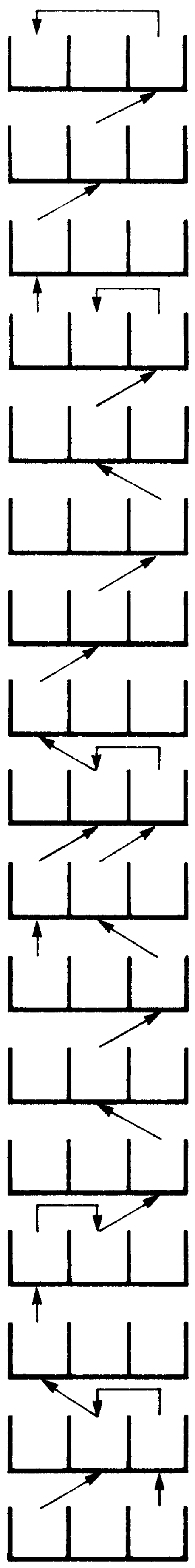

FORTH DIMENSIONS

FORTH INTEREST GROUP
P.O. Box 1105
San Carlos, CA 94070

Volume 1
Number 6
Price $2.00

INSIDE

FORTH DIMENSIONS

Published by Forth Interest Group

Volume 1 No. 6 March/April 1980

Publisher Roy C. Martens

Subscription to FORTH DIMENSIONS is free with membership in the Forth Interest Group at $12.00 per year ($15.00 overseas). For membership, change of address and/or to submit material, the address is:

Forth Interest Group
P.O. Box 1105
San Carlos, CA 94070

HISTORICAL PERSPECTIVE

FORTH was created by Mr. Charles H. Moore in 1969 at the National Radio Astronomy Observatory, Charlottesville, VA. It was created out of dissatisfaction with available programming tools, especially for observatory automation.

Mr. Moore and several associates formed FORTH, Inc. in 1973 for the purpose of licensing and support of the FORTH Operating System and Programming Language, and to supply application programming to meet customers unique requirements.

The Forth Interest Group is centered in Northern California, although our membership of 950 is world-wide. It was formed in 1978 by FORTH programmers to encourage use of the language by the interchange of ideas through seminars and publications.

PUBLISHER'S COLUMN

This is a special issue of FORTH DIMENSIONS. It is the regular issue but it is also very special. It includes the complete text of Charles Moore's speech at FORTH Convention, October 1979, in San Francisco. The founder of FORTH has given us a historical and futuristic view of FORTH. Thank you, Chuck!

This issue completes Volume 1 of FORTH DIMENSIONS and what a way to finish. The largest issue to date and the complete Charles Moore article. Look for Volume 2, Number 1 soon.

Roy Martens

FORTH, The Last Ten Years and The Next Two Weeks ...

Charles H. Moore
Chairman of the Board
FORTH, Inc.

WELCOME

Thank you. You honor me just by being here and being so involved in something that I never really expected would be this interesting to a group of people. I think way back in the Dark Ages I had in mind maybe some day addressing the Rotarians about FORTH. This is a rather more select group.

It turns out that FIG's estimate of this being the tenth birthday party for FORTH is remarkably accurate. By way of explanation, this is not intended to be a history of FORTH. For one reason, I do not have a very good memory for such events and I am not going to be particularly accurate nor particularly complete -- I am just going to give you my impression of what's happened for ten years. I am not up with what's happening in Europe or even in San Francisco and I apologize for that, but there never seems to be a need to delve into the history of FORTH. There is a history but first I want to talk a little bit about what FORTH is. This has been a subject of some speculation.

ASPECTS

Is FORTH an operating system? Is it a language? Is it a state of mind? I propose to trace five threads of history through ten years. I am going to do it in such an order that if we cut off the end nobody will care.

The five aspects of FORTH are philosophy, language, implementations, computers, and organizations, (meaning groups like FIG). If you talk about FORTH, the language, you can talk about some of these things and if you talk about FORTH the company you can talk about other things. This is a reasonable organization for what is really a broadly based attack upon the problems of society.

When I was very young I don't think I would have liked myself very much. I recollect being rather arrogant — that is a little bit too strong -- I wanted to do things my way, I was not convinced that I should not be permitted to, and I think I was a bit hard to get along with. That's all changed now. But in particular I was insecure. I was promoting certain ideas which everyone told me were wrong and that I thought were right. But if I were right, then all those other people had to be wrong and there were a lot more of them than me. It took a lot of arrogance to persist in the face of rather massive disinterest.

You may have noticed that FORTH is a polarizing concept. It is just like religion or politics, there are people who love it and people who hate it and if you want to start an argument, just say — "Boy, FORTH is really a great language".

I think there were some of you around ten years ago who may be aware of the problems that a programmer would encounter. They are exactly the same problems that a programmer encounters today! There has been no progress in the software industry for the last twenty years. This was apparent ten years ago and it was unsettling. It did not seem that the last thought had been "thunk" when FORTRAN was invented and yet nobody seemed to question that. It was the unspoken assumption that things are the way they are and they cannot become substantially different.

Speech at FORTH Convention, October 1979, San Francisco. CA.

PHILOSOPHY

Let me about philosophy now. I was a free-lance programmer once upon a time (1968). I went to work for a carpet manufacturer and learned COBOL partly out of financial necessity and partly with the thought "here's a language I don't know — let's pick up one more". These people acquire a graphics system. It was an IBM 1130 with a 2250 graphic display unit, a very nice state-of-the-art outfit, expensive! Speculation was that this would help us design carpets or maybe furniture. Nobody was really sure but they wanted to try. The 1130 was a very important computer. It had the first cartridge disk. It also had a card reader, a card punch, and a console typewriter. The backup for the disk was the card punch! I don't think I ever backed-up the disk but I do remember reloading the operating system numerous times.

The 1130 went away one day because without color it really wasn't worth anything for manufacturing carpets. They also had a Burroughs 5500 which was running ALGOL at a time when ALGOL was not popular. A very nice machine. They were programming in COBOL, which was the first good COBOL. These were progressive people and want to credit them -- Mohasco Industries. I put FORTH on the 5500 -- this is fairly unusual. It was cross-compiled to the 5500 from the 1130, since there is no assembler on the 5500. There is a dialect of ALGOL called ESBOL that Burroughs used to compile operating systems (it was not available to the users). But I learned about push-down stacks from this machine and had a lot of fun. It was third-shift work because the 5500 was busy. It was replaced by a Univac 1108, so I implemented FORTH on the 1108. It controlled the interactions of a bunch of COBOL modules which did all the real work. The 1108 was cancelled due to anticipated financial reverses; the programming staff quit; and I went to work for the National Radio Astronomy Observatory (NRAO).

Before I left -- my last week -- I wrote a book. It was entitled Programming a Problem Oriented Language and it expressed my philosophy at the time. It is very amusing reading. It also describes what was then FORTH. It makes amusing reading because, unknown to anyone, I was expressing opinions, attitudes, not designing a programming language.

FORTH first appeared on that 1130 and it was called FORTH. It had all the essential characteristics of FORTH and I will return to that point later.

F-O-R-T-H is a five letter abbreviation of "fourth," standing for fourth-generation computers. This was the day, you may remember, of third-generation computers and I was going to leapfrog all of that. But FORTH ran on the 1130 which only permitted five-character identifiers.

The first thing that was modern FORTH was the Honeywell 316 program at NRAO. I was hired by George Conant to program a radio telescope data acquisition program. I was given a computer (to the envy of other people who felt they deserved it). I was turned loose to do whatever I could with it, provided I came up with a product. I just went off and nobody really wanted or appreciated what they ended up with.

We developed a number of systems at NRAO and encountered the issue of patenting software. Programs cannot be patented; ought not to be patented; would be very expensive to patent. NRAO has an agreement with Research Corporation, a company that tries to pull from universities some technology spinoffs that can be used to better mankind. (They patent things for people who don't know (or care) how.) FORTH seemed like something that perhaps should be patented, so we spent a year writing proposals, investigating and getting lawyers' opinions. The conclusion was that maybe it could be patented, but it would take Supreme Court action to do it. NRAO wasn't interested. As inventor, I had fall-back rights but I didn't want to spend $10,000 either, so FORTH was not patented. This probably was a good thing. I think that if any software package would qualify for patenting FORTH would. It has no really innovative ideas in it, yet the package would not otherwise have been put together. If you apply this reasoning to hardware, hardware is patentable. It is one of my disillusionments that the establishment refuses to provide any effective protection for software. Probably it is the lack of vocal objection from within the industry and the willingness to acquiesce, knowing that today's software will be obsolete in a year anyway.

