THE USE OF SKELETON PROGRAMS IN TEACHING COBOL

by
Wendell L. Pope
Department of Computer Science
Utah State University
Logan, Utah 84321

Persons teaching COBOL encounter the problem of giving students sufficient practice in the use of newly introduced language constructs without assigning a new program for each one. The way to practice COBOL is, of course, to write COBOL, and we typically assign programs for the students to write. However, in programming assignments, certain constructs, particularly in the IDENTIFICATION DIVISION and the ENVIRONMENT DIVISION are practiced repeatedly and become routine, while new constructs are practiced very little. Also, there is a lot of repetition of previously learned constructs just to practice one new one each time a program is written. This 'overhead' in each program means that the proportion of practice on new concepts in each program is discouragingly low. Our experience has been that students can, and do, lean on textbook examples, other students and consultants to such an extent that the successful completion of a programming assignment is not a very good measure of the amount of learning that has taken place. We also find that students tend to equate the benefit from programming assignments with the amount of time it takes them to complete the assignments, thus over-evaluating programming exercises as learning experiences. This results in disappointing performance on examinations. It is not uncommon for students to perform at or above the 90 percent level on homework and around the 70 percent level on examinations. This means that they feel they have done work worth an A and end up with a C in the course. Not a pleasant experience for them or for the teacher.

Increasing the amount of practice on new concepts and decreasing the amount of dependence on others then become worthwhile goals for the person teaching COBOL. This should result in a more realistic balance between the amount of work the student does and the amount of learning that is taking place.

An effort to address these goals has been made through the creation of skeleton programs for various concepts and constructs in COBOL. The skeleton program is an adaptation of an idea given me by a colleague.

A skeleton program consists of an almost-complete COBOL program that requires the addition of a few statements, and enough additional code to check the correctness of the result of executing the added statements. The missing statements are associated with a new concept or construct of the language, and providing the missing statements gives practice in the use of the new construct. Skeleton programs can be used as quizzes or as homework. If used as quizzes they measure a student's understanding of a concept and his skill in using it. Used as homework the skeleton programs provide practice of a construct without the necessity of writing a whole program, but almost everyone who does the exercise can get it right after a few trials. In the quiz environment the student is given a question to respond to in class, the response is written down on paper, with a carbon copy. The original is left with the teacher and the student is instructed to take the copy with him and enter his answer in the skeleton program, compile and run it. The student must note all changes that had to be made in his answer in order to get the skeleton program to compile, link and run correctly. He then turns in the carbon copy with the changes noted. This becomes a measure of his success in generating the right answer to the question. The original acts as a control to prevent attempts to circumvent the purpose of the assignment.

Several advantages of using skeleton programs have been discovered. First, the student is given practice concentrated on the new concept or construct without the overhead of writing an entire COBOL program. Second, the student gets immediate feedback on the value of his answer to the question by submitting it to the impartial review of the compiler and the testing built in to the skeleton program. Third, the amount of help the student may have in creating the answer to the question is

under control of the teacher. Fourth, the number of skeleton programs and their frequency of use can be adapted by the teacher to the needs of the students. Fifth, skeleton programs can be created to explore features of the language that are not easy to incorporate into programming assignments, and sixth, there need be no interference with the normal assignment of programming projects for the course. Skeleton programs are meant to provide practice in a controlled environment, to supplement rather than to replace, the student's experience in writing complete programs.

