technical contributions

STRUCTURED PROGRAMMING IN COBOL UNDER IBM 360/370 OS

Melvyn Feurman,
Mary R. Dallal, and Lillian Liebling
c/o Dept. of Air Resources
The City of New York
51 Astor Place
New York, N. Y. 10003

Introduction

In this work, we illustrate structured programming techniques for IBM COBOL 360/370 OS by tracing the development and associated documentation of a COBOL edit program through four steps: problem definition, program specification, program design, and program coding. We have also outlined, in an appendix, the OS JCL required to compile and execute a COBOL main program with subroutines.

Programming Technique

In our opinion, the single most important feature of structured programming is highly readable code. In this regard, we have implemented many of the suggestions made by Baker and Mills (1) who have written that a readable program permits

> "the chief programmer to <u>read</u>, <u>understand</u>, and validate all programs and data developed by <u>other programmers</u> on the team; this motivates better programming. The other programmers in turn read and understand programs written by the chief programmer that define the program stubs which they must interface. While this organization results in the benefits of 'ego-less programming', as described by Weinberg, it goes farther in insuring that at least two programmers fully understand <u>every</u> line of the program." (Emphasis added)

We have also adopted the well-known suggestion made by Dikjistra as to programming without the GO TO statement (2).

Modular Programming

In our programming, we have used CALL and ENTRY statements to write modular programs. These two statements are proposed extensions to ANS COBOL (3) that already have been implemented under IBM 360/370 COBOL (4). Our modular design techniques are similar to those proposed by W.P. Stevens, G. S. Meyers, and L. L. Constantine (5). Consider the following specifications for controlling charges for water and sewer usage in The City of New York.

Problem Definition

"A customer record will be transmitted from one of five borough offices. This record must be checked for accurate, reasonable, and valid information. Incorrect records must be returned with an accompanying errors list for correction. Correct records will be converted according to specifications."

This is a common problem for an edit program: edit the file and create an exceptions list for error records. In the next step we translate the problem into program specifications.

Program Specifications

A file called OFFSET-TRANSACTIONS, containing customer credits, is input to the edit program, where the validity of all fields in the record is checked. If the record is valid, it is converted to a specified format called FINANCE format. If the record is invalid--where one or more fields in the record is in error--the record is printed with an appropriate error message. The program must also maintain counters for the number of records checked, converted, printed, etc.

Design Technique

W. P. Stevens et al have illustrated the division of the design process into a general program design and a detailed program design:

"General program design is deciding what functions are needed for the program (or programming system). Detailed design is how to implement the functions."

We begin by examining the overall flow of control dictated by the problem definition. Then we prepare a general and detailed program design; finally, we code the program into clear meaningful COBOL program statements.

A Flowchart

In Figure 1, the overall flow of control is defined. At (1) a record is read. The validity of the record is then checked against pre-determined specifications at (2). If the record is invalid, we process the record at (3). If the record is valid, we process it at (4). At (5) we specify that the entire process is to be repeated until the end of the input file.

General Program Design

We begin, with the aid of the flowchart, by dividing the program specifications into distinct functions. For each overall function of the program, we assign a module name:

Module Name	Function
FRONT1	Read Input
FRONT2	Check Input
FRONT3	Print Invalid Records
FRONT4	Convert Valid Records
FRONT5	Write Output

Figure 2 represents our implementation of this structured design technique, where each step in the flow becomes a module.

Detailed Program Design

The next task is to translate the general program design into a detailed design. This process involves defining a main program, subroutines, and the parameters that are passed between calling and called programs. In Figure 3, we present a "structure chart" (a term used by Constantine) for the five modules; here we shall refer to them as subroutines. A sixth module named FRONT11 has been added as a main program. Later we shall show how a main program such as FRONT11 can call and pass parameters to subroutines.

Parameter Table

In Figure 3, the parameter linkages between main program and subroutines have been labelled 1 through 5. Figure 4 represents a parameter table or program stub that contains the parameters that are passed between FRONT11 and the called programs. For example, FRONT1, a module designed to read and count records, returns three parameters to FRONT11, namely:

(1) a COUNT-OF-OFFSET-RECORDS-READ
(2) an OFFSET-TRANSACTION record
(3) an EOF-SWITCH-OFFSET-FILE.

Therefore, these three parameters are shown under the OUT column in the table. Similarly, FRONT1 is passed as input a COUNT-OF-OFFSET-RECORDS-READ parameter. Note that this parameter is both an input and output parameter because--as we shall see later--it is received, incremented, and returned by FRONT1 every time a record is read. In a similar manner, parameters are listed for the other four modules that are called by the main program FRONT11.

FRONT2 is used to check the validity of an offset record; it is passed the following parameters as input from FRONT11:

(1) an OFFSET-TRANSACTION-RECORD
(2) a COUNT-OF-RECORDS-CHECKED
(3) a COUNT-OF-RECORDS-FOUND-VALID
(4) a COUNT-OF-RECORDS-FOUND-INVALID

FRONT2 returns as output the following parameters to FRONT11:

(1) a twenty-one element EDIT-ERROR-TABLE, where each element contains a '0' or '1'. A '1' in EDIT-ERROR (I) indicates that the Ith field is invalid; a '0' indicates the Ith field is valid.

