A Curriculum for a University Course in Advanced COBOL

John C. Molluzzo
Division of Mathematics and Science
St. John's University
Staten Island, N.Y. 10301
(212) 447-4343

Introduction

Recently many colleges and universities have either begun or expanded their offerings in computer science and data processing. Many of these institutions have recognized that the COBOL langugae should be an integral part of their programs. This recognition stems from the widespread use of COBOL in the business community. While many texts on COBOL are available representing several approaches to the language, there are few books that can be used as a basis for a second, more advanced course in COBOL. There is, therefore, a problem for the developers of such an advanced course - what should be included, and in what order?

This paper presents a curriculum for a university course in advanced COBOL. The curriculum is based on a course (CUS 36 - Advanced Commercial Computing) developed by the author at the Staten Island Campus of St. John's University.

The curriculum we present is for a university course as opposed to a two-year college or continuing education course. It emphasizes understanding COBOL and producing efficient code, as well as how COBOL interfaces with its operating environment. We, therefore, cover programming techniques, data and file structures, operating system concepts, and the relation of COBOL statements to the computer's architecture. The aim of the course is not just to produce COBOL coders, but rather individuals with a detailed and profound knowledge of COBOL and its relationship to the operating system.

The only absolute prerequisite for this course is, naturally, an introductory COBOL course. Students, therefore, should be familiar with the elementary data movement and arithmetic verbs, output data editing, elementary control statements, serial file processing, tables, and table searches. It is helpful, although not necessary, for students to have taken courses in computer architecture and assembler language, operating systems, and finite or computer mathematics.

It is important to have an overall programming philosophy for this course. We are strict adherents to top-down, structured programming techniques, and insist all projects be written using them. See, for example, Yourdon (1975), Van Tassel (1978), Chmura and Ledgard (1976), or Shelly and Cashman (1978). When developing solutions to illustrative programming problems we use several program design strategies. Included are discussions of data flow-oriented design, see DeMarco (1978), Yourdon and Constantine (1978), or Weinberg (1980), and data structure-oriented design, see Jackson (1975), Warnier (1976), Pressman (1982), or Molluzzo (1981). Data flow diagrams or data structure diagrams can be part of the required documentation for all programming projects. This aspect of the course provides excellent preparation for a course in systems analysis and design.

In the next section we present an outline of the topics covered in the course. We expand and comment on the outline more fully in the last section.

Course Outline

Unit I. Review and Extensions of Elementary COBOL

- a. Review of elementary COBOL
- b. Arithmetic efficiencies
  1. USAGE clauses
  2. Mixing numeric data types
  3. Decimal point alignment
  4. When to use COMPUTE
  5. When to use indexes rather than subscripts
- c. Input data editing
  1. Types of data editing
  2. Mod-11 check

d. Conditional statements

1. Compound conditions
2. Implied subjects and objects

Unit II. The Report Writer

a. Report structure

b. Report definition

1. RD entry; CONTROL and PAGE clauses
2. Report group descriptions; TYPE clause, group and elementary entry descriptions
3. Control breaks and sum counters

c. Report processing; INITIATE, GENERATE, TERMINATE

Unit III. Sequential Files

a. Operating and file system interfaces

1. SELECT and RESERVE clauses
2. I-O-CONTROL paragraph
3. FD entry; BLOCK and RECORD CONTAINS clauses
4. EXTEND opening mode

b. Sorting and merging

1. ASCII and EBCDIC collating sequences
2. Sort and merge algorithms
3. Internal and external sorts
4. SELECT clause for work files
5. SD entry
6. SORT and MERGE verbs with and without input and output procedures; RELEASE and RETURN verbs
7. Work file restrictions (if any)

c. Application

1. Multiple input files
2. Sequential file updating

Unit IV. Random Access Files

a. Direct access devices

b. Hashing functions

c. File organizations

1. Indexed files
2. Direct files
3. Relative files

d. Processing random access files

1. ORGANIZATION and ACCESS MODE clauses; RECORD, NOMINAL, and ACTUAL keys; I-O-CONTROL paragraph
2. READ, WRITE, REWRITE, DELETE AND START

