

Multi-Languages: A Different Approach to Programming

An Honors Thesis (TD 499)

By

Parola A. Brown

Thesis Director

Dr. Clinton P. Fuelling

Clinton P. Fuelling

Ball State University

Muncie, Indiana

November 1978

Spelt
Thesis
11
2999
-A
-B76

TABLE OF CONTENTS

Introduction	1
Reasons for Multi-Language Programming	2
Method of Implementation	5
Appendix A Program Description	9
Appendix B Input/Output Description	12
Appendix C COBOL Main Program	14
Appendix D FORTRAN/MACRO Subroutines	31
Appendix E Sample Output	33
Bibliography	47

INTRODUCTION

The following is a look at computer programming through the use of multi-languages. Using the DEC 1090, a program has been designed for use as a student grade record. The programming has been accomplished through the use of three languages, COBOL, FORTRAN, and MACRO, the DEC 10 assembly language.

The contents of this report are aimed at showing why and how the use of several languages can be implemented in to one computer program. Through the example of a multi-language program included, some methods and limitations of the multi-language concept can be observed.

REASONS FOR MULTI-LANGUAGE PROGRAMMING

Ever since computers were invented, computer scientists have been faced with the problem of a means of communication with the machine. Early programmers were left with the task of writing code in the machine language of the computer. This was a difficult job; when a change was made to a program the result was that most of the other coding also needed changing. This was necessary because the addressing was in relation to the machine and the addresses were changed with different code.

From the machine language, most computer manufacturers developed an assembly language which made life easier for the programmer. With the assembly language, it was not necessary to change the addressing with each correction. The addressing was relative to the program instead of the entire machine. An address could also be referenced by a name assigned to a location. Another simplification developed in assembly language was the use of mnemonics instead of numbers in reference to the operational code. Assembly language was better than the machine language but a lot of coding for a small procedure was still necessary. Languages were still on a low level and with the increased usage of the computer, something of a higher level was needed to meet the needs.

With this in mind, many groups went about the task of designing higher level languages. In the past few decades, hundreds of computer languages have been developed, but only a small number are used to any extent. Many languages never made it, while others have been replaced by more recent advancements. The first widely used high-level language was FORTRAN, for use in scientific and numerical computing. Developed for execution efficiency, the earliest versions were designed and implemented in the mid-

1950's.¹ Since that time, several versions of the language have been developed.

The language achieving the most success has been COBOL, which stands for Common Business Oriented Language. Developed in the late 1950's through the backing of government and computer manufacturers, the language was out on the market in the early 1960's and is still backed by a maintenance committee. In August of 1968, COBOL became a standardized language.² This has allowed for a wider use of the language since the programmer is able to use one language on several machines with little variation. A non-standardized language can vary greatly between computers.

Another language developed in the mid-1960's was PL/I by International Business Machines. IBM's attempt in designing PL/I was to balance two sets of conflicting goals. The problem was the need for a multipurpose language which would handle generality and flexibility without a loss of execution efficiency.³ The structure of PL/I draws heavily on programming languages. It was designed to meet the needs of the business organization and the scientific computing organization.

To some extent PL/I does meet the needs of both types of organizations, but in spite of this many businesses use COBOL for programming where scientific groups use FORTRAN. At times, the need for a combination of scientific and business programming arises that can not be

¹ Terrence W. Pratt, Programming Languages: Design and Implementation. (Englewood Cliffs: Prentice-Hall, Inc., 1975) p. 315.

² Fundamentals of COBOL. U.S. Navy Programming Languages Group, UNIVAC Division of Sperry Rand Corporation (Sperry Rand Corp., 1968) p. 1.6.

³ Pratt, p. 325.

answered by PL/I. COBOL and FORTRAN individually fall short in answering the problem. In instances where large character strings and reports are needed along with large numerical computations, it would be convenient to be able to use two or more languages to solve the problem. This solution is possible and is permitted by most computer manufacturers.

Most languages have special features that would be beneficial to programs written in other languages. FORTRAN, for instance, has a set of library calls for trigonometric functions and other numerical functions such as the square root and absolute value functions which will be used later in the example. Allowing these functions to be accessed by other languages saves time for both the programmer and the computer.

In the remainder of the report, the solution of how to write programs with multi-languages will be explained.

METHOD OF IMPLEMENTATION

```
.LOAD GRADE.REL,AVERAG.MAC,MEANST.REL
```

To be able to use the result of the load, the combined machine language must be saved. This is done by entering SAV after the programs are loaded. A name is then assigned to the execution file. In the example, the name assigned is STUDEN. The name was chosen from the COBOL program, GRADE.CBL, since it was the first module in the load. STUDEN was obtained from the PROGRAM-ID name. GRADE.REL was the first program in the list because it is the main module.

At this point the program is ready to be executed which can be done by entering

```
.RUN STUDEN
```

The above process is a relatively simple task. The problem arises in setting up the programs to be executed. Certain standards and idiosyncrasies must be dealt with so the code of each subprogram can be understood by the others. One restriction set upon the program is that only one subprogram or main program can be used for all of the input and output of all files. In the example, all of the input/output is handled by the main program.

Another restriction encountered in the use of multi-language programming is that the type of parameters passed between modules is limited. From personal experience and information received from Henry Newsom and Dan Fortriede, personnel of Ball State University Computer Center, a variable must be in computational or computational-3 mode to be used as a parameter. This means a word must be binary integer or packed decimal form. It has also been shown from previous experience that a parameter can not be a table or an element of a table. Digital Equipment's COBOL

Programmer's Reference Manual shows that the use of real value parameters

But results from testing reveal that real values do not work will so for safety the variables should be defined as integers. The above limitations can only be said about Ball State's DEC 10 and ways of removing some of the limitations may be possible, but if this is true, the information was unknown to the author.

