Computing prior to FORTRAN

by R. W. BEMER
Honeywell Information Systems
Phoenix, Arizona

ABSTRACT

The life of the programmer in pre-FORTRAN days is characterized in modern terminology, indicating how strongly FORTRAN has changed the programmer's condition and working habits.
The 25 years since the introduction of FORTRAN covers most of programming as we know it, certainly in volume of usage. To minimize any possible communications gap, I have chosen to describe how it was before that watershed event by means of some of the terminology and buzzwords of today:

1. Conferences and published papers
2. Computer science education
3. Stored programming
4. Structured programming
5. Program portability
6. Performance measurement
7. Communications and timesharing
8. Compilers
9. Data independence
10. Software piece parts
11. Software packages

The technical history of early programming languages has been covered by many authors (it became a popular subject), so I'll confine my contribution to more general areas.

CONFERENCES AND PUBLISHED PAPERS

Publication of software papers in pre-FORTRAN days was far less prolific than now. And it wasn’t yet “software.” Papers on software techniques prior to FORTRAN are given,2-42 as found (mostly) in Youden’s “Computer Literature Bibliography 1946 to 1963.” They’re given in best chronological order. To avoid duplication, sources with multiple papers are referenced separately, and the individual papers are given decimal notation.

Doing an analysis of the paper content of the early Joint Computer Conferences (the only continuing national meetings of that era) yields the counts shown in Table I. The last entry is the meeting at which FORTRAN was presented.

The summary pre-FORTRAN count is that of Table II.

<table>
<thead>
<tr>
<th>Year</th>
<th>JCC</th>
<th>Hardware</th>
<th>Applications</th>
<th>Software</th>
<th>H/A</th>
<th>H/S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1951</td>
<td>Eastern</td>
<td>16</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1952</td>
<td>Eastern</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1953</td>
<td>Western</td>
<td>8</td>
<td>11</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1953</td>
<td>Eastern</td>
<td>18</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1954</td>
<td>Western</td>
<td>8</td>
<td>14</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1954</td>
<td>Eastern</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1955</td>
<td>Western</td>
<td>6</td>
<td>16</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1955</td>
<td>Eastern</td>
<td>6</td>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1956</td>
<td>Western</td>
<td>18</td>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1956</td>
<td>Eastern</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1957</td>
<td>Western</td>
<td>28</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

COMPUTER SCIENCE EDUCATION

This was just starting, and in just a few schools. When you hired a programmer then, you didn’t ask about a degree in computer science; there weren’t any. IBM used its Programmer’s Aptitude Test as one screening method, and it worked somewhat, but people had a tendency to read more into it than was warranted.

A lot of us had our own pet questions, for we were taking them off the street. Magazine writers were curious about how one became a programmer. Dave Sayre had been a crystallographer, and Sid Noble and Art Bisguier were hired when I, an ex-movie set designer, advertised for chess players.

Although there may not have been enough collected theories to support specific degrees, the university people were all busy creating courses. The summer sessions at MIT and Michigan brought many practitioners together. Language processors were being built there and at Purdue, Pennsylvania, Carnegie Tech, Case, UCLA, and many others.

STORED PROGRAMMING

Programs have always been “stored programs.” The only difference is in where they were stored. In desk calculator days—in our heads. To program the IBM 601, one had to file notches in a phenolic strip, and they were stored in a box or hung on the machine. The IBM 604 was programmed by wires placed in plugboards, and often we stored them for reuse, if they were general enough. More often they were unwired for a new program (I wired about 700–800 60-step boards for the 604).

For the CPC the program was obviously in the cards. Bob Bosak and I devised a card system with 4 different tracks of 3-operand instructions, and so could feed a deck of cards continuously in a loop.
STRUCTURED PROGRAMMING

Structure in programs is generally ascribed to Wilkes, Wheeler, and Gill, in their book on programming for the EDSAC. The subroutine was the first element of structure, and was generally accepted by programmers, particularly those writing interpretive systems.

We had no DO UNTILs or semaphores at our disposal, but many programs had a structure that's all but forgotten now. It was called "optimum programming," a method of placing sequential instructions just right on a magnetic drum, so they would be ready to read just after the previous instruction was completed.

PROGRAM PORTABILITY

The first way used to reconcile the differences between two types of computer was to recode the problem. The second way was to write a programmed interpretive emulator for one machine in the code of the other. When this resulted in performance degradation of 100:1 up to 1000:1 it lost a certain amount of favor.

The third way was to use the source language of the interpreter and write another interpreter for the second machine. This had some success, because the degradation was often not very high (except for extremely dissimilar machines), and it could even run faster! Several of these were made. If machines of today's speeds had suddenly been introduced then, this may have become commonplace; compilers might have a different role. Even now, after thousands of compilers, interpreters still enjoy a considerable vogue. The fourth way, with different compilers, did not to my knowledge receive substantial usage until FORTRAN, and even there the portability path from a 704 to a 650 was difficult because the 650 supported fewer index registers.

