NEVADA COBOL

ELLIS COMPUTING

SOFTWARE TECHNOLOGY

NEVADA COBOL

Programmers' Reference Manual

Edition II for use with Rev 2.1 or higher diskettes

Ellis Computing
3917 Noriega Street
San Francisco, CA 94122
(415) 753-0186

PREFACE

This manual is organized for quick reference to the most frequently needed information. Sections I through IV correspond to the four COBOL DIVISIONS while Section V is the INTRODUCTION.

If you are reading this manual for the first time start with SECTION V.

This manual assumes you already know how to program in COBOL and have read the CP/M operating system manuals. If you are a beginner be sure to read the COBOL Primer section.

* CP/M is a registered trademark of Digital Research Corp.

Printed in the U.S.A.

ACKNOWLEDGMENT

This acknowledgment has been reproduced from the "CODASYL COBOL Journal of Development, 1978-79" and "American National Standard Programming Language COBOL, X3.23-1974" as requested in those publications.

Any organization interested in reproducing the COBOL standard and specifications in whole or in part, using ideas from this document as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce the following acknowledgment paragraphs in their entirety as part of the preface to any such publication (any organization using a short passage from this document, such as in a book review, is requested to mention "COBOL" in acknowledgment of the source, but need not quote the acknowledgment):

COBOL is an industry Language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and Language. Moreover, no responsibility is assumed by any contributor, or by the Committee, in connection therewith.

The authors and copyright holders of the copyrighted material used herein

"FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC(R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell"

have specifically authorized the use of this material in whole or in part, in the COBOL specifications in programming manuals or similar publications.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

SECTION I

IDENTIFICATION DIVISION

IDENTIFICATION

FUNCTION: To identify the source program for documentation purposes.

FORMAT:

IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.

[AUTHOR. comment entry.]

[INSTALLATION. comment entry.]

[DATE-WRITTEN. comment entry.]

[DATE-COMPILED. comment entry.]

[SECURITY. comment entry.]

RULES:

1. This entire Division is for documentation purposes only and is treated as comments by the compiler. However, the required key words are checked, so all text must be in upper-case and follow the COBOL rules.

EXAMPLE:

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID. TEST1.
0003 AUTHOR. ELLIS COMPUTING.
0004 INSTALLATION. SAN FRANCISCO PROGRAMMING CENTER.
0005 DATE-WRITTEN. JANUARY 11, 1979.
0006 DATE-COMPILED. JULY 15, 1981.
0007 SECURITY. COPYRIGHT 1979 ELLIS COMPUTING.
0008* comment lines with * in column 5 can be lower-case.
```

COPY

COPY

FUNCTION: The COPY statement inserts text into the source program at compile time.

FORMAT:

COPY u:file-name .

RULES:

1. A COPY cannot occur within another COPY.

2. The disk unit (u:) is optional. The current logged-in disk drive will be used as the default if the unit is not specified.

3. The COPY statement should be preceeded by a space and terminated by a period, normally, starting in column 7.

4. The file type is not part of the COPY statement but must be type CBL.

EXAMPLE:

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID. TESTCOPY.
0003  COPY A:FILE1.
0008  COPY A:FILE2.
0015  COPY B:FILE3.
```

the following represents a separate file named FILE1.CBL to be included (copied) by the above copy statement line 0003.

```
0004 AUTHOR. ELLIS COMPUTING.
0005 INSTALLATION. SAN FRANCISCO PROGRAMMING CENTER.
0006 DATE-WRITTEN. JANUARY 25, 1982.
0007 DATE-COMPILED. JANUARY 25, 1982.
```

SECTION II

ENVIRONMENT DIVISION

ENVIRONMENT

FUNCTION: To identify the computer upon which the program is to be compiled and executed.

FORMAT:

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.  comment    [WITH DEBUGGING MODE].

OBJECT-COMPUTER.  comment

                                  { MODULES    }
                                  { WORDS      }
          [MEMORY SIZE integer-1  { CHARACTERS }]

          [MEMORY BEGINNING integer-1 ENDING integer-2]

          [ PROGRAM COLLATING SEQUENCE IS ASCII] .