The method used to call a FORTRAN or MACRO subroutine from a COBOL program is

```

ENTER {
  MACRO
  FORTRAN-IV
  FORTRAN
  COBOL
} program-name

[ USING {
  identifier-1
  literal-1
  procedure-name-1
} [ , {
  identifier-2
  literal-2
  procedure-name-2
} ] ... ]4

```

The example uses FORTRAN instead of FORTRAN-IV. A COBOL subroutine which is equivalent to a CALL statement has also been excluded in the example. When entering the program-name, the identifier should be placed within quotation marks. The USING clause is optional, but is necessary if any information is to be passed between the programs.

When the COBOL program is compiled the ENTER instruction is converted to MACRO code. Looking at the ENTER for the FORTRAN example, the code generated is comparable to that below.

Programmer's Reference Manual shows that the use of real value parameters, but results from testing reveal that real values do not work will so for safety the variables should be defined as integers. The above limitations can only be said about Ball State's DEC 10 and ways of removing some of the limitations may be possible, but if this is true, the information was unknown to the author.

The method used to call a FORTRAN or MACRO subroutine from a COBOL program is

$$\text{ENTER } \left\{ \begin{array}{l} \text{MACRO} \\ \text{FORTRAN-IV} \\ \text{FORTRAN} \\ \text{COBOL} \end{array} \right\} \text{ program-name}$$

$$\left[\text{USING } \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \\ \text{procedure-name-1} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{identifier-2} \\ \text{literal-2} \\ \text{procedure-name-2} \end{array} \right\} \right] \dots \right] \quad 4$$

The example uses FORTRAN instead of FORTRAN-IV. A COBOL subroutine which is equivalent to a CALL statement has also been excluded in the example. When entering the program-name, the identifier should be placed within quotation marks. The USING clause is optional, but is necessary if any information is to be passed between the programs.

When the COBOL program is compiled the ENTER instruction is converted to MACRO code. Looking at the ENTER for the FORTRAN example, the code generated is comparable to that below.

```

                MOVEI    16,%LIT0605
                PUSHJ   17,MEANST

%LIT060: XWD      440,PASS-SGRADE
%LIT  { XWD      440,PASS-MGRADE
%LIT  { XWD      0,PASS-TOT-GRAD
%LIT  { XWD      0,MEAN-FIND
%LIT  { XWD      0,STD-FIND

```

⁴ COBOL Decsystem 10 Programmer's Reference Manual (Maynard: Digital Equipment Corporation, 1975) p. 6-33.

⁵ Ibid., p. C-2.

The `MOVEI 16,%LIT060` loads the address of the parameter address list into accumulator 16 where it can be referenced from in the subprogram. The `PUSH 17,MEANST` places the address of the next sequential instruction on to the top of the stack. The address of the subprogram `MEANST` is then loaded into the program counter which keeps track of the next instruction. The subprogram is executed until it reaches, in this case, a `RETURN` instruction. This FORTRAN instruction compiles into `POPJ 17,,` which returns control to the main program. The same process is done when branching to the `MACRO` program. The return is the statement `POPJ 17,.`

Once the contact has been achieved from the main program to the subroutine, the subroutine is able to make use of the data passed. By moving the resulting information to one of the parameters, the main program can make use of what the subroutine generates.

Appendix A

PROGRAM DESCRIPTION

The following program updates a student grade file and produces a report of the students' grades. It runs interactively on Ball State University's DEC 10. All programs and files are stored on disk.

The first input into the system is optional. A title card of up to 79 characters can be entered by following the title by an 'H' in the eightieth position. The next input, the first if no header card, is the control card. The control card is checked carefully for any errors since it decides in what directions the program will go. If any errors are found, the program will terminate. The control card contains information^{on} whether all or only the present runs grades shall be listed. It also controls the mean and standard deviation output. All, just the present grades, or no mean and standard deviation can be chosen. The third controlling feature is to print the grade average or not. The remaining 16 bits that are used tell how many tests are to be entered that run, from 0 to 7, and which grades are to be entered specified by a 0 or 1, a 1 meaning that the grade is to be added. A 'C' is entered in position 80.

After the control card is verified, a student card is read in and verified. The social security number is compared to the social security number of a record read from the old file. After determining whether the entry is less than, equal to, or greater than the old record, action is taken depending on the student type of the entry. A student can be added when the social security number is less than the old record. If a student is to be updated, deleted, or have previous grades changed, the social security numbers must be equal.

A new file is created from old records, updated records, and new

records. The new file will be moved after execution to the old file where it can be used on the next run. If the grade average is asked for, all the grades for a particular student are added together and a MACRO subroutine is called to calculate the average. This is done after each student is written on the new file. Also at this time, the grades are added up in a table according^{to} how the mean is specified. Another table handles the sum of the squares for calculation^{of} the standard deviation. The tables are for use at the end of the file. The final action on a student is writing a report line with the social security number, student name, and grades. If requested, the average will also be written. The above processes will be repeated until all student entries are made.

To end the program, it is necessary to place a 'X' in position 80 of the input line. When this is done, all remaining students on the old file are written to the new file and go through the above calculating process. When all are written, the mean and standard deviation tables are used according to the control card. The FORTRAN subroutine will be called and the mean and standard deviation printed at the end of the report.

The program will finish up by writing an audit page. A total of the types of records entering the program was tabulated along with the number of records written. All this information will be printed on the error/audit page.

MACRO subroutine AVERAG.MAC

The subroutine clears the accumulator 0 and by indirect addressing, moves the sum of the student grades to the accumulator. This number is multiplied by 100 so the answer will have more precision when returned to the main. The number is then divided by the indirect addressing of the total number of grades. The result is moved back to an indirect address in relation to accumulator 16 where it can be referenced in the main program. Control is then returned to the main program.

FORTRAN subroutine MEANST.FOR

The mean is calculated by dividing the total of a grade column by the total number of grades. This is multiplied by 10 to carry back more precision. The variance is calculated by dividing the sum of the squares of the grades, which was calculated in the main program, by the total number of grades. From this number the square of the mean is subtracted (before the mean is multiplied by 10). The standard deviation is found by taking the square root of the variance and multiplying by 10. A return to the main program then occurs.

