that program in C, because it has to be run every day and has to run reasonably fast. But the fact was that it was far easier to write the program in Icon to do the initial analysis because, again, my time was valuable. [Program was recoded in C.]

I have also generated a program for our installers that one of the typesetters that we talked to has a separate command language that allows users to get certain types of effects through this command set. The problem is that the character set for the commands, of course, is different from the character set that's used to get characters on the typesetter. And there's this whole code conversion problem. And what they did is they published some tables of character codes, and our application engineers and these installers would have to take a string of 30 or 40 characters and go look it up and figure out what the octal equivalents are. And I mean, this is really something that should be automated. And I wrote an Icon program to do that - that basically they say what typesetter they are going to, what number base would they like the output to be in, and a few other issues that ask them questions. And if they don't give a reasonable response it tells them, "No, that's not reasonable. Answer the question again." And then they

basically type in the strings to convert, and then it does all the conversion for them and prints out the results. And this has saved them a lot of work. And I coded it in Icon because originally I coded it for myself, because it was helping me out. And then they realized I had something that they could use and so I spruced it up a bit and now these people are running around with the iconx interpreter and the ICX files and are running this program. They have no idea, of course, what Icon is all about, but they are using the program.

One of the people who reports to me is a SNOBOL as well as an Icon programmer, and we had been moving our software from among various PC platforms and we had a need to eliminate some space, squeeze some static area space down in our C programs. And one of the ways to do that was to go through and eliminate duplicate strings. And he wrote an Icon program that went through our whole source file system and tabulated the strings and found out where the duplicates are and managed to do major restructuring of how we handled string constants within our C source code. So I think it was very valuable - something that wouldn't be rational to do in C. It would have been too expensive. We couldn't have afforded the time. But in Icon it was very easy, and of course it was very easy to debug and verify that it worked.

So we basically use Icon at work as a set of programming tools to help us do our jobs, and it's really valuable in that perspective. And interestingly enough, while some of the utilities may be considered throw away, some of them are definitely not throw away and are run regularly, which I find very interesting, because I certainly didn't write them with that intention. And particularly in the case of the code converter for the typesetter, I didn't expect it to be run all the time. I expected to run it once or twice myself and be done with it. And all of a sudden it was turned into not a production program but a program that's run regularly. And of course it helps again that Icon is free.

CARGO: These systems that your installation engineers were using in the field, I assume they are 386 systems?

GOLDBERG: Yes, they are networked together for newspaper front-end systems, which are what newspapers use to produce their papers. Reporters enter their stories; ad takers take ads; production people make up display ads; editorial staff lay out editorial pages, and someone might lay out classified pages. And there's a variety of

opportunities for conversion projects and bringing in outside copy that's tabulated, that's in some format that's not quite the way you want it to write an Icon program to do that. Our application engineers are not programmers though, so they're not able to do that. It really had to come through the people who are the real programmers in our organization.

Of the nine people currently there in my department, there are four people who know Icon. And three of us have written utility programs using Icon. The fourth may have; I am not sure. But three of us have definitely used Icon at work to write utility programs. Again, it's just the right tool for the job at the time.

If we were to build a real program that had to run regularly on a customer installation, I might have some difficulties in promoting that. A variety of reasons that come to mind are: one, will it run fast enough? I mean, that's always an issue. And then, two, since not even a majority of the department knows Icon, there's the issue of how do you train the rest of the employees, or how do you get them to support a program that's not in the standard language that we use? I think that's really less of an issue with the talents of the people that are in the department, but nevertheless, it's something that management has to be concerned about at some level. But it's more than acceptable for writing the utility programs, and we do that on an as-needed basis. And it happens regularly.

CARGO: About how big do those programs get to be?

GOLDBERG: Oh, the program I wrote was probably 150 lines. That includes comments, so I am not talking major development. I am not sure how big the string analyzer was. I never really looked at it. I mean, I just knew it was running in Icon and I said, "That's fine; that's the way to do it. Let's do it." So I never even saw the code. I just saw the results. I am assuming that was a relatively larger program, although I don't know how much of the syntax the program actually parsed of the C programs. It may have done a naive parse.

CARGO: Did those utility programs tend to run on the ENCORE?

GOLDBERG: These programs all run on 386 PCs. They may use 386 Icon, or they may use the standard MS-DOS Icon. It's basically what's available and what's needed. The ENCORE is really only used for IIT coursework.

CARGO: Oh, right. I got confused about that.

GOLDBERG: But Icon is on a number of the programmers' machines and I put it out on our Novell server and made it available.

CARGO: Did you have to go through any kind of justification to your management about using Icon?

GOLDBERG: No, I am very lucky in that situation. Historically, I have worked in at least one organization where I had to justify certain activities using SPITBOL. Many years ago I wrote a cross-assembler for a large bank I was working for at the time, and it was for some sort of bizarre NCR machine that was cassette-based or something, and their assemblies took 30 minutes. And I said, "Well, I could write a cross assembler and get it done." And they called me on it and said, "Okay, do it." That's why I ended up using SPITBOL, and it ran fine and saved them an awful lot of development time. But yes, you have to justify it in situations like that. In situations where I am now, it's a small company, 80 employees, and I can pretty much use whatever tools I want to use. I don't have too many problems there. We do have ground rules that we agree to, like we are going to use Novell for our network server and things like that. But at the level of, "Well, gee, is it assembler language or Icon or C?" that's not really a problem. [Referring to utility program, not major products.]