An EDIT-ERROR-TABLE pictured: '010000011000000000000' indicates that fields two, eight, and nine have errors; while the remaining eighteen fields are correct.

(2) a RECORD-VALIDITY-INDICATOR. Although there is a redundancy here--since EDIT-ERROR-TABLE will likewise indicate invalid records--we shall show that this parameter makes the main program more readable.

(3) a COUNT-OF-RECORDS-CHECKED incremented by 1 for every record checked by FRONT2.

(4) a COUNT-OF-RECORDS-FOUND-VALID incremented by 1 for every record found valid.

(5) a COUNT-OF-RECORDS-FOUND-INVALID.

Note that in this list, the last three parameters are both input and output parameters. In a similar manner in Figure 4, we define parameter lists for FRONT3, FRONT4, FRONT5.

Subroutine Entry Points

In the final step of the design process we use CALL and ENTRY statements to further modularize the functions of each subroutine. A list of entry point names and associated functions is shown in Figure 5. In Figures 6 and 7, we show a structure chart and corresponding parameter table to illustrate the final modularization of FRONT1.

In the remaining sections of this paper, we present a detailed analysis of three of the modules--FRONT11, FRONT1, and FRONT2--as we translate the structure chart into clear COBOL statements. It is useful to re-write the original program specifications introduced above to show how English language statements can be made analogous to the paragraphs in the Procedure Division of a COBOL program.

Revised Program Specifications

Open input and output files; initialize counters to zero. Read an OFFSET record; if an EOF is reached, close all files, display counters, stop run.

Check the validity of a record; if the record is invalid, print an appropriate error message, keep a count of bad records; if the record is valid, convert the record to FINANCE format, then write the record-as-converted to an output file, keeping a count of records converted and written.

Continue this process until an EOF has been read.

Translation of the Structure Chart into COBOL Statements

We begin by examining in Figure 8 the FRONT11 main program that calls the five subroutines listed in the structure chart of Figure 4. FRONT11 contains the fundamental control logic of the system; it determines when each of the subroutines should be called.

The COBOL source code in FRONT11 is a good example of the internal documentation techniques available in COBOL. The data names that are defined in the Working Storage Section and used in the CALL statements correspond almost exactly with the English phrases used in the parameter table of our structure chart. Descriptive data and paragraph names such as RECORDS-FOUND-VALID and READ-AN-OFFSET-RECORD were chosen to make the program completely transparent to all the programmers on the project. Note also the flow of control in FRONT11: top down, with no GO TO statements. Here we use PERFORM UNTIL... and IF...ELSE to control program loops.

Call Statements

The use of subroutines allows us to develop a main program that is relatively short--the Procedure Division length is approximately one page. FRONT11 is uncluttered with special tests; we have thus avoided confusing main-line logic with specific subroutine functions.

Nested PERFORM Statements

This technique also involves the use of nested PERFORM statements. For example, the MAIN-SEGMENT paragraph contains a PERFORM statement at 006500, which invokes the PROCESS-RECORDS paragraph which in turn performs the READ-AN-OFFSET-RECORD paragraph which finally calls FRONT1 to read a record from the OFFSET file.

The input logic can be clarified by using the structure chart of Figure 9. At (1) the MAIN-SEGMENT paragraph performs the PROCESS-RECORDS paragraph. At (2) PROCESS-RECORDS performs READ-AN-OFFSET-RECORD. At (8) READ-AN-OFFSET-RECORD calls FRONT1B to read an OFFSET record.

We again use nested PERFORM statements to edit input records. For example, in the PROCESS-RECORDS paragraph at 007100, the PERFORM CHECK-A-RECORD THROUGH WRITE-A-FINANCE-RECORD causes the CHECK-A-RECORD, PRINT-ERROR-MESSAGE-IF-ERROR, CONVERT-TO-FINANCE-FORMAT, WRITE-A-FINANCE-RECORD paragraphs to be performed. The flow of control for processing valid records can again be clarified by examining the path (1)-(2)-(8)-(3)-(9)-(4)-(10)-(5)-(11)-(6)-(12)-(7) in the structure chart.

End-of-File Logic

A switch named EOF-SWITCH-OFFSET-FILE (defined at statement 004100) is returned to FRONT11 from FRONT1 after the READ-AN-OFFSET-RECORD paragraph is performed. A value of 0 for EOF-SWITCH-OFFSET-FILE indicates that a record has been read from the OFFSET file; a value of 1 means that FRONT1 has read an end-of-file in this file.

We define a level-88 entry name END-OF-OFFSET-FILE to utilize the PERFORM UNTIL option, avoiding the customary use of the GO TO statement after an end-of-file test. This technique also adds to the readability of the program.

How Parameters are Passed Between Programs

There must be a one-to-one correspondence between parameters in CALL and ENTRY statements. Consider for example the CALL FRONT1A, CALL FRONT1B, and CALL FRONT1C statements at 005600,009800, and 006600 in FRONT11. For each of these CALL statements in the main program there must be corresponding ENTRY FRONT1A, ENTRY FRONT1B, ENTRY FRONT1C statements in a subroutine.

A Linkage Section in the Data Division of the called program is used to define parameters that are passed from the calling to the called program. A Linkage Section should only be included in a program when that program receives a parameter from a calling program.