Appendix B

INPUT/OUTPUT DESCRIPTION

INPUT

STUGR1.FIL - Old Tape File
 STUGR2.FIL - New Tape File

Social Security Number	9
Student Name	30
Type of student	1
'A' Add	
'B' Drop	
' ' Regular	
Add/Drop number	2
Grade table 15 times	
Grade Score	3
	<u>45</u>
Total	87

Report Heading Entry

Title	79
Heading code - H	<u>1</u>
Total	80

Control Card

Grades listed	1
'P' Present only	
'A' All	
Mean listed	1
'P' Present only	
'A' All	
' ' None	
Average listed	1
'Y' Yes	
'N' No	
Number test to be entered	1
0 through 7	
Test number to be added - 15 grades	15
0 not added	
1 added	
Filler	61
Control code - C	<u>1</u>
Total	80

Student Record Card

Social Security Number	9
Student Name	30
Add/ Drop number	2
Grade table 7 times	
Grade number 2	
Grade score 3	35
Filler	2
Student type	1
'D' change Previous grade	
'D' Drop student	
'A' Add student	
'U' Update student - add grades	
Student code - 3	<u>1</u>
Total	80
Stop entry	
Filler	70
Stop code - X	<u>1</u>
Total	80

OUTPUT

STUGR2.FIL -New Tape File

Described under input.

Error Line

Error card	80
Filler	2
Error message	<u>50</u>
Total	132

Report Line

Filler	2
Social Security Number	11
Filler	3
Student name	30
Filler	73
Print table 15 times	
Filler 2	
Grade 3	75
Filler	2
Grade average	<u>6</u>
Total	132

Appendix C

COBOL MAIN PROGRAM

GRADE.CBL

IDENTIFICATION DIVISION.

PROGRAM-ID. STUDENT GRADE FILE.
AUTHOR. PAM BROWN
DATE-WRITTEN. OCTOBER 7, 1978
DATE-COMPILED.
REMARKS.

THE FOLLOWING PROGRAM IS PART OF A THESIS ON MULTILANGUAGE PROGRAMMING. THE PROGRAM CALLS UPON SUBROUTINES WRITTEN IN FORTRAN AND MACRO. THE PROGRAM USES TO FILES FOR STUDENT RECORDS. RECORDS CAN BE ADDED, UPDATED OR DELETED. IF THE RECORD IS DELETED, IT IS STILL ON THE FILE PUT CONTAINS A 'D' IN A DROP BIT. THE PROGRAM CAN CALCULATE THE AVERAGE, THE MEAN AND THE STANDARD DEVIATION FOR NONE, SOME, OR ALL OF THE GRADES DEPENDING ON THE CONTROL CARD.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

SOURCE-COMPUTER. DECSYSTEM-10.
OBJECT-COMPUTER. DECSYSTEM-10.
SPECIAL-NAMES. CHANNEL (0) IS TOP-OF-PAGE.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT OLD-GRADE-FILE ASSIGN TO DSKB.
SELECT NEW-GRADE-FILE ASSIGN TO DSKB.
SELECT GRADE-CARDS ASSIGN TO TTY.
SELECT REPORT-FILE ASSIGN TO LPT.
SELECT ERROR-FILE ASSIGN TO LPT.

DATA DIVISION.

FILE SECTION.

FD OLD-GRADE-FILE
LABEL RECORDS ARE STANDARD
RECORDING MODE IS ASCII
DATA RECORD IS OLD-FILE
VALUE OF ID IS 'STUGR1FIL'.

01 OLD-FILE.
05 SOC-SEC-NUM PIC S9(9).
05 STUDENT-NAME PIC X(30).
05 TYPE-STU PIC X.
88 TYPE-DROP VALUE 'D'.
88 TYPE-ADD VALUE 'A'.
88 TYPE-REG VALUE ' '.
05 TYPE-NUM PIC 99.
05 T-GRADE-TABLE OCCURS 15 TIMES
PIC S999.

FD NEW-GRADE-FILE
LABEL RECORDS ARE STANDARD
RECORDING MODE IS ASCII
DATA RECORD IS NEW-FILE
VALUE OF ID IS 'STUGR2FIL'.

01 NEW-FILE.
05 SOC-SEC-NUM PIC S9(9).
05 STUDENT-NAME PIC X(30).
05 TYPE-STU PIC X.
05 TYPE-NUM PIC 99.
05 GRADE-TABLE OCCURS 15 TIMES
PIC S999.

FD GRADE-CARDS
LABEL RECORDS ARE OMITTED
DATA RECORDS ARE HEADER-CARD, CONTROL-CARD,

```

      STUDENT-CARD, STOP-CARD.
01  HEADER-CARD.
    05  HEADER-MES          PIC X(79).
    05  CARD-CODE          PIC X.
      88  HEAD-CODE        VALUE 'H'.
01  CONTROL-CARD.
    05  FILLER              PIC X(79).
    05  CARD-CODE          PIC X.
      88  CONTROL-CODE    VALUE 'C'.
01  STUDENT-CARD.
    05  CD-SOC-SEC-N       PIC  89(9).
    05  CD-NAME            PIC X(30).
    05  CD-TEST-NO        PIC  99.
    05  CD-GRADES OCCURS 7 TIMES.
      10  CD-GRD-NO        PIC  99.
      10  CD-GRD-SCORE    PIC  999.
    05  FILLER              PIC XX.
    05  CD-CODE            PIC X.

      88  CD-PREVIOUS     VALUE 'P'.
      88  CD-DROP         VALUE 'D'.
      88  CD-ADD          VALUE 'A'.
      88  CD-UPDATE       VALUE 'U'.
    05  CARD-CODE          PIC X.
      88  STUDENT-CODE    VALUE 'S'.
01  STOP-CARD.
    05  FILLER              PIC X(79).
    05  CARD-CODE          PIC X.
      88  STOP-CODE      VALUE 'X'.
FD  REPORT-FILE
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS REPORT-LINE.
01  REPORT-LINE            PIC X(133).
FD  ERROR-FILE
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS ERROR-LINE.
01  ERROR-LINE            PIC X(133).