Given interest from other astronomers, a few believers formed FORTH, Inc. We developed miniFORTH (FORTH on minicomputers) with the idea to have a programming tool. The first important realization of that tool came when we put an LSI-11 and FORTH into a suitcase. I think I became the first computer-aided programmer, in that I had my computer and took it around. I talked to my computer, my computer talked to your computer and we could communicate much more efficiently than I could directly. Using this tool we put FORTH on many computers. My goal in all of this was to make myself a more productive programmer. Before all this started, I had figured that in forty years I could write forty programs at the rate I was going. That was it. Period! That was my destiny but I wanted to write more programs than that. There were things out in the world to be done and I wanted to do them.

It has taken a long time. I still don't have the computer I want, but I'm working at ten times that rate and I see other computer-aided programmers now. I am amazed that it should not have been obvious that programmers had to be computer-aided. To expect the programmer to deal with an intrinsically unfriendly machine on his own is not in keeping with the attitude that we preach for other people to follow.

As time went on it became apparent that FORTH is an amplifier. A good programmer can do a fantastic job with FORTH; a bad programmer can do a disastrous one. I have seen very bad FORTH and have been unable to explain to the author why it was bad. There are characteristics of good FORTH: very short definitions and a lot of them. Bad FORTH is one definition per block, big, long, dense. It is quite apparent, but very hard to point to an example of something that went awry or explain why or how. BASIC and FORTRAN are much less sensitive to the quality of the programmer. I was a good FORTRAN programmer. I felt that I was doing the best job possible with FORTRAN and it wasn't much better than what everyone else was doing. I indented things a little more nicely, maybe, and I declared some things that everybody else left to get declared by default. What more can you do? In a sense I said, "let me do it right. Let me use a tool which I appreciate and if everyone can't use this tool well, I am sorry, but that is not my goal." In that sense FORTH is an elitist language. On the other hand, I think that FORTH is a language that a grade-school child can learn to use quite effectively if it's presented in the proper bite-size pieces, with the proper motivation.

Finally, polyFORTH is a condensation of everything we have learned in the last ten years of developing FORTH. I think it is a very good package. I foresee no fundamental changes in the design of the language except for accommodation to the standards which are becoming increasingly important. Up until now there has been no reason for standards. There are internal standards of FORTH, Inc. internal standards at Kitt Peak for the effectiveness of the organization, but there has never been a demand for portability. In fact I know very few programs that portability has ever been seriously attempted with. The time has clearly come to change that.

There will be developments in other areas and one was brought home most forcibly today. It may be that FORTH is not merely a programming language. It may be saying something much more important about communication — between people, between computers, between animals. This is startling! It had never occurred to me that anyone would really "speak FORTH" in an attempt to communicate with anything else than a computer; it is not any longer clear that that is the case. There may be concepts embodied in FORTH of greater general utility to the basic problem of communication.

Now in concluding the philosophy section, I would like to read a poem. This is a poem that some of you have heard. It is a translation of a classic of English literature and it goes as follows:

```
: SONG
  SIXPENCE !
  BEGIN RYE @ POCKET +! ?FULL END
  24 Ø DO  BLACKBIRD I + @ PIE +! LOOP
  BAKE BEGIN ?OPENED END
  SING DAINTY-DISH KING ! SURPRISE ;
```

The author is Ned Conklin, who is very good at that sort of thing and is the first FORTH poet. Is there a place for this in the world of communication? I don't know. It is remarkably easy to come up with such paraphrasing of just about anything that you care to paraphrase. It's not clear that it's not an efficient means of communication.

INTRODUCTIONS

Let me introduce two people since I have touched upon the subject. It is ten years since there was one FORTH programmer. I would estimate that there are now 1,000 FORTH programmers, which is 2 to the 10th power and comes out nice and round -- a doubling time of one year. Actually, I think the doubling time is slightly shorter than that -- 10 times in three years and that comes out to 2,000 as some people would prefer. What we conclude is that next year there will be twice as many programmers; the year after that twice as many, and if you believe numerology these projections are unarguable. There is a curve, you extrapolate the curve and draw the conclusions. We don't know how it is going to come about. FORTH, Inc. can't train twice as many people next year — well, maybe we can. But, somehow the FORTH community as a whole has got to train twice as many people next year and thereafter. Maybe the Apples and the Radio Shacks are going to be the method of accomplishing that. It seems that capabilities come along just about quickly enough to keep the exponential curve growing. I have fairly great confidence that 1) the doubling time is a year, and 2) it is going to continue. Now there is collateral evidence to support this, if you plot the number of FORTH systems or the dollar value of FORTH systems or percent penetration of markets. Each way, you get about the same growth curve, so I think the growth curve is honest.

Ten years ago there was one FORTH programmer. The second FORTH programmer is in the audience; please meet Elizabeth Rather. Now that is quite a quantum jump, from one to two. The next step was four and they came out of Kitt Peak and the growth can be traced from there, for awhile, if anyone cares to. Actually the first FORTH user is in the audience and that is Ned Conklin. He was head of the station at Kitt Peak for NRAO, running the telescope, responsible for committing his telescope to this risky venture. It is an important telescope because it is responsible for half of the interstellar molecules discovered in the last ten years.

Again, I didn't exactly ask permission to commit these people to this course of action. Nobody realized what the consequences were going to be. It doesn't seem to have worked out too badly. FORTH is still running on that telescope at Kitt Peak and on a lot of other telescopes.

LANGUAGE

Now let's talk about the language and how FORTH came to be what it is today. There is a pre-history which goes back much further than ten years and I have some slides showing that time. These are strictly pre-history -- I found an old pile of listings and I photographed them. The first component of FORTH to occur was the interpreter. [Figure 1] This is an example of an early interpreter programmed in ALGOL. This was done at Stanford Linear Accelerator Center back in the early sixties. This is a program which still exists and it is called TRANSPORT. It designs electron-beam transport systems. You see an early dictionary there. The word ATOM shows the LISP influence. ATOM is an indivisible entity, which we now call a "word." Having read a word DRIFT from an input card, I would execute the drift routine and so on. I have looked through innumerable listings and found this style of programming quite consistent — it's the way I wrote programs in those days. I had an input deck which got interpreted with a structure pretty much as you see it today: words separated by spaces, no particular limits on the length of the words (as you can see from SOLENOID), only the first characters, however, were significant.

```
IF ATOM="DRIFT" THEN DRIFT
ELSE IF ATOM="QUAD" THEN QUAD
ELSE IF ATOM="BEND" THEN BEND
ELSE IF ATOM="FACE" THEN FACE (-1)
ELSE IF ATOM=ROTATE" THEN ROTATE
ELSE IF ATOM="SOLENO" THEN SOLENOID
ELSE IF ATOM="SEX" THEN SEX
ELSE IF ATOM="ACC" THEN ACC

ELSE IF ATOM="MATRIX" THEN BEGIN IF NOT FITTING THEN BEGIN
   REAL A;
   WRITE1(3,0,0,CORE[S]); LINE (-(8+42x(ORDER-1)));
   FOR J=1 STEP 1 UNTIL 6 DO BEGIN
      FOR K=1 STEP 1 UNTIL 6 DO WRITE1(2,8,R1[J,K]xUNIT[K]/UNIT[J1,2]);
      LINE(0) END;
   IF ORDER=2 THEN FOR C=1 STEP 1 UNTIL 6 DO BEGIN
```

FIGURE 1.

Here is another example, quite similar. [Figure 2.] Here ATOM has become W and I am looking up + and - and T, R, A and I — which represent an early version of our text editor. That again is ALGOL. I am not completely clear what was being edited. I think it was some kind of files sort program, maybe on cards that were getting printed or rearranged.