FRONT11, as a main program, contains no Linkage Section; as a calling program only, it does not receive parameters. FRONT1, FRONT2, FRONT3, FRONT4. and FRONT5 contain Linkage Sections because they receive parameters from FRONT11.

FRONT1

In this section, we examine the Linkage Section and the Procedure Division of FRONT1, shown in Figure 10.

The data names in the CALL and ENTRY statements need not be the same in the calling and called programs; however, in FRONT11 and FRONT1, corresponding data names have been made identical to add to program readability. For example, COUNT-OF-OFFSET-RECORDS-READ is defined in both the Working Storage Section of FRONT11 and the Linkage Section of FRONT1.

Linkage Section in FRONT1

The parameters that are in an ENTRY statement must be defined in the Linkage Section of the called program. For example, the 01 entry for COUNT-OF-OFFSET-RECORDS at 003250 defines the parameter passed in the ENTRY FRONT1A statement. At statement 003300 and 003400, we define OFFSET-TRANSACTION and EOF-SWITCH-OFFSET-FILE as the two parameters used in the ENTRY FRONT1B statement.

Note that the definition of parameters in the Linkage Section does not have to be in the same order as they appear in the corresponding ENTRY statement. For example, EOF-SWITCH-OFFSET-FILE appears before COUNT-OF-OFFSET-RECORDS-READ in the Linkage Section.

Upon entry to the called program, the current values of all the data items in the parameter list of the ENTRY statement are copied into the called program from the calling program. Later when the called subroutine executes a GOBACK statement, the current values of the items in the parameter list are copied back into the calling program, overriding the original values of the parameters in the calling program.

For example, when the CALL to FRONT1A is made, the value of COUNT-OF-OFFSET-RECORDS-READ in FRONT11 is copied into FRONT1 and set equal to zero. After the GOBACK is executed at statement 004100, COUNT-OF-OFFSET-RECORDS-READ is copied back into FRONT11, setting COUNT-OF-OFFSET-RECORDS-READ in FRONT11 equal to zero.

Later, when FRONT1B is called by FRONT11 to read a record, COUNT-OF-OFFSET-RECORDS-READ is again copied into FRONT1, incremented by 1 and copied back into FRONT11.

FRONT2

In Figure 11, we show an abbreviated version of FRONT2, which contains its Working Storage Section, Linkage Section, and Procedure Division. As noted above, the function of FRONT2 is to identify all of the fields in the input record that are in error.

FRONT2 fills EDIT-ERROR-TABLE, a 21-element table (for the 21 fields in the record) with elements 1 or 0, depending on the result of the validity check performed on each field.

We again use ENTRY points to modularize the functions of FRONT2. For example, FRONT2A is used to initialize counters and switches in the program.

FRONT2B is called to check the validity of an input record. As each paragraph is performed, an INDEX is incremented by 1 to keep track of the field being tested. If a field is in error, the STORE-INDEX-PARAGRAPH is performed, which sets the corresponding element in EDIT-ERROR-TABLE to 1, and sets the RECORD-VALIDITY-INDICATOR to 1 as well.

Conclusions

In this work, we have made the basic concepts of structured programming design more concrete by examining a COBOL main program and its subroutines. We have also discussed the case of structured documentation techniques to clarify the design process.

All of the programs that were presented were written in a productive environment under "real life" conditions. We have found that the techniques presented resulted in a system of easily modifiable programs with few errors; developed within a group environment in which junior programmers, working in liaison with a senior programmer, improved and strengthened their COBOL programming and structured programming ability.

APPENDIX

JCL to Compile a Main Program and Subroutines

Under IBM 360/70 OS, the COBOL compile-link-go procedure (COBUCLG) and the COBOL compile (COBUC) procedure are used to run a COBOL main program and subroutines.

Overview

In the first step of the job, we use the COBUC procedure to translate the main program into an object module that is stored on disk.

In subsequent steps, we again execute the COBUC procedure to translate the subroutines; in each step we can compile only a single subroutine. In addition, we use special JCL statements to connect the object module created in each step into a single object module.

In Figure 11, we show the JCL required to compile FRONT11 and the five subroutines in our job.

At (1) we execute the COBUC with the LIB parameter because we use COPY statements in our programs.

The COB.SYSIN DD card at (2) is used to override the SYSLIN DD card in the COB step of the COBUC procedure. Here we allocate SPACE on disk (UNIT=3330) for a temporary file named &OBJ. We specify PASS to pass &OBJ to the next step.

At (3) we define EPA.FRN.SOURCE as the library which contains the members' reference in the COPY statements of our programs.

At (4) we define FRONT11 as the first program to be compiled in the job. Note that the first program to be compiled using the technique described here must be the main program.

At (5) we use the COBUC procedure to compile FRONT1.

At (6) we specify on the COB.SYSLIN DD card the MOD parameter because we are extending &OBJ to include the machine language equivalent of FRONT1.

At (7) we use the COBUC procedure to compile FRONT2, extending &OBJ further to include the FRONT2 machine language code.

This process continues until the last subroutine is to be compiled, whereupon we use the COBUCLG procedure to finish the job. Thus at (8) we use the COBUCLG to compile FRONT9. At (9) we insert the JCL that defines the input and output files for our programs.