```

WORKING-STORAGE SECTION.

```

77 END-CARDS          PIC 9          VALUE 0.
77 END-FILE          PIC 9          VALUE 0.
77 NO-CARDS          PIC 999        VALUE 0.
77 NO-REC            PIC 999        VALUE 0.
77 REP-PAGE          PIC 99         VALUE 1.
77 ERR-PAGE          PIC 99         VALUE 1.
77 INV-TEST          PIC 9          VALUE 0.
77 TEST-SUB          PIC 99         VALUE 0.
77 T-NUM-SUB         PIC 99         VALUE 0.
77 CHANGE-REC        PIC 9          VALUE 0.
77 TOT-ERR           PIC 999        VALUE 0.
77 TOT-CARDS         PIC 999        VALUE 0.
77 TOT-RECORDS       PIC 999        VALUE 0.
77 TOT-UP            PIC 999        VALUE 0.
77 TOT-DEL           PIC 999        VALUE 0.
77 TOT-ADD           PIC 999        VALUE 0.
77 TOT-PRE           PIC 999        VALUE 0.
77 TOT-WRT           PIC 999        COMP VALUE 0.
77 LEO-SUB           PIC 999        VALUE 0.
77 ERR-LINE-CNT      PIC 99         VALUE 0.
77 REPORT-LINE-CNT  PIC 99         VALUE 0.
77 MEAN-FIND         PIC 9999       COMP.
77 STD-FIND          PIC 999        COMP.
77 PASS-SGRADE       PIC S9(8)      COMP.
77 PASS-MGRADE       PIC S9(7)      COMP.
77 PASS-TOT-GRAD     PIC 9999       COMP.
77 STD-TRAN          PIC 99999.     COMP.
77 GRADE-TRAN        PIC S9(6)      COMP.
77 TOT-GRADES        PIC S9(6)      COMP.
77 GRADE-AVE         PIC S9(6)      COMP.
77 PREV-SOC-SEC-NUM PIC S9(9).
77 PASS-MFIND        PIC 999V9.
77 PASS-SFIND        PIC 99V9.
01 CT-CONTROL-CARD.
   05 CT-GRADE        PIC X.
      88 PRE-GRADES   VALUE 'P'.
      88 ALL-GRADES   VALUE 'A'.
   05 CT-MEAN         PIC X.
      88 PRE-MEAN     VALUE 'P'.
      88 ALL-MEAN     VALUE 'A'.
      88 NO-MEAN      VALUE ' '.
   05 CT-AVE          PIC X.
      88 GR-AVE       VALUE 'Y'.
      88 NO-AVE       VALUE 'N'.
   05 NUM-TESTS       PIC 9.
   05 TEST-ADD OCCURS 15 TIMES
      PIC 9.
   05 FILLER          PIC X(61).
01 HEAD-1.
   05 FILLER          PIC X(84)     VALUE SPACES.
   05 FILLER          PIC X(6)      VALUE 'GRADES'.
   05 FILLER          PIC X(36)     VALUE SPACES.
   05 FILLER          PIC X(5)      VALUE 'GRADE'.
   05 FILLER          PIC X         VALUE SPACE.
01 HEAD-2.
   05 FILLER          PIC X(2)      VALUE SPACES.

```

```

05 FILLER PIC X(11) VALUE 'SOC-SEC-NUM'.
05 FILLER PIC X(11) VALUE SPACES.
05 FILLER PIC X(12) VALUE 'STUDENT NAME'.
05 FILLER PIC X(15) VALUE SPACES.
05 FILLER PIC X(36)
  VALUE '1 2 3 4 5 6 7 8'.
05 FILLER PIC X(36)
  VALUE ' 9 10 11 12 13 14 15'.
01 HEADER-SPACE.
05 FILLER PIC X(27) VALUE SPACES.
05 HEADER=RPT PIC X(79) VALUE SPACES.
05 FILLER PIC X(15) VALUE SPACES.
05 FILLER PIC X(5) VALUE 'PAGE '.
05 REP=PAGE=PRINT PIC Z9.
05 FILLER PIC X(5) VALUE SPACES.
01 ERROR=HEAD.
05 FILLER PIC X(39) VALUE SPACES.
05 FILLER PIC X(4) VALUE 'CARD'.
05 FILLER PIC X(55) VALUE SPACES.
05 FILLER PIC X(7) VALUE 'MESSAGE'.
05 FILLER PIC X(16) VALUE SPACES.
05 FILLER PIC X(5) VALUE 'PAGE '.
05 ERR=PAGE=PRINT PIC Z9.
05 FILLER PIC X(5) VALUE SPACES.
01 ERR=STATE.
05 FILLER PIC X.
05 ERR=CARD PIC X(80) VALUE SPACES.
05 FILLER PIC XX VALUE SPACES.
05 ERR=MES PIC X(50) VALUE SPACES.
01 AUDIT=LINE.
05 FILLER PIC X(5) VALUE SPACES.
05 AUD=MES PIC X(25) VALUE SPACES.
05 AUD=TOT PIC Z99.
05 FILLER PIC X(100) VALUE SPACES.
01 REPORT=DET=LINE.
05 FILLER PIC X(2) VALUE SPACES.
05 PR=SO=SEC=NUM .
  10 SOC=1 PIC 999.
  10 FILLER PIC X VALUE '-'.
  10 SEC=1 PIC 99.
  10 FILLER PIC X VALUE '-'.
  10 NUM=1 PIC 9999.
05 FILLER PIC X(3) VALUE SPACES.
05 PR=STUDENT=NAME PIC X(30) VALUE SPACES.
05 FILLER PIC X(3) VALUE SPACES.
05 PRINT=TAB OCCURS 15 TIMES.
  10 FILLER PIC XX.
  10 PRINT=TABLE PIC ZZZ.
05 FILLER PIC X(2) VALUE SPACES.
05 PR=GRADE=AVE.
  10 GRAD=QUO PIC ZZ9.
  10 GRAD=PER PIC X.
  10 GRAD=REM PIC 99.
01 MEAN=LINE.
05 FILLER PIC X(20) VALUE SPACES.
05 FILLER PIC X(4) VALUE 'MEAN'.
05 FILLER PIC X(25) VALUE SPACES.
05 MEAN=PR=TAB OCCURS 15 TIMES.
  10 MEAN=VAL=LINE PIC ZZ9.9.