e. Comparison of file organizations

1. When each should be used
2. Relative speeds of different file organizations in sequential and random processing

Unit V. Additional Topics

a. Subprograms

1. Subprogram Linkage
2. LINKAGE SECTION
3. CALL, CANCEL, and EXIT PROGRAM verbs

b. Library facilities

1. Sharing files among programs
2. COPY verb

c. String manipulation

1. INSPECT
2. STRING
3. UNSTRING

d. Debugging

1. Debugging features of COBOL
2. Status keys and DECLARATIVES
3. ABEND debugging and core dumps
4. Interactive debugging

Comments and Suggestions

The following comments are keyed to the major subdivisions of the topical outline of the previous section.

Unit I.

We have found one of the better ways to review elementary COBOL is to discuss, line-by-line, a complete program. The program should include as many elementary features of COBOL as possible, such as data movement, editing, conditional statements and flow of control, table processing and searching, and single level control breaks.

The topics in I.b, and I.c are not usually discussed in an elementary course. They, nevertheless, are simple extensions of elementary topics, can be covered quickly, and can be interleaved with the review. We give a detailed explanation of USAGE clauses in terms of the resulting internal bit (or byte) representations and when to use each type of USAGE. We also discuss how proper decimal point alignment reduces the number of object code statements produced by the compiler. It is important to stress these topics because their proper implementation can significantly reduce the execution time of a program with virtually no effort on the part of the programmer. See Grauer and Crawford (1978 and 1979) and Olsen and Price (1982).

In most elementary COBOL courses, it is assumed that data input to a program is "clean", that is the data contain no errors. Fields that are supposed to contain alphabetic data do so, numeric fields do contain numbers, etc. This is not, unfortunately, the case in the "real world". Input files contain errors. If an input file is not already cleaned up by another program, any program processing that file should check the validity of the data it processes. We, therefore, include in I.c a complete discussion of input data editing.

Input data editing includes checking for a blank field, for numeric data (the class test), for specific values (using condition names), for reasonableness of numeric data, and verifying a total against a check figure. We also discuss the mod-11 check for transportation and transcription errors. See Shelly and Cashman (1978). The mod-11 check is a good application of arrays and the use of DIVIDE with the REMAINDER option.

The IF...ELSE... construction and nested IF's are usually treated in elementary COBOL. We include, however, in I.d the frequently omitted topics of compound conditions, the rules for their evaluation, and the rules for implied subjects and relations. See IBM (GC28-6396-5). After covering this topic some of the better students are tempted to write compound conditions as compactly as possible. The instructor should, however, emphasize the need for clarity rather than brevity.

Unit II.

The Report Writer is one of the most useful but, nevertheless, neglected features of COBOL. There is not much literature available on the Report Writer. It is, however, worth the effort to seek it out. See Chai and Chai (1976), Lyons (1980), Grauer and Crawford (1979), and Phillipakis and Kazmier (1982). We discuss multiple level control breaks, rolling sums forward, cross footing, and the use of the Report Writer in both detail and summary reporting. If time permits, and if the code can be made available, we show how the Report Writer expands the REPORT SECTION into executable code.

All reports produced by course projects must be generated by the Report Writer.

At this point in the course we usually assign the first major project. Programming assignment 1, Chapter 2, in Shelly and Cashman (1978) can be easily expanded to include almost all topics in Units I and II.

Unit III.

Units III and IV, sequential and random access files, form the core of our course. We begin with a general discussion of file storage and access methods and take care to distinguish between them. We cover blocked records, interblock gaps, multiple buffering, I/O interrupts, and the operating system's access methods. These topics are important for efficient coding of the ENVIORNMENT DIVISION. It is appropriate at this point to discuss the SELECT clause and how it provides an interface between the program and the file system. It may be necessary, depending on the hardware and operating system, to give a detailed explanation of the file system. At St. John's, for example, we operate in the Honeywell MULTICS environment. The MULTICS I/O system is device independent and the MULTICS file system is hierarchically constructed. This has a great effect on the form, interpretation, and use of the SELECT clause.

