
# The World's First COBOL Compilers

Talk by:
Harold "Bud" Lawson and Howard Bromberg

Recorded June 12, 1997
Palo Alto, CA

CHM Reference number: 102639620

# "The World's First COBOL Compilers"
# By Harold "Bud" Lawson and Howard Bromberg

## The Computer Museum History Center Lecture Series
## Gates Computer Science Building, Stanford University
## June 12, 1997

LEN SHUSTEK:

Thank you all for coming. My name is Len Shustek. I'm a volunteer with The Computer Museum History Center. For those of you who don't know, The Computer Museum History Center opened about a year ago in this area; we're an offshoot of the Computer Museum in Boston, and our mission is to preserve and study and celebrate the history of computers and computer technology. Eventually we're going to be opening an adults-oriented museum somewhere in Silicon Valley, we don't know exactly where yet. In the meantime we have offices over at Moffett Field, the ex-Naval Air Station you can see from the Freeway. But more importantly, we have a warehouse over at Moffett Field where we're collecting tons and tons of old computers.

Since we don't have a museum yet, we're doing a bunch of intermediate activities, one of which is this lecture series, of which this is one, and the next lecture will be, in fact, over in our warehouse.

We have some information on it here, if you want to come up later, but it is June 24th, at 5:30, at The Computer Museum warehouse at Moffett Field, and that will be a talk by NASA engineers on the ILIAC IV, and in fact we have some pieces of the ILIAC IV, not the entire thing because it would fill up our warehouse over there.

We also have various toys. Gwen, do you want to talk about the items we have here? Gwen Bell is the "soul" of The Computer Museum, and our guiding light, and she can talk about some of the toys we have here today.

GWEN BELL:

But the other thing I want to say is, to come to Moffett Field you have to be badged in, and to be efficient about that, send email to allison@tcm.org and Zoe Allison will make sure that your name is there at the gate, so that you can get in as efficiently as you can. That's what you have to do for that.

We do have products! We did develop a product here, last fall. We spent most of last fall doing this work to determine all of the microprocessors that were ever done, and this is a poster of them at four times scale. All of them are in scale. It has been corrected a number of times. We still, but, you know, someone will tell us something else is wrong with it

sometime. [Comment: ... will be corrected.] Right. We have order forms for it. We also have videotapes that - we've had lectures since 1979. And, this is on the pioneer computers and computer pioneers. In the first three years we had Grace Hopper, and George Stibitz, and John Atanasoff, a number of people who are no longer with us. And so we put this together and taken the high points off into a series of two tapes. Here you can see, hear Grace herself, telling the "bug story", and it's much better than anybody else. And that's the way to have it. So, we have order forms....

This is another poster we have, which is on memories, and was done on all of the kinds of memory devices we have in the collection. And a Bowl - the official Computer Bowl Trivia Book, because every year we do the Computer Bowl - we won't have another one until April. But we will do that again. And there's an order form. Thank you very much, and now on with the ....

LEN SHUSTEK:

We're privileged today to have two people who really are pioneers. Certainly from the early days of computers, but from the very beginning days of high level languages for computers. And they're going to talk to us today about the first COBOL compilers, that were done in the late '50s - early '60s timeframe.

We have two speakers today. Let me introduce our second speaker first: Howard Bromberg, who was one of the original authors of COBOL as a representative on the original CODASYL Committee from 1959 to 1962. And then he was the first chairman of the ANSI Standards Committee for COBOL. He worked on the compiler for the RCA 501 for COBOL at the same time that the UNIVAC compiler was in process. After that he was founder and president of International Computer Technology Corporation, in San Francisco, which does software development, and was a consulting firm. They did design and development of commercial applications, COBOL compilers, and custom programming. Howard has been active in the ACM for many years, including a stint as a national lecturer for the ACM. He is a fellow of the British Computer Society and was the chief U.S. delegate to the ISO committee on programming language standardization. He is currently a computer industry consultant. And Howard will speak second.

Our first speaker is Harold Lawson, otherwise known as Bud Lawson, who was assistant programmer for Remington Rand-Univac, starting in 1959. He worked on the other COBOL compiler that was in progress at the same time, working under Grace Hopper, and he'll tell us about the compiler and what it was like working for Grace in this process. After that, he worked for IBM; he was Director of Programming Research and Development at the Standard Computer Corporation; had a number of university positions: visiting professor, a professor at Brooklyn Poly, U.C. Irvine, the University of Stuttgart, Royal Institute of Technology in Stockholm, Polytechnical University in Barcelona, and Linkoping University (did I pronounce that right - Linkoping?). He's also been an ACM national and overseas lecturer, a Fellow of the IEEE, and since 1988 has been the Managing Director of his own consulting firm, Lawson Konsult.

It's a great privilege for me to introduce Bud Lawson, because when I was a student at Brooklyn Poly in the late '60s he was my professor, and was the person who was instrumental in convincing me that computers were a lot more fun that physics, and I should switch from being a physics major to being a computer science major, which was a switch I never came to regret. So I'd like to introduce Bud Lawson for the first part of the talk.

BUD LAWSON:

It's a great pleasure to be here, and to be able to share with you some of these experiences from the early days, and particularly that you see that your former students have succeeded as well as Len has done. It's quite a great kick also.

In any event, I want to speak to you about the first UNIVAC COBOL compiler today. As Len said, I happen to be living in Sweden at the moment; I've been there for about - well, since 1971. That's a long time, but that's another story; I won't say why. I went there to design a machine for a Swedish computer manufacturer, and never got away.

But anyhow, some of the early days of computing:  I started my career, as was mentioned, in the Philadelphia area. As well, a large part of Howard's time was also there. I'd like to dedicate this to our common former boss, Grace Murray Hopper. She was a great inventor, a great teacher; she was always questioning things, and she always inspired people in many ways that I think were important then. All of us who grew up in Aunt Grace's boudoir are eternally grateful. This word "boudoir" I think is very interesting because Herb Grosch, who was a very controversial figure - I guess he's still around, somewhere - in an article in Datamation, in the time frame about '60 - '61, called the Automatic Programming Department at Univac "Aunt Grace's Boudoir." I thought it would be interesting to cite that point.

What I'd like to speak about today is to give you a little introduction, talk about some of these early UNIVAC languages and compilers, a little bit before the time of COBOL, just to give you a little framework as to what happened in advance - it's not a complete history; talk to you a little bit about the characteristics of the UNIVAC II, so you get an appreciation for the extreme resource limitations that one was dealing with in designing a compiler for a language of the size of COBOL at that point in time. Then I would like to talk about a little bit about the strategy of how one would look at -- how Grace looked at compilers -- which developed kind of a way of looking at compilers from about the time Howard came there in about '55 or '56, and was used as a philosophy of how one developed the UNIVAC II COBOL compiler, looking at it as a data processing system. That has some interesting implications.

Then I'll talk about, and go through, the actual compiler structure a little bit, and then I'll look at one of the heaviest parts of the compiler, the code generation part, which I had the responsibility for doing, which we happened to call "DUZ" - and remember the old soap commercial, "DUZ does everything", and that's why it got its name. And it was quite a challenge to get that to fit into the UNIVAC II's memory, but I'll tell you about that. And then I'll surprise you all to tell you that this was probably the world's first compiler written in a

higher-level language. Which seems strange for such a small machine, but the compiler was actually written in FLOW-MATIC, which was a previous higher-level language. Very interesting.