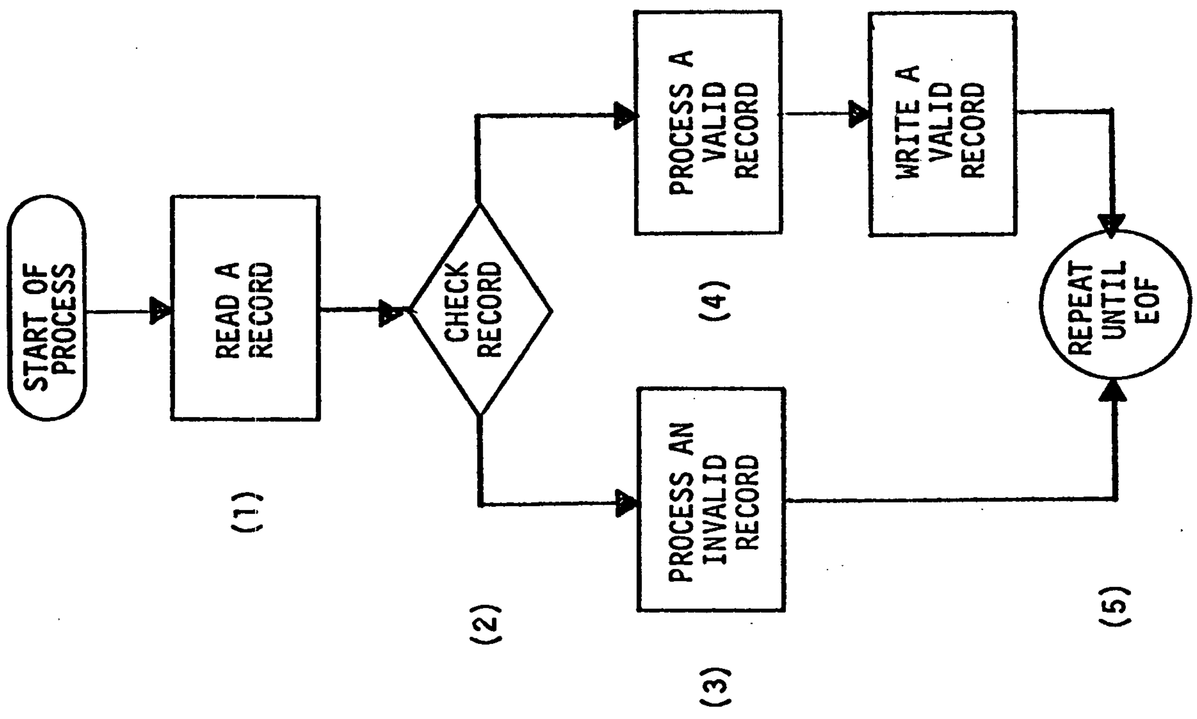


FIGURE 1

FLOW OF CONTROL
FOR EDITING OFFSET RECORDS

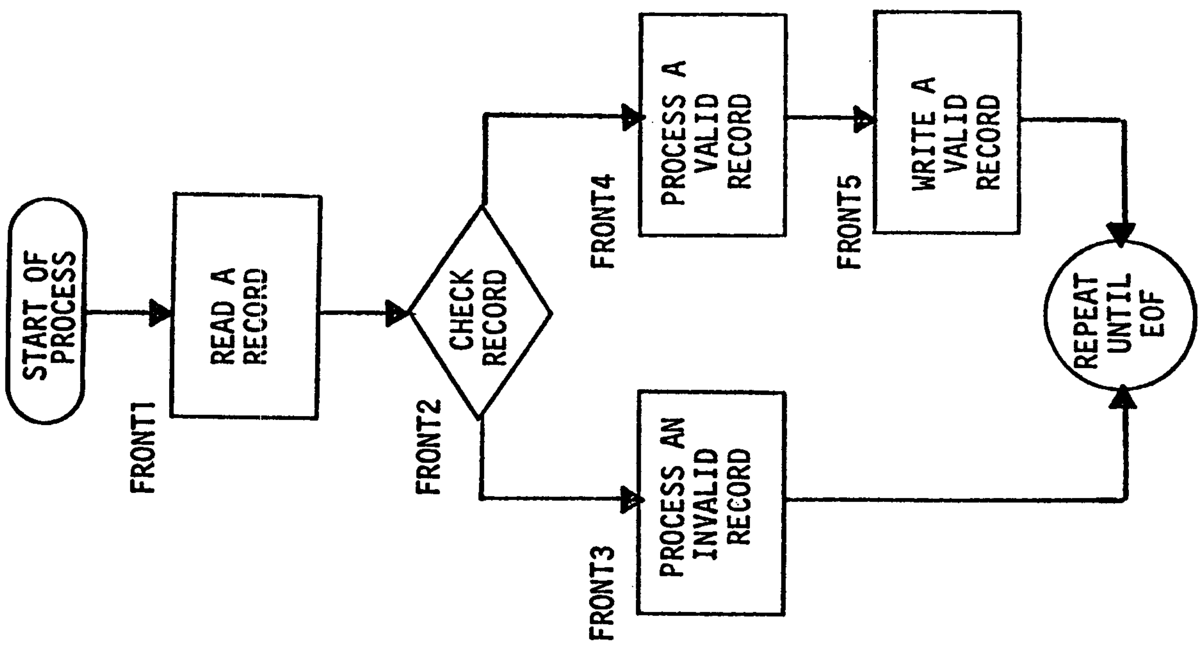


FIGURE 2

FLOW OF CONTROL
FOR EDITING OFFSET RECORDS
AFTER ASSIGNING MODULE NAMES
TO PROGRAM FUNCTIONS

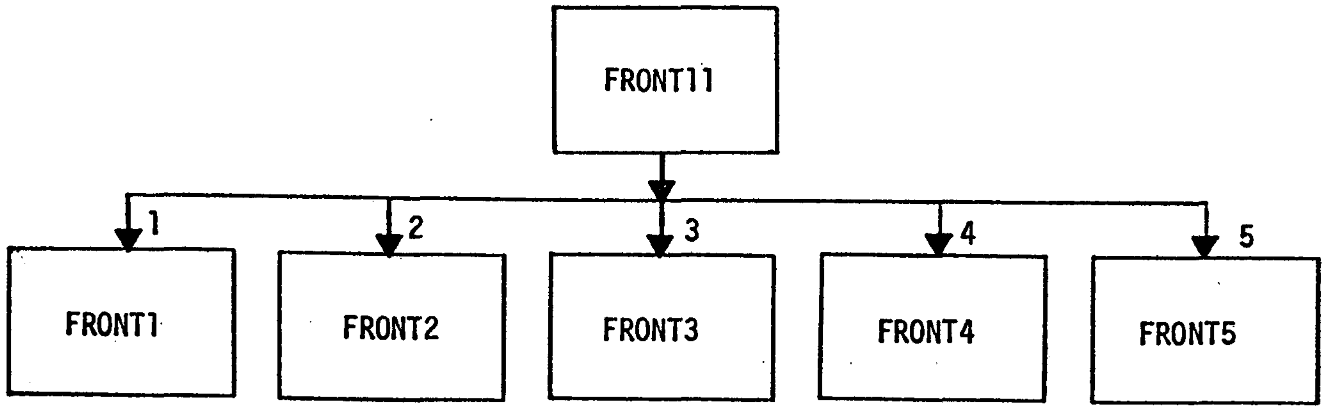


FIGURE 3

STRUCTURE CHART FOR FRONT11 AND SUBROUTINES

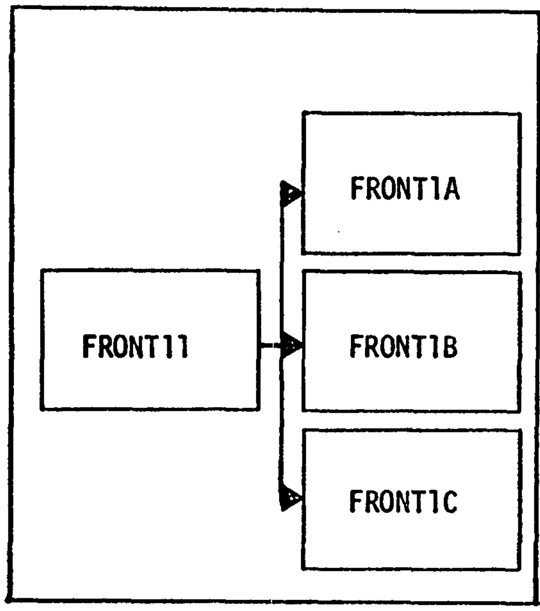
SUBROUTINE	INPUT PARAMETERS	OUTPUT PARAMETERS
FRONT1	COUNT-OF-OFFSET-RECORDS-READ	COUNT-OF-OFFSET-RECORDS-READ OFFSET TRANSACTION EOF SWITCH OFFSET FILE
FRONT2	OFFSET-TRANSACTION COUNT-OF-RECORDS-CHECKED RECORDS-FOUND-VALID RECORDS-FOUND-INVALID	EDIT-ERROR-TABLE COUNT-OF-RECORDS-CHECKED RECORDS-FOUND-VALID RECORDS-FOUND-INVALID RECORD-VALIDITY-INDICATOR
FRONT3	OFFSET-TRANSACTION EDIT-ERROR-TABLE RECORDS-ON-ERRORLIST	RECORDS-ON-ERRORLIST
FRONT4	OFFSET-TRANSACTION COUNT-OF-RECORDS-CONVERTED RECORDS-ON-CREDITLIST RECORDS-ON-DEBITLIST	FINANCE-TRANSACTION COUNT-OF-RECORDS-CONVERTED RECORDS-ON-CREDITLIST RECORDS-ON-DEBITLIST
FRONT5	FINANCE-TRANSACTION GOOD-CREDITS-ONTO-TAPE GOOD-DEBITS-ONTO-TAPE	GOOD-CREDITS-ONTO-TAPE GOOD-DEBITS-ONTO-TAPE

