COBOL: Limitations and Deficiencies — Julian Stacey

Introduction

Cobol, (COmmon Business Orientated Language) was originally defined by Captain Grace Hopper of the US Navy in 1959 - historically Cobol was the second high level (non-assembler) language (the first being FORTRAN (FORmula TRANslation) for scientific applications). Cobol was a significant development in its day, but is now considered 23 years later in 1982, to have inherent deficiencies in many areas, despite the efforts of the ANSI Cobol Committee to enhance this product of the 'punched card and valve computer' era.

Detail

1
Cobol is a verbose lauguage, and in general does little effective work for a lot of source code, all of which has to be both initially typed in, and later read by maintenance programmers.

2
Cobol is an insufficiently structured language with a considerable number of eccentricities, the cumulative effect of which is to make programs harder to control during development, enhancement and maintenance/error correction phases of a project. Some eccentricities or structure problems are:-

2.1
Non Procedural : Cobol is not an algorithmic procedural language and thus does not lend itself easily to structured programming; attempts by some programmers to make Cobol 'structured' result in even more occurences of the (justly maligned) 'GOTO' statement, making the resultant code less structrued.

2.2
GOTOs : An excess number of 'GOTO's necessitates a high number of labels, programmers are thus forced to invent a large number of label names and usually come up with label names meaningless to future maintenance programmers and current design colleagues.

2.3
Constructs
Cobol compilers allow the following dangerous constructs :-

```
PERFORM A THROUGH B
        A: some statements ...
           ......
           GOTO  C
        B: ......
```

where B and C can be anywhere in the program.
Whilst most procedural languages will allow a programmer 'GOTO's if unavoidable by the programmer, they are often 'warned of' by the compiler, (particularly if it goes more than one level in or out), furthermore as GOTOs form such potentially dangerous constructs, they immediately leap to the programmer's eye when checking for errors, whereas in Cobol an erroreous GOTO is not easily seen, hidden amongst all the normal GOTO's; - this point becomes more significant after initial program development, when the project enters a maintenance and update/amendment phase.

Dangerous Cobol 'PERFORM' constructs such as:-

```
    PERFORM A THROUGH B VARYING X FROM 12 BY 6 UNTIL X GREATER THAN 3

                A           ....
                            GOTO  C
                            .........
                            GOTO  D
                B:          .........
```

can be much better replaced by a typical algorithmic language construct such as:-

```
                FOR X = 12 STEP 6 UNTIL X )| 73 CALL PROCEDURE A
```

2.4
Cobol has obscure 'magic' numbers of deep significance: data division declarations must be proceeded by a level number ranging from 0 to 80, some are special reserved numbers, some are not; those that are special are:-

```
     Level 0 & 1)  Special significance to some, but not all compilers
*              8)  Normal outermost statements
              77)  Automatic switches such as 'PASS VALUES ARE 8 THROUGH 10'
              88)  Constants
```

Numbers on an inner level must have a higher number than all outer levels. When a piece of source text is moved or copied (using an editor) to a new place in the program, it is thus often necessary to renumber many lines, further it is interesting to speculate what would happen if many levels of nesting were required, in excess of 69 (=77-8).

2.5
The comment delimiter in different program divisions varies between: 'COMMENT', 'REMARK', & 'NOTE' this lack of uniformity is unimportant, but is synonymous of other irregularities.

3
Cobol lacks a number of the facilities of newer language, facilities such as:-

3.1
There is no protective 'scoping'- ie. no local variable that can only be a accesed by its own subroutine/procedure and not the rest of the program.

3.2
No dynamic storage is available, thus:

*3.2.1
A second calling procedure cannot call a target procedure until the first calling procedure has finished, else the variable values will be overwritten. Cobol is thus both not re-entrant and not recursive.

3.2.2
Stacks rely on dynamic allocation, and are thus are not available, buffers are non-expandable beyond a finite point.

3.3
As no ASCII string passing is available, dynamic file or structure pointing becomes impossible, except by time consuming chaining through every possible element in a list of things possible to point at, or by jumping to an assembler routine written in another language: neither of which expedient would be necessary if a modern, quality language were used for a whole task from the the outset.

3.4
Data typing is limited:-

3.4.1
There are only a limited number of data types in Cobol:-

1 Alpha and Alpha Numeric ASCII fixed length character arrays.
2 Numerics (stored in ASCII)
3 Integers (Computational)

3.4.2
The following data types found in most scientific engineering languages are not available in Cobol:-

Signed and Unsigned Integers;
Short, Normal and Long Integers;
Floating Point and Double Floating Point;
Pointers to all other data types, including pointers to pointers.
Arrays of " " " " , " arrays of arrays.

3.4.3
Unlike Cobol newer languages like Pascal, Ada, and Chill allow a user to specify data types of their own, ie. in Pascal:-

```
TYPE workday = (monday, tuesday, wednesday, thursday, friday saturday, sunday)
VAR holiday, workday: day;
```

although Cobol can specify a list, or boundary limits of acceptable values that may be assigned to a variable, what Cobol cannot do is check that a variable of the right type is being allocated to a variable. (Thus Cobol could allow 'Granny Smith' as an assignment to an employee-name field, whereas it might be of type "apple-variety', and thus would be rejected at compile time by Pascal.

4
Application Areas
Cobol is appropriate for applications involving little real processing, but a lot of copying, some semi-automatic sorting, and precise output formatting, applications such as payroll cheque printing are typical. Though some salesmen claim many of their 'systems programs' are written in Cobol, in fact their definition of a 'systems program' is generally very loose, and often covers system environment software seen by a user; on closer investigation Cobol is found to be totally inappropriate for true kernel systems design programming, and is not used statements such as "if Cobol is used for a system supplier's own software it must be good enough for our applications" are thus erroneous

Whilst Cobol programmers are plentiful in commercial data processing environments, Cobol experience is less often found in engineering staff, who are more likely to have skills in procedural languages such as Algol, PL1, Pascal, C, PL/M & PL/Z, Coral, Chill, Ada and TAL. Cobol (and Basic) applications programmers are the least skilled, lowest paid programmers in computing, a disproportionate number of databases use Cobol only because first simple database requirements were in the DP world, but now databases are becoming more complex (relational instead of hierarchic etc.), and fast interactive response times are becoming required instead of batch usage, the ratio of Cobol databases is decreasing, with the advent of many database management systems written in modern languages.

* = check if allways valid