The Open Channel is exactly what the name implies: a forum for the free exchange of technical ideas. Try to hold your contributions to one page maximum in the final magazine format (about 1000 words).

We'll accept anything (short of libel or obscenity) so long as it's submitted by a member of the Computer Society. If it's really bizarre we may require you to get another member to cosponsor your item.

Send everything to Jim Haynes, Applied Sciences, UC Santa Cruz, CA 95064.

Heresy on high-level languages

R. N. Caffin

For many years now a barrage of criticism has been leveled at such "primitive and unstructured" languages as Fortran and Basic, and the virtues of "goto-less" programming have been loudly proclaimed. Supposedly sophisticated languages such as Algol, or new languages such as Pascal, have been extolled for their elegance and style. Typical of this theme was the panel discussion, reported in the April 1978 Computer,* in which "Pascal emerged as a serious contender to replace Basic" and was touted for its "'elegant' notational form" and its "completeness." Reality intruded slightly with the grudging concession that Basic would "probably also be supported." In a subsequent flight of fantasy, however, Fortran was declared "definitely out of serious consideration for most micros, now and in the foreseeable future."

I do not use Pascal, Algol, or APL at present for very good reasons: basically they use too much memory space and run too slowly (or not at all) on my PDP-11/03 (28K-word plus floppy disks). In terms of computing power, they offer me nothing that Fortran IV does not. Admittedly, Fortran may take longer to say something than APL, but I can't type APL on a standard keyboard very easily (and I can't read it when I've written it).

Similar criticisms of APL and Pascal for micros—too big and too slow—can be found in early 1978 issues of Byte. Perhaps too many academics and others in a position to speak and be heard are doing too much speaking and not enough hearing. Surely the workshop comment that Fortran is definitely out for most micros falls flat on its face in the light of the full-page advertisement on the back cover of the very same issue of Computer—a Cromemco advertisement for its Z-80 disk system featuring Fortran IV, Basic, Macro, and a linking loader. Were the workshop participants right and Cromemco wrong, or has Cromemco, which stands or falls on its sales to the final arbiters (buyers/users), done its homework? One could ask the same of other companies selling Fortran for microcomputers: companies such as Processor Technology (Sol Computer), Extensys (EX3000), Technical Design Labs (Xitan 280), and Microsoft.

Why does this disparity between theory and practice exist? I suggest that a reappraisal is necessary by the theoreticians: they have become too concerned with "elegance" and structured programming, and that it is automatically created in every program. This is close in concept to that of the total social security state, where you can't fail even if you try. The aim in Algol and Pascal is, in part, to produce a "readable" form of program. In consequence the operating system becomes larger (the ultimate example may be with APL). Yet, it is possible to produce an unbelievably unreadable program in any of those languages—and a perfectly readable program in Fortran, Basic, or even Macro.

I suggest that the approach is wrong and that we must go back to teaching programmers to think, to plan, and to document. I suggest that you can't produce a language for performing complex logical and mathematical operations that makes the operations automatically readable. I suggest that a preoccupation with eliminating all "gotos" from a language is futile. Indeed, I have come across flow diagrams where the use of some sort of "goto" was obviously optimal. Apropos of which, the continuing healthy state of branch, jump, and skip instructions in machine language and microcode demonstrates the true nature of program flow. In fact, one might suggest that these instructions are on the increase—compare for instance the PDP-11 with the PDP-8, and look at recent microprocessor instruction sets.

We have seen Algol and Basic evolve from their early beginnings. We have seen Fortran II become Fortran IV and Fortran IV become Fortran IV+ or Fortran V. We have even seen that outrageous pun, changing WATFOR to WATFIV. There is evolution, as well as revolution, in the realm of software. For producing a more powerful language, I prefer evolution. Here, some of the alternate languages can play a part. When a language has a particularly useful feature, that feature is usually copied. Virtual arrays are spreading into Fortran, for instance. (Perhaps a new ANSI standard on Fortran is overdue.)