FIGURE 2.

```
CYCLE; FILL OUTPUT WITH BUFFER[1].BUFFER[2]:
WHILE WORD NEQ "END    " DO
IF W=GM1 THEN REPLY ("OK   ")
ELSE IF NUMERIC THEN L:=MIN(W-1, EOF)
ELSE IF W="+       "THEN L:=MIN(L+WORD.EOF)
ELSE IF W="-       "THEN L:=MAX(L-WORD.O)
ELSE IF W="T       "THEN BEGIN
   IF WORD=G 1 THEN W:=1; W:=MIN(L+W-1, EOF);
   FOR L:=L STEP 1 UNTIL W DO BEGIN
      POSITION: TYPE END; L:=L-1 END
ELSE IF W="R       "THEN BEGIN
   POSITION; REPLACE END
ELSE IF W="A       "THEN BEGIN
   L:=EOF:=EOF+1; REPLACE END
ELSE IF W="I       "OR W="D       "THEN BEGIN
   IF NOT RECOPY THEN BEGIN
      RECOPY:=TRUE; REWIND(CARD) END;
   POSITION; (F W ="I       "THEN BEGIN
      PLACE: REPLACE END
   ELSE BEGIN EMPTY:=TRUE; IF WORD NEQ GM1 THEN BEGIN
         L:=MIN(1+W-1, EOF); SPACE(CARD, L-LO+1); LO:=L+1
         END END END
```

Here is another way of setting up a dictionary. [Figure 3.] I am filling an array with strings of text and I am going to search that array for a match, take the index and vector through a computed GO-TO.

FIGURE 3.

```
LABEL UNDEFINED, BACKWARD1, TYPE2, FIND, INSERT, DELETE, ERASE,
START, REPEAT1, BOUNDARY1, BEGIN2, END2, QUIT1, ALGOL, FORTRAN,
COBOL, DATA, PACK1;
SWITCH SW:=UNDEFINED, BACKWARD1 TYPE2, FIND, INSERT, DELETE, ERASE,
BACKWARD1, TYPE2, FIND, INSERT, DELETE, ERASE, START, REPEAT1, BACK,
BOUNDARY1, BEGIN2, END2, QUIT1, ALGOL, FORTRAN, COBOL, DATA, PACK1;
ALPHA ARRAY COMMAND[1:32];
FILL COMMAND[*] WITH
"-      ", "T      ", "F    ", "I       ", "D      ", "E      ",
"BACKWAR", "TYPE   ", "FIND ", "INSERT  ", "DELETE ", "ERASE  ",
"START  ", "REPEAT ", "      ", "BOUNDAR ", "BEGIN  ", "END    ",
"EXIT   ", "ALGOL  ", "FORTRAN", "COBOL ", "DATA   ", "PACK   ",
"       ",  COUNT:=0; RETURN; COPY ("EDIT#RE" U "ADY   ");
TRANSMIT;
BACK: SOURCE(1); WORD1;
   IF W="        "THEN GO QUIT1; GO TO SW[MEMBER(COMMAND.W)+1];
```

Here is another way of implementing the dictionary. [Figure 4.] This is the first appearance I have on record of a stack. I am looking up the words in a conditional statement and setting NEXT to the index. And that's the first appearance of NEXT which I can find.

FIGURE 4.

```
PROCEDURE RELEVANCE; BEGIN REAL T,KO;
   J:=0; I:=-1; WHILE WORD NEQ "END                 "DO
   IF W="=       "THEN NEXT:=3
   ELSE IF W="GT     "THEN NEXT:=4
   ELSE IF W="LT     "THEN NEXT:=5
   ELSE IF W="NOT    "THEN NEXT:=6
   ELSE IF W="AND    "THEN NEXT:=7
   ELSE IF W="OR     "THEN NEXT:=8
   ELSE IF W="+      "THEN NEXT:=9
   ELSE IF W="-      "THEN NEXT:=10
   ELSE IF W="*      "THEN NEXT:=11
   ELSE IF W="/      "THEN NEXT:=12
   ELSE IF KO:=SEARCH1(W) GEQ O THEN BEGIN
      NEXT:=1; NEXT:=k:=KO END
   ELSE BEGIN
      NEXT:=2;
      IF BASE[K]=" " THEN NEXT:=WORDS[0]
      ELSE NEXT:= W END;
   NEXT:=0 END;
```

Here is the other half of that -- this is the implementation of the stack. [Figure 5.] This is a variant of ALGOL called BALGOL that lets you put assignment statements inside other statements. "Stack of J replaced by J-1" is how you push something onto the stack. One of my "less-liked" features of ALGOL was that I had to play games like "real of boolean of stack of J and boolean of ..." just in order to get around the automatic typing that ALGOL was insisting that I apply. Now this was specifically intended to let me manipulate parameters that were interpreted from the card deck as arguments to the routines. In other words, if I wanted the sine of an angle, I could say ANGLE SINE but if I wanted to convert the angle from one unit to another, I needed at least some simple arithmetic operators and this provided them. This is again at Stanford.

FIGURE 5.

```

BOOLEAN PROCEDURE RELEVANT; BEGIN
   I:=J:=-1; STACK[0]:=1; DO CASE NEXT OF BEGIN
      J:=-1;
      STACK[J:=J+1]:=CONTENT;
      STACK[J:=J+1]:=NEXT;
      STACK[J:=J-1]:=REAL(STACK[J]=STACK[J+1]);
      STACK[J:=J-1]:=REAL(STACK[J] GTR STACK[J+1]);
      STACK[J:=J-1]:=REAL(STACK[J] LSS STACK[J+1]);
      STACK[J]:=REAL(NOT BOOLEAN(STACK[J]));
      STACK[J:=J-1]:=REAL(BOOLEAN(STACK[J]) AND BOOLEAN(STACK[J+1]));
      STACK[J:=J-1]:=REAL(BOOLEAN(STACK[J]) OR BOOLEAN(STACK[J+1]));
      STACK[J:=J-1]:=STACK[J]+STACK[J+1];
      STACK[J:=J-1]:=STACK[J]-STACK[J+1];
      STACK[J:=J-1]:=STACK[J]xSTACK[J+1];
      STACK[J:=J-1]:=STACK[J]/STACK[J+1];
      END UNTIL J LSS 0;
   RELEVANT:=BOOLEAN(STACK[0]) END;

```

Now here is a PL/1 program doing very much the same thing at a considerably later date. [Figure 6.] At the top you see JCL (Job Control Language) which was also not a pleasant thing to deal with. One of the criticisms of programming languages that I mentioned in my book was that a programmer at a typical computer center, in order to function, needed to know nineteen languages. This covered writing Fortran programs, submitting card decks, etc. These languages were all subtly different with commas here and spaces there and equal signs meaning different things — nineteen languages, just to function. Nobody advertised the fact. Nobody sat down and took a course in nineteen languages, but you had to pick them up in the course of several weeks or several months in order to be effective. FORTH, I figured could replace all of them.

Here is NEXT: PROCEDURE CHARACTER. [Figure 7.] I don't remember that syntax but that I think it is the first definition of NEXT as a procedure that went off and got the next word and did something with it. This is still all pre-FORTH. We haven't gotten to what I would consider the first FORTH system.

FIGURES 6 & 7

```
//UTILITY       JOB SYSTEM OVERHEAD
//              EXEC PGM=IEBUPDTE, PARM=NEW
//SYSPRINT      DD   SYSOUT=A
//SYSIN         DD   DATA
./             ADD   NAME=WORD,LEVEL=00,SOURCE=0,LIST=ALL
NEXT: PROCEDURE CHARACTER(4);
   DECLARE KEYBOARD STREAM INPUT, PRINTER STREAM OUTPUT PRINT;
   DECLARE (1 TEXT CHARACTER (81) INITIAL((81)" "),
      2 C(81) CHARACTER(1), I INITIAL(f`,W CHARACTER(4),
      WORD CHARACTER(32) VARYING BASED(),P,NUMERIC BIT(1)) EXTERNAL;
P=ADDR(C(I));
IF C(I)="-" OR C(I)="." OR "0" LE C(I) THEN BEGIN; NUMERIC="1"B;
   IF C(I) NOT="." THEN DO I=I+1 BY 1 WHILE "O" LE C(I); END;
   IF C(I)="." THEN DO I=I+1 BY 1 WHILE "0" LE C(I); END; END;
ELSE DO; NUMERIC="0"B;
   IF "A" LE C(I) THEN DO I=I+1 BY 1 WHILE "A" LE C(I) OR C(I)="-"
   END; ELSE; I=I+1; END;
W=WORD; RETURN(W);
```

Here is a rather later version of FORTH coded for the IBM 360. [Figure 8.] Those are the routines PUSH and POP. PUSH cost 15 microseconds on an IBM 360-50. It includes stack limit checking, which doubled the cost and was one of the things that led me to feel that execution-time stack checking was not desirable and in fact not necessary. However, up to that point, the consequences of a runaway stack were terrifying. POP is there also. It was coded in a macroassembler that did not have stack operations. It was not possible to refer to a previous "anything" so the deck is full of "L19 data constants, address, AL2(*-L18)" to give me a relative jump to the previous one. It could all be done but it wasn't pleasant.