I left Temple University in June, 1959, and started work for Grace there. It was an interesting point in time. There was an old Exide battery warehouse on 19th and Allegheny Avenue; dusty and dirty, with the railroad tracks running down the back side. And that's where Howie was also. I remember showing up there, the first day of work, getting a pile of hardware and software manuals and an assignment to build a new generator for the FLOW-MATIC compiler. And if you imagine - I knew a bit about computing; I'd taken some programming courses the year before at the United States Census Bureau in Washington. So, anyhow, that was quite a challenge. And you had a kind of sink or swim attitude about, well, learn it and do it. And we learned a lot by doing it. But we had also a lot of fun colleagues there that helped to get that.

This was the important time that the COBOL compiler was being developed, so after getting this training, I was first put on this team to develop the COBOL compiler. There were four really senior designers that worked for Grace directly: they were Bill Finley, Tom Jones, Dan Goldstein, and Dick Miner. These were my bosses, basically, and did the really high-level systems analysis and design of the compiler from Grace's higher-level ideas. And then there were a number of people which I have cited - I've actually written a preliminary paper that everybody can get a copy of here, which I hope will appear in the Annals of the History of Computing within the next period of time also. It will contain this and probably some other things also.

That was my start in the business there. There were some earlier work that was done, and a lot of the programming language and compiler history - a fair amount of it - actually is attributed to the work that went on at Univac, and with Grace. Her first compiler, which I have the documentation for myself, was A0, that was done in the '51-'52 time frame. And I think probably - I'm not sure - but this may be one of the first papers that the word compiler ever appeared in. I know Don Knuth is sitting over there; he might agree with that. [Comment: I don't think it was a compiler.] No, I agree, that's what I was just going to say, it was not what we would call a compiler. It was a way of collecting together a set of subroutines that evaluated mathematical and statistical functions. So it had some type of little control mechanism for bringing things from a tape library and putting them together. Okay, compiling is the thing of putting things together, so in that regard...

From those humble beginnings, Grace went further, and eventually evolved the language called MATH-MATIC, which looked much more like what we would call a language than a compiler. And MATH-MATIC at least was paralleled, or maybe before, or in the same time frame, with FORTRAN, and was available in the UNIVAC I.

In about 1955 Grace was interested in turning her attention to a type of compiler for dealing with data processing, which was becoming the far dominant use of computers at the time. I have here a document that was produced on a preliminary definition of a data processing compiler, which is stamped Feb. 3, 1955. And that was the origin of the B0

effort, in about the same time frame that Howard joined Univac. Interestingly enough, I could give another side of that, there's an example of a program that's very close to FLOW-MATIC written in the last three pages, there's this English version, a French version, and a German version of the program. So she felt that it was important at that time that things be written in your own language that you would express this in. That hasn't turned out to be the case, but it is interesting that at that point in time, her point of view was exactly that.

In any event, FLOW-MATIC took hold actually within the UNIVAC world; it became a very important compiler; a lot of people did a lot of important work with FLOW-MATIC. There were implementations even done in the Wright-Patterson Air Force Base to use FLOW-MATIC to generate code not only for the UNIVAC I and II but also to generate code for an IBM 705. So you could compile code actually for producing 705 assembly code and assemble it from FLOW-MATIC. This was called the AIMICO system.

FLOW-MATIC became an important influence - one of the important influences - along with COMTRAN from IBM and some other languages, that became inputs to the CODASYL efforts.

But Howard, I think, will give you much more deeper insight than I can give you; he was really a key person in that whole activity. I, being a junior programmer at Univac, only heard these fantastic histories about Jean Sammet and other people that were working on this COBOL CODASYL Committee at the time. It was interesting to learn about that.

Those were some of the earlier parts. I'll just give you a few characteristics of the UNIVAC II, which will give you I think a picture of what that was - here's kind of a view of the configuration of the UNIVAC II. Large central processing unit, and memory, with water-cooled system; was the first use of a - one of the early uses of core memory - the UNIVAC I had a mercury delay line memory - with 1K words, and a word in the UNIVAC I and II was represented in character form [by] 6-bit binary-coded decimal; the decimal part was so-called excess-three decimal, for various reasons, but it was really a true decimal machine. 12 decimal digits - 12 characters to a word; all arithmetic was performed in decimal.

You dealt with input-output on these Uniservos, tape drives, you could have I think up to 10 of them. If you were fortunate, five or six worked (laughs) at a time. You had a supervisor control panel which was much like flying an airplane (laughs). And you learned to listen to programs as well as to observe the lights and see what things were going on. You could really do a lot of debugging by these visual and hear - aspects of dealing with your program. There were a lot of some peripheral equipment which things were done off-line with. [Question: How many words of core?] 2K in the UNIVAC II, of 12K [sic] words, and 1K in the UNIVAC I.

The character set had 64 characters, and here we see the numeric digits were there and that they were in excess-3 because you see that they don't start until this point so the actual binary code at this point three will be equivalent to zero. That's why it was called excess-3 binary coded decimal.

I didn't have an instruction card left, but I do have a UNIVAC II data processing manual, and that's where I found this instruction list. This is all the non-input/output instructions, for the whole machine. The machine basically was very easy to learn to program in assembly language, strangely enough, because all these things like A is Add, S is Subtract, M is Multiply and D is Divide. After a while it became very intuitive because it was decimal: only referenced decimal addresses, you referenced everything with decimal, it was not very difficult to program in machine language. There was an assembler called X-1, which did very minimal things, and you could get assembly code done like that. But it was a rather simple machine to learn to program.

I think we can put that in context with what has happened and where we are today, and one would say that this gives us something to think about in respect to our current world of complex software systems based upon more primitive instruction sets provided by the contemporary microprocessors we saw on the view here. Have we made progress in this area? One could really question that. Having microprocessors which have an instruction set like they were, dealing with the x86 doesn't really make for software-friendly systems. This was - the UNIVAC II was, I think, a software-friendly computer. Interesting thought to think about; I give that as an aside.

So that's a little bit about the UNIVAC II. I could say a lot more about it but I think that will give you a feel for some of the flavor of the UNIVAC II. Instructions - there were two instructions per word, a left half and a right half. Six digits per instruction. As I said, it was fairly straightforward to learn to program.

Compilers are data processing systems. The idea of building a compiler for these machines that would be able to deal with a language like FLOW-MATIC or eventually like COBOL etc. - and even right at the beginning when Grace did this experimentation in '55 with what we called B0, you see this data processing type of thinking. We have got to make a lot of tapes, because we didn't have much memory to deal with, so we couldn't include too much program, or too much of the program that we were compiling, in memory at one point in time. You have to look at it as a data processing system. And at Univac, data processing systems of the day were something like the following.

Here we have these Uniservos, where you mounted up live tapes, and this is where a typical inv - type of data processing system you have an inventory tape and an updated inventory tape where you pass transactions against that and do an updating. So that type of thinking permeates also the compiler, because here in this case the source document - this is source code - written in COBOL, and you actually typed this on a Unityper, and that's recorded onto a tape - conversion here - and then you mount that and start up the compiler, and run it. So it runs very much like that.

The input-output system also was quite simple. Everything was read in 60-word blocks. There was nothing else; that was the only way you could get stuff in and out of these 12-character words. There was conventions to indicate end of file, end of volume, etc., in the beginnings of these blocks which were called sentinels, that would allow you to deal with the treatment of these tapes. I bring this up to show you that you really want to look at the

compiler now as a data processing system. Now, in retrospect, knowing what I know about compilers today, I probably would not have done it that way, but at the time....

And since he's here, I just want to remember because the first time that I ever heard the word Don Knuth was in about 1960. I had heard that you had done a compiler, and won your case. It was about the time that we were working on this, and we said, "Wow - this system was so complicated" - about how we will build compilers, and I knew that you had done the one by yourself. So it seemed to us rather interesting challenge that such a person could write one compiler by himself. But eventually, of course, I met you, so I can understand how you did that work at that point in time. That was the first time I heard your name, Don, anyhow.