Several sample skeleton programs follow. The first is on use of the arithmetic verbs ADD, SUBTRACT, MULTIPLY and DIVIDE. The student is asked to write the statements necessary to evaluate an expression without destroying any of the operands (which may necessitate the addition of extra declarations of identifiers for intermediate results in the DATA DIVISION), and store the result rounded in one storage location and truncated in another. The skeleton program includes code that checks to see that the answer was generated correctly and that none of the operands were destroyed. An alternative is to put the code that does the checking in a separate file and include it at compile time with the COPY command. One leans toward the COPY alternative as students get more sophisticated. The skeleton program is written for use at a terminal, and the results of the run are displayed at the terminal along with the correct answer so that the student may know when he has successfully completed the assignment. A similar skeleton program was written for the COMPUTE verb, but is not included due to space limitations.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  TEST-ARITHMETIC-VERBS.
AUTHOR.  W. L. POPE.
*     Using only ADD, SUBTRACT, MULTIPLY
*and DIVIDE verbs, write the statements
*necessary to evaluate:
*              A + B*C
*              -------
*               D - E
*Store the answer rounded in X, truncated
*in Y. Do not destroy any of the operands.
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  VAX-11.
OBJECT-COMPUTER.  VAX-11.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE-OUT ASSIGN TO "ARITH.OUT".
*
DATA DIVISION.
FILE SECTION.
FD  FILE-OUT LABEL RECORDS ARE OMITTED.
01  RECORD-OUT     PIC X(50).

WORKING-STORAGE SECTION.
01  GIVEN-VALUES.
    03  A        PIC S999V99 VALUE -432.75.
    03  B        PIC S99V99  VALUE 29.74.
    03  C        PIC S99V99  VALUE -97.29.
    03  D        PIC S999V999 VALUE 498.53.
    03  E        PIC S999V999 VALUE 519.86.

01  RESULT-VALUES.
    03  X        PIC S99999V99.
    03  Y        PIC S99999V99.
    03  TEMP1    PIC S9(5)V9999.
    03  TEMP2    PIC S999V999.

01  EDITED-RESULTS.
    03  X-OUT    PIC ---,--9.99BB.
    03  Y-OUT    PIC ---,--9.99.
01  SAVE-VALUES.
    03 ASAVE     PIC S9(6)V99.
    03 BSAVE     PIC S9(6)V99.
    03 CSAVE     PIC S9(6)V99.
    03 DSAVE     PIC S9(6)V999.
    03 ESAVE     PIC S9(6)V999.

*
PROCEDURE DIVISION.
MAIN-PARA.
    PERFORM SET-UP-PARA.
* Enter answer here.

    PERFORM CHECK-PARA.
    PERFORM DISPLAY-CORRECT-RESULTS.
    CLOSE FILE-OUT.
    STOP RUN.

CHECK-PARA.
    DISPLAY " ".
    IF A = ASAVE AND B = BSAVE AND
       C = CSAVE AND D = DSAVE AND
       E = ESAVE NEXT SENTENCE
    ELSE MOVE
    "One or more operands were altered."
     TO RECORD-OUT
         DISPLAY RECORD-OUT
         WRITE RECORD-OUT.
    MOVE
      "Here are the results you computed:"
        TO RECORD-OUT.
    DISPLAY RECORD-OUT.
    WRITE RECORD-OUT.
    MOVE "       X           Y"
          TO RECORD-OUT.
    DISPLAY RECORD-OUT.
    WRITE RECORD-OUT.
    MOVE X TO X-OUT.
    MOVE Y TO Y-OUT.
    DISPLAY EDITED-RESULTS.
    WRITE RECORD-OUT FROM EDITED-RESULTS.

DISPLAY-CORRECT-RESULTS.
    DISPLAY " ".
    MOVE SPACES TO RECORD-OUT.
    WRITE RECORD-OUT.
    MOVE
       "Here are the correct results:"
         TO RECORD-OUT.
    DISPLAY RECORD-OUT.
    WRITE RECORD-OUT.
    MOVE 155.94 TO X-OUT.
    MOVE 155.93 TO Y-OUT.
    DISPLAY EDITED-RESULTS.
    WRITE RECORD-OUT FROM EDITED-RESULTS.
```