FIGURE 4

PARAMETER TABLES
FOR FRONT11 SUBROUTINES

ENTRY POINT NAME	FUNCTION
FRONT1A	OPENS INPUT FILE; INITIALIZES COUNTERS
FRONT1B	READS INPUT RECORD;SETS EOF SWITCH
FRONT1C	CLOSES INPUT FILE
FRONT2A	INITIALIZES COUNTER FOR RECORDS CHECKED
FRONT2B	VALIDATES A RECORD;CREATES AN ARRAY INDICATING ERROR FIELDS; SETS ERROR INDICATOR TO 0 OR 1
FRONT3A	OPENS ERROR PRINTFILE; INITIALIZES COUNTER
FRONT3B	PRINTS ERROR RECORD; INCREMENTS COUNTER BY 1
FRONT3C	CLOSES ERROR PRINTFILE
FRONT4A	OPENS PRINT FILE; INITIALIZES COUNTER
FRONT4B	CONVERTS AND PRINTS VALID RECORDS
FRONT4C	CLOSES PRINT FILE
FRONT5A	OPENS OUTPUT FILE
FRONT5B	WRITES OUTPUT RECORD; INCREMENTS COUNTER BY 1
FRONT5C	CLOSES OUTPUT FILE

FIGURE 5: ENTRY POINT NAMES AND ASSOCIATED FUNCTIONS



INPUT PARAMETERS	OUTPUT PARAMETERS
COUNT-OF-OFFSET-RECORDS-READ	COUNT-OF-OFFSET-RECORDS-READ
COUNT-OF-OFFSET-RECORDS-READ	OFFSET-TRANSACTION-OUT EOF-SWITCH-OFFSET-FILE COUNT-OF-OFFSET-RECORDS-READ
NONE	NONE