[Comment, probably from D. Knuth: It was easier, because you didn't have to explain to anybody else what you were doing.]

That's it! (Laughs.) Thank you. That's good. It sounded like, to us who worked on this compiler, that that was an interesting accomplishment to deal with.

Let's take a look at how this compiler was actually organized. This is a picture of the original design; it's not readable, but I just show it, and I've made a more readable version of this. To give you an idea, that everything starting at the top here with a COBOL source code. You'll see this a little bit larger in a minute. And everything filters through here in a number of passes. Now all the actual passes are not here, because as we'll see, because we key to this as the data processing system we have to get keys on things. So we split stuff up and give sequence numbers to parts of the COBOL program so you could then start processing and then a lot of the passes were actually sorts and merges. You could sort everything, you get all the verbs of one type together, and then run them through, and process them, and then you eventually have to merge things back together and get the program back together again in sequential order. So this doesn't represent even all the passes, but it represents the major phases. I think you could get up to about 60 actual passes, that could be run during a single compilation.

But what I've done is to make life easier: instead of from this original document, I've taken the trouble to make one which we can hopefully read. So we begin here with the COBOL program, and phase B is an initial phase where we do a lexical analysis of the COBOL program, and I happened to write this lexical analysis; this B1 I did myself, so I know that very well. But actually this split the program up from free format into fixed format and isolated parts - the parts of speech, you might say, we called it "the digestive problem" - as the tape that we produced from there. It also produced somewhat a cleaned-up version - "pretty printer", I guess you would say - view of the COBOL program which could then be put back in which you could also run that out to print, later on. So from that "digestive problem" it went into a phase called B2 where we actually assigned sequence numbers to different parts of the COBOL program. You have explicit constants that came out, data descriptions and procedural descriptions from the various different divisions of the COBOL program.

From that, we go into phase C, where it split out things into some types of selective statements; these were particularly related to input-output and environment type statements, then the procedural statements and procedural names. So these were split up so we could process names in a particular way.

Phase D kept the data side of this, mostly from the Data Division, and processed this, calculated sizes of things, etc., and produced the data names list, and a data description that went later on. And an environment and file description which was used for the input-output generation. So that went on to separate tape files. So you can start to see that we had to really have a problem getting the number of servos working properly. There's a lot of tape when your COBOL program is split over a number of different tapes at the same time. So then, this comes up here to generating also Working Storage constants that were there, as well as these descriptions. That was Phase D.

Phase E would put these data names, the procedural statements, the procedure names here, and we generated one of the most important files, was this macro instructions, where we then for each of the verbs have information about what the data operands were, associated with that thing. We called that a macro instruction at this time. We also produced a jump list, with all the GOTOs and their implied GOTOs, cross-reference listing, implicit procedure constants, and then the selected macros are basically things that are going to be related into I-O commands.

Then we go off into another phase called G here. I didn't get everything onto one page, so we now have to take the bottom of this and go into the top of this. Let's see - we were up to E. Phase F here took these macro instructions, the jump list, and the data description far apart from here, and actually could merge these basically so to put them in an order that they would be prepared for, that we could start to use that to go into code generation. This is macro instructions and data descriptions together, and actually at this point in time is the first time that we could detect semantical errors - that you hadn't really used the COBOL language properly. This is quite late in the compilation process actually, that you could actually - because there you could start to get everything together again, and have enough information. Before, we could do all syntactical checks, that was done fairly early on, but semantical errors, this was about the first time you could find out that the program really made any sense.

And from there, these macro instructions went - well, then for F, let me go into G first. We select the macros here into a phase called G, which was the Input-Output; we called it actually TIOGA as a name:  Tape Input-Output Generation and Assembly. Dick Miner, who was the expert for Input-Output, taught me a lot about that and I was also working together with he, and I wrote a large part of both the object - all the object parts of execution for Input-Output as well as this part of this TIOGA. I could tell you some funny stories about testing that in Pittsburgh, but that's another thing. This was the Input-Output generation. This could also generate some constants, that also generated the Input-Output code, which could come into the code generation part here.

Actually, we just passed through code generation, because all the code was already there.

Now, for the rest of the things that are not I-O, we have these macro instructions and then we come into H1. This H1 is DUZ. "DUZ does everything." Because this was really the toughest part of the compiler, getting this code generation to work, with 2K memory for all the verbs we had to deal with. I'm going to treat that separately later on, since I did all of that myself, and that's probably a large reason that I spent a year and a half working 80-hour weeks, was getting that thing to work. It was no easy task, I'll tell you.

Interestingly enough, these macro instructions we called "micro instructions"; we generate the code from here, as well as constants that come out of here. Then there's a Phase H2 where we took these constants, explicit constants, and I-O constants coming in here, and built a pool of constants. Then the J1 phase here was to start to put things together. Here was the Working Storage part, the programs, the constant pool. That finally put together the final object program, in final executable form. Then we did push that over to J1 [sic], where you had your cross-reference list, the original COBOL program in "pretty-print" form (if you want to say it that way), and that finally could be merged out and put a record of compilation and you could print that out on the off-line printer. So, that's the basic structure of the whole compiler.

Q: [Was the object program supposed to fit into 2K, or is it segmented ... ?]

A: No, it could; if it did not fit into 2K, there was a mechanism for segmenting it. But you had to indicate that in your program - there was no automatic part. But it was possible to segment your object program also. It's a good question.

The next part was this DUZ part. Since I got that job, that was really, as I said, a very tough thing to do. It was not easy to deal with that, but I tried to find out a way of analyzing what I could feel would be the most frequently used configurations of COBOL statements, and the most common cases, and prior to then to find a mechanism where these generators for those parts could sit in core. So as I passed this macro instruction tape through this, I didn't have to go out and fetch other tape in. But then there was a library of generators for the other cases, which was in this picture before here. Here we have a library of generators which sat outside here, for the cases you couldn't get into here, which was then read into an overlay area and executed from there, to generate those cases.

Now we called that DUZ, as I say, because "DUZ does everything", and I think that was really where I learned a lot about programming art, and how to deal with programs, and structure them, and deal with this complexity in other ways. This is actually original, which I found and I still have, written by myself, July 1st, 1960. This was my documentation, to describe how a set of internal subroutines, which I developed, which I could then call, as I was compiling these codes, getting macro instructions, calling these subroutines, which would provide services in generating the code. Here, when I needed to Load Register - there were actually four registers in the UNIVAC II; two of them were the primary registers, L and A. We had Load and Unload Register A where we [2 words inaud.] the lists, and then we had a thing where we had any type of instruction which I could call and put together and put it in the output, etc., and then did a lot of things, there were some shift and align

things, when I had to do a little shifting and alignment instructions that had to be generated.

I guess that was kind of my first more or less scientific paper I created - I still have it, actually. I didn't bring it along with me. I tried to deal with a set of generalized algorithms for dealing with shifting and alignment of decimal arithmetic. I couldn't find any references to anything, about how to deal with that, and it's strange, so it's probably a unique paper, which has never been published, about how to deal with the case of generalized decimal arithmetic. That was important in getting this thing to work properly.

There's some editing service, implied calls that we have generated, containing internal references, constant processors, this put things out for eventual processing for constants, etc. And eventually things came down into this [1 word.] cycle output, where I actually put out the code, and collected it into buffers and put it out. So it turned out to be, this was my saving grace, was having this, because this - I think I got this to put into about three or four hundred words of code. So I was able to get a fair amount of the rest of the code generators in there, and deal with that in a proper way. So it turns out. There was a very convenient subroutine call mechanism called the R and U instructions, that allowed you to deal with subroutines, but of course there was no formal method for parameter passing or things of that variety. I should also say an interesting aspect: this machine had of course no index registers, so if you were really dealing with vectors, generating in terms of vectors, you had to generate self-modifying code. And we simulated this by memory locations, which got the name B-Boxes. I guess Howard remembers B-Boxes; I think that was a carry-on from some of the earlier compiler discussions at Univac also.