FIGURE 8.

```
L18 DC AL2(*-L17)
   NAME 3,X'+4555)',0  DUP
          DC    AL1(3),x'445550 '
          DC    X'0'
          ORG   *-2-VO
          DS    OH
          ORG   *+ 0+1
          DC    AL1(0*X'40'+X'40').AL2(4)
PUSH A SP,MFOUR   COSTS 15 OS
   ST T,0(,SP)
   CB SP,DP
   BCR 2,NEXT  BHR
      B ABORT
L19 DC AL2(*-L18)
   DC AL1(4),X'4402CF50',X'40',AL2(8)   DPCP
   LA SP,4(,SP)
POP L T,4(,SP)   COSTS 21 US
   LA SP,4(,SP)
   C SP, SPOO
   BCR 12, NEXT  BNHR
      B ABORT
```

Here is a version of FORTH coded in COBOL. [Figure 9.] This was done at Mohasco, of course. I am setting up a table of identified words which I am going to interpret from an input string. The attitude is so pervasive that I begin to think that I was talking myself into something here. COBOL is fairly difficult to write subroutines for. They have subroutines, they can be performed, but they may not have any parameters. This makes it a little bit awkward to do anything meaningful.

FIGURE 9.

```
MOVE "CONFIGURATION" TO IDENTIFY(4);
MOVE "DATA" TO IDENTIFY(5);
MOVE "FILE" TO IDENTIFY(6);
MOVE "FD" TO IDENTIFY(7);
MOVE "MD" TO IDENTIFY(8);
MOVE "SD" TO IDENTIFY(9);
MOVE "WORKING-STORAGE" TO IDENTIFY(10);
MOVE "CONSTANT" TO IDENTIFY(11);
MOVE "PROCEDURE" TO IDENTIFY(12);
MOVE "INPUT-OUTPUT" TO IDENTIFY(13);
```

Here's the first example of FORTH text. [Figure 10.] This came out of Stanford again. The word DEFINE begins (that is,:) a definition and the word END (that is,;) ends it. The "OPEN is obscure. "NAME seems to be the way the name was introduced. Apparently there did not have to be a space between the quote and the word. There are the definitions of a number of stack operators. Top of the line is CODE - "OPEN DEFINE MINUS + END ; I guess that is subtraction. SEAL was an early word for sealing the dictionary for some reason. BREAK, I guess broke the seal. "< : OPEN DEFINE - < END ; is the same definition we use today, in a very early state. That was from a thing I called "Base Two," intended to be some kind of base programming language — I can't remember any more about it.

FIGURE 10.

```
"- "OPEN DEFINE   MINUS  +    END
SEAL   "< "OPEN DEFINE    - <    END BREAK
"NOT "OPEN DEFINE    MINUS 1+    END
"< "OPEN DEFINE   .<    END
"AND "OPEN DEFINE   x    END
"OR "OPEN DEFINE  NOT   .NOT AND NOT    END
   "T 1 1 "REAL DECLARE
"= "OPEN DEFINE   T=; DUP T<  . T> OR NOT   END
"≠ "OPEN DEFINE   = NOT   END
"≤ "OPEN DEFINE,  > NOT   END
"≥ "OPEN DEFINE   < NOT   END
"DUMP "OPEN DEFINE   NAME 10 "ALPHA WRITE; 3 10 "REAL WRITE 0 LINE  END
```

This is a version of FORTH source for the 5500. [Figure 11.] Again, very early — the second computer that FORTH was put on. Apparently ¢ stands for CODE and these are the code definitions of the stack operation on a 5500. Now the 5500 is a stack machine at a time when stack machines were not at all popular. They did a very good job with their stack. All of these were implemented with one 12-bit instruction and the present names of these operations are directly derived from the names of the 5500 operations. That's where DUP came from, for instance. Notice that the ¢OR was a way of distinguishing the assemblers OR from the FORTH OR before vocabularies were available.

FIGURE 11.

```
LIST
(   'PRIMITIVES'  26 LAST=  30 SIZE=)
¢ =   S RETURN
¢ @  <SD RETURN
¢ +V 241, RETURN

¢ OR    ¢OR RETURN
¢ AND  ¢AND RETURN
¢ NOT  115, RETURN
¢ DUP  ¢DUP  RETURN
¢ SWAP ¢SWAP  RETURN
¢ DROP ¢DROP  RETURN
¢ +  +1  RETURN
¢ -  -1  RETURN
¢ MINUS  ¢MINUS  RETURN
¢ *  *1  RETURN
¢ /  /1  RETURN
¢ MOD ¢MOD  RETURN
```

Here's an example of FIND coded for the 5500. [Figure 12.] Notice that the word SCRAMBLE is referred to, which is a colon-definition for doing a hashed search. Apparently here I had eight threads, just as we put in polyFORTH last year. These ideas go way, way back. This is FORTH after the threshold was crossed, ten years ago, almost exactly. One can become a little bit depressed at the "tremendous" rate of progress in the last ten years when you see that it was all back there.

FIGURE 12.

```
¢SM ¢FIND   SCRAMBLE  <SD  ¢DUP
    41 >A  41 >B  ¢BEGIN  V  <J  1771,  ¢IF
      ¢BEGIN  VO <U  1771,  ¢IF
          1 <L  RESULT
        ¢THEN   ADDR  ¢DUP  1 <L  <S
        US WORD  <J  ¢EQUAL  ¢IF
          V1   J  US  RESULT
        ¢THEN  ¢DUP  <SD  ¢BACK
    ¢THEN  GET  ¢BACK
:  FIND  TOP  ¢FIND  ¢IF  UR <UD ¢B  ¢THEN;
```

Here is another example of source. [Figure 13.] This was from the Univac 1108. These are very early record descriptions. This is the layout of a record in a file with the name of the field and the number of bytes in the field. That was the Dun & Bradstreet reference file for looking up bad debts.

FIGURE 13.

```
DBI   DBI/MOORE 33 33
    DUNS 8  NAME 24  STREET 19 CITY 15 STATE 4 ZIP 5
    PHONE 10  BORN 3  PRODUCT 19  OFFICER 24  SIC 4 SIC1  4  SIC2
    SIC3 4 SIC4 4 SIC5 4  TOTAL 5.0 EMPL 5.0 WORTH 9.0 SALES 9.0
    SUBS 1 HDQ 1 HEAD 8 PARENT 8 MAIL 19 CITY 15 STATE 14
    NAME1 19
END
```

This is the last slide. [Figure 14.] This is a modern FORTH source from FORTH, Inc. Vector arithmetic coded for the Intel 8086. CODE V+ with comments listing the arguments that are on the stack with the results being obvious. Fairly nicely spaced out code. It just looks like good, clean pure FORTH instead of that other gibberish. Not at all cryptic, perfectly obvious to the discerning reader and that's the end of the slides.

FIGURE 14.

```
   (VECTOR ARITHMETIC)
CODE V+ ( Y X  Y X)  0 POP   1 POP   2 POP   3 POP
   2 0 ADD  3 1 ADD  1 PUSH  0 PUSH    NEXT
CODE V- ( Y X  Y X)  2 POP   3 POP   0 POP   1 POP
   2 0 SUB  3 1 SUB  1 PUSH  0 PUSH    NEXT

CODE UMINUS  0 POP  1 POP   1 NEG   1 PUSH
   0 NEG   0 PUSH  NEXT
: UMIN   ROT MIN >R MIN R> ;

CODE V*/   ( Y X M D)  7 POP   3 POP   1 POP   0 POP
      3 IMUL   7 IDIV  0 PUSH
   1 0 MOV   3 IMUL  7 IDIV   0 PUSH   NEXT



COPYRIGHT FORTH, INC  OCT. 1979
```

Let me now sketch in time sequence the operations that were implied by some of those slides. The first thing to exist was the text interpreter reading punch cards. The next thing to exist was the data stack with the manipulations of the operators. Those took place very early, around 1960.

