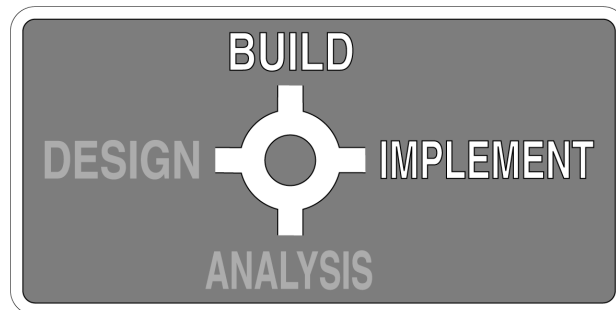


COBOL Programming



Helen Lahey
Bob Wicks

5.1 COBOL Programming Overview

This chapter discusses programming standards in relation to general formatting, naming conventions and the requirements for each division of a COBOL program.

COBOL has been widely adopted for programming large business and government applications since it was conceived in 1959. A committee of the Conference of Data Systems Languages developed the COmmon Business-Oriented Language to be a machine-independent and readable language to overcome the problem of the multiple manufacturer-dependant programming languages available at the time. To a large extent it has been standardised by the American National Standards Institute, although it has not achieved the complete machine-independence desired by its designers. COBOL is more English-oriented than other languages and it handles input and output operations efficiently, however, its mathematical capabilities are limited. Modern development tools have enabled COBOL to be extended into the client/server and Internet environments.

Many people question whether COBOL is still relevant and worthy of inclusion in the University's teaching program. Despite the enduring cry of COBOL's imminent demise, extensive commercial applications need maintenance and enhancement. Additionally, based on the number of licenses sold in 1998, COBOL still had almost ten percent of market share of computer languages (*Computerworld*, 29 Mar 1999, p. 6). The Department of Information Systems' curriculum design committee reviews this issue on a regular basis and accepts the advice of employers. As long as the Department is convinced that having COBOL skills advantages our students, we will continue to include this computer language in our curriculum.

The standards presented in this handbook have been designed to develop sound programming practices and improve the readability of COBOL programs. These standards have evolved from the diverse experience of the various staff members from the Department of Information Systems who have taught COBOL programming over the years.

Information Technology Services (ITS) here at USQ have kindly allowed us to include a number of pages from their MIS 1.0 Standards Handbook in Appendix C. These pages provide an example of the COBOL standards that are implemented within ITS and have been included to demonstrate standards that are in use in industry. Notice that these specialised standards are not exactly the same as the standards used within the Department of Information Systems. However, there should be no difficulty adapting to specific requirements once the habit of using standards has been adopted.

5.2 General Format

Standardised formatting of source code enhances readability, thus facilitating the development and maintenance of programs.

Visual separation must be used between major segments of code to enhance readability. A blank line must precede divisions and paragraphs.

Indentation can be used, rather than punctuation such as commas and semicolons. Punctuation is often a source of confusion and error and is not necessary.

Write only one sentence per line. Multiple clauses in a sentence must be written on separate lines and indented after the first clause.

Comments must only appear at the beginning of the program, before paragraphs, and before 01 levels. Do not embed comments within logically contiguous code. Comments must only describe the code that follows and be sufficient for program maintenance. Specific, descriptive and meaningful data and paragraph names contribute to good documentation.

5.3 Naming Standards

5.3.1 File Names

In order to standardise file names, we normally use the format XXXXnn where XXXX is a file-type and nn is the program number. Typical file types include:

- SKEL skeleton
- PROG program
- ASSN assignment
- DATA data
- MAST master
- TRAN transaction
- PRNT printed output
- SUB subprogram

For example, problem number 5 may consist of the files PROG05, DATA05 and PRNT05.

5.3.2 Paragraph Names

The main controlling paragraph is called the CONTROL-MODULE. All other paragraph names must have a prefix, a verb and a noun, for example, BAC-CALCULATE-PAYMENT. The prefix (BAC-) indicates the level of this paragraph and the paragraph from which it was called (BA-...). The verb and noun must be specific and descriptive of the processing within the paragraph. The prefix hierarchy diagram at the end of this section represents the prefixing structure in a graphical form.

Adding an adjective is optional and often improves readability. For example, B-ACCEPT-VALID-PROFILE-DATA implies that this paragraph accepts and validates profile data from keyboard input. The single prefix indicates that it has been called from the CONTROL-MODULE.

If a paragraph is called from more than one other paragraph, then its prefix must include Z. For example, if a paragraph is called from BA and also from BBA, then the prefix must be BZ. A paragraph called from two different parts of the program structure, such as BBB and CA, must have a Z prefix.

Each module of a hierarchy or structure chart is coded as either a paragraph or a sub-program. A typical hierarchy chart of a program, using COBOL paragraphs to represent the modules, could be represented by the following module prefixes.