Q: This was written in FLOW-MATIC?

A: Yes.

Q: And after FLOW-MATIC compiled it, it was 300-400 words?

A: Yes. I'm going to come back to that in a minute. (Laughs.)

So, anyhow, that was the idea of DUZ. Now, how we actually got to that topic was the implementation of FLOW-MATIC. Now, it seems surprising to do this, but Grace was very insistent that we write this compiler in FLOW-MATIC. It was not the easiest thing to do, as Don just made a comment about. But I guess the only time I really got into a fight with Grace was, I introduced the facility into the FLOW-MATIC compiler when I changed the thing, and I introduced two statements called SUPPLEX1 and SUPPLEX2, and these were allowed to put me in [1 word] in one machine instruction or two machine instructions in certain critical places. So the structure in these couple-300 lines was FLOW-MATIC, but occasionally I had to put a few things in in order to make the thing fit. So she found out about this SUPPLEX1 and SUPPLEX2, and I was called in to her office, to explain why I had put in this facility. There was another facility put in X-1 code, which was assembly language; that was a large amount of code. This was just to put in one or two instructions in critical places. The SUPPLEX1 and SUPPLEX2 actually survived in FLOW-MATIC, and

people used it after that - my new generators in the FLOW-MATIC compiler. It was through this mechanism that I was able to get these subroutines in reasonable size, and be able to use them in effective form.

After that, Grace admitted that "all right, yeah, that was okay" because when we actually came to the demonstrations of this, and was running, she said, "How the hell did you get all that stuff in - all that code generation in that thing?" She really didn't understand how I managed to do it. I explained how it worked, etc., so.... Then we came - from that time on we had a very interesting professional relationship, she - had a mutual respect for each other for a long period of time. It was really a challenging thing. I remember the demonstration days, and this was an important part of course to have working.

That's a little bit about the compiler. I'd just like to give a little epilogue about a certain few other aspects of this that might be interesting to think about. The compiler, as was discussed earlier, was developed concurrently with the COBOL language. It was not easy, because we did have a little bit of a moving target. This document here came out in April of 1960, which they had worked very hard, on the CODASYL Committee, to get out. While, during the late '50s and early '60s when we were developing the compiler, our representative, Dan Goldstein, came back from these meetings, and we would have to sit down and say, "Oh, well, let's all scratch our heads and say what are we going to have to change in order to get this stuff." The basic structure of the compiler held, so it was mostly details that got changed. So, we were fortunate, but it was not a really easy environment to work in, in that regard.

As we said, RCA also developed a COBOL compiler for the 501, in December, 1960, the first time - and this is - I want to make a qualification of this - the first time a COBOL program was both translated and executed on computers of two different manufacturers. Since we sent this out, Doug McIlroy has reminded me that Jean Sammet wrote in her book that it was the Philco 2000 FORTRAN compiler that was probably working in about April of 1960. And so FORTRAN programs were being executed on two different manufacturers. So this should be qualified maybe to say this was the first time a COBOL program, or a data processing language program, was executed on two different compilers. In any case it's certainly a historic event. I don't remember the exact date - do you know? [Howard Bromberg: December sixth and seventh.] Sixth and seventh - yes, because my daughter was born on the ninth. And Grace was so happy this thing worked, and when my daughter was born, she came around with a lot of bracelets and some suits that she had knitted herself, and gave me as presents to give to my daughter Catherine, who's now a research worker with Brookhaven National Laboratories, if I may say also, Long Island, in molecular biology.

About that time, 1961, as we did this work - did you have a comment, Don? [Reply: It was the same program that was running on both?] That's right. [Question: You guys were cooperating, you weren't competing?] In a sense cooperating, because in the morning it was run - it was two different days, sixth and seventh, on the sixth it was run in Philadelphia on the UNIVAC II and on the seventh, in Cherry Hill. Was it the other way, Cherry Hill first? Okay, sixth and seventh. All right, thank you, I stand corrected. (Laughs.)

That's not too important, I think. Anyhow, it was really, as I said, there were some very minor changes I think made in the program. I do not have a copy of the program. Turns out, because of the announcement of this that Warren Simmons, I think, has indicated that there is some copy of this. If you could get a copy of the source text, I don't remember the source program, I would love to see it again. Of course at that time I studied it, but I don't have it any more. It would be fun to put it into this Annals of History Computing [sic] article, that source program, also.

Q: Do you have any recollection of how long it took to compile, a program like that?

A: Yes - I put into my paper a five to eight minutes for a null program, basically, that a very little apart from there, but I think it was a little bit longer, I think probably eight to 10 minutes, because most of it was tape movement, getting the phases initialized, and things like that required a fair amount of time. To do a null program was probably on the order of eight to 10 minutes. That's my recollection. To do anything reasonable was half an hour or an hour. If you had any reasonable sized program, it took quite a while. [Comment: I think Howard's compiler was a little bit slower.] [Laughter.] Oh really? [Inaudible comments, laughter.]

Q: Since you were thinking of this as sort of a data processing system, and you had all the tape movement, were you able to do any batch processing and take multiple programs and move them through together, through the batch, or just a single program that had to go all the way through the pipeline?

A: No. Single source program, we dealt with at a time.

It's interesting that in this time frame - that was December '60; '61, Remington Rand-Univac was bought out by Sperry. This was a very bad thing for the Automatic Programming Department at Univac, and I think Howard will tell you a little bit more about that. It was a time a lot of us did a lot of soul-searching, saying "Well, is this the place we really want to be?" because Grace fell into disfavor with her new management under Sperry, and a number of us had left. Tom Jones went over to RCA to work with Howie, Howie and Charlie Katz at General Electric in Phoenix tried to recruit me, a number of other places. I happened to wind up at IBM, and worked for several years on compiler research, and then eventually on the PL/I programming language. And that is the first time I really got to know Professor Knuth, at that point in time, and did a lot of the work - I guess you might say that the pointer variable concept in high-level programming languages is my invention. That was put into PL/I, with a lot of comments from both Don and from Doug McIlroy, during that period of time.

When I started the compiler I didn't really lose sight of some of this work in doing on COBOL, so I did spend some of my time trying to study, because I knew some of the problems that we ran with that COBOL compiler was processing data structures and data descriptions. So I tried to sit down and figure out some data structures and algorithms that would be interesting for organizing processing of such things, and that wound up being my first professional publication: [it] was, The Use of Chain-List Matrices for the Analysis of

COBOL Data Structures, which Don was kind enough to take up as being one of the first papers in multi-linked data structures. In Volume I of Knuth you'll find a reference to this paper. I think that was - there were a few people doing doing similar things around the same point in time, but I think it was the first publication of algorithms to deal with those - that type of multi-linked data structures. Which also gave me a lot of insight into thinking about how to deal with pointers eventually also, when I was given the assignment to put that into the PL/I programming language.

That's, I think, a little bit - most of what I have to say. I think it gives you a fair view of things, and I would like to turn over the work to Howie now to continue on, unless there are some further questions, but we can continue and if anything comes to mind later on, we'll come back to it.

HOWARD BROMBERG:

I'm not going to talk about compilers. What I thought I'd talk about is the environment of the times, within which all of this kind of work was being done. It was a different world, in the late '50s in the computer environment. Most of the things that we were doing, we were doing by "seat of the pants." There weren't a lot of tools available, there weren't a lot of people that we could talk to, certainly no pioneers - my definition, Len, of a pioneer of course is different from yours: just an old guy who's still working. [Laughter]

I thought I would share with you some remembrances that I have about Grace Hopper, and put those little anecdotes in the framework of what was going on in the development of COBOL language and compilers and standards at that time.