```


PROCEDURE DIVISION.

OPEN INPUT OLD-GRADE-FILE, GRADE-CARDS,
 OUTPUT NEW-GRADE-FILE, REPORT-FILE, ERROR-FILE.
 DISPLAY TTY-CONTROL-LINE.

*
 * STUDENT RECORDS READ AFTER CONTROL CARD FROM TERMINAL.
 *

READ-CARD-IN.

READ GRADE-CARDS AT END MOVE 1 TO END-CARDS,
 GO TO FINISH-COPY.
 IF STOP-CODE MOVE 1 TO END-CARDS,
 GO TO FINISH-COPY.

*
 * VALIDATE CONTROL CARD
 *

PROCESS-HEAD-CONTROL.

PERFORM ERROR-HEAD-RPT.
 IF HEAD-CODE MOVE HEADER-MES TO HEADER-RPT
 PERFORM READ-CARD-IN.
 MOVE CONTROL-CARD TO CT-CONTROL-CARD.
 IF NOT CONTROL-CODE
 MOVE 'NO CONTROL CARD' TO ERR-MES,
 PERFORM ERR-PRINT-SEC
 GO TO AUD-REP-SEC.

IF NOT PRE-GRADES AND NOT ALL-GRADES
 MOVE 'INCORRECT GRADE CODE' TO ERR-MES
 PERFORM ERR-PRINT-SEC
 GO TO AUD-REP-SEC.

IF NOT PRE-MEAN AND NOT ALL-MEAN AND NOT NO-MEAN
 MOVE 'INCORRECT MEAN CODE' TO ERR-MES
 PERFORM ERR-PRINT-SEC
 GO TO AUD-REP-SEC.

IF NOT GR-AVE AND NOT NO-AVE
 MOVE 'INCORRECT AVERAGE CODE' TO ERR-MES
 PERFORM ERR-PRINT-SEC
 GO TO AUD-REP-SEC.

IF NUM-TESTS NOT NUMERIC OR NUM-TESTS GREATER THAN 7
 MOVE 'INCORRECT NUMBER OF TESTS' TO ERR-MES
 PERFORM ERR-PRINT-SEC
 GO TO AUD-REP-SEC.

PERFORM CHECK-VAL-TEST VARYING TEST-SUB FROM 1 BY 1
 UNTIL TEST-SUB GREATER THAN 15.

IF INV-TEST EQUAL 1
 MOVE 'INVALID CHOICE OF TEST' TO ERR-MES
 PERFORM ERR-PRINT-SEC
 GO TO AUD-REP-SEC.

PERFORM PRINT-HEAD-SEC.

*
*
*

VALIDATE STUDENT RECORD FROM TERMINAL

```

PROCESS-RECORD-CARDS,
  DISPLAY TTY-CARD-LINE,
  PERFORM READ-CARD-IN,
  ADD 1 TO TOT-CARDS,
  IF NOT STUDENT-CODE
    MOVE 'INVALID CARD CODE' TO ERR-MES
    PERFORM ERR-PRINT-SEC
    GO TO PROCESS-RECORD-CARDS,
  IF CD-SOC-SEC-N NOT NUMERIC
    MOVE 'INVALID SOCIAL SECURITY NUMBER' TO ERR-MES
    PERFORM ERR-PRINT-SEC
    GO TO PROCESS-RECORD-CARDS,
  IF NOT CD-PREVIOUS
    PERFORM TEST-GRADE-NUM THRU TEST-GRADE-SCORE
      VARYING TEST-SUB FROM 1 BY 1 UNTIL
        TEST-SUB GREATER THAN NUM-TESTS,
  IF INV-TEST EQUAL 1
    MOVE 'INVALID TEST OR SCORE ' TO ERR-MES
    PERFORM ERR-PRINT-SEC
    GO TO PROCESS-RECORD-CARDS,
  IF CD-PREVIOUS
    PERFORM TEST-GRADE-SCORE
      VARYING TEST-SUB FROM 1 BY 1 UNTIL
        TEST-SUB GREATER THAN 7,
  IF NOT CD-PREVIOUS AND NOT CD-DROP AND NOT CD-ADD
    AND NOT CD-UPDATE
    MOVE 'INCORRECT STUDENT CODE ' TO ERR-MES
    PERFORM ERR-PRINT-SEC
    GO TO PROCESS-RECORD-CARDS,
  IF (CD-DROP OR CD-ADD) AND CD-TEST-NO NOT NUMERIC
    MOVE 'INVALID ADD/DELETE TEST NUMBER ' TO ERR-MES
    PERFORM ERR-PRINT-SEC
    GO TO PROCESS-RECORD-CARDS,

```

```

READ-OLD-REC,
  READ OLD-GRADE-FILE AT END MOVE 1 TO END-FILE,
  GO TO FINISH-CARDS,
  MOVE 1 TO OLD-WRITE,
  ADD 1 TO TOT-RECORDS,

```

*
 * UPDATE, DROP, OR ADD STUDENT RECORD IN SEQUENTIAL
 * ORDER ON THE STUDENT-RECORD FILE
 *

PREPARE-NEW-FILE.