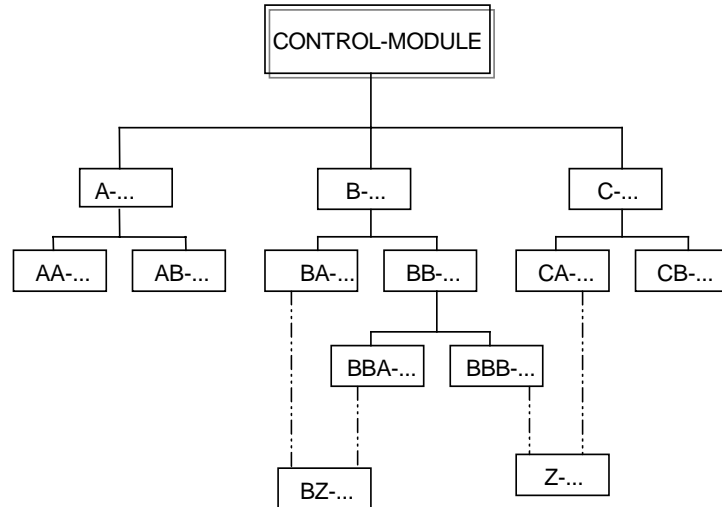
```
CONTROL-MODULE .
  PERFORM A- ...
  PERFORM B-ACCEPT-VALID-PROFILE-DATA .
  PERFORM C- ...
  ...
  STOP RUN .

...

B-ACCEPT-VALID-PROFILE-DATA .
  PERFORM BA-ACCEPT-PROFILE-DATA
  UNTIL ...
  ...

BA-ACCEPT-PROFILE-DATA .
  ...
  PERFORM BZ-VALIDATE-DATE-AND-TIME .
  ...
```

The BZ prefix on the paragraph BZ-VALIDATE-DATE-AND-TIME indicates that this module is re-called, from within another paragraph that has a prefix beginning with the letter B. The prefix hierarchy diagram on the next page represents the above modules and expands this program structure.



The reader of the program is aware, when this convention is used, where a particular module belongs in the module hierarchy. For example, BBA-... is a third level module and was called by module BB-... which in turn was called by B-..., and so on.

Alternative paragraph prefixing standards may be more useful in larger, more complex programs. Many COBOL programmers favour a simple ascending numeric system, with no hierarchical links between paragraphs. The sole purpose of the prefix in this case is to facilitate the location of a paragraph within the listing. Locating a paragraph in a large program listing can be quite frustrating when prefixing of this type is not used. The prefixing standards in use within ITS, (see their page 16 of the MIS 1.0 Standards Handbook in Appendix C) make use of a combination alphanumeric prefix. Nevertheless, the prefixing standards described previously in this section are mandatory for students studying unit 75123 COBOL Programming.

5.3.3 Data Names

All data items must have a prefix. The following prefixes are to be used:

- Input fields WI-
- Output fields WO-
- Input/Output fields WIO-
- Sort fields WS-
- Error messages WE-
- Flags WF-
- Table items WT-
- Linkage items LS-
- Working variables WW-

Data names within the file section must be prefixed similar to the record name. To distinguish between the file and the record, add the suffix, FILE, to all file names and RECORD to record names.



```
FD PAYROLL-FILE.  
01 WI-PAYROLL-RECORD.  
    03 WI-EMPLOYEE-ID      ...  
    03 WI-EMPLOYEE-LASTNAME ...  
    03 WI-OVERTIME-HOURS   ...  
    ...
```

Fields that occur in two or more different records must be differentiated from one another by the prefix only. The rest of the data name must be the same. This eliminates the need to develop different data names for the same data. For example, the program using the payroll file shown above is likely to have a WO-EMPLOYEE-ID in the output record.

An example of an alternative naming convention can be found on page 8 of the MIS 1.0 Standards Handbook in appendix C.

Always use descriptive names. Data names should be composed of two or more descriptive words. The field name WI-HOURS must indicate the type of hours, such as overtime, class, or regular hours. For example, use WI-OVERTIME-HOURS for the field name, rather than WI-HOURS.

5.3.4 Abbreviations

To allow shorter data names without loss of clarity, a programmer must use consistent abbreviations. Abbreviations must be clear, acceptable, meaningful and chosen very carefully, as poor abbreviations will cause confusion. Abbreviations must have commonly understood meanings. The following list provides a number of commonly used business terms:

ADDR	Address	HDG	Heading
AMT	Amount	HR	Hour
CALC	Calculate	IND	Index
CHAR	Character	MAST	Master
COL	Column	NO	Number
CUST	Customer	QTY	Quantity
DEL	Delete	REC	Record
DEPT	Department	SUB	Subscript
DESC	Description	TRAN	Transaction
EMP	Employee	YR	Year
EOF	End Of File	YTD	Year To Date
ERR	Error		

5.3.5 Procedural Verbs

A set of commonly understood verb definitions will help program clarity. The following list has a reasonable degree of acceptance in the industry.