Nothing substantial then happened until 1968 and that was the 1130 and the ability for the first time to totally control the way the computer interacted with the programmer. This machine had the first console I had ever seen. I had always submitted card decks -- now I had a typewriter. Do you know what I did? I submitted card decks. It took a long time to figure out how to use the keyboard and whether or not the keyboard would do anything for me. I was very good at a keypunch and I really didn't care if there was a console or not.

The first FORTH was coded in FORTRAN. Very shortly thereafter it was recoded in assembler. Very much later it was coded in FORTH. It took a long time to feel that FORTH was complete enough to code itself. The first thing to be added to what had already existed was the return stack and I don't remember why that was. I don't remember why I didn't just put the return information on the parameter stack. It was an important development to recognize that there had to be two stacks -- exactly two stacks, no more, no less.

The next thing to be added was even more important. I don't know if you appreciate it but it was the invention of the dictionary. In particular, the dictionary in the form of a linked list. In particular, the existence of the code field in the header. For control, up until then, flags had been set, computed GO-TOs executed, some mechanism for associating a subroutine with a word. Now the existence of the address of the routine (rather than an index to the routine) made an incredibly fast way of executing a word once it had been identified. No other language has a code field or anything resembling it. No other language feels obliged to quickly implement the code that the word identifies. You can go about it at your own precious time, but even in these days it was important that FORTH be efficient. The whole purpose of this system was to draw pictures on the 2250 display. The 2250 was a stand-alone minicomputer interfaced with the 1130. What came out of the 1130 was a cross-assembler which assembled the instructions which were then to be executed by the 2250. I think the 2250 had its own memory. Again, very sophisticated things being done very early in a very demanding environment. IBM software, in 16K of memory, could draw pictures on the 2250 fairly slowly. What I accomplished in 4K would draw three-dimensional moving pictures on the 2250. But it could only do that if every cycle were accounted for and if the utmost was squeezed out -- that's why FORTRAN had to go. I couldn't do an impressive enough job with FORTRAN, an assembler was the requirement.

At this time colon definitions were not compiled -- the compiler came much later. The text was stored in the body of the definition and the text interpreter reinterpreted the text in order to discover what to do. This kind of contradicts the efficiency of the language but I had big words that put up pictures and I didn't have to interpret too much. The cleverness was limited to squeezing out extraneous blanks as a compression medium and I am told that this is the way that BASIC executes today in many instances.

This machine had a disk -- I can't prove it but I am almost certain that the word BLOCK existed in order to access records off the disk. I do remember that I had to use the FORTRAN I/O package and it wouldn't put the blocks where I wanted them -- it put the blocks where it wanted them and I had to pick them up and move them into my buffers. It had an assembler for assembling 1130 code, it had a target assembler for assembling 2250 code and clearly B-5500 code. The B-5500 program was taken far enough to recompile itself. Beyond that there was no application because there was no way I was going to get access to that machine outside of the third shift.

This was the transition point between something that could not be called FORTH and something that could. All the essential features except the compiler were present in 1968. It took a long time for the next step.

The first compiler occurred on the Honeywell 316 system at NRAO several years later. It resulted from the recognition that, rather than reinterpreting text, the words could be compiled and an average of five characters per word could be replaced by two bytes per word at a compression of a factor of two or three. Execution speed would be vastly faster. Again, if it was that easy, why hadn't anyone else done that? It took me a long time to convince myself that you could compile anything and everything. The conditional expression, for instance, had to be compiled somehow. Before compilation, if you came to an IF you could scan ahead in a text string until you came either to an ELSE or a THEN. How did you do an IF if you were going to compile things? But, it worked! Again, I can't remember the sequence.

It may be that the 316 compiled the 360, but I think the 360 compiled the 316. Again, in the early days of FORTH, the idea of of today was there -- cross compiling, cross assembling between different computers was there. Interrupts came at about this time. It was important to utilize the interrupt capability of the computer but it had not been done by me before that. I didn't know anything about interrupts, but I/O was not interrupt-driven. Interrupts were available for the application if it wanted them. FORTH didn't bother.

The multi-programmer came along a couple of years later when we put an improved version of the system into the Kitt Peak PDP-11. This multiprogrammer had four tasks. Input was still not interrupt-driven, which was unfortunate. Interrupt-driven I/O came along when FORTH, Inc. produced its first multi-terminal system. It did not speed things up particularly. If you count cycles it was much more efficient, and it prevented any loss of characters when many people were typing at the same time. FORTH didn't have to look quickly to get each character before the next one came along, as they were all buffered and waiting.

Data-base management came along at this time. It has been extensively changed, just like FORTH has, but fundamentally nothing has changed. The concept of files and records and fields that I outlined this afternoon dates back from 1974 or so.

The first target compilers came along later with microFORTH. They are very complex things, much more so than I had expected them to be.

I think that completes the capabilities that I think of as FORTH today and I think you can see how they dribbled in. At no point did I sit down to design a programming language. I solved the problems as they arose. When demands for improved performance came along I would sit and worry and come up with a way of providing improved performance. It is not clear that the process has ended, but I think it is clear that that process has now got to be carried into the hardware realm.

HARDWARE

I have designed some computers. This is expensive because I am supposed to be earning money by writing software. I think that hardware today is in the same shape as software was 20 years ago. No offense, but it's time that the hardware people learned something about software and there is an order or two magnitude improvement in performance possible with existing technology. We do not need pico-second computers to make substantial, really substantial, improvements in speed. And faced with that realization, there is no point in trying to optimize the software any further until we have taken the first crack at the hardware. The hardware redesign has got to be as complete as the software redesign was. The standard microprocessors did not have FORTH in mind. Those minicomputers that can be microprogrammed cannot be microprogrammed well enough to even be worth doing. The improvements available are much greater than you can achieve by these half measures.

IMPLEMENTATIONS

All right, let's switch gears a little bit. I would like to talk about the implementations of FORTH of which I am aware. I have touched on them already but I want to rattle off a string of CPUs which have come to mind just to dazzle you with the capabilities. It is actually a tour through the history of computers and it is fascinating that this could all have happened in ten years.

FORTH has been programmed in FORTRAN and in ALGOL and in PL/1 and in COBOL and in assembler and in FORTH. I am sure some of you can come up with other languages with the same history. It has been done on the IBM 1130, the Burroughs 5500, the Univac 1108, the Honeywell 316, the IBM 360, the Nova, the HP 2100 (not by me, but by Paul Scott at Kitt Peak), the PDP-10 and PDP-11 (by Marty Ewing at Cal-Tech), the PDP-11 (by FORTH Inc.), the Varian 620, the Mod-Comp II, the GA/SPC-16, the CDC 6400 (by Kitt Peak), the PDP-8, the Computer Automation LSI-4, the RCA 1802, the Interdata, the Motorola 6800, the Intel 8080, The Intel 8086, the TI 9900 and coming soon the 68000, the Z8000, the 6809 -- I know you people have 6502s and Four Phase. (Audience: And Illiac!) I've raised the question -- is it the case that FORTH has been put on every computer that exists?

COMPUTERS

We speak now about FORTH computers -- there are FORTH computers. The first one I know of was built at Jodrell Bank in England around 1973. It is a redesign of a Ferranti computer that I think went out of production. They were going to build their own bit-slice version and they discovered FORTH about the same time, modified the instruction set to accommodate FORTH, and built what I am told is a very fast FORTH computer. I have never seen it. I have talked to its designer, John Davies, who is one of the early FORTH enthusiasts and eminently competent to do this.

In 1973 came General Logic and Dean Sanderson. The machine qualifies as a FORTH computer because it has a FORTH instruction set and there is a story there. Dean showed me his instruction set and there was this funny instruction that I couldn't see any reason for. I figured it was some kind of no-op or catch-all because it had the weirdest properties. It couldn't possibly be useful -- it was NEXT. It was a one-instruction NEXT -- it was beautiful. And it was a very simple modification to the instruction set. A few wires here and there and that was the first time I saw a FORTH computer. Here was the ability to change an ordinary computer to make it into a FORTH computer.