```
SET-UP-PARA.
    OPEN OUTPUT FILE-OUT.
    MOVE A TO ASAVE.
    MOVE B TO BSAVE.
    MOVE C TO CSAVE.
    MOVE D TO DSAVE.
    MOVE E TO ESAVE.
```

Another skeleton program involves the use of condition names. The student is asked to create some condition names and then use them in an IF statement. A similar program could be written for other conditions and their corresponding IF statements.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  TEST-COND-NAMES.
AUTHOR.  W. L. POPE.
*     This program tests for the use of
*condition names.  The user must supply
*some condition names and an IF statement.
*
*     The identifier RESPONSE contains a
*single character.  Use condition names
*and an IF statement to perform TRUE-PARA
*if RESPONSE contains a T, FALSE-PARA if it
*contains F, and ERROR-IN-RESPONSE if it
*contains any other character or is blank.
*
*     a.  Write the condition names needed.
*     b. Write the IF statement(s) to
*          PERFORM the correct paragraphs.
*
*     If all is done correctly you should
*see messages on the screen indicating
*which paragraphs have been performed.
*The TRUE paragraph should show as the
*first performed, the FALSE second, ERROR-
*IN-RESPONSE third and fourth.
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  VAX-11.
OBJECT-COMPUTER.  VAX-11.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT RESULT-DISPLAY
           ASSIGN TO "CONDNAME.OUT".
*
DATA DIVISION.
FILE SECTION.
FD  RESULT-DISPLAY.
01  RESULT-LINE       PIC X(70).

WORKING-STORAGE SECTION.
01  OUTLINE.
    03  FILLER      PIC X(17)
           VALUE "YOU HAVE ENTERED ".
    03  PARA-NAME   PIC X(17).
    03  FILLER      PIC X(20)
           VALUE ", RESPONSE CONTAINS ".
    03  RESPONSE-OUT PIC X.

01  OUTLINE-ERROR  PIC X(25)
        VALUE "WRONG PARAGRAPH ENTERED!!".

01  GIVEN-VALUE.
    03  RESPONSE    PIC X.

***** enter answer to part a here *****
01  TEST-VALUES.
    03  TRUE-VALUE    PIC X VALUE "T".
    03  FALSE-VALUE   PIC X VALUE "F".
    03  BLANK-VALUE   PIC X VALUE SPACE.
    03  WRONG-VALUE   PIC X VALUE "G".
    03  RESULT-VALUE PIC X.

*
PROCEDURE DIVISION.
MAIN-PARA.
    OPEN OUTPUT RESULT-DISPLAY.
    MOVE TRUE-VALUE TO RESPONSE.
    MOVE SPACE TO RESULT-VALUE.
    PERFORM ANSWER-PARA.
    IF RESULT-VALUE NOT = "T"
          PERFORM WRONG-PARA-MSG.
    MOVE FALSE-VALUE TO RESPONSE.
    MOVE SPACE TO RESULT-VALUE.
    PERFORM ANSWER-PARA.
    IF RESULT-VALUE NOT = "F"
          PERFORM WRONG-PARA-MSG.
    MOVE BLANK-VALUE TO RESPONSE.
    MOVE SPACE TO RESULT-VALUE.
    PERFORM ANSWER-PARA.
    IF RESULT-VALUE NOT = "E"
          PERFORM WRONG-PARA-MSG.
    MOVE WRONG-VALUE TO RESPONSE.
    MOVE SPACE TO RESULT-VALUE.
    PERFORM ANSWER-PARA.
    IF RESULT-VALUE NOT = "E"
          PERFORM WRONG-PARA-MSG.
    CLOSE RESULT-DISPLAY.
    STOP RUN.

ANSWER-PARA.

***** enter answer to part b here *****

TRUE-PARA.
    MOVE "T" TO RESULT-VALUE.
    MOVE "TRUE-PARA" TO PARA-NAME.
    PERFORM OUTPUT-PARA.
FALSE-PARA.
    MOVE "F" TO RESULT-VALUE.
    MOVE "FALSE-PARA" TO PARA-NAME.
    PERFORM OUTPUT-PARA.
ERROR-IN-RESPONSE.
    MOVE "E" TO RESULT-VALUE.
    MOVE "ERROR-IN-RESPONSE" TO PARA-NAME.
    PERFORM OUTPUT-PARA.
WRONG-PARA-MSG.
    DISPLAY OUTLINE-ERROR.
    WRITE RESULT-LINE FROM OUTLINE-ERROR.
OUTPUT-PARA.
    DISPLAY SPACE.
    MOVE RESPONSE TO RESPONSE-OUT.
    DISPLAY OUTLINE.
    MOVE SPACES TO RESULT-LINE.
    WRITE RESULT-LINE.
    WRITE RESULT-LINE FROM OUTLINE.
```