I went to Philadelphia, where Bud worked, with the Navy Department; I was working for them and we were doing calculations for the reactor that was going into a nuclear submarine. Our sub-contractors were Westinghouse Atomic Power Lab and GE's Knoll Atomic Power Lab at the time. We had, at the Navy - at that time I was working at the David Taylor Model Basin, which is just outside of Washington, D.C. - we had a UNIVAC I, of all things. But that UNIVAC I had to serve a lot of different requirements, other than this nuclear submarine activity. So when we had this responsibility, it was necessary to find computer time. Now, that sounds funny, because today it's easy to find computer time. In those days, there was no excess computer time. It just didn't exist. There was only a handful of mainframes around. We were a UNIVAC shop; we couldn't go and find a 702 or a card-punching calculator. So we had to find a UNIVAC II, or a UNIVAC I. And where the devil do you find it?

The person in the Navy responsible for our activity was a guy by the name of Hyman Rickover, who was, like Grace Hopper, a "piece of work." [Laughter] Rickover said to the Univac people, "I am sending a team to 19th and Allegheny Exide Battery factory, and I want you to give them computer time because it's important." You know, big stuff. The Univac manufacturing facility at that time was on the second floor of this ugly, ugly warehouse, and there were four assembly stations. As one machine was assembled, it went

through two months of testing. When that was finished, and got shipped, they started again; and meantime another machine from the second station was going through testing, and so forth. We were able to take our tapes and programs, and we would be given one of these test machines, on the test floor, so instead of being tested for two months, it was only tested for one month, and turned over to us. We were running this installation 24 hours a day, six days a week. And the seventh day, the maintenance people came in. Although the maintenance people were there, all the time [laughter].

As Bud said, these things were water-cooled machines. They were in an un-air conditioned facility, windows wide open, all kinds of things running in. The opportunity to run through a program without an error not created by the programmers, or even the hardware guys, was a million to one. Every 10 minutes or so, every six minutes or so: read-write error. Bingo! Everything stops. What's the problem? We're over there with, literally, brushes, air things, trying to clear out that head, that was the only problem. And, of course, the condensation from, you remember that big power supply? The condensation would run off and fill up this little trough around it, and when it would overflow, there was some kind of water sensor, and that would send off an alarm that if you had a heart pacemaker or so it would turn that thing off - it was just frightening. And this is the conditions under which we were working.

Just a little aside: in those days there were electrical engineers, technicians, and other people - us. The EE's did not talk to us. Did not speak English. They were not from this planet. [Laughter] The technicians understood, and they had a job to do, they saw that we had a job to do, and it was okay. So they were very, very helpful. One day, because of all these read-write errors from all the dirt and dust that was coming in the windows, I decided to buy some suction cups. We had six servos, and I bought 12 suction cups, and I stuck them on with good old spittle, onto the back of the servo, underneath where the tape would pass on its way to and from the read-write head with a little bit of pressure on it, very little. And then I put 10-15 layers of a Chem-wipe, which was like a Kleenex but no lint on it. And I affixed it on each end with rubber bands. And it virtually eliminated the read-write errors - virtually - until the engineers came by. [Laughter] That was the end of that. Those things had to go out - they weren't to spec, they didn't bla bla bla. So they took them off.

We had a technician who was really very good and friendly, and he was the guy that told us we could put them back until he saw one of the engineers come in, and then we had to rip them off. [Laughter] So that's how we worked it.

While all this was going on, we had the phone company - you know you didn't have cell phones in those days - the phone company came in and they hung our phone from the ceiling over the console so at least we could communicate with the outside world. I had four of them, one over each of the stations, which the engineers didn't like that either. Every month we had to move this installation. And the tapes in those days were metal tapes; they weighed 12 pounds or so. We had hundreds of these tapes that we had to move. The reason they were metal tapes: these engineers had the idea that every so often they would put these metal tapes in a safe and burn the safe and then be able to read the tape. I

never saw a tape in a safe before. But they had this idea that that's where tapes were kept - it was backup taken to an nth degree. It's crazy.

It was a major problem every month or so to move. One day, as I was sitting at this little console, a little old lady comes over to me. She says, "What are you doing?" I explained what I was doing, and she said, "Do you think I could have some computer time?" I said, "Are you mad?" [Laughter] I said, "This is national defense! This is important kind of stuff. What do you do?" She told me what she did, and I said, "Well, that's pretty interesting." And from then on, I gave Grace all the computer time that we could find for her. She was just absolutely marvelous. And said to me, as we became friendly and now on a first-name basis, "If you ever get tired of the Navy, you can come to work for me." Shortly thereafter, one of these machines became Navy property and was shipped down to the Navy, and I went back with the machine. I tried to explain to them that machines have some kinds of personalities. Like Bud was saying, these old UNIVACS didn't like sorting. They hated it - you know, chk! chk! chk! chk! chk! [Laughter] I said, if we could organize the processing work that we have, in such a way as not put a stress upon these dear little creatures, that that would be a much more effective and efficient way to run. They thought that that was really stupid, and it might have been.

So I then went to work for Grace. In those days, Grace was working on the mathematical compiler, the Univac answer to FORTRAN. There was A0, A1, A2 and A3. At the same time, we started working on something called B0, which was the business compiler, the business language, that we all thought was going to be the major emphasis for these mainframe computers. You must realize that in those days we were operating in a very competitive environment. The competition was a company called IBM. All the rest of us tried to do things better than IBM, before IBM, instead of IBM, whatever. This was just the way it was.

When the idea of a common language came up, of course all of the IBM competitors thought that was a marvelous idea. Obviously it was a marvelous idea. Here's a way that we could take Big Blue and say, "We're just as big as you are. Our color's different, but we're just as good." So the organizational meeting for having this common business language was held down in Washington, and there were about three dozen or more people that came to that organizational meeting. The interesting part at the time was that I think that there was one university attending. And that also shows what the times were all about. We were not computer scientists. Computer scientists worked for universities. We were computer practitioners, for want of a better name. We didn't know anything about elegance. We knew about bottom line: the damn thing has to run; we gotta get it out in such-and-such a time; the budget is this, let's not over-run it by more than 200%. [Laughter] So there was always this unofficial race to get things done, and as I say, not done as a computer scientist would do it today.

There were about eight computer manufacturers who participated in the definition of this language. We knew it had to accommodate what we had a feeling was business data processing. Again, we were not business data processors. I don't think anybody had written a payroll program.

Payroll to me was business data processing: multiply hours times rate, and cut a check. Payroll was in those days very complex. And it was probably the most prominent of all the business data processing applications. There were 70-some different payrolls, we counted at one time, payroll applications. Today there must be two thousand of them.

So we had this idea that business data processing is different from numerical calculation because it didn't use numbers. So what our idea was, we would talk to our customers, our users, most of whom were in the so-called business data processing area, and find out from them what are the operations. What do they do? What kinds of things do they run? There were very few surprises.

They did a lot of comparisons, a lot of reading and writing, very little calculations. Our plan then was to create first a language that would represent the needs of business data processing users. Second, we wanted a language that was understandable. We said that since our users in the data processing area communicate in a language called English, that would be the most convenient language for us to use to write our specification language in. So COBOL became an English-language business data processing language. Finally, we said, wouldn't it also be nice if everybody - all the manufacturers - would be able to translate this language spec into their machine code, so that if we need computer time, we don't need to stick with our manufacturer and run around like crazy and go to a battery factory in Philadelphia.