PERFORMANCE MEASUREMENT

Although no hardware instrumentation was available for probes, much performance measurement did occur. It was vital because the computers were too slow for the amount of calculation waiting to be performed. While working at Marquardt, I was chastised one day by my boss, for not shaving. It was caused by being up since the previous morning running a trajectory simulation on the CPC. Under such circumstances, everyone wanted programs to run as fast as they could. That is why the program optimizers for drum machines (like SOAP) were so heavily used.

When the 701 superseded the CPC, the balance between user and machine changed. One man at the RAND Corporation took two years to program a problem that ran in two minutes. He experienced considerable culture shock.

There was competition everywhere to have the fastest program for a given task, quite often a mathematical subroutine. When published, those subroutines always had timing associated so the user could plan wisely. The situation was much the same as in the early days of microcomputers. jewel work was needed, and the domain was small enough to see and measure something. There was even competition between software and hardware people. The 705 engineers were shocked when a programmed divide ran faster than the hardware instruction—without firmware, they could not program a Newtonian iteration.

I suspect that FORTRAN itself had much to do with the temporary hibernation of performance evaluation. After programming in the other languages, it gave so much power because of the ease of use (and the efficiencies were incorporated for you in the compiler), that the number of user of computers could expand much more rapidly. It wasn't until operating systems came into heavy use that we rediscovered the need to prevent waste.

COMMUNICATIONS AND TIMESHARING

It wasn't Ethernet, but George Stibitz had tied into a relay computer by way of a Teletype—in 1940. SAGE was one of the first major projects to use direct inputs from communications lines. FORTRAN wasn't available when it began, and couldn't have been used for much of the job if it had, for it wasn't just a scientific problem.

Timesharing was just talk. The first time I find the word appearing is in a J. W. Forgie paper on the input-output system for the Lincoln TX-2 computer, concurrent with the 1957 FORTRAN paper. I proposed such usage in an article the next month; it was suggested that IBM should fire me, because that wasn't in line with their policy.

COMPILERS

Compilers existed before FORTRAN, but they were all rudimentary in comparison. Grace Hopper, chief pioneer of the concept, might have gone faster further if she had had the type of support given to Backus and his group. IT, A2 and A3 were true compilers, but they avoided interactions and optimization.

DATA INDEPENDENCE

This concept arose with the commercial compiler languages. Grace Hopper and company wrought the Data Division concept. Scientific languages all stuck to floating point, with integers for loop control.

Data structure was usually built into the program, and it didn't seem important, because hardly any interchange of programs took place between different computers. Even if that were possible one could not necessarily get the same answers due to different hardware characteristics.

SOFTWARE PIECE PARTS

Piece parts for software first came to attention at the first Software Engineering conference in 1968, proposed by Doug McIlroy. However, Bob Glass makes a convincing case that they were in existence before FORTRAN, certainly via the SHARE organization. Indeed they were necessary to counteract the inefficiencies of working without such compilers.
SOFTWARE PACKAGES

In the modern sense the software package did not exist, for today they cost money. Before FORTRAN it was unthinkable to sell software, although the packages did exist. They were traded or given away. Examples are several general CPC boards, plus the many 650 packages published in the IBM Technical Newsletter No. 10. 27

There is no doubt that packages existed. They were source programs for interpretation, not compiled source as today. A buzzword of the times was “abstraction.” Douglas Aircraft had a “matrix abstraction,” for example. 23 It manipulated matrices and performed combinatory functions. Ergo, if your problem could be expressed in matrix form, it could be solved. So it was urged that all problems be expressed this way, a not altogether natural way of use. But many of today’s software packages have similar contortional requirements upon the user.

Codes for nuclear computation also fell in the category of software packages, even if they were exchanged in machine language form. Hundreds of these codes were disseminated.

SUMMARY

I’m enjoying the developments of today, but my pleasure is a bit spoiled by the terrible waste in software development, and so much poor software. It’s tempting to recall Miniver Cheevey, who loved “the medieval grace of iron clothing.” Software before FORTRAN could be considered quite medieval, even primitive, but there were certain graces. From my starting in the computer field in early 1949, until FORTRAN arrived, I was either working too hard to see the Peter Principle in effect, or else it didn’t exist in such a virulent form. It was exciting to build software then. We had management support and trust for whatever we thought was possible. The number of levels of management was low, and the control tenuous. I reported to John Backus in FORTRAN days, but never felt the slightest pressure. I looked upon him as a friend, not a menace. So today we have better tools and knowledge, and theories of program correctness and such. I don’t think that they have added to the fun and excitement of Computing Prior To FORTRAN!

REFERENCES

7.3 Strachey, C. S. “Logical or non-mathematical programmes.” 46–49.
From the collection of the Computer History Museum (www.computerhistory.org)