The following skeleton program is to demonstrate skills in handling a one-dimensional table. It also involves the ACCEPT statement and some editing.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DATE-TABLE.
AUTHOR. W. L. POPE.
*    It is required to get the date from
*the system and print it in the form: MAY
*20, 1986; with the correct date printed
*each time the program is run.
* a. Write DATA  DIVISION  entries to
*  declare the fields to receive the date.
* b. Write DATA  DIVISION entries neces-
*     sary to edit the date as required.
* c.  Write DATA  DIVISION  entries to
*     declare a table of month names.
* d.  Write the PROCEDURE DIVISION  state-
*     ments to  get the date from the sys-
*     tem and to fill the  fields declared
*     in b. above.
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. VAX-11.
OBJECT-COMPUTER. VAX-11.
*
DATA DIVISION.
WORKING-STORAGE SECTION.
01  TODAYS-DATE.

01  DATE-OUT.

*
PROCEDURE DIVISION.
MAIN-PARA.

    DISPLAY
      "This is the date from the system:".
    DISPLAY "YYMMDD".
    DISPLAY TODAYS-DATE.
    DISPLAY " ".
    DISPLAY
"This is the way the date was converted:".
    DISPLAY DATE-OUT.
    STOP RUN.
```

The last example is a skeleton program demonstrating the use of the SORT verb with the USING and GIVING options. We have used a similar skeleton program for the SORT verb with INPUT and OUTPUT procedures, but it is not included due to space considerations. Students must be supplied with an unsorted file. Enough information about the format of the file to sort it is given in the comments in the skeleton program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SORT-USING-GIVING.
AUTHOR.  W. L. POPE.
*This is a test of SORT/USING/GIVING.  The
*output can be written to your terminal.
* a. Write the SELECT statements that will
*    be needed.  The data to be sorted are
*    in a file named PAGE154.DAT, records
*    are 35 characters long.
* b.Write the FD/01  pairs for the input
*    and outpt files, and the SD/01 pair
*    needed.  The two keys are customer
*     number in positions 1-7, and quant-
*     ity in positions 23-25.
* c.Write the SORT statement to sort the
*    file in ascending order by customer
*    number and descending order by quant-
*    ity.
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  VAX-11.
OBJECT-COMPUTER.  VAX-11.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
*
DATA DIVISION.
FILE SECTION.
*
PROCEDURE DIVISION.
MAIN-PARA.
    SORT
    STOP RUN.
```

We have used skeleton programs in our COBOL class several times, and different instructors have used them. Both students and instructors have responded favorably to them.

****

COMMUNICATION-- continued from page 60

[5] T.R. Lewis, "Listening To Learn", Santa Ana, CA: Toastmasters International, 1969.

[6] R.G. Nichols and L.A. Stevens, "Listening to people", Harvard Business Review, pp. 112-119, Sep.-Oct. 1957.

[7] C.D. Sigwart and G.L. Van Meer, "The art of the user interview", in Proc. 17th SIGCSE Technical Symposium on Computer Science Education, Feb. 1986, pp. 127-130.

[8] W.J.Taffe, "Teaching Computer Science Through Writing", SIGCSE Bulletin, Vol 18, No 2, Juin 1986

[9] Toastmaster International, "Speaking to inform", in Advanced Communication and Leadership Program, Toastmasters International, 1978.

[10] Toastmaster International, "Technical Presentations" in Advanced Communication and Leadership Program, Toastmasters International, 1984.