Those were the ideas that we had, the goals that we had in mind. What we thought was that there would be an evolution, that this language was only going to be a language that would exist and survive for a very short time. The organization under whose auspices we were working, a government organization called Conference on Data Systems Languages, actually created three committees, or task groups. We were the short-range group, because their mentality said, "What can these guys do in six months' time, or something like that? It's only going to be an idea, a target, a platform, or what have you. And we'll have an intermediate-range committee, and a long-range committee, and probably the real, lasting work will be done by the long-range committee," after they see what we have turned out.

So our goal was to produce this specification in a short period of time, because if we had gone to the powers that be and said "Well, you know, we think in two years we could really refine this, and it would look neat and good, and we could develop something that would last," we would not have been funded, if you will. That would have been the end of it, and somebody else would have taken over. So we knew what the parameters were, and we set to work doing our thing.

While all this is going on, we had two advisors. These advisors to the committee were people who knew more about what should be done than we knew. One of the advisors was a fellow by the name of Bob Bemer from IBM, and the other advisor was Grace Hopper from Remington Rand-Univac. Now, mind you that we still felt that we were in this kind of race, this competitive, beat-'em-at-all-costs kind of race, and consequently what would happen is that every individual manufacturer would bring to the committee those procedures, those verbs, those constants, those formats, that would show his or her

particular machine in the best light. We would fight like crazy over these problems. Grace, for example, through her rep, Danny Goldstein, wanted to make certain that we were recognizing a binary machine. "Well, I don't want to recognize binary machines." So, that was one of the little hang-ups that we had. Bemer wanted something else, and on and on and on.

While Bud and his people were working as he's described, and trying to create a system out of these incomplete language specifications, the committee kept on revising them, changing them, refining them. We would meet every month or six weeks or so, and sometimes come back to our implementation groups with major changes or additions that hadn't been thought of.

I approached the 501 compiler development in a slightly different way than the way Bud explained his, in that what I did is, I hired a lot of smart guys like Bud to do that work [laughter]. So I was busy doing all the language specification, and these guys were back there sweating out - trying to figure out, how the hell do we put this into our compiler design. We essentially had the same kind of structure: we had phases that handled each of the divisions, created a meta-language from which we could go into an assembler or directly to machine code. We had at that time on the drawing boards a specification for a new machine, a 601, which was ten times the power, and prettier, and cheaper, and all that kind of stuff, and I wanted to make sure that what we were doing for this machine - that the reason I created the meta-language was because we could then port it over to the 601. Of course, they were never able to sell the 601, so that was an extra amount of effort for naught.

Finally, the specifications were finalized at this time, and published as Bud showed you. Everybody raced to produce his interpretation of that COBOL language specification. It in a sense culminated, from a political standpoint, by our doing a compatibility demonstration to this CODASYL Committee - to this executive committee who had no idea what business data processing was, much less compiler development. But none the less they understood that if you take a program essentially written for this machine and run it on that machine, and take a program written for that machine and run it on this machine - that's got to be good. So, indeed, we demonstrated that. My recollection of compiler time is a little different from yours; my recollection was that it took slightly less than an eternity. I remember that during the demonstration, what I did was, I had each of the program managers who were responsible for each of the phases standing at the computer and saying, "When this is spinning, what it is doing is such-and-such." And, I said, just speak slowly and long [laughter] because I don't know when this thing will ever finish. As I recall, it was about an hour or hour and a half for that program.

Working for RCA was a little different than working for Remington Rand. RCA was in what they liked to call the communications and electronics and entertainment business - color TV was the big thing; they liked flash and dash and what have you. And our computer center, at the time, was something like this, nice seats out in front, and a glass-enclosed computer center with curtains. And the curtains operated electrically, so I could stand here at this lectern and push a button and the curtains would part. Very, very dramatic.

So now we got this compiler through, and we got the listing, and I gave out to everybody a copy of everything - they hadn't the foggiest idea what it was they were carrying home, except that it was heavy and it had their name on it. [Laughter] And then I had them all come into the computer room, and press a button, because they had survived this compatibility demonstration, and we printed out a little diploma for them, suitable for framing, with their name on it, and the date, and the fact that we did that. [Inaudible question/comment from audience] (Laughs) As a matter of fact.

We then, the next day, went over to Grace's shop, and essentially she did the same dog and pony show for the same group: ran through the compiler, Grace explained everything, what was happening, through all the phases, and finally got the printout, turned it over to them, and everybody was happy and off they went.

After that demonstration the playing field started to level a little bit, now that everybody was in the business data processing business with the same tool. Everybody had a COBOL compiler; one had a little bit more of this, a little less of that, but basically the core specification was identical throughout all of the interpretations of that language. And you could indeed ship programs around.

The intermediate- and longrange committees were sort of disbanded, and a very nice approach to the maintenance of this language was adopted. What we decided is that we'd like to have an idea of upward compatibility, so we had language elements that the committee had discussed and described - defined. Those elements that were going to be put into the next generation of the language were put in something like an on-deck circle, so that everybody in the user and development community had an idea of what was coming next, so that they can operate in a period of relative calm. Which was novel for this kind of stuff.

Then we had to turn our attention to the standardization effort. At that time I left the COBOL committee business and started working with the standardization activity under the ANSI - the American National Standards Institute. The Secretariat for Programming Languages was held by the United States in the form of the industrial organization called CBEMA - Computer Business Equipment Manufacturers' Association. The fellow, to my surprise, a kinder word, who was the head guy at the CODASYL effort, Charlie Phillips, who was coordinating the work of the development of COBOL, moved over to become the head of CBEMA, and as a result the titular head of the standards activity.

Which brings me to the tombstone story. I was working, as I mentioned, for the RCA Corporation; RCA was a different kind of company. They were into a lot of different pursuits, and at that time when you wanted anything done at RCA it had to have some kind of marketing flavor to it. Or color TV flavor, which was really hard to produce at that time.

One day, when I was particularly frustrated with a conversation that I had with Charlie Phillips, about how slow these decisions were coming, and how difficult it was to maintain an implementation group on any kind of schedule and do any kind of meaningful work

without these decisions, and hanging up the phone on him as I had done 50 times in the past, I left work early and was driving on the freeway and I saw a monument company, making tombstones. I said, "That's a good idea," easy off, easy on, and off I went and I said to the guy, "Can you make me a tombstone?" He said, [snicker] [laughter]. So he said, "What name do you want on it?" And I said, "COBOL." [Laughter] And he said, "What kind of name is that?" I said, "It's a Polish name." [Loud laughter] So I picked out a little, it was a white marble rectangular type thing on a pedestal, with a little lamb on the top, which I thought was a nice touch. [Laughter] And I paid the guy, and he said "I'll call you when it's done." And I went home.

A couple of weeks later he calls me and he says, "Tombstone's ready." I was still angry, and I went back and picked it up and took it home. On the sidewalk I built a little crate for it, and I shipped it UPS to Charlie Phillips at the Pentagon. Very pleased with myself. And I don't hear anything from him: no calls, no nothin'. Did he get it? Something happened to the truck? Or is he playing a mind game with me? [Laughter] Finally I called him. I couldn't stand it any more. "Charlie, did you get anything from me?" He said, "Yeah, as a matter of fact I did." I said, "Well, what do you think?" He says, "I don't know what you meant by that." [Laughter]

Grace, in her travels, when she would give talks, lectures, what have you, would tell this tombstone story. And people would call me, all over the country, people called me and said "What is this tombstone?" I said, "I don't know what you're talking about." [Laughter] I denied that story - deny, deny, deny. A picture of it was on an ACM Communications cover; it was all over the place. "I don't know anything. My hands are clean!"