IF SOC-SEC-NUM OF OLD-FILE LESS THAN CD-SOC-SEC-N
 AND NOT-WRITE
 MOVE 0 TO CHANGE-REC
 MOVE CORRESPONDING OLD-FILE TO NEW-FILE
 PERFORM MOVE-OLD-FILE VARYING TEST-SUB FROM 1 BY
 1 UNTIL TEST-SUB GREATER THAN 15
 PERFORM WRITE-NEW-RECORD
 MOVE 0 TO OLD-WRITE
 PERFORM READ-OLD-REC
 GO TO PREPARE-NEW-FILE.
 IF (SOC-SEC-NUM OF OLD-FILE EQUAL TO CD-SOC-SEC-N) AND CD-ADD
 MOVE 'RECORD ALREADY EXISTS' TO ERR-MES
 PERFORM ERR-PRINT-SEC
 PERFORM PROCESS-RECORD-CARDS
 GO TO PREPARE-NEW-FILE.
 IF SOC-SEC-NUM OF OLD-FILE EQUAL TO CD-SOC-SEC-N AND CD-DROP
 ADD 1 TO TOT-DEL
 MOVE 'D' TO TYPE-STU OF OLD-FILE
 MOVE CD-TEST-NO TO TYPE-NUM OF OLD-FILE
 MOVE CORRESPONDING OLD-FILE TO NEW-FILE
 PERFORM MOVE-OLD-FILE VARYING TEST-SUB FROM 1 BY
 1 UNTIL TEST-SUB GREATER THAN 15
 MOVE 0 TO CHANGE-REC
 PERFORM WRITE-NEW-RECORD
 MOVE 0 TO OLD-WRITE
 PERFORM PROCESS-RECORD-CARDS
 PERFORM READ-OLD-REC
 GO TO PREPARE-NEW-FILE.
 IF SOC-SEC-NUM OF OLD-FILE EQUAL TO CD-SOC-SEC-N AND
 CD-UPDATE
 ADD 1 TO TOT-UP
 PERFORM PREPARE-MOVE-REC
 VARYING TEST-SUB FROM 1 BY 1 UNTIL TEST-SUB
 GREATER THAN NUM-TESTS
 MOVE CORRESPONDING OLD-FILE TO NEW-FILE
 PERFORM MOVE-OLD-FILE VARYING TEST-SUB FROM 1 BY
 1 UNTIL TEST-SUB GREATER THAN 15
 MOVE 1 TO CHANGE-REC
 PERFORM WRITE-NEW-RECORD
 MOVE 0 TO OLD-WRITE
 PERFORM PROCESS-RECORD-CARDS
 PERFORM READ-OLD-REC
 GO TO PREPARE-NEW-FILE.
 IF SOC-SEC-NUM OF OLD-FILE EQUAL TO CD-SOC-SEC-N AND
 CD-PREVIOUS
 ADD 1 TO TOT-PRE
 PERFORM PREPARE-MOVE-REC
 VARYING TEST-SUB FROM 1 BY 1 UNTIL TEST-SUB
 GREATER THAN 7
 PERFORM PROCESS-RECORD-CARDS
 GO TO PREPARE-NEW-FILE.

IF SOC-SEC-NUM OF OLD-FILE GREATER THAN CD-SOC-SEC-N
AND CD-ADD
ADD 1 TO TOT-ADD
PERFORM PREPARE-MOVE-REC-ADD
VARYING TEST-SUB FROM 1 BY 1 UNTIL TEST-SUB
GREATER THAN NUM-TESTS
PERFORM MOVE-ADDED-REC
MOVE 1 TO CHANGE-REC
PERFORM WRITE-NEW-RECORD
PERFORM PROCESS-RECORD-CARDS
GO TO PREPARE-NEW-FILE.
IF SOC-SEC-NUM OF OLD-FILE GREATER THAN CD-SOC-SEC-N
AND NOT CD-ADD
MOVE 'RECORD OUT OF ORDER' TO ERR-MES
PERFORM ERR-PRINT-SEC
PERFORM PROCESS-RECORD-CARDS
GO TO PREPARE-NEW-FILE.

*
 * COPY REMAINDER OF RECORDS FROM OLD FILE TO NEW FILE
 *

FINISH-COPY.
 IF END-FILE EQUAL 1 GO TO PROCESS-MEAN.
 MOVE CORRESPONDING OLD-FILE TO NEW-FILE.
 PERFORM MOVE-OLD-FILE VARYING TEST-SUB FROM 1 BY
 1 UNTIL TEST-SUB GREATER THAN 15
 MOVE 0 TO CHANGE-REC.
 PERFORM WRITE-NEW-RECORD.
 PERFORM READ-OLD-REC.
 GO TO FINISH-COPY.

*
 * ADD REMAINDER OF STUDENT RECORDS FROM TERMINAL
 * IF NOT ADD CODE = ERROR IN RECORD
 *

FINISH-CARDS.
 IF END-CARDS EQUAL 1 GO TO PROCESS-MEAN.
 IF CD-SOC-SEC-N LESS THAN PREV-SOC-SEC-NUM
 MOVE 'RECORD OUT OF ORDER' TO ERR-MES
 PERFORM ERR-PRINT-SEC
 GO TO FIN-READ.
 IF NOT CD-ADD
 MOVE 'INVALID RECORD TYPE' TO ERR-MES
 PERFORM ERR-PRINT-SEC
 GO TO FIN-READ.
 MOVE 1 TO CHANGE-REC.
 ADD 1 TO TOT-ADD.
 PERFORM PREPARE-MOVE-REC-ADD
 VARYING TEST-SUB FROM 1 BY 1 UNTIL
 TEST-SUB GREATER THAN NUM-TESTS.
 PERFORM MOVE-ADDED-REC.
 PERFORM WRITE-NEW-RECORD.
 FIN-READ.
 PERFORM PROCESS-RECORD-CARDS.
 GO TO FINISH-CARDS.

*
* IF MEAN IS ASKED FOR PREPARE AND WRITE MEAN AND
* STANDARD DEVIATION
*

PROCESS-MEAN.

PERFORM CALL-FORT-MEAN
VARYING TEST-SUB FROM 1 BY 1 UNTIL TEST-SUB
GREATER THAN 15.
MOVE MEAN-LINE TO REPORT-LINE.
PERFORM WRITE-REPORT-LINE.
MOVE STD-LINE TO REPORT-LINE.
PERFORM WRITE-REPORT-LINE.
GO TO AUD-REP-SEC.