FIGURE 6

FRONT1 ENTRY POINTS

FIGURE 7

PARAMETER TABLE FOR ENTRY POINTS IN FRONT1

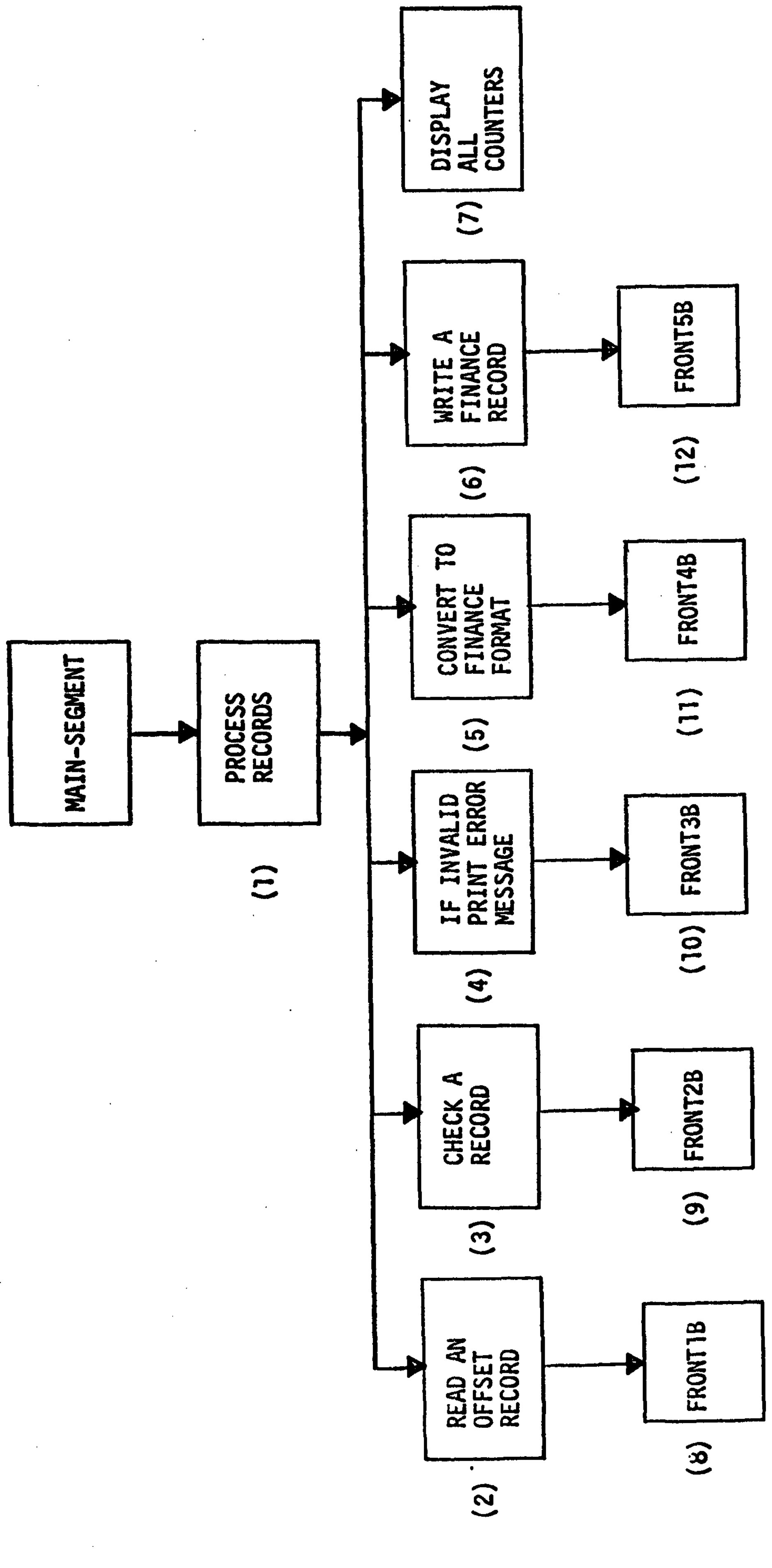


FIGURE 9

STRUCTURE CHART FOR FRONT11

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FRONT1.
REMARKS. THIS SUBROUTINE IS CALLED BY A MAIN PROGRAM TO
             READ THE FRONTAGE FILE. THIS PROGRAM IS PART OF
             THE FRONTAGE SYSTEM FOR COLLECTING WATER AND
             SEWER TAXES. THIS PROGRAM HAS THE FOLLOWING
              THREE ENTRY POINTS
                 (1) FRONT1A - OPENS THE FRONTAGE TRANSACTION
                               FILE
                 (2) FRONT1B - READS A RECORD FROM THE FRONTAGE
                               FILE
                 (3) FRONT1C - CLOSES THE OFFSET FILE AFTER THE
                                   LAST RECORD HAS BEEN READ.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370-155.
OBJECT-COMPUTER. IBM-370-155.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
      SELECT FRONTAGE-OFFSETS ASSIGN TO UT-2400-S-OFFSETS.
DATA DIVISION.
FILE SECTION.
FD   FRONTAGE-OFFSETS
      LABEL RECORDS ARE STANDARD
      RECORD CONTAINS 80 CHARACTERS
      BLOCK CONTAINS 0 RECORDS
      DATA RECORD IS OFFSET-TRANSACTION.
01   OFFSET-TRANSACTION COPY OFFSET.
LINKAGE SECTION.
01   COUNT-OF-OFFSET-RECORDS-READ    PIC  9(5) COMPUTATIONAL.
01   OFFSET-TRANSACTION-OUT COPY OFFSET.
01   EOF-SWITCH-OFFSET-FILE   PIC 9.
      88  END-OF-OFFSET-FILE          VALUE IS 1.
      88  MORE-RECORDS-IN-OFFSET-FILE VALUE IS 0.
01   EOF-SWITCH-FINANCE-FILE          PIC 9.
      88  END-OF-FINANCE-FILE VALUE IS 1.
      88  MORE-RECORDS-IN-FINANCE-FILE VALUE IS 0.
PROCEDURE DIVISION.
OPEN-ALL-FILES.
      ENTRY 'FRONT1A' USING COUNT-OF-OFFSET-RECORDS-READ.
      OPEN INPUT FRONTAGE-OFFSETS.
      MOVE ZEROES TO COUNT-OF-OFFSET-RECORDS-READ.
      GOBACK.
READ-AN-OFFSET-RECORD.
      ENTRY 'FRONT1B' USING OFFSET-TRANSACTION-OUT
                           EOF-SWITCH-OFFSET-FILE
                           COUNT-OF-OFFSET-RECORDS-READ.
      READ FRONTAGE-OFFSETS AT END
           MOVE 1 TO EOF-SWITCH-OFFSET-FILE
           GOBACK.
      MOVE CORRESPONDING OFFSET-TRANSACTION TO
           OFFSET-TRANSACTION-OUT.
      ADD 1 TO COUNT-OF-OFFSET-RECORDS-READ.
      MOVE 0 TO EOF-SWITCH-OFFSET-FILE.
      GOBACK.
CLOSE-FILES.
      ENTRY 'FRONT1C'.
      CLOSE FRONTAGE-OFFSETS.
      GOBACK.
```

FIGURE 10

```
001800 WORKING-STORAGE SECTION.