One day I get a call from the secretary of the Marketing Vice President, corporate Marketing Vice President for RCA, for whom I worked. And that tells you something else, where the Automatic Programming Department was. I was asked to come to the executive offices. I did, and waited an appropriate amount of time, and the guy finally ushers me in and says to me, "Someone in Rockefeller Center, New York, has heard that you sent a tombstone to someone at the Department of Defense, and they think that this may inhibit our ability to get government contracts." [Laughter] He said, "Did you do that?" I said I did. He said, "Would you like to tell me why?" I said, "No." [Loud laughter] So he said, "Thank you."

That was it. So I went back to my office, and I got things nice and neat [laughter]. I didn't put them in boxes, exactly, but sort of neat, ready, pointing toward the door. And I never heard a word from that time on. That was great, and I think, to their credit, that was the end of it. But Grace kept on churning it up with this story.

This is only the second time I've told the story; the first time I told this story was at the Computer Museum in Boston, in '86 or something like that, when Grace was there.

Meantime Grace, as Bud mentioned, had a little thing with the new management at Sperry, and she decided to leave and go back to her original love, which was the Navy Department. That was great. At that time - [Q: Didn't she have a heart attack about then?] It was not, yeah, she had a problem, like an angina, but I don't think it was a real heart attack.

To back up for just a second: when I worked for Grace, it was unbelievable. I lived just a couple of blocks from where Grace did, there in these days, so I used to pick her up in the morning, and bring her home at night. And we would have a problem, some technical issue, during the day and we'd talk about it on the way home. I said, "That is really an interesting problem, everybody is quite interested in it. I'm sure we'll get to that solution." Eight o'clock the next morning, when I picked her up, she has it solved! And I said, "Grace, that's not fair." [Laughter] I said, "You know, we're all set now, got our sleeves all rolled up, and we're going to attack that problem, and you've got it all solved." She said, "That's your problem, not mine." [Laughter]

Well, now Grace was back at the Navy Department. And I'm out here in San Francisco, trying to eke a meager living out of devloping some tools and software packages, doing this and that, and Grace now is recruiting, finding these sailors that she gets, and she says, with no budget, and no particular authority, she started teaching these sailors how to be programmers. And they're producing software packages for the commercial marketplace, identical to what I'm doing. Only, she's giving them away. Giving them away! "Grace! You're taking the bread from my childrens' mouths!" [Laughter] "What do you mean?" I said, "How can you do this? You are in the Navy. The Navy's job is to sail boats. It is not to make software and to give it away." She said, "Well, it's not really giving it away - there's no documentation, and, you know...." I said, "I go around to these big corporations, and try and peddle my little wares, and they say to me, 'But we have one of those. We got it from the Navy, for nothing.'" So I said, "You must stop." She says, "Well, go talk to my boss about this." (The conversation was not as polite as I recalled to you.)

I did indeed go to Washington and I had a meeting with her boss, who was a Navy captain. I said to him, "Would you like to become an admiral?" I said, "You're not going to." I said, "As I leave here, I'm going to visit my congressman, because of what your Department is doing, namely, you are building software and distributing it into the commercial market. There are certain things the Navy does not do. (1) You don't build battleships, you sub-contract that out. (2) You don't build uniforms, you sub-contract that out. And (3) You don't build software."

Shortly thereafter, that phase of Grace's activity calmed down a little bit. I don't know whether it was because of her captain boss, or because she just got tired of that, but she stopped doing that, and moved into what became a major contribution to this whole effort. And that was in the validating of COBOL compilers. The work that she started at the Navy gave everyone - computer manufacturers and users - an opportunity to be guaranteed that what they were buying, what they were using, what they were betting their companies on, were indeed valid, standardized COBOL compilers. And I think that was a super accomplishment.

Having not spoken to Grace for five years because I was in that snit, she was invited by the San Francisco ACM chapter to come out to give a talk, and they called me and said, would I introduce her, and I said, “Sure.” And I called her up and I said, "Do you forgive me? I still

love you. And I'll pick you up at the airport." And I picked her up, and I gave her a big kiss, and since then everything was fine and dandy. The hatchet was then buried, and we became good buddies again.

As I said, Grace was unique in so many ways. Grace - if there was such a thing as a "glass ceiling" in those days, she would have broken it, right away. She would never have accepted it, allowed it, considered it, spoken it. That just was not her way. Grace had a way of looking at you and knowing what is going to motivate you, what you want to hear, and how she's going to present it. She used to give talks to upper management, executives from banks, insurance companies, all these different kinds of large corporations that had huge inventories of COBOL programs, but hadn't the foggiest idea where their computer center was. She would talk to them and tell them what microseconds mean; see, she had this hunk of wire, did you ever see that? And she used to show it to these guys, you know, the President of US Steel: "This is a nanosecond, mister." It sounds silly to us, but it really made sense to these people, and they got some kind of understanding from her of what the devil was going on, and why they were spending all these millions of dollars every year, to support the information technology activities.

The last time I saw Grace was at this Boston Computer Museum shindig. The day before, the Navy decommissioned the Constitution, which was the oldest active duty ship in the Navy. They had her as part of the ceremonies, as she was the oldest active duty sailor in the Navy, or so. Bud and I were just talking about the fact that the Navy was really a major part of Grace's life, and she loved it - she loved all of the people that had anything to do with the Navy. If there were two people or ten people that came for a job with her, if one guy had Navy experience, bingo! He's got the job.

A propos of that, just one other little remark. One day we got a fellow in who was sort of an applicant, he wanted to come to work. She says to me, "What do you think?" I said, "Well, this guy is a personal family friend of Pres Eckert," who was the vice president at the time. I said, "Everything that we do here is going to go right back to Eckert." And she says to me, "Now you understand." That was Grace, and we all miss her. Thank you.

LEN SHUSTEK:

Thank you very much. Thank you both. Great talks!

A couple of things. One is, we would have had the tombstone here, in person, except the tombstone is still in Boston, in the Computer Museum behind glass as part of the exhibit. So we couldn't bring it out here. The other thing is that I have copies here, probably enough for all of you, of the paper that Bud is going to submit to the Annals of the History of Computers, on the first UNIVAC compiler, and you're all welcome to come up and get some.

GWEN BELL:

[Several words inaudible] thing is, I think it's September ninth when the Hopper is going to be commissioned in San Francisco, and we are trying to organize the event, from the Computer History Center. We are in contact with the people, and I hope everybody can hear about that so we can really visit Grace and ...

BUD LAWSON:
I'd just add to that, that I have sent materials about Grace to the Navy Department, and it's in the hull of that ship. They put a time capsule in the hull of that ship. I was actually invited to the launching of that hull, at the Bath Iron Works, in Maine. I've sent material for that contract.

LEN SHUSTEK:
Why don't we take a few questions? I'd like to start off with one question I have, about the test program. How big a test program was that? Was it a really good exerciser of the compilers? Did you have it in advance, and how far in advance, and did you tweak your compilers to make sure you could compile it, the way people do these days with benchmark programs? Either one of you can answer that.

BUD LAWSON:

There was only one program to compile (laughs). No, obviously the people did a lot of local testing on their own parts, their own phases, and generated their own inputs and outputs, which was a problem. But you had to do that - that was the only way to do it. I remember in this DUZ phase, it required a lot of my time just generating test cases, to try to test out that code generator, which was not very easy. The COBOL program that was actually compiled didn't exercise all the facilities, as I recall, so that we did spend a lot of time analyzing that particular program. I think that became a common thing, also people doing compilers - I had some good friends at the Burroughs Corporation a few years later, were coming with COBOL compilers also. There was a big COBOL program called Big Bird, that was a very famous program that everybody doing COBOL compilers had to manage - Big Bird. This was really a real big challenge, that these people talked about.