Input	Meaning
ACCEPT	Accept data from the keyboard
READ	Read data from a file
RETURN	Obtain a sorted record from the sort file.
Output	Meaning
DELETE	Remove record from file
DISPLAY	Display data on screen
PRINT	Prepare a line for printing
RELEASE	Pass an unsorted record to the sort file
REWRITE	Overwrite a record on an existing file
WRITE	Write to a file
Processing	Meaning
CALL	Call a subroutine
CLEAR	Clear the screen
FORMAT	Prepare an output field
INITIALIZE	Set initial values to variables
TALLY	Count the number of items
UPDATE	Modify records in a master file
VALIDATE	Check that a field is valid.

5.4 Identification Division

The identification division must contain PROGRAM-ID, AUTHOR and a brief comment describing the purpose of the program. Other statements are not required.



The standards described in this chapter are demonstrated in the numerous examples and solutions provided in the study materials for the COBOL Programming unit.

5.5 Environment Division

The environment division contains the INPUT-OUTPUT SECTION and FILE CONTROL only. If the program does not use files, then the environment division is empty, although the division heading is still required.

5.6 Data Division

The optional RECORD CONTAINS and DATA RECORD clauses of the FD must be omitted as they add little to program documentation.

Each data-item subdivision should be indented four spaces. When this consumes too much space on the coding line, indent two spaces.

The gap level-numbers should be increments of two starting with level 01 (for example, 01, 03, 05, 07, etc.). This allows easy insertion of new levels if necessary.

Vertically align all picture clauses (PIC). All clauses for elementary data items, except USAGE and VALUE, must appear on different lines and be aligned.

The USAGE and VALUE clauses must remain on the same line as the PIC clause, unless it is too long for the line. If the VALUE clause is greater than 10 characters in length, then begin the clause on the next line. Keep the reserved word, VALUE, on the same line as the PIC clause.

Numeric integer fields, not involved in calculations or subject to numeric editing, must be described as alphanumeric rather than numeric. This will be more efficient due to the requirements of arithmetic sign handling.

All field lengths, except for single digit fields, must be expressed in the form 9(3) rather than 999. Edited fields, however, are usually described without parentheses. For example, PIC \$\$\$,\$\$9.99CR is much easier to read than PIC \$(3),\$ (2)9.9(2)CR.

The optional word IS must not be used in the VALUE clause.

Since the DISPLAY clause is the default value for the usage clause, this statement must be omitted.

Output records must be defined in the WORKING-STORAGE SECTION, rather than the FILE SECTION. Normally a number of different output lines, such as heading, detail, sub-total, and total, will be required.

5.7 Procedure Division

The CONTROL-MODULE is the only paragraph which should contain STOP RUN or EXIT PROGRAM commands. Each program module must only have one entry point and only one exit point.

Use indentation consistently to reflect the logical structure.

Subordinate clauses to PROCEDURE DIVISION statements must be indented four columns to the right of the start of the statement, with successive clauses aligned.

Each new command must commence on a separate line in column 12. Where a statement exceeds one line, it must be indented from column 12 on successive lines. Words must not be broken over lines.

Open and close all files with single OPEN and CLOSE statements followed by a file list with each file name on a separate line and vertically aligned.

The READ statement must be written with the AT END phrase on a separate line and indented. The imperative command(s) that are controlled by the AT END clause must be written on separate lines and indented under the AT END clause.

The WRITE statement must be written with the AFTER ADVANCING phrase on a separate line and indented.

Any MOVE statement that will not fit on one coding line must be written with the reserved word TO and the receiving field(s) on a separate line and indented.

Any MOVE statement that has multiple receiving fields must be written with each receiving field, after the first one, on a line of its own. Align them vertically under the first receiving field name.

Indent statements following an IF or ELSE clause. Each ELSE must line up with the corresponding IF.

With COMPUTE statements, use parentheses to control the sequence of arithmetic operations, rather than relying on the normal sequence of operations. Parentheses make the expression easier to read and understand and help ensure that the correct computation is made.

Normally, the PERFORM statement is used only to perform a paragraph. However, the in-line PERFORM may be used if the paragraph contains only a simple statement, such as changing a flag or setting an index.

Always use a full stop at the end of a sentence unless there is a good reason not to, for example, within an IF statement.



Do not use the GO TO statement or the GO TO DEPENDING ON statement.

PERFORM THRU is not to be used.

In subprograms, COMMON STORAGE must not be used.

5.8 References and Suggested Readings



- Brandel, M. 1999, 'The Creation of Cobol', *Computerworld*, Mar 15.
- Levite B.L. 1995, *Structured COBOL Programming: Interactive and Batch Processing*, Boyd & Fraser Publishing Company, Massachusetts.
- Orenstein, D. 1999, 'Development Tools Keep Cobol Current', *Computerworld*, Jan 18, p. 14.
- Stern, N. & Stern, R.A. 2000, *Structured COBOL Programming For the Year 2000 and Beyond*, John Wiley and Sons, Inc., New York.