The solution for more general work does not lie in fool-proof, very high-level, pseudo-English languages. We must accept, for the present at least, that programming requires thought. The
Comment on high-level heresy

I've been meaning for some time to write a commentary along the lines of Dr. Caffin's article. Just recently I wanted to show the computer to a 12-year old friend. We sat down together at a timesharing terminal. I decided it would be immoral to corrupt this young mind with Basic or Fortran, so we called up Algol. Thought it would be nice to start with a completely trivial program and then introduce more powerful concepts by successively modifying it. So, as in Example 1...

Then I thought to make a loop so the program would print the numbers from 1 through 10 and stop. A pedagogical problem arises here: structured programming suggests that I should immediately introduce the FOR loop; but I would rather use a GO TO to illustrate the simpler concept of looping, and then afterward introduce an IF test to terminate the loop. Remember, I'm not teaching a programming course; I'm merely trying to show in a step-by-step way the kinds of things that a computer can do (see Example 2).

I started anticipating my young friend's questions. Why do you have to say BEGIN? What's all that stuff about a FILE? Why the INTEGER statement? (Why do we build machines with separate representations for integers and reals?) Why the LABEL declaration and the label, instead of using the line number? Why the colon in front of the equals sign? What's all that junk in the WRITE statement? Why all the semicolons at the end of the lines? Why did you have to put a semicolon after the WRITE statement when it wasn't needed there before? Why don't you need a semicolon at the end of the IF statement? These are much the same questions I had asked years ago when I first encountered Algol, after having learned Basic first.

And I rebelled at having to type all those "unnecessary" keystrokes. I think now it all boils down to a "two cultures" situation. When I'm writing a program that is to be used repeatedly, to last for years, and to be maintained by somebody else, I write in Algol or Pascal or PL/I using lots of forethought, careful structuring, plenty of comments, and everything else I can do to turn out a professional product. But at other times I'm in the role of an engineer with a problem to solve, just once, as quickly as possible. Then I tend to prefer Basic or Fortran; I compose the program at the terminal while I enter it, and successively modify it until it does what I want done. In fact I may not know in advance how to write the program until I have tried out pieces of it—e.g., I may be trying to read a tape that someone has brought in, with no knowledge of its character code, block length, record format, or anything else. Maybe for situations like this I ought to write a well-structured utility that everyone could use, that would read the tape and list all its characteristics; but then this only happens about once a year and the tape format is always different, and it really doesn't take more than a few minutes to throw something together.

Probably most mature computer professionals have by now realized that the single all-purpose glorified and perfected programming language is a kind of Holy Grail. It's just that inventing new languages and arguing their relative merits is easier and more fun than solving real problems. Progress is being made; we are learning some things about programming quality and style; and the newer language features all serve legitimate purposes. What we can learn from Fortran and Basic is that there is a real need for languages which can be learned quickly and which are relatively easy to use, having a minimum of computer artifacts. Someone who has to hunt-and-peck on a terminal wants a programming language that requires a minimum of keystrokes. Let's continue to have more elegant languages for building our programming monuments; but let's not give up the languages that are more suitable for writing throwaway programs. And let's try to get more ease-of-use into our new languages.

---

Example 1:

**Algol Program**

```
BEGIN
  FILE TTY REMOTE (1,10):
  INTEGER I:
  I := 1:
  WRITE (TTY, <F4.0>,1)
END.
```

(74 keystrokes)

**Basic Equivalent**

```
LET I = 1
PRINT I
```

(21 keystrokes)

---

Example 2:

**Algol Program**

```
BEGIN
  FILE TTY REMOTE (1,10):
  INTEGER I;
  LABEL L;
  I := 1:
  WRITE (TTY, <F4.0>,1);
  I := I + 1;
  IF I <= 10 THEN GO TO L
  END;
```

(123 keystrokes)

**Basic Equivalent**

```
LET I = 1
200 PRINT I
LET I = I + 1
IF I <= 10 THEN 200
END
```

(54 keystrokes)

(In the programs, and in counting keystrokes, I have omitted the line numbers that are required by the timesharing system for both programs; except that line number 200 has to be shown in the Basic program for it to make sense.)

---

J.H.

March 1979