It is instructive to give examples of SELECT clauses in COBOL programs implemented on several systems to stress their system dependent nature.

We begin topic III.b, sorting and merging, with a discussion of sort and merge algorithms such as the selection, bubble, heap, quick and merge sorts. See Tremblay and Bunt (1979). We also give examples of the use of multiple sort keys, and how the choice of a collating sequence can affect the outcome of a sort.

To use COBOL's SORT and MERGE correctly, especially when using input and output procedures, students must understand how these verbs work. We, therefore, give a careful explanation of which files are opened and closed, by whom and when during execution of SORT and MERGE. See Olsen and Price (1982). We also include how the SORT/MERGE utility works. See Davis and Fisher (1979).

Finally, we discuss the SORT/MERGE work files and their allocation problems, if any. (This topic is both hardware and software dependent.)

Perhaps the most important type of sequential file processing program is the sequential file update. Students should have a thorough understanding of the problem and know how to construct a solution. Therefore, in topic III.c we discuss several sequential file update algorithms. See McCracken (1976), Shelly and Cashman (1978), Dijkstra (1976), Dwyer (1981), and Inglis (1981). Most of these algorithms rely heavily on the structure of the files being processed. This topic can, therefore, be used quite effectively to illustrate data structure-oriented program development techniques such as those of Jackson (1975) and Warnier (1976). The file update problem is also an excellent topic for term papers, should the instructor decide to assign one.

At this point, our second project - sequentially updating a master file - is usually assigned.

Unit IV.

Because of their more complicated organization, random access files, Unit IV, are usually more difficult for students to understand than are sequential files. Random access files, however, are really not more difficult to use than sequential files. Without getting tied to a particular hardware device, we discuss the general characteristics of direct access devices including techniques of direct addressing. The discussion of hashing functions is made easier if the students have the necessary mathematical prerequisites. See Molluzzo and Buckley. If not, they can be covered in a short time.

The most important random file organization is indexed organization. We, therefore, discuss in detail at least one implementation of indexed files. See Tremblay and Sorensen (1976) for a discussion of IBM ISAM and Control Data indexed file organizations. We carefully distinguish the ENVIRONMENT DIVISION statements necessary for each type of file organization and which processing verbs can be used with each access made. Great care should be taken here. We have found that students easily confuse the different kinds of keys. Finally, we discuss the use of each organization in both sequential and direct file processing. Note that if the course is IBM based, the difference between ISAM and VSAM files should be discussed. See Grauer and Crawford (1979).

Unit V.

The topics in Unit V are independent of the rest of the course material and can be introduced at anytime during the course. We usually treat topics V.a, V.b, and V.c after covering Units I, II, III, and IV. Topics V.b and V.c, however, can effectively be covered at the beginning of the course, for example during or immediately after Unit I.

How and when the instructor covers debugging depends on the background of the students and the environment in which they develop programs. Almost all COBOL compilers have debugging facilities. For example, IBM's READY TRACE and EXHIBIT statements, the DEBUG card, etc. At the very least, these features should be covered. If available in the version of COBOL being used, the instructor should discuss the use of status keys and DECLARATIVES to process I/O and Report Writer errors. The use of DECLARATIVES with the COBOL debugging feature should also be included. See Phillipakis and Kazmier (1982).

If the students have taken a course in assembler language, the instructor should discuss ABEND debugging using core dumps. See Grauer and Crawford (1979) and Rindfleisch (1976).

Many business and academic computer installations now develop COBOL programs interactively rather tha in a batch environment. Frequently, such interactive systems include a package such as IBM's TSO interactive COBOL debugger or MULTICS' PROBE. If available, an interactive debugger should be introduced early in the course and its use encouraged in program development.

