CUC

A DEFENSE OF COBOL FOR DOCUMENTATION

by
JOHN RYNES

John Rynes is an Analyst in the Corporate office. He has designed and implemented the new CUC payroll system in DOS COBOL for the IBM 360 and, over the past three years has written programs in 12 languages for nine different machines.

John joined CUC in 1964. He holds a B.S. degree in mathematics from the University of Wisconsin. John, his wife Beverly and their son, George live in Mt. Kisco.

Author's Note: I had been thinking about writing an article on the advantages of COBOL for documentation for some time. But being involved with writing programs in that language, I made that article a future project.

After reading Marty Hopkins' article in the last CU-BITS *I felt that the question of COBOL's suitability for documentation was pertinent and should now be discussed from the other side.*

No source program will ever provide all the documentation needed-not even a COBOL program can do that. All source programs are concerned with detail. Therefore, they can supply much of the required detail documentation but they can't provide equally essential general documentation.

When put to proper use, however, the inherent features of COBOL will yield a better documented source program than any other programming language. To prove this point, you need only compare some of its features with those of other programming languages.

One of the best documentation tools available in COBOL is the thirty character tag for field and paragraph names. The number of meaningful combinations of characters is virtually limitless. All assembly languages are more restrictive. Most of them allow only six to eight characters in names. As a result, COBOL names can be much more meaningful than assembly language names. Consider the recognition time required when looking through an assembly language listing and coming across these two tags:

```
CLCNYITX
EXMFEDTX
```

Now consider the corresponding two tags in COBOL:

```
CALCULATE-NEW-YORK-INCOME-TAX
EXEMPTIONS-FOR-FEDERAL-TAX
```

The meanings of the COBOL tags are obvious. You can even identify them as a paragraph name and a field name. Good COBOL programmers will take advantage of this feature to write meaningful tags for all fields and paragraphs in their programs.

Even in COBOL, however, it is possible for a programmer to write:

```
GARB (A,G,E)
```

When he means:

```
ZIP-CODE (POST-OFFICE, CITY, STATE)
```

And thereby lose all semblence of meaning and documentation. Perhaps these programmers don't like to contribute the additional effort required or perhaps they carry to their COBOL programming those habits acquired in writing assembly language programs. Perhaps they should not be COBOL programmers. At any rate, the tool is available and any well documented COBOL program makes use of it.

An equally valuable documentation feature in COBOL is the stratified structure of the Data Division, something not found in most other programming languages. By insisting on level indicators, each one less inclusive than the other, the CODASYL committee has removed from COBOL the necessity of indirect addressing and its documentation problems. Anyone who has tried to document an assembly language program filled with indirect addressing knows what I mean.

As an example of the power of documentation in stratified data, consider a program which combines meaningful tags, stratified data and another COBOL feature, qualification. For example a programmer has a twenty-five character field on a payroll master record which contains an employee's name. He wishes to move the first three characters of that name to the first three positions of a ten character code on the employee's check stub. In assembly language he would have to write somethir like:

```
MCW NAME1-22,NAME2-7
```

But in COBOL he can write:

```
MOVE FIRST-THREE CHARACTERS OF
PERSON-NAME IN PAYROLL-MASTER-RECORD
TO FIRST-THREE-CHARACTERS OF
CHECK-CODE IN CHECK-STUB.
```

Obviously the assembly language statement needs more explanation. The COBOL statement supplies its own documentation -- and what's more, needs no relative addressing.

The above examples of coding point out another advantage of COBOL as a documentation vehicle. Its verb set containing words like MOVE, ADD, DIVIDE and PERFORM is made of common words whose meanings are easy to grasp for anyone. In contrast, many other programming languages, especially assembly languages, have verb sets that can be understood only by men trained in the use of the language. Such mnemonic op codes as:

```
OCT
STC
CTC
```

Taken from the verb set of the EZCODE Assembly System for the RCA 501 or:

```
BXLE
SRDL
EDMK
```

Taken from the verb set of the Basic Assembly Language System for the IBM 360, clearly point out one advantage of the COBOL verb set.

Another advantage of COBOL's verb set is that a programmer need write only one line of program instruction rather than the five or ten lines he would need to code in an assembly language. As a result, anyone reading a COBOL listing for its documentation value, or any other reason, can understand what is happening by reading one line rather than many lines. For example, here's a comparison of the following instructions:

```
MCW  GRSPY,WORK # 8
S    DED,WORK
A    +5,WORK-1
LCA  @bbb,b$0.bb-@,NTPY
MCE  WORK1,NTPY
```

```
SUBTRACT DEDUCTIONS FROM GROSS-PAY GIVING
NET-PAY ROUNDED.
```

Although both examples could yield the same results, it is obviously easier to understand the COBOL sentence than the AUTOCODER statements.

Another good documentation feature of COBOL is its separation of the Data Division and the Procedure Division. Because of this separation there are no hidden data implications buried in the procedures as there can be in assembly languages. All conversion, scaling, truncation, removal of signs, editing and padding are implied by Data Division clauses. For any given field these clauses can be found by examining the field and its redefinitions in the Data Division. This feature might appear an inconvenience to an assembly language programmer trying to write in COBOL, but to the maintenance programmer it is an invaluable aid to documentation. He does not have to read the entire program to find out what is happening to one field.

Finally, the coding freedom given to COBOL programmers is an advantage not found in most programming languages. A good COBOL programmer will use this freedom to develop techniques that will help him write a well-documented program. One of these might be a unique numbering system for paragraph names. Another might be a policy of starting all verbs in column 12 and all continuation phases in column 16. Still another might be the use of EXIT as termination points for all PERFORM statements.

I will admit that there are some documentation tools available in other programming languages which are not available in COBOL. The asterisk used to indicate comments in most assembly languages does stand out more than the COBOL word NOTE. This feature would be nice to have, but it is not essential since any COBOL programmer knows he can make his entire first line, or entire comment, a solid set of asterisks if he chooses.

In COBOL there is no way to skip to a new page at compilation time. This feature would have advantages, but it is not essential since (in most COBOL compilers) you can skip a predetermined number of lines by inserting one blank card in the source deck for each line skip desired.

In summary, COBOL inherently contains many outstanding documentation features. The good COBOL programmer is in fact the only requirement for getting good documentation from COBOL. With that requirement, a COBOL listing will provide outstanding documentation because the inherent documentation features in COBOL tend to clerify meaning rather than obscure it.

CUC **COMPUTER USAGE COMPANY, INC.** 344 Main Street, Mount Kisco, N. Y. 10549