SPECIAL-NAMES.  [CURRENCY SIGN IS literal-1]

          [DECIMAL-POINT IS COMMA] .
```

RULES:

1. The generated object code uses memory up to integer-1 CHARACTERS (upper-address limit), if specified. Format 2 specifies a MEMORY BEGINNING address and an ENDING address used to relocate CALLed programs. If these clauses are not used the generated object code will use all available contiguous memory.

2. At compile time the Compiler uses all available contiguous memory.

3. When WITH DEBUGGING MODE is specified, lines with "D" in column 5 are also compiled.

4. PROGRAM COLLATING SEQUENCE IS ASCII is treated as comments by the compiler since the machine collating sequence is ASCII.

5. The literal which appears in the CURRENCY SIGN IS literal clause is used in the PICTURE clause to represent the currency symbol. The literal is limited to a single character and must not be one of the following characters.

a. digits 0 thru 9;
b. alphabetic characters A, B, C, D, L, P, R, S, V, X, Z, or the space;
c. special characters '*', '+', '-', ',', ';', '(', ')', '"', '/', '='.
If this clause is not present, only the currency sign is used in the picture clause.

6. The clause DECIMAL-POINT IS COMMA means that the function of comma and period are exchanged in the character-string of the PICTURE clause and in numeric literals.

7. Integer-1 and integer-2 in the MEMORY SIZE clause are addresses. Users with relocated versions please remember to adjust these addresses upwards.

EXAMPLE:

```
0011 ENVIRONMENT DIVISION.
0012 CONFIGURATION SECTION.
0013 SOURCE-COMPUTER. 8080-CPU
0014     WITH DEBUGGING MODE.
0015 OBJECT-COMPUTER. 8080-CPU
0016     MEMORY SIZE 16383 CHARACTERS.
0017* the following line would be used for called programs.
0016     MEMORY BEGINNING 16384 ENDING 32767.
```

INPUT-OUTPUT

INPUT-OUTPUT

FUNCTION: To name each file and to specify the associated external hardware devices.

FORMAT:

```
INPUT-OUTPUT SECTION.

FILE-CONTROL.

  SELECT file-name-1 ASSIGN TO

                        {PRINTER}
                        {DISK   }

                              {SEQUENTIAL}
    [, ORGANIZATION IS        {RELATIVE  }]

                              {SEQUENTIAL}
    [, ACCESS MODE IS         {RANDOM    }]

    [RELATIVE KEY IS data-name-1]

    [ RECORD DELIMITER IS STANDARD]

    [, FILE STATUS IS data-name-2].

I-O-CONTROL.

    SAME [RECORD] AREA FOR file-name-1, file-name-2...
```

RULES:

1. Each file-name-1 must be unique.

2. The RECORD DELIMITER statement cannot be used with the PRINTER.

3. When the RECORD DELIMITER statement is specified, each record is variable length and separated by a carriage return and line feed.

4. On a delimited write, the record to be transferred is first searched from right to left for the first non-blank character and the delimiter is placed one position to the the right of it. The record including the delimiter is then transferred.

5. On a delimited read, the record is transferred from left to right until the record area is filled or until a delimiter is detected in the incoming data. The delimiter is not transferred to the user area. If the data record is shorter than the record area space, the previous data remains unaltered.

6. Data-name-2 must be defined in the WORKING-STORAGE section as a two (2) character alphanumeric data item.

Position 1 (STATUS KEY 1)	Position 2 (STATUS KEY 2)
0=Successful completion	0=No information availiable
1=AT END	X=SEE ERROR CODES
2=INVALID KEY	
3=PERMANENT ERROR	
9=SEE STATUS KEY 2	

7. ORGANIZATION IS RELATIVE applies only to fixed length DISK files. If this clause is not specified then ORGANIZATION IS SEQUENTIAL is assumed.

8. The RELATIVE KEY uniquely identifies each record in a RANDOM file by an integer greater than zero which specifies the records logical ordinal position in the file. For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area.

9. The RELATIVE KEY is multiplied by the record size and divided by the physical block size and the block is retrieved.

10. The RELATIVE KEY is always an unsigned integer with size 7 or less in the WORKING-STORAGE SECTION.

11. SAME RECORD AREA is for documentation purposes only.

12. A RELATIVE file is created with a fixed length sequential write program to allocate the file space.

13. When RECORD DELIMITER is not specified the records are output in fixed length format each one the size of the longest record description for that file.

14. On INVALID KEY the user record area results are unspecified (filled with padding 1AH characters).

15. On fixed length read when the last record is short the remainder of the user area is filled with padding characters.

16. On a DELIMITED read when a short record is read the results to the right of the last valid input character are unspecified (whatever was there from before the read). Its a good idea to move spaces to the record area before each read.

17. On a DELIMITED read if the input data contains a tab character (09H) it is passed to the user unchanged. If we expanded the tabs then we could not use packed decimal data types because of the possibility of 09H a valid combination in packed decimal, so we don't process the tabs. This allows the use of packed decimal (COMP-3) data types in DELIMITED files. CP/M has a program called PIP that can be used to expand tab characters. See CP/M manual for PIP (T) option.

EXAMPLE:

```
0021 INPUT-OUTPUT SECTION.
0022 FILE-CONTROL.
0023     SELECT OLD-PAYROLL-MASTER-FILE
0024        ASSIGN TO DISK
0025        ORGANIZATION IS SEQUENTIAL
0026        ACCESS MODE IS SEQUENTIAL
0027        RECORD DELIMITER IS STANDARD
0027        STATUS IS STA-1.
0028     SELECT LISTING ASSIGN TO PRINTER.
0029     SELECT NEW-PAYROLL-MASTER-FILE
0030        ASSIGN TO DISK
0031        ACCESS MODE IS RANDOM
0032        RELATIVE KEY IS KEY3
0033        STATUS IS STA-2.
```

COPY COPY

FUNCTION: The COPY statement inserts text into the source program at compile time.

FORMAT:

COPY u:file-name .

RULES:

1. A COPY cannot occur within another COPY.

2. The disk unit (u:) is optional. If not specified the default drive is used.

3. The COPY statement should be preceeded by a space and terminated by a period, normally, staring in column 7.

4. The file type is not part of the COPY statement but must be type CBL.

EXAMPLE:

```
0011 ENVIRONMENT DIVISION.
0012  COPY A:FILE4.
0013* the following copy looks for FILE5.CBL on the default
0014* drive
0015  COPY FILE5.
```

SECTION III

DATA DIVISION

DATA DIVISION

FUNCTION: To specify the particular characteristics of each file.

FORMAT:

```
DATA DIVISION.

FILE SECTION.

FD file-name

                                        {RECORDS    }
     [, BLOCK CONTAINS integer-1        {CHARACTERS}]

          {RECORD IS OMITTED    }
     LABEL{RECORDS ARE STANDARD}

                                  {data-name-1}
     VALUE OF FILE-ID IS          {literal-1  }

          {RECORD IS  }
     [ DATA {RECORDS ARE} record-name-1 [record-name-2]].
```

RULES:

1. BLOCK CONTAINS clause is for documentation purposes only.

2. LABEL RECORDS ARE STANDARD must be used for all disk files and may be used for printer files.

3. VALUE OF FILE-ID must also be used for all disk files and may be used for printer files.

4. Literal-1 is a 1-14 character file name and disk unit. The disk unit is optional and if not present at run time the currently logged-in disk unit will be used.

5. To send output directly to the printer specify VALUE OF FILE-ID IS "A:PRINTER". Any other file-name sends the output to the disk.

6. LABEL RECORD IS OMITTED can be used for printer files only and sends output directly to the printer. If this statement is used then the statement VALUE OF FILE-ID must not be used.

EXAMPLE:

```
0041 DATA DIVISION.
0042 FILE SECTION.
0043 FD  NEW-PAYROLL-MASTER-FILE
0044          LABEL RECORDS ARE STANDARD
0044          VALUE OF FILE-ID IS "A:MASTER.ACT"
0045          DATA RECORDS ARE HOURLY, SALARY.
0046* note record descriptions go here. see next examples
0066 FD  LISTING  LABEL RECORDS ARE STANDARD
0067*  note the next line sends data directly to the printer
0067          VALUE OF FILE-ID IS "PRINTER"
0068          DATA RECORD IS PRINT-LINE.
0100 FD  THE-SOURCE  LABEL RECORDS ARE STANDARD
0101          VALUE OF FILE-ID IS THE-FILE
0102          DATA RECORD IS DISK-IN.
0103 FD  LIST-SPOOL
0104          LABEL RECORDS ARE STANDARD
0105*  note the next line sends data to disk file for later
0105*  printing.  see cpm type command using control-p.
0105          VALUE OF FILE-ID IS "B:LIST.TXT"
0106          DATA RECORD IS PRT-LINE.
0107 FD  LIST2
0108*  note the next line sends data directly to printer
0108          LABEL RECORD IS OMITTED
0109          DATA RECORD IS PRT-LINE2.
```

RECORD DESCRIPTION

RECORD DESCRIPTION

FUNCTION: To specify the particular characteristics of data items.

FORMAT:

```
                {data-name-1}
level-number    {FILLER     } [REDEFINES data-name-2]

                    [, OCCURS integer-1 TIMES]

                       {PIC    }
                    [, {PICTURE} IS {character-string-1}]

                     {SYNC         }  {LEFT }
                    [{SYNCHRONIZED} [{RIGHT}]]

                     {JUST      }
                    [{JUSTIFIED} RIGHT]

                    [BLANK WHEN ZERO]

                                      {COMP           }
                                      {COMP-3         }
                                      {DISPLAY        }
                                      {COMPUTATIONAL-3}
                    [[, USAGE IS]     {COMPUTATIONAL  }]... .
```

WORKING-STORAGE SECTION.

same as above and

```
                                {[ALL] literal}
                                {QUOTE}{HIGH-VALUE}
                                {ZERO}{LOW-VALUE}
                    [, VALUE IS {SPACE  }]... .
```

LINKAGE SECTION.

same as above without value clauses.

RULES:

1. Level-number must be an integer between 01 and 49 or 77.

2. The VALUE clause cannot be used in an item which also contains an OCCURS or REDEFINES clause.

3. The OCCURS clause cannot be used in a 01 or 77 level entry.

4. The WORKING-STORAGE area must be initialized before use, as its initial value is unspecified.

5. The plural forms of SPACE, ZERO, HIGH-VALUE, LOW-VALUE and QUOTE can be used.

6. A PICTURE clause must be specified only for elementary items.

7. The maximum number of characters allowed in character-string-1 is 30.

8. The character-string-1 describes the characteristics and editing requirements of the data. It describes the size of the data, the editing to be performed on the data, and the category of the data. There are five types of data that can be described with a picture clause:

A. Alphabetic character strings contain the symbols 'A' and 'B'. The contents of the alphabetic described item can be any combination of the (26) letters of the Roman alphabet and the space character from the COBOL character set.

B. Numeric character string contain the symbols '9', 'S', and 'V'. The number of digit positions that can be described must range from 1 to 18 inclusive. The contents of the numeric described item can contain the Arabic numerals 0-9 and +,- signs.

C. Alphanumeric character strings contain the symbols 'A', 'X', '9'. Its contents can be any printable ASCII character.

D. Alphanumeric edited character strings contain the symbols 'A', 'X', '9', 'B' '0' '/'.

E. Numeric edited character strings contain the symbols 'B', '/', 'V', 'Z', '0', '9', ',', '.', '*', '+', '-', '$', 'CR', 'DB'.

The individual characters are described as follows:

Each A represents a character position that can contain only a letter of the alphabet or a space.

Each B represents a character position into which the space character will be inserted.

The S indicates the presence (but not the representation nor the position) of an operational sign, and must be written as the leftmost character in the picture string.

The V indicates the location of the assumed decimal point and may appear only once in a character string.

Each X indicates a character position that may contain any allowable character from the ASCII set.