We have had some ideas about other such modifications that could be made effectively, but I don't know that any of them have been carried out. I am told that Cybek has got its own machine now, built by Eric Fry. These are rumors that I'm just passing on. Child, Inc. does raster graphics systems. I am told they were working on a FORTH computer that was supposed to be available last February -- I haven't heard. Again, I think they are competent to do it and do it well. A blindingly fast FORTH computer on a board, probably biased towards graphics applications, raster graphics.

I have built a FORTH computer called BLUE. It's small. It has never executed any FORTH yet. The design changes as fast as the chips can be plugged into the board, but it's not hard to do.

What are the characteristics of a FORTH computer? It does not need a lot of memory. 16K bytes is about right. Half PROM, half RAM, maybe. It does not need a lot of I/O ports -- it does not need any I/O ports except for the application requirements. A serial line is nice; a disk port is nice. We have put FORTH on an 8080 with disk replaced by enough core to hold 8 blocks. Quite viable, no particular problem with system crashes, a somewhat protected environment. Bubble memories are coming, and Winchester drives of course. We don't need much mass memory, we only need on the order of 100 to 250 blocks. The fact that FORTH can exist quite happily on a very small machine by contemporary standards should be exploited.

ORGANIZATION

Finally, I would like to run through the history of the organizations which have been involved with FORTH. They form another thread to the tapestry. Mohasco, of course, and the National Radio Astronomy Observatory. They birthed it and they rejected it. Thats pretty much why I am here today instead of in South America programming telescopes. They had what we have learned to identify as the NIH Syndrome but in a weird mutated case because it was invented there! It is their loss. You have perhaps read about the VLA, a very large array of antennas in New Mexico -- a very exciting project. Something that I really would have liked to have programmed. It wasn't in the cards (and they have software problems today). On the other hand, there was Kitt Peak. Astronomers are a conservative lot. This may be surprising. I think they are surpassed only by nuclear physicists in being conservative. We have been unable to scratch the nuclear physics field here although I am told that CERN is interested. NRAO is a sister laboratory to Brookhaven. One would think that there would be some communication between them and there isn't. They are both managed by the same University Association. We couldn't interest Brookhaven and we couldn't interest NRAO, but we could interest Kitt Peak and Elizabeth Rather is the one who did it. She liked FORTH and talked a lot of other people into liking it, too. Kitt Peak adopted FORTH — gave it the impetus because Kitt Peak is the show place of the astronomy world. Kitt Peak put FORTH on a whole batch of Varians. It's fun because I remember Varians breaking down and spares being wheeled in -- there were 14 of them lined up in the hall. Reliability through redundancy? Pipeline architecture? A lot of other observatories picked it up.

We were deluged by requests for FORTH systems from astronomers and went into business to try to exploit that market. It is a market we would still be in today except that there are so few new telescopes in the world, and you can't support a company on that market.

The formation of FORTH, Inc. was important because I don't think we would be here today if it weren't for FORTH, Inc. We worked very hard to try to sell this thing. We didn't know what we were getting into; we were your classic naive, small business folk. I caution any of you against thinking it is easy to go into business for yourself. It is fun, but the advice is true -- do not go into an area where you must create the demand for your product. But that was the least of the problems really that FORTH has faced. The next step was probably DECUS. Marty Ewing gave his PDP-11 FORTH system to DECUS. I didn't know if that was a good idea at the time — free FORTHs floating around. It was important because a lot of people were exposed to FORTH who otherwise would not have been. I imagine we picked up a few sales through that channel. Cybek came along. Cybek is probably the savior of FORTH, Inc., in that it provided us lucrative business at a time where we desperately needed it to stay alive. Art Gravina, the President of Cybek is the one that designed (if that's the word) our data-base management system. We can argue about who did what, but Art provided the opportunity to do commercial systems. He got a good deal because he could handle 10 times as many terminals as he could with the BASIC program that preceded FORTH; we learned everything we know about data base management from him. We also acquired our distaste for commercial programming at that time. I commend those of you who are involved in it. I find it much to heavy in system analysis for my taste. You've got to go in there and tell the businessman what he has to do, talk him into it, and hold his hand all through the process of installation. It is a different talent from that of writing programs. Don't underestimate the cost of support.

I think that is about the time that the International Astronomical Union met and agreed on FORTH as a standard language. That was a boost in the world of astronomy although the world of astronomy was no longer the major driving force in the popularity of FORTH. I think EFUG came along about that time -- this was '76 or so -- the European FORTH Users Group. It turned out, to our surprise, that Europe was a hotbed of FORTH activity of which we were largely unaware and perhaps really still are unaware, in that we are not involved in that world and don't quite appreciate the level of interest. FST (FORTH Standards Team) probably began in EFUG's first meetings. Later, a couple of years ago, FIG was started and now we have FORML (FORTH Modification Laboratory), which is an idea-generating organization. The tendency seems to be for people to organize themselves into groups. Some of these groups are companies, some of these groups are associations. It looks like FORTH is going to be a communal activity in the sense of unstructured clusterings of like minded people. The suggestion is that this whole world of FORTH is going to be quite disorganized, uncentralized, uncontrollable. It's not bad, it's perhaps good.

CONCLUSION

To close on a philosophical note: power to the people. This is the first language that has come up from the grassroots. It is the first language that has been honed against the rock of experience before being cast into bronze. I hesitate to say it is perfect. I will say that if you take anything away from FORTH, then it isn't FORTH any longer, that the basic components that we know are all essential to the viability of the language. If you don't have mass memory, you've got a problem and it can't be waved away. I hesitate to predict. I don't know what is going to happen. I think my view of the future is more unsettled tonight than it has been for years. Promising, yes, confusing, and perplexing. The implications are perhaps as staggering now as they were ten years ago. The promise of realization is much higher. This is ten years of FORTH.

My original goal was to write more than 40 programs in my life. It think I have increased my throughput by a factor of 10. I don't think that that throughput is program-language limited any longer, so I have accomplished what I set out to do. I have a tool that is very effective in my hands -- it seems that it is very effective in others' hands as well. I am happy and proud that this is true.

I wish that the future smiles on you and all of your endeavors.

(Mr. Moore's address concluded to an extended standing ovation.)

DISCUSSION PERIOD

Question: When did <BUILDS and DOES> come along?

Mr. Moore: That's a good question -- ;CODE came first. I think it began way back in 1130 days with the notion that you could define a word that would define other words. That was staggering. I couldn't grasp the implications. ;CODE was a very esoteric word. I explained it to people proudly but I couldn't express the potential I saw in it. I didn't know what ;CODE should do. (It specified the code to be executed for a previously defined word.) I don't have it but I think the initial assembler code for ;CODE was three or four lines long. One of the driving forces behind the address interpreter was making it possible to code ;CODE cleanly. This had all kinds of implications as to what registers should be available. W should be saved in a register instead of (pardon the expression) direct threaded code recovered somewhere because that was expensive. I had a lot of trouble with ;CODE. That was the most complicated routine I had coded in this systems programming fashion. Not so much later it seemed that there ought to be an analog of ;CODE which specified the code to be interpreted when you executed a word. It seemed the natural balance, but I hadn't the foggest idea of what the implementation should be. The first definitions of ;: required three or four lines of code. You had to do what ;CODE did and then more and this couldn't be explained to anyone. Out of that grew the distinction between compile time action and execute-time action and the present form of CREATE and DOES>. This was due to Dean Sanderson, again. It was very convenient for words to be coded to act this way, but it was expensive. It required not only the address of the code to be executed, but the address of the code to be interpreted as well as the parameter to be supplied to the code to be interpreted. Questions included: should the parameter be put on the stack or should the address of that parameter be put on the stack? Should that parameter be at the beginning of the parameter field where it was a little bit awkward or one word in? Now I don't know if you are aware of the new DOES>? It now has full symmetry with ;CODE.