001810 77  SCC-CHECK                       PIC 9.
001900 77  INDX                            PIC 99 VALUE ZERO.
002400 LINKAGE SECTION.
002455 01  COUNT-OF-RECORDS-CHECKED    PIC 9(5)  COMPUTATIONAL.
002456 01  RECORDS-FOUND-VALID         PIC  9(5)  COMPUTATIONAL.
002457 01  RECORDS-FOUND-INVALID   PIC 9(5) COMPUTATIONAL.
002462 01  RECORD-VALIDITY-INDICATOR          PIC 9.
002465         88  RECORD-IS-VALID                         VALUE IS 0.
002466         88  RECORD-IS-INVALID                       VALUE IS 1.
002500 01  OFFSET-TRANSACTION COPY OFFSET.
002700 01  EDIT-ERROR-TABLE.
002800     05  EDIT-ERROR              PIC 9 COMPUTATIONAL
002900         OCCURS 21 TIMES.
003000 PROCEDURE DIVISION.
003100 CLEAR-THE-COUNTERS.
003105     ENTRY 'FRONT2A' USING COUNT-OF-RECORDS-CHECKED
003106                           RECORDS-FOUND-VALID
003107                           RECORDS-FOUND-INVALID.
003110     MOVE ZEROES TO COUNT-OF-RECORDS-CHECKED
003111                    RECORDS-FOUND-VALID
003112                    RECORDS-FOUND-INVALID.
003115     GOBACK.
003120 BEGIN-THE-EDIT.
003125     ENTRY 'FRONT2B' USING OFFSET-TRANSACTION
003200                           EDIT-ERROR-TABLE
003300                           RECORD-VALIDITY-INDICATOR
003310                           COUNT-OF-RECORDS-CHECKED
003320                           RECORDS-FOUND-VALID
003330                           RECORDS-FOUND-INVALID.
003400     MOVE ZERO TO INDX.
003500     MOVE ZEROS TO EDIT-ERROR-TABLE.
003550     MOVE 0 TO RECORD-VALIDITY-INDICATOR.
003560
003600 CARD-CODE-TEST-CC1.
003700     ADD 1 TO INDX.
003800     IF CARD-CODE NOT EQUAL TO 5
003900         PERFORM STORE-INDEX.
004000 NEW-AUTHORITY-TEST-CC2-CC6.
004010     ADD 1 TO INDX.
004100     EXAMINE NEW-AUTHORITY-NUMBER REPLACING LEADING
004200         SPACES BY ZEROS.
004300     IF NEW-AUTHORITY-NUMBER EQUAL TO '00000'
004400         PERFORM STORE-INDEX.
004500     IF NEW-AUTHORITY-NUMBER NOT NUMERIC
004600         PERFORM STORE-INDEX.
004700 OFFSET-ACTION-TO-BE-TAKEN-CC7.
004800     ADD 1 TO INDX.
004900     IF OFFSET-ACTION-TO-BE-TAKEN NOT EQUAL TO '1'
005000     AND OFFSET-ACTION-TO-BE-TAKEN NOT EQUAL TO '2'
005100         PERFORM STORE-INDEX.
*
************************* REMAINDER OF PROCEDURE DIVISION  *******
*
021200 STORE-INDEX.
021300     MOVE 1 TO RECORD-VALIDITY-INDICATOR.
021400     MOVE 1 TO EDIT-ERROR (INDX).
```

FIGURE 11