Nearly all sizeable data processing systems consist of many programs that must communicate and work together to produce the end product of the system. It is important for students to understand how programs communicate and to gain experience in designing and coding at least a small system of programs. Topics V.a and V.b are, therefore, extremely important for student in order to obtain an understanding of how "real world" systems work.

To fully explain subprogram linkage, we include a discussion of the housekeeping responsibilities of both calling and called programs. See Phillipakis and Kazmier (1982). Depending on the preparation of the class, this topic can be covered at the assembler languae level, see Yarmish and Yarmish [1979], and can include calls to programs in languages other than COBOL.

We discuss the use of COPY to reduce coding in systems of programs, and the sharing of files at both the program and device levels. Since students will write in the final project several large programs that will communicate with each other, it is important that students write well-structured modular programs. Thus, we include careful definitions of module cohesion and independence with appropriate examples. See Yourdon (1975).

We now assign our final project. The project involves writing a system of programs driven by a mainline program. Calls are made to subprograms, which share files of several types of organization. Each subprogram in the system is of substantial length. We, therefore, assign this project to teams of four or five students, each student responsible for a subprogram. A small library of data and file descriptions is created for use in the subprograms via the COPY verb. Structured walkthroughs are conducted of both program design and code. See Shelly and Cashman (1978).

Conclusion

It is difficult to judge the success of any course. Student evaluations are frequently based on their attitudes toward the instructor, the text book, and many factors in the classroom environment. The validity of such evaluations is, therefore, suspect. Perhaps the most accurate evaluation of the effectiveness of a course is one made by an independent third party who can evaluate what students have learned. This is particularly true of a professionally oriented course such as the one we have described. The performance of students on examinations administered to them by prospective employers is, therefore, a good indication of the quality of our curriculum.

Employers are, understandably, reluctant to release information on results of their interviews are tests. Hard statistical data was, therefore, not available for analysis. However, based on informal interviews with graduates who took our course and who are now employed as COBOL programmers or systems analysts of COBOL-based systems, we can judge our course a success. Most of these former students were hired or offered a job by the first company that gave them a written COBOL examination or an oral COBOL technical interview. A few graduates, in fact, scored high enough on these examinations to be hired as programmer/analysts rather than programmer trainees.

References

1. Chai, W.A. and Chai, H.W. (1976). Programming Standard COBOL, Academic Press, N.Y.

2. Chmura, L.J. and Ledgard, H.F. (1976). COBOL with Style, Hayden Book Co., Rochelle Park, N.J.

3. Davis, W.S. and Fisher, R.H. (1979). COBOL: An Introduction to Structured Logic and Modular Program Design, Addison-Wesley, Reading, Mass.