Again, this grew out of a clear perception of what the word is and what it does. I know of no way of speeding the process from initial thought to development except to let a certain amount of time pass. We sat and debated this thing endlessly and missed the obvious. The current implementation of DOES> does not require the address of the code to be interpreted. That is supplied by a different mechanism and therefore the parameter can occupy the parameter field as it is supposed to. Therefore you can "tick" it and change its value, which is wonderful, except that it is going to be in ROM and won't matter. We save two bytes per DOES> definition. Two bytes per word for a very common class of words and for three years we didn't realize that we had missed the optimum by so much! Although this is proprietary to FORTH, Inc., I am sure that given these clues all of you will proceed to go off and invent the new DOES>. Maybe that's the way things should be.

<u>Question</u>: When you were in the philosophy section, you said that you still don't have the computer that you want and you sort of alluded to that at various times. Can you tell us just what computer you would like to have?

<u>Mr. Moore</u>: Well, first and foremost it has to be reliable. I want an MTBF of ten years. I see around me computers that fail in six months and that is preposterous. You do not get an MTBF of ten years by taking ten computers with an MTBF of six months and lining them up in parallel. I want a computer that I can drop in the ocean and fish out and it doesn't care. I want a computer without an on/off switch. I want it to be small. By small I guess I mean I want it to fit in my pocket, I don't really want it on my wrist. These considerations have consequences. I consider that a reliable computer is one with small parts count. The probability of failure is proportional to the number of parts. So if you only have six parts you can last maybe ten years -- I don't know. I would like it to have voice input/output. A terminal is acceptable but this isn't small. Are you familiar with the Write-Hander? A very nice input device which, if it's perfectly matched to your fingers and if you can train your reflexes to depress two keys at the same time, is a good idea. I am afraid the implementation is not perfect. It probably has to be customized to your hands somehow and even then it is only a substitute for voice input. No power supply, of course, it should run off body heat.

The potentials in the field of communication are enormous. I don't like telephones much. I speculate that I would talk on the telephone if there were a computer between me and it. I speculate that a personal computer like this could be an interface between an individual and society. If I wanted a new driver's license I would tell my computer to get me a new driver's license and it would deal with the bureaucracy.

<u>Question</u>: Can you develop a version of FORTH that will be machine-independent?

<u>Mr. Moore</u>: The premise is wrong. The equivalence of FORTH on different machines requires meticulous attention to the characteristics of these machines. You must use all the hardware capabilities of each machine and you must then work to force it into the mold specified by FORTH's virtual machine.

The internal characteristics of every machine can and must be exploited. You do not need any particular number of registers or stacks, as they can all be simulated; but if you neglect the capability of the machine, you can end up a factor of two down from where you might otherwise be.

Question: So far I've heard you say that things can be done differently but I wanted to hear what you had to say about the architecture of the processor itself.

Question: I would like to support that question with one more question. I have a customer who ran 64 users on a 5500 for ten years and bought a VAX and can only run 16. My question is -- which is progress and what is a suitable architecture for a FORTH machine?

Mr. Moore: If you measure the size of those two machines, we could put 64 users on a VAX, no problem. The fact that he couldn't is a condemnation of the software, not the hardware.

The characteristics of a FORTH processor? I don't want to go into detail but I will say it is substantially simpler than machines like the 8086, 6800s, simpler than the 8080, probably. You don't need much more than the ability to execute microcode. Very simple microcode will serve for this very simple architecture and the performance comes out of the simplicity. If you add complexity, you are going to decrease your reliability and increase power consumption, all those bad things. I would like to see some studies made of the tradeoffs. I would like to know how complicated FORTH is. Take FORTRAN. You cannot measure its complexity. You can't say a FORTRAN program is 10 to the sixth power in complexity. You can't measure that, you can't separate FORTRAN from the operating system, from the floating-point hardware. You can't determine how complicated a problem it is that that FORTRAN is trying to solve. If we had a standard FORTH implemented on a chip with enough RAM, PROM, and CPU and all FORTH words on that chip, we could measure the area of that chip and get a measure of the difficulty of the task that we were trying to solve. We could compare two implementations with different hardware/ software tradeoffs based on the common measure of the area it took to implement. I think that would be a very, very interesting thing to do.

There is a speculation that human brains have a great capacity, 10 to the 13th bits, some preposterous number. I suspect that human brains have a much smaller capacity than that, on the order of 10 to the 9th bits maybe. We make up for our bad memories by "faking it" a lot; people don't remember what happened ten years ago. I reconstruct what must have happened based on little clues that I pick up here and there. Those reconstructions are so accurate and so impossible to contradict that we get the impression that we can remember a very great deal. I suspect that we can build a computer as complicated as a human brain now and it won't need to be powered by Niagara Falls. It will be a small box.

Question: Assuming someone were to design a FORTH small micro-engine of some sort, how much faster or more capable do you think it would be over the present implementations? Just assuming that someone has done that. No guesstimate?

Mr. Moore: No, and I am not saying and don't you say either, Dean.

Question: Can we assume that it is less than 100 and more than 1?

Mr. Moore: A substantial improvement. We've got to have some edge. The rest of the world is going to come up and clobber us.

Question: You mentioned before the philosophy of seeking the least complicated solution for a problem. Would you comment on simplicity versus flexibility for a solution. How do you judge, how do you make tradeoffs?

Mr. Moore: If you have a thing that does one thing well it is of no interest or value unless the one thing that does well is FORTH. I have no idea. It never occurred to me to ask how you measure flexibility.

Question: Could you extrapolate present trends to the future?

Mr. Moore: I think the obvious extrapolation is telepathy. That in 20 years we will have the functional equivalent of telepathy. If you want to talk to anyone in the world you will be able to do so with minimal apparatus and you will be able to talk to them in the sense of not intruding upon them as telephones do now, but rather relaying messages from your computer to their computer and holding vast dialogues in the way computer mediated conferencing is being done today through terminals. This will be a very natural way of conducting our lives. It requires nothing in the way of technological breakthroughs! It merely wants the implementation of optical communication links worldwide.

I'm reluctant to quote science-fiction books, but there was one recently dealing with the first manned Mars mission and the fact that the United States came up with project management techniques of an advanced order through the use of computers — and completely outstripped the rest of the world. We gained an ascendant position in the world and earned the hatred of the rest of the world for our superiority, tried desperately to export technology and failed. The same thing can happen here. If we have a very tightly integrated community in terms of human resources and the rest of the world is excluded, we are going to have one helluva problem. I think this will happen. I think it will happen without any deliberate planning, and I think it is going to cause problems as great as those we have today between the haves and have-nots. But it will probably also generate the solution.

As to the twists that might occur, there are two wild speculations. One, that computers become intelligent or aware. Nobody has a clue as to what that means, but it is conceivable. Two, that we learn to record human personalities in machines. That merely requires the ability to detect and record the requisite volume of information which, as I say, I don't think is all that great.

Question: I wonder if you care to comment about the possibility of a new direction of architecture having to do with associative memory, which the brain seems to have.

Mr. Moore: I've studied these associative memories: there is such a chip on the market. I'd like to say they ought to be useful in implementing a FORTH computer but, I'm afraid I'm stuck in the mold. When I first encountered LISP I couldn't conceive of a language that didn't have a store operator. How could you do anything if you couldn't store your results somewhere? I can't conceive of a piece of data that is its own label, if you will. I am so used to thinking of data having addresses that my mind just doesn't grasp the possibilities if they don't. So, I'm afraid I can't say anything useful about associative memories.

(The evening concluded with an extended ovation.)

INFORMATION

Convention Transcriptions

Audio tape transcripts of the FORTH Convention held in San Francisco, October, 1979 are available. FIG member Jim Berkey recorded the technical sessions and the banquet speech by Charles Moore.

Our thanks to Jim for this significant contribution to FORTH history.

There are four tapes and they may be ordered at $4.00 each, postpaid from: Audio Village, PO Box 291, Bloomington, IA 47402.

Tape 1 -	Bill Ragsdale, Welcome and Introductions; Standard Teams Report; General Announcements; Case Statement Contest.
Tape 2 -	More Standards Team Reports; FORML; Applications of FORTH: Language Concepts.
Tape 3 -	Continuation of Language Concepts; Technical Workshop with Charles Moore.
Tape 4 -	Banquet Speech by Charles Moore, "FORTH, The Last Ten Years and The Next Two Weeks." (Printed in this issue of FORTH DIMENSIONS.)

Manuals

Here's an update of FORTH manuals currently in print and commercially availabe i.e. not part of licensed material.