For the conversion program, the issue, certainly for internal use, is not a problem at all. Again, the only problem that I would have, or could potentially have, is that if we did some large conversion program was going to be run on a regular basis from now and for a long time hereafter, I would have to make sure that I could support that program. So that's really the major issue. I would have to be able to support it. It's not that it wouldn't run. It's a question of, what happens if it breaks or it needs to be changed. I would have to be able to justify it and support it.

CARGO: Yes, I have found myself thinking about that particular problem and thinking about delivering the Icon version as a prototype, which would mean that it would get to the customer's location that much more quickly, and then continuing to work on the C program and eventually delivering that.

GOLDBERG: One of the problems, of course, is that prototypes have a habit of becoming final versions. And this is not always true. But once you have got the coding done, even if it's a prototype form, often there's this tendency that you're forced into going on to your next project.

CARGO: I was thinking from the standpoint of your customer, it would mean that instead of having to wait a month for a C version, he might have an Icon version in a week.

GOLDBERG: Right. It might be to their advantage, yes. Typically, I try to pad the scheduling of the conversion projects to give us enough breathing room. It doesn't always work that way. We don't always have that luxury. I think that the common thread here is that there are two factors. One is the cost of programming time, and Icon can definitely diminish that in many cases. And then, your point about the fact that since you can diminish the programming time, then you can reduce the schedule in some way and deliver a product quicker. I agree with that. Many of the products we build, though, cannot be in anything but C. I mean, we have to access databases on a network server and all these other problems that Icon is really ill-equipped to deal with. But in terms of conversions, it's at the top of the list.

I have used SPITBOL once for a conversion, and I am not even sure if I could recall at this point why I used SPITBOL instead of Icon. It really should have been in Icon, and that's really the last SPITBOL program I have done in quite some time. That was about two years ago. Everything else I do at this point is in Icon for those types of projects.

CARGO: Out of curiosity, is there an increasing number of Icon-literate people in the people that report to you? In other words, are some of the people who are already there learning Icon in addition to the ones who already know it?

GOLDBERG: At this point, the people who know Icon are coming in from IIT. I would certainly like it for the other staff members to learn Icon. We don't really have the luxury to tear them away from their work at this point, to allow them to learn it. Some of these people have talked about doing it on their own at home, but that is hard to do and certainly not something that you can require. And I wouldn't dream of requiring it in that form. But I would, once that we have a scheduling situation that's a normal situation where there's some slack. I would very seriously consider handling an internal Icon course to get people up on it. I think it's a very reasonable thing to do. We have really had a long two years at my company, because we have converted our systems, not once, but twice in the last two years as part of a master plan to move us off proprietary hardware, which we built, to PCs running our software, including operating systems, to now PCs running Novell with just our applications software. And there hasn't been enough slack to entertain things like this. I would think over the next year that could happen, and I certainly encourage people. I have an Icon book on my bookshelf, and I have Icon on all my machines, and whoever wants to have access to it is more than welcome to have the book and read it. But I would really prefer to have a short course on using Icon. I think that would probably be the way to get people rolling - to spend a few hours with them as experienced programmers and talk about it and have them look at some programs, have them write a couple little programs. And then if it stuck, that was great if they wanted to use it, and if it wasn't, I don't think that I would force them into it.

CARGO: I think those circumstances indicate why I brought up education and particular materials for teaching experienced programmers about Icon.

GOLDBERG: Yes, you see, I go back to a statement I made earlier, and I really do feel (even the first generation of the Icon book is still far more readable than many of the other books on programming languages) that a reasonably bright programmer can pick up Icon very quickly. That doesn't mean they're an expert, but they can definitely use it to do reasonable things. Some people need more guidance, if you will, than others than a short course can handle. I haven't really seen a primer on Icon. That really doesn't exist yet to help out here. The *Icon Analyst* may serve that purpose. Of course, its cycle is such that we may have to wait for the first two years' worth of issues before we have a complete primer. So that may not be acceptable in terms of my timeline or other people's timelines. Just thinking off

the top of my head, you could clearly in a four hour-session move them through some of the major aspects of Icon. And in two four hour sessions, with experienced people again, I think you could cover a number of the salient features of the language in a rational way with sample programs and appropriate handouts to give people a real flavor of the language. I really think it could be done. It would require good planning; it couldn't just be done off the top by walking in and doing it. That's something I would enjoy doing at work, and we're setting up, as it turns out, an in-house employee education center. And we are going to be teaching various internal classes in there. They could range from anything from "Introduction to DOS and PCs" to "Running Lotus 1-2-3" to running our application software, which requires special knowledge and configuration knowledge. And I could use that facility to teach Icon in-house. I could technically teach non-programmers, though I think that would be a lot harder. I did once teach a SNOBOL course in a commercial environment to non-programmers, and I found that somewhat of a challenge [laugh]. I am not sure how successful it was either, but it made me think about different types of projects and programs we have people do. As an example of a program that I had them do at that time was to read in a number and write out its value in English - to have to think about the process of doing the algorithm, plus it's a very natural program for even Icon as well, but at that time SNOBOL. It was very, very difficult to get through it. It was very difficult to get done. So I am not sure that I am the right person to teach non-programmers Icon. I would give it thought, but I really... I think a short course for the in-house programming staff would be extremely valuable for many of them. I think it may turn into a tool they would then use.

CARGO: Now, I think I am going to quit.

GOLDBERG: Okay.

CARGO: Well, good luck.

END OF INTERVIEW