*
* MOVE VARIABLES TO PARAMETERS AND CALL FORTRAN SUBROUTINE
* MEANST TO CALCULATE THE MEAN AND STANDARD DEVIATION
*

CALL-FORT-MEAN.

IF TOT-GRAD (TEST-SUB) NOT EQUAL 0
MOVE STD-GRADE (TEST-SUB) TO PASS-SGRADE
MOVE MEAN-GRADE (TEST-SUB) TO PASS-MGRADE
MOVE TOT-GRAD (TEST-SUB) TO PASS-TOT-GRAD
ENTER FORTRAN 'MEANST' USING
PASS-SGRADE, PASS-MGRADE, PASS-TOT-GRAD
MEAN-FIND, STD-FIND
DIVIDE MEAN-FIND BY 10 GIVING PASS-MFIND
DIVIDE STD-FIND BY 10 GIVING PASS-SFIND
MOVE PASS-MFIND TO MEAN-VAL-LINE (TEST-SUB)
MOVE PASS-SFIND TO STD-VAL-LINE (TEST-SUB).

```
*
*   PREPARE AUDIT PAGE AND PRINT AT END OF PROGRAM
*
AUD-REP-SEC.
  MOVE 'TOTAL RECORDS READ' TO AUD-MES.
  MOVE TOT-RECORDS TO AUD-TOT.
  WRITE ERROR-LINE FROM AUDIT-LINE AFTER
    ADVANCING TOP-OF-PAGE.
  MOVE SPACES TO AUDIT-LINE.
  MOVE 'TOTAL CARDS READ' TO AUD-MES.
  MOVE TOT-CARDS TO AUD-TOT.
  WRITE ERROR-LINE FROM AUDIT-LINE AFTER
    ADVANCING TOP-OF-PAGE.
  MOVE SPACES TO AUDIT-LINE
  MOVE 'TOTAL RECORDS UPDATED' TO AUD-MES.
  MOVE TOT-UP TO AUD-TOT.
  WRITE ERROR-LINE FROM AUDIT-LINE
    AFTER ADVANCING 1 LINES.
  MOVE SPACES TO AUDIT-LINE.
  MOVE 'TOTAL RECORDS DELETED' TO AUD-MES.
  MOVE TOT-DEL TO AUD-TOT.
  WRITE ERROR-LINE FROM AUDIT-LINE
    AFTER ADVANCING 1 LINES.
  MOVE SPACES TO AUDIT-LINE.
  MOVE 'TOTAL RECORDS ADDED' TO AUD-MES.
  MOVE TOT-ADD TO AUD-TOT.
  WRITE ERROR-LINE FROM AUDIT-LINE
    AFTER ADVANCING 1 LINES.
  MOVE SPACES TO AUDIT-LINE.
  MOVE 'TOTAL PREVIOUS ADDS' TO AUD-MES.
  MOVE TOT-PRE TO AUD-TOT.
  WRITE ERROR-LINE FROM AUDIT-LINE
    AFTER ADVANCING 1 LINES.
  MOVE SPACES TO AUDIT-LINE.
  MOVE 'TOTAL RECORDS WRITTEN' TO AUD-MES.
  MOVE TOT-WRT TO AUD-TOT.
  WRITE ERROR-LINE FROM AUDIT-LINE
    AFTER ADVANCING 1 LINES.
CLOSE-PROGRAM.
  CLOSE OLD-GRADE-FILE, NEW-GRADE-FILE,
    GRADE-CARDS, REPORT-FILE, ERROR-FILE.
  STOP RUN.
```

*
* MAKE MOVES FOR PRINT LINE
*

MOVE-TEST-GRADES-ALL.
MOVE GRADE-TABLE (TEST-SUB) TO
PRINT-TABLE (TEST-SUB).
MOVE-TEST-GRADES-PRE.
MOVE CD-GRD-NO (TEST-SUB) TO T-NUM-SUB.
MOVE GRADE-TABLE (T-NUM-SUB) TO
PRINT-TABLE (T-NUM-SUB).

*
* PREPARE DATA TOTALS FOR CALCULATIONS OF AVERAGE, MEAN
* AND STANDARD DEVIATION
*

LOAD-AVE-TABLE.
IF GRADE-TABLE (TEST-SUB) NOT EQUAL TO 0
ADD GRADE-TABLE (TEST-SUB) TO
GRADE-TRAN
ADD 1 TO TOT-GRADES.
LOAD-MEAN-TABLE-ALL.
IF GRADE-TABLE (TEST-SUB) NOT EQUAL TO 0
ADD 1 TO TOT-GRAD (TEST-SUB)
MULTIPLY GRADE-TABLE (TEST-SUB) BY
GRADE-TABLE (TEST-SUB) GIVING STD-TRAN
ADD STD-TRAN TO STD-GRADE (TEST-SUB)
ADD GRADE-TABLE (TEST-SUB) TO
MEAN-GRADE (TEST-SUB).
LOAD-MEAN-TABLE-PRE.
MOVE CD-GRD-NO (TEST-SUB) TO T-NUM-SUB.
IF GRADE-TABLE (T-NUM-SUB) NOT EQUAL TO 0
ADD 1 TO TOT-GRAD (T-NUM-SUB)
MULTIPLY GRADE-TABLE (T-NUM-SUB) BY
GRADE-TABLE (T-NUM-SUB) GIVING STD-TRAN
ADD STD-TRAN TO STD-GRADE (T-NUM-SUB)
ADD GRADE-TABLE (T-NUM-SUB) TO
MEAN-GRADE (T-NUM-SUB).