...Using FORTH, 1979, Carolyn Rosenberg and Elizabeth Rather, 160 pp, $25.00. Order from FIG or FORTH, Inc., 2309 Pacific Coast Highway, Hermosa Beach, CA 90254.

...microFORTH^{TM} Primer, 2nd Edition, 1978, 60 pp plus glossary. Order from FORTH, Inc. or Miller Microcomputing Services, 61 Lake Shore Road, Natick, MA 01760.

...Kitt Peak Primer, Feb. 1979, Richard Stevens, 200 pp plus glossary. Order from FIG.

...Introduction to STOIC, Mar. 1978, Jonathon Sachs. Order from Stephen K. Burns, MIT, Room 20A-119, Cambridge, MA 02139 or Wink Seville (714) 452-0101.

...Cal Tech FORTH Manual, June 1978, M.S. Ewing. Send $6.00 to Cal Tech Bookstore, 1-51, California Institute of Technology, Pasadena, CA 91125.

...URTH Tutorial, University of Rochester. Send $20.00 to Software Ventures, 53 Arvine Heights, Rochester, NY 14611.

...FORTH Introduction Reprints, 38 pp, summary and collection of published and unpublished articles about FORTH. Send $10.00 to John S. James, P.O. Box 348, Berkeley, CA 94701.

Editor...

Know anymore? Send us your lists.

MEETING NOTICES

NORTHERN CALIFORNIA

FIG Monthly meetings are held the fourth Saturday of each month at the Special Events Room of the Liberty House department store in Hayward. Informal lunch at noon in the store restaurant, followed by the 1:00 p.m. meeting. Directions: Southland Shopping Center off Highway 17 at Winton Avenue in Hayward, CA. Third floor rear of the Liberty House. Dates: 3/22/80, 4/26/80, etc. All welcome.

THREE NEW GROUPS

FORTH Ottawa Group
c/o W. Mitchell
39 Rockfield Crescent
Nepeah, Ontario, K2E 5L6, CANADA

FORTH UK Group
c/o William H. Powell
16 Vantorts Road
Sawbridgeworth, Herts
CM21 9NB, ENGLAND

Mr. Edward J. Murray
Department of Computer Science
University of South Africa
P.O. Box 392
Pretoria 0001, Union of South Africa

Congratulations! Let us know what you're doing.

People who want to organize local groups can write to FIG for organizational aids and names of other members in your areas. Start a group!

MASSACHUSETTS

Dick Miller of Miller Microcomputer Services announces monthly meetings of the MMSFORTH Users Group. Meetings are on the third Wednesday of the month at 7:00 pm in Cochituate, Mass. Call Dick at (617) 653-6136 for the site and more information.

Incidentally, Dick offers a TRS-80 FORTH System and Z80 and 8080 assembler, data base manager and floating point math extensions. Enthusiastic comments have been received.

FORTH DAY AT NCC

Tuesday, May 20, is FORTH DAY at the PERSONAL COMPUTING FESTIVAL at the National Computer Conference which is being held at the DISNEYLAND HOTEL, ANAHEIM, CA on May 19-22. FIG members wishing to break out their portable soapbox are invited to participate. Speakers may address the audience on any FORTH related topic or may be part of a panel discussion. FIG will have a table in the exhibit area for distribution of FIG literature and individual discussions. Figgers with running FORTH systems with a good demonstration package may show off their efforts at the exhibition area. Members with a paper may present it and have it become a part of the proceedings. FIG has committed to this effort on a short deadline. Those wishing to participate should contact Jim Flornoy (415) 471-1762 immediately. There will also be a FIG meeting at the show.

FIFTH COMPUTER FAIRE

Look for FIG at the Fifth West Coast Computer Faire! We'll be at Booth 1028 in Brooks Hall, San Francisco (the lower exhibit area), on March 14-16. We'll also have a Users Group Meeting on Saturday. All FIG Publications will be available. This will be an ideal time to sign up for FORTH DIMENSIONS, Volume II.

The Booth and meeting site are through the generous consideration of Jim Warren and the Faire management. Thanks again!

FIG MEMBER PAPER

Joel Shprentz of the Software Farm, Reston, VA, will present a paper at the Fifth Computer Faire entitled "Solving the Shooting Stars Puzzle". Joel sells TRS-80 tiny-FORTH and features a FORTH solution to this network problem, using a modification of Dijkstra's algorithm for the shortest path problem.

OLD NEWS BUT INTERESTING

The West Coast Computer Faire is one of the best personal computing conferences and exhibitions. The Fourth Computer Faire was held in San Francisco on May 11-13, 1979 with more than 14,000 people attending.

FIG had a booth where it released the first of the public domain fig-FORTH implementations. FIG also sponsored a FORTH Users Meeting at which about 100 people attended to hear seven vendors talk and answer questions about FORTH systems.

A four hour technical session was also held on the topics of FORTH introduction, Extensibility, Standards, FIG implementation, poly-FORTH, multi-tasking in URTH, and ARPS. The conference proceedings are available for $14.78 from Computer Faire, 333 Swett Road, Woodside, CA 94062.

EUROPEAN MEETINGS

Promptly after the Fourth West Coast Computer Faire, several FIG members, including Kim Harris, Bill and Anne Ragsdale, John James and John Bumgarner, sallied forth (sic) to Amsterdam for the annual European FORTH User's Group Meeting, May 1979.

The meeting was held at the State University at Utrecht, under the hospitality of Dr. Hans Nieuwenhuizen. In addition to papers from FIG members that were also presented at the Computer Faire, there were status reports on standards efforts and reviews of European developments.

The next meeting is planned for Nancy, France, hosted by TECNA. Incidentally, TECNA is one of the most technically progressive firms in using FORTH as a base for a variety of operating systems and application dialects.

REVIEW BY KIM HARRIS

During the European User's Group Meeting at Utrecht, it was exciting to meet other FORTH users and learn of their activities. Attendees shared the results of projects including improved user security, an innovative file system, a microcoded FORTH interpreter, and the extension of FORTH to new languages. This last topic illustrated a powerful use of FORTH. A translator was written in FORTH which converted a completely different computer language into FORTH source which was subsequently compiled and interpreted. The new language was designed to be readable by bank managers in France (it resembles French COBOL). But this language can be easily controlled and expanded because the FORTH translator is small, structured and itself extensible.

We also met some University of Utrecht students who have done a lot with FORTH on an Apple II. They added floating point software to FORTH and wrote some excellent high resolution graphic words. We were shown a timing benchmark of 500 floating point additions using the same floating point software but called from FORTH or microsoft BASIC.

European FORTH users are very active in experiments for improving FORTH. FIG members were pleased to meet some of them and we all look forward to sharing past, present and future developments.

STANDARDS TEAM

Twenty seven people, comprising the 1979 Standards Team, met at Avalon, CA (Catalina Island), October 14-18, 1979. The scope of their work has significantly expanded! FORTH-77 and FORTH-78 were primarily standardized glossaries for common program expression. FORTH-79 has been extended to assure (hopefully) program portability. The glossary notation has been improved, definitions of terms added, English vocalization specified for symbols, rules of usage added and address space specified.

The intent of FORTH-79 is for a common form for publication and interchange of FORTH programs. The steps before release include a final Technical Referee review and a mail vote of the Standards Team. Watch FORTH DIMENSIONS for the availability announcement.

FORTH CONVENTION

On October 20, 1979, FIG sponsored a one day convention in San Francisco. This date allowed European members attending the Computer Faire to attend. In all there were 255 attendees with 110 at a dinner for Charles Moore's "Tenth Birthday of FORTH". Technical meetings were held and Jim Berkey has tape transcripts. $16.00 for 4 tapes. Audio Village, P. O. Box 291, Bloomington, IA 47402.

FORML MEETING

The first meeting of the FORTH Modification Laboratory (FORML) was held at Imperial College, London, January 8-10, 1980. Representatives of both the European FORTH Users Group and FIG attended.

The participants identified categories of limitations and approaches toward solutions. Detailed work is to be done by individuals with progress reports to the team. Topic areas included: Virtual FORTH Machine, concurrency, lanaguage, correctness, documentation, file system, I/O and programming methodology.

Contact Kim Harris or Jon Spencer through FIG for further information.