```
        //D1R4FRM5 JOB (B826,FWRB,10,2),'MARY R. DALLAL',CLASS=E
        /*SETUP        SETUP EPAPAK
        /*MESSAGE      THIS IS A RUN
        /*MESSAGE      TIME = 10   LINES = 2000
        /*
 (1)    //STEP1     EXEC COBUC,PARM.COB=(LIB)
 (2)    //SYSLIN          DD DSNAME=&OBJ,UNIT=3330,
        //                SPACE=(400,(100,100),,,ROUND),
        //                DCB=(LRECL=80,BLKSIZE=400,RECFM=FBS),
        //                DISP=(MOD,PASS)
 (3)    //SYSLIB          DD DSNAME=EPA.FRN.SOURCE,DISP=OLD
 (4)    //COB.SYSIN       DD DSNAME=EPA.FRN.SOURCE(FRONT11),DISP=OLD
        /*
 (5)    //STEP2     EXEC COBUC,PARM.COB=(LIB)
 (6)    //COB.SYSLIN      DD DSN=&OBJ,DISP=(MOD,PASS)
        //SYSLIB          DD DSN=EPA.FRN.SOURCE,DISP=OLD
        //COB.SYSIN       DD DSNAME=EPA.FRN.SOURCE(FRONT1),DISP=OLD
        /*
 (7)    //STEP3     EXEC COBUC,PARM.COB=(LIB)
        //COB.SYSLIN      DD DSN=&OBJ,DISP=(MOD,PASS)
        //SYSLIB          DD DSN=EPA.FRN.SOURCE,DISP=OLD
        //COB.SYSIN       DD DSNAME=EPA.FRN.SOURCE(FRONT3),DISP=OLD
        /*
        //STEP4     EXEC COBUC,PARM.COB=(LIB)
        //COB.SYSLIN      DD DSN=&OBJ,DISP=(MOD,PASS)
        //SYSLIB          DD DSN=EPA.FRN.SOURCE,DISP=OLD
        //COB.SYSIN       DD DSNAME=EPA.FRN.SOURCE(FRONT4),DISP=OLD
        /*
        //STEP5     EXEC COBUC,PARM.COB=(LIB)
        //COB.SYSLIN      DD DSN=&OBJ,DISP=(MOD,PASS)
        //SYSLIB          DD DSN=EPA.FRN.SOURCE,DISP=OLD
        //COB.SYSIN       DD DSNAME=EPA.FRN.SOURCE(FRONT5),DISP=OLD
        /*
 (8)    //STEP6     EXEC COBUCLG,PARM.COB=(LIB)
        //SYSLIB          DD DSNAME=EPA.FRN.SOURCE,DISP=OLD
        //COB.SYSIN       DD DSN=EPA.FRN.SOURCE(FRONT9),DISP=OLD
        /*
 (9)    //GO.OFFSETS      DD DSN=OFFSETS,DISP=OLD
        //GO.FINANCE      DD DSN=FINANCE,DISP=(NEW,CATLG,DELETE),
        //                UNIT=TAPE9,VOL=SER=EPA999,LABEL=(1,SL)
        //GO.ERRORS       DD SYSOUT=A
        //GO.VALIDREC     DD  SYSOUT=A
```

FIGURE 12

References

(1) F. Terry Baker and Harlan D. Mills, "Chief Programmer Teams", Datamation, December 1973, pp.58-61.

(2) E. W. Dikjistra, "GO TO Statement Considered Harmful", Communication of the ACM, Vol. 11, No. 3, March 1968,pp. 147-8.

(3) ANSI X3J4 Committee, "Proposed Revision of Standard COBOL", SIGPLAN, Vol. 7, No. 6, June 1972, pp.25-40.

(4) IMB OS Full American National Standard COBOL (GC 28-6396, May 1972, pp. 227-40.

(5) W. P. Stevens, G. S. Meyers, L. L. Constantine, "Structured Design", IBM Systems Journal, Vol. 13, No. 2, 1974, pp. 115-39.