4. DeMarco, T. (1978). Structure Analysis and System Specification, Yourdon, Inc., N.Y.

5. Dijkstra, E.W. (1976). A Discipline of Programming, Prentice-Hall, Englewood Cliffs, N.J.

6. Dwyer, B. (1981). One More Time - How to Update a Master File, CACM, v.24, no. 1, 3-8.

7. Grauer, R.T. (1981). A COBOL Book, Prentice-Hall, Englewood Cliffs, N.J.

8. Grauer, R.T. and Crawford, M.A. (1978). COBOL: A Progmatic Approach, Prentice-Hall, Englewood Cliffs, N.J.

9. Grauer, R.T. and Crawford, M.A. (1979). The COBOL Environment, Prentice-Hall, Englewood Cliffs, N.J.

10. IBM, OS Full American National Standard COBOL, GC28-6396-5, IBM Corporation.

11. Inglis, J. (1981). Updating a Master File - Yet One More Time. CACM, v.24, no. 5, 299.

12. Jackson, M.A. (1975). Principles of Program Design, Academic Press, N.Y.

13. Lyons, N. (1980). Structured COBOL for Data Processing, Glencoe Pub. Co., Encino, CA.

14. McCracken, D. (1976). A Simplified Guide to Structured COBOL Programming, John Wiley & Sons, N.Y.

15. Molluzzo, J.C. (1981). Jackson Techniques for Elementary Data Processing, ACM SIGCSE Bulletin, v. 13, no. 4, 16-20.

16. Molluzzo, J.C. and Buckley, F. Computer Mathematics with Applications, in preparation.

17. Olsen, J.L. and Price, W.T. (1982). Elements of Structured COBOL Programming, 2nd ed., Holt, Rinehart, Winston, N.Y.

18. Phillipakis, A.S. and Kazmier, L.J. (1982). Advanced COBOL, McGraw-Hill, N.Y.

19. Pressman, R. (1982). Software Engineering: A Practitioners Approach, McGraw-Hill, N.Y.

20. Rindfleisch, D.H. (1976). Debugging System 360/370 Programs Using OS and VS Storage Dumps, Prentice-Hall, Englewood Cliffs, N.J.

21. Shelly, G. and Cashman, T. (1978). Advanced Structured COBOL, Anaheim Press, Fullerton, CA.

22. Tremblay, J. and Bunt, R.B. (1979). An Introduction to Computer Science, McGraw-Hill, N.Y.

23. Tremblay, J. and Sorensen, P.G. (1976). An Introduction to Data Structures with Applications, McGraw-Hill, N.Y.

24. Van Tassel, D. (1978). Program Style, Design, Efficiency, Debugging, and Testing, Prentice-Hall, Englewood Cliffs, N.J.

25. Warnier, J. (1976). Logical Construction of Programs, Van Nostrand-Reinhold Co., N.Y.

26. Weinberg, V. (1980). Structured Analysis, Prentice-Hall, Englewood Cliffs, N.J.

27. Yarmish, R. and Yarmish, J. (1979). Assembler Language Fundamentals, Addison-Wesley Pub. Co., Reading, Mass.

28. Yourdon, E. (1975). Techniques of Program Structure and Design, Prentice-Hall, Englewood Cliffs, N.J.

29. Yourdon, E. and Constantine, L.L. (1978). Structured Design, 2nd ed., Yourdon, Inc., N.Y.

(Continued from page 43)

References

[Alsp72] Alspaugh, Carol Ann. "Identification of Some Components of Computer Programming Aptitude." Journal for Research in Mathematics Education, March 1972, pp. 89-98.

[BaMV68] Bauer, Roger, William A. Mehrens, and John F. Vinsonhaler. "Predicting Performance in a Computer Programming Course." Educational and Psychological Measurement, 28 (1968), pp. 1159-64.

[FoGl81] Fowler, George C. and Louis W. Glorfeld. "Predicting Aptitude in Introductory Computing: A Classification Model." AEDS Journal, Winter 1981, pp. 96-109.

[IPAT79] Institute for Personality and Ability Testing. Administrator's Manual for the 16 PF. Champaign, Ill: IPAT, 1979.

[John72] Johnson, Richard T. Rev. of the Computer Programmer Aptitude Battery. In the Seventh Mental Measurements Yearbook. Highland Park, N. J.: Gryphon Press, 1972.

[MuWa71] Mussio, Jerry J. and Merlin W. Wahlstrom. "Predicting Performance of Programmer Trainees in a Post-High School Setting." Proceedings of the Annual Computer Personnel Research Conference, 1971, pp. 26-45.

[NieN75] Nie, Norman H., et al. Statistical Package for the Social Sciences. 2nd ed. New York: McGraw-Hill, 1975.

[Palo74] Palormo, Jean M. Computer Programmer Aptitude Battery: Examiner's Manual. 2nd ed. Chicago: Science Research Associates, 1974.

[PeHo79] Petersen, Charles G. and Trevor G. Howe. "Predicting Academic Success in Introduction to Computers." AEDS Journal, Fall 1979, pp. 182-91.

[Wein71] Weinberg, Gerald. The Psychology of Computer Programming. New York: Van Nostrand Reinhold, 1971.