Who Needs a Business-Oriented Language?

BY JIM MOORE

A business-oriented language is desperately needed these days. I wish someone would invent a common, business-oriented language. Maybe we could call it "COBOL." Not catchy enough? OK, how about Hot Pepper? We could have dialects like Jalepeno, Habanero and Cayenne. One feature that this hot new language *must* have is the ability to document itself. Additionally, it must be able to run just about anywhere. It should be easy to learn and shield programmers from a great deal of complexity while still allowing sophisticated interaction with the surrounding operating system. Oh yeah, I almost forgot! It must be designed to solve common, every day business problems.

MAINFRAME GOODIES

Joke over. Of course, I'm having a little fun with the "reinvent the wheel" crowd. "COBOL is dead!" they shout. Well, somebody better tell IBM this because they keep improving it. I have been writing about ISPF Version 4 for more than a year now. This month, I'm shifting gears to examine more programming stuff: COBOL for MVS, High Level Assembler, edit macros and other interesting, late 20th century mainframe goodies. I get a big kick out of being out on the bleeding edge of mainframe technology. Why? Simply because I've been doing mainframe work for almost 22 years and I've never seen better, more usable or more sophisticated software than that being released right now for OS/390. To be sure, I am a huge PC fan. I bought my first PC (a 48K Apple II+) in 1980. I've owned a PC in one form or another ever since. I currently have three; two for business, one for fun. I have seen the rise and fall of BASIC, the shift from C to C++, the first commercial GUI (the Lisa, from Apple circa 1983) and all of the other wild, woolly and way too fast changes that have occurred in the last 20 plus years. I couldn't run my business or write this column without a small computer.

I get a big kick out of being out on the bleeding edge of mainframe technology. Why? Simply because I've been doing mainframe work for almost 22 years and I've never seen better, more usable or more sophisticated software than that being released right now for OS/390.

However, and this is a very big however, I believe that it takes a lifetime to achieve virtuosity in anything. Other human undertakings such as painting, music and writing don't keep changing the rules every two to three years. To become a master of your craft, you must take the time to master your craft. If you aren't given the time to do so, well, that would be like Matisse having to relearn his painting technique every other year.

COBOL II REVIEW

The first programming language that I learned was BASIC. COBOL followed soon after. I wrote my first commercial COBOL program in 1976. The versions of MVS back then were all 24-bit and the COBOL standard was COBOL 74. In 1985, COBOL II (or COBOL 85) was introduced. I first used the COBOL II compiler in 1986. In 1989, IBM released Version 1 Release 3 of COBOL II. This was a watershed release that introduced many new language features. One of my favorites was reference modification. Also, many new compiler options were added, and several other options were renamed. For example, in older COBOL releases PMAP would produce an Assembler equivalent listing of the COBOL program. In V1R3 COBOL II, this option became LIST. Compile options such as SSRANGE and NUMPROC() were added.

I don't plan to spend a lot of time writing about COBOL II since it will not be supported after January 1, 2000. However, if you have a good handle on the features and power of COBOL II, you will have no problems with COBOL for MVS.

COBOL II GRAB BAG

In general, it has been my experience that many COBOL programmers have very little knowledge of how to exploit modern COBOL dialects. Unfortunately, this is the fault of the companies that employ these programmers. I know of many major businesses that never converted to COBOL II. Only now, as the century winds down are they converting to COBOL for MVS. They are using code-converting software to clean up the old COBOL 74 code just so that it can compile correctly. They are not training COBOL programmers in any of the new features. Newly graduated Computer Science majors don't even know COBOL. Some companies have done a real half-baked COBOL II conversion and are now blessed with a crazy-quilt of COBOL 74, COBOL II and COBOL for MVS.

So, to finish this month's column, what follows is just a rapid fire list of some of the

many sundry changes that have been introduced into COBOL since 1985.

1. The **WHEN-COMPILED** special register allows access to the date and time of program compilation at run-time. Display it at the start of job and save some debugging effort when you are invoking the incorrect version of the program.