After my experience with COBOL at Univac, and beginning at IBM, I didn't really deal with COBOL so much, and I haven't since then, but I hear a lot of stories about it. But testing was just done as best you could at that point in time; there were not a large base of COBOL programs to try. This was - I don't know who actually wrote the - did you as a committee write that program, Howie? Who wrote the test program? I mean the demonstration program, in COBOL. You don't remember who wrote it?

HOWARD BROMBERG: Which program are you talking about?

BUD LAWSON: The one we compiled on the sixth and seventh. Who actually put that program together? The sixth and seventh of December.

HOWARD BROMBERG: Those were actual, valid, user customer programs.

BUD LAWSON: Where did you get them from?

HOWARD BROMBERG: They came from US Steel - I think it was Warren Simmons. And one came from one of our customers ...

BUD LAWSON: Okay, so there were two programs. I think we had them in advance, so we were able to - so we were working, trying to get those to go through our compilers in both cases, obviously.

Q: Were you able to use the Unityper yourself? The reason I'm asking is that I worked for Univac in the summer of '62, and at that time I couldn't - I would have to fill out forms that would have my program on it, and submit it to the people there, who would enter it, then they would give me the thing back and then I could proofread that and give my corrections for the next day, and so it was impossible to write software under these circumstances and I - this was with cards on the UNIVAC III, Univac solid-state, but so we had to get our keypunch in our department so that we could prepare the input ourselves. But were you allowed to hands-on, as you were doing all the software work?

BUD LAWSON: Yeah, we did. We had at least two or three Unitypers in our department, where we could prepare it, just exactly as I showed that picture there. We prepared on the Unityper the code and although, yeah, it was depending on your typing skills, how well you did that; some people are good at typing - I happened to be very lucky, at age 16 or 17 I had taken a course in typing. So I was very good at it. But some of my colleagues were not as good as that, and made a lot of mistakes, so some of the people did actually do as you said, put them in forms, gave them to people, some girls typed them up, and got them back. Corrections you could make - typically when you go down to the machine, you're on there and suddenly you find there were some problems, and you could make some on-line corrections very early, on the machine also by keying in.

I had one story, my maintenance of the UNIVAC II. The carriage-return command on this on-line typewriter that you saw at the supervisor control panel was always not working. We had around these big metal tapes that were 12 pounds each, an elastic band which I took off and I tied down to one end of the typewriter on the carriage. So it would build up pressure as it was printing across here, and when the carriage-return command came, whish! (Laughs) It would come back again. It slammed [1 word] the other way . So that was my one really major repair of the equipment.

The other story, reminded me here when Howard was talking about these 12-pound tapes was, when we were dealing with this Tape Input Output Generation Assembly, we exactly had the same problem that Howard talked about. We had a hard time getting time on the UNIVAC too. And there weren't a lot of them around. So, we went out to our customers. Almost all the TIOGA work was done in Pittsburgh, and I still remember driving out across the Pennsylvania Turnpike. We were four or five people that went: Dick Minor and I plus two others, one who was helpful with keypunching and typing, Peg Hart - not Peg Hart, her name was Peg but I don't remember her last name. We went out across the Pennsylvania Turnpike with about 25 tapes, in the trunk of this car, which went across the Turnpike this

way! (Laughs) It was really - they were heavy stuff. You used to get a hernia lifting them up and putting them on the Uniservo. Don?

Q: When Charlie Katz came to Case, where I was, [inaudible] he told us that Grace Hopper took the tape with the MATH-MATIC compiler on it at night, every night, and slept on it, or the next morning [several words inaudible] [laughter]. The other question I had was, did you every hear of UNISAP, in Philadelphia? This was an assembly program for UNIVAC that was developed in Cleveland.

BUD LAWSON: Was this your product? [Reply: Mel Conway.] Mel Conway did it, okay.

Q: I just wondered if you ever got it.

BUD LAWSON: What time frame, do you know?

A: This was '59.

BUD LAWSON: UNISAP. Was for UNIVAC II?

A: Symbolic Assembly Program.

BUD LAWSON: No, I didn't - I don't remember, we - when we did symbolic, even if we were going to write something in machine language, we used X-1, which was the normal UNIVAC assembler. How high-level was Unisap?

A: It was - looked like a [inaudible] - looked like [several words inaudible] - the people in Philadelphia said that the machine language was so mnemonic anyway ...

BUD LAWSON: Yeah, exactly the point I made earlier, it was fairly intuitive to learn to program the machine, so, right. Okay. Interesting point though, I didn't hear that. Yes?

Q: With the identical program running on two architectures, this the first benchmark, to call it that, did it spur any competition or economic direction by either company?

BUD LAWSON: I'll leave that one over to Howie. (Laughs) I don't know if there was any - you know, these guys to me were like mystical people; the name Howard Bromberg, and the name Warren Simmons, and from my point of view I would hear through Dan Goldstein and about Jean Sammet and all those people, so but as far as the business aspect, or whether it spurred further competition between Univac and RCA, I don't know.

HOWARD BROMBERG: There was only one competition. IBM. There was IBM against Univac. IBM against RCA. IBM against Honeywell. IBM against GE. IBM against NCR. IBM against Sylvania. That was it. We were just struggling to keep our noses a little bit above the water, so that we could say, "We have now an arsenal as good as what IBM has. Even though they have something called Commercial Translator, or PL/I, which was a grand idea. We are with the industry standard." That was the message.

Q: So you said you developed COBOL in a six-month timeframe, a short-term timeframe, do you think that that hurt your specification of the language, or do you think that you ended up with something that you were happy with?

HOWARD BROMBERG: It hurt it in one sense, that the specification was not as elegant as it could have been. It helped, in the other sense that it was something out early, and you could "kick the tires." And, it worked. So, again, it was the conflict of, you want to make something nice, right, or do you want to make something elegant and beautiful and something that your grandchildren will be proud of. [Laughter] We opted for the former. [Laughter]. With all the trouble we got into!

Q: And nobody has even mentioned Year-2000 yet! [Laughter]

A: Good point. Bill?

Q: And it probably also helped it from getting overloaded, because PL/I was designed to be elegant, and had features that were just murder, not only for the compiler writers, but for the users, because of unpredictable behavior for the sake of elegance.

BUD LAWSON: Well, I wouldn't have called PL/I elegant. [Laughter] I will tell you a story about - this is a COBOL lecture, but I'll tell you a PL/I story. I was involved in the original PL/I specification, their maybe goal was to do that, but it certainly was not elegant. I added these pointer variable concepts to the language.

One time, when we had a Language Control Board, all of a sudden there came down a very, very long letter to New York, we were in Manhattan at the time, from somebody sitting over on my left here, that looked at the specifications, who thought it was very interesting, this new language, PL/I. And the whole letter was stating, "Well, on page so-and-so you state so-and-so, and on page so-and-so you state so-and-so. These seem to be in conflict." Now, if you know Don Knuth, you know that that's the type of comments that - and he was right! In 95 percent of the cases. And so we hired him as a consultant [laughter] to help us get the specification better, anyhow. But I still don't think I would call it - there were some concepts which we tried to have some degree of cleanness with, but we turned out - the whole storage class mechanism, after putting in the pointer variable concept, was no longer really - it should have been re-thought, let's put it that way, and there wasn't time to re-think it. If it had been re-thought, we could have probably done what you said.

Q: I was thinking of some of the type conversions.

BUD LAWSON: Yeah, exactly, and even those [inaudible]. I know you were very much into PL/I in those fairly early days.

Q: But I ignored the type conversions.

BUD LAWSON: Okay [laughter].

END OF TAPE.

Transcribed by John Amos, San Antonio, Texas, July-August 1997,
with invaluable help from Len Shustek