* WRITE RECORD ON TO NEW FILE AND PREPARE FOR OUTPUT
*

```

WRITE-NEW-RECORD.
  ADD 1 TO TOT-WRT.
  WRITE NEW-FILE.
  MOVE SOC-SEC-NUM OF NEW-FILE TO PREV-SOC-SEC-NUM.
  IF ALL-GRADES
    PERFORM MOVE-TEST-GRADES-ALL
      VARYING TEST-SUB FROM 1 BY 1 UNTIL TEST-SUB
      GREATER THAN 15.
  IF PRE-GRADES AND CHANGE-REC EQUAL 1
    PERFORM MOVE-TEST-GRADES-PRE
      VARYING TEST-SUB FROM 1 BY 1 UNTIL TEST-SUB
      GREATER THAN NUM-TESTS.
  IF GR-AVE
    PERFORM LOAD-AVE-TABLE
      VARYING TEST-SUB FROM 1 BY 1 UNTIL TEST-SUB
      GREATER THAN 15
    MOVE ZEROES TO GRADE-AVE
    ENTER MACRO 'AVERAG' USING GRADE-TRAN,
      TOT-GRADES, GRADE-AVE
      TOT-GRADES, ' ', GRADE-AVE.
    MOVE ZEROES TO TOT-GRADES
    MOVE ZEROES TO GRADE-TRAN
    MOVE GRADE-AVE TO PRE-GRADE-AVE
    MOVE '.' TO GRAD-PER.
    MOVE CORRESPONDING PRE-GRADE-AVE TO PR-GRADE-AVE.
  IF ALL-MEAN
    PERFORM LOAD-MEAN-TABLE-ALL
      VARYING TEST-SUB FROM 1 BY 1 UNTIL TEST-SUB
      GREATER THAN 15.
  IF PRE-MEAN AND CHANGE-REC EQUAL 1
    PERFORM LOAD-MEAN-TABLE-PRE
      VARYING TEST-SUB FROM 1 BY 1 UNTIL TEST-SUB
      GREATER THAN NUM-TESTS.
  MOVE SOC-SEC-NUM OF NEW-FILE TO PRINT-S-S-N.
  MOVE CORRESPONDING PRINT-S-S-N TO PR-SO-SEC-NUM.
  MOVE STUDENT-NAME OF NEW-FILE TO PR-STUDENT-NAME.
  MOVE REPORT-DET-LINE TO REPORT-LINE.
  PERFORM WRITE-REPORT-LINE.
  DISPLAY REPORT-DET-LINE.
  MOVE SPACES TO NEW-FILE.
  PERFORM ZERO-NEW-RECORD VARYING TEST-SUB FROM 1 BY 1
    UNTIL TEST-SUB GREATER THAN 15.

```

*
* PREPARE AND WRITE TO PRINTER AND/OR TERMINAL ERROR MESSAGES,
* STUDENT RECORDS, FINAL REPORT AND AUDIT PAGE
*

ERR-PRINT-SEC.
 DISPLAY ERR-MES.
 IF ERR-LINE-CNT GREATER THAN 52
 PERFORM ERROR-HEAD-RPT.
 MOVE HEADER-CARD TO ERR-CARD.
 MOVE ERR-STATE TO ERROR-LINE.
 WRITE ERROR-LINE AFTER ADVANCING 1 LINES.
 ADD 1 TO ERR-LINE-CNT.
 ADD 1 TO TOT-ERR.
WRITE-REPORT-LINE.
 IF REPORT-LINE-CNT GREATER THAN 52
 PERFORM PRINT-HEAD-SEC.
 WRITE REPORT-LINE AFTER ADVANCING 1 LINES.
 ADD 1 TO REPORT-LINE-CNT.
ERROR-HEAD-RPT.
 MOVE ERR-PAGE TO ERR-PAGE-PRINT.
 WRITE ERROR-LINE FROM ERROR-HEAD
 AFTER ADVANCING TOP-OF-PAGE.
 MOVE SPACES TO ERROR-LINE.
 WRITE ERROR-LINE AFTER ADVANCING 1 LINE.
 ADD 2 TO ERR-LINE-CNT.
 ADD 1 TO ERR-PAGE.
PRINT-HEAD-SEC.
 MOVE REP-PAGE TO REP-PAGE-PRINT.
 WRITE REPORT-LINE FROM HEADER-SPACE AFTER
 ADVANCING TOP-OF-PAGE.
 WRITE REPORT-LINE FROM HEAD-1 AFTER
 ADVANCING 2 LINES.
 WRITE REPORT-LINE FROM HEAD-2 AFTER
 ADVANCING 1 LINES.
 ADD 4 TO REPORT-LINE-CNT.
 ADD 1 TO REP-PAGE.

WARNINGS:

0633 MOST SIGNIFICANT DIGITS TRUNCATED ON PASS-MFIND
0634 MOST SIGNIFICANT DIGITS TRUNCATED ON PASS-SFIND
0813 MOST SIGNIFICANT DIGITS TRUNCATED ON STD-TRAN
0822 MOST SIGNIFICANT DIGITS TRUNCATED ON STD-TRAN

NO FATAL ERRORS, 4 WARNINGS

Appendix D

FORTRAN/MACRO SUBROUTINES

MEANST.FOR

AVERAG.MAC

```
00100      SUBROUTINE MEANST(NSQ,NX,N,MEAN,NSTD)
00175      XMEAN=FLOAT(NX)/FLOAT(N)
00275      MEAN=10*XMEAN
00375      VAR=(FLOAT(NSQ)/FLOAT(N))-(XMEAN**2,0)
00387      NSTD=IFIX(SORT(ABS(VAR))*10)
00400      RETURN
00500      END
```

```
00050 AVERAG: SETZ 0
00100 MOVE 0,@(16)
00200 IMULI 0,*D100
00300 IDIV 0,@1(16)
00400 MOVEM 0,@2(16)
00500 POPJ 17,
00600 END
```

Appendix E

SAMPLE OUTPUT

FOUR RUNS

First Run

Control Card(1st 19 positions)

PPY4111100000000000

Report Page

Audit Page

New File

Note: STUGR1.FIL created after each run by

.COPY STUGR1.FIL STUGR2.FIL