2. The **LENGTH OF** special register: Want to have a program dynamically determine how many entries are in an OCCURS structure? Use COMPUTE MAX-ENTRIES = **LENGTH OF** BIG-OCCURS / **LENGTH OF** ONE-ROW (1).

3. The **ADDRESS OF** special register: Wait until you see what this can do! In future columns, I will demonstrate many memory addressing techniques using **ADDRESS OF** in conjunction with **USAGE IS POINTER** data items and the **SET ADDRESS** and **NULL** instructions.

4. The **SPECIAL-NAMES** clause allows you to define your own "figurative constants." Figurative constants in COBOLese are words such as SPACES, LOW-VALUES, QUOTE, etc. You can define a class of characters under **SPECIAL-NAMES** to create your own.

5. The word **TALLY** can be used as a generic, temporary counter whenever you need a fullword, 9(08) BINARY data item such as the following:

```
PERFORM VARYING TALLY FROM 1 BY 1
UNTIL TALLY > LENGTH OF BUFFER
```

6. **ALPHABETIC-UPPER** and **ALPHABETIC-LOWER** are valid class tests.

7. **NEGATIVE** and **POSITIVE** can be used in numeric compares. For example:

```
If Packed-Field NEGATIVE
```

8. **TRUE** and **FALSE** can be used to set 88 level values.

9. The word **BINARY** can be substituted for COMP(UTATIONAL). **PACKED-DECIMAL** can be used in place of COMP-3. COBOL is the only language that uses these oddball "COMP" terms. Start using **BINARY** and **PACKED-DECIMAL** like the rest of the world!

10. Learn about reference modification:

    - address character fields anywhere within their defined length
    - can be used in both the sending and receiving field
    - can be used in conjunction with OCCURS items
    - arguments can be literals, variables or expressions

    Example: MOVE 'MIDDLE' TO PRINT-LINE (65:6). Read as: Move the literal "MIDDLE" to position 65 of PRINT-LINE for a length of 6 positions.

11. Scope Terminators: **END-IF, END-READ, END-COMPUTE**, etc. I promise full coverage of these tricky little devils. They are great!

12. Great new sub-program linkage features that include: CALL by reference OR by content, the **IS INITIAL** clause of the **PROGRAM-ID**, and of course, the **CALL identifier-1** format.

13. Use hex literals in character fields. For example:

```
05  Bit-OR-Value          PIC X(01)
VALUE X'0C'.
```

14. Relative subscripts are allowed. Also, indexes and subscripts can be commingled.

15. Mixed-case source statements are allowed.

CONCLUSION

Boy, these only scratch the surface. I didn't even include many of the new verbs such as **INITIALIZE, EVALUATE**, the use of inline **PERFORMS, INSPECT...CONVERTING** and many others. De-editing moves, value clauses on OCCURS fields, new uses of relational operators — the list goes on. Anyway, the point is that every great new feature that was introduced in COBOL II is still available in COBOL for MVS. Even better, there are numerous new features that are only available in COBOL for MVS! Stay tuned for all the latest COBOL for MVS innovations. ts

NaSPA member Jim Moore is the president of Concentrated Logic Corporation, a Glendale Heights, Ill.-based software development firm specializing in TSO/ISPF/PDF and database design. He can be reached at conlogco@ix.netcom.com.

Tip of the Month

Be very careful with the NUMPROC(PFD) compile setting. PFD stands for preferred sign, or the C, D and F sign nibbles on all numeric fields. With preferred signage turned on at compile-time, the compiler will not insert any sign ORing instructions before it does a numeric compare. Also, all packed-decimal compares will use the CLC instruction. Why such care? If your data has mixed signs, you could compare unequal on fields that really were equal, such as packed fields that contain zeros. For example, one field might have an "F" sign and the other might have a "C." With NUMPROC(PFD), these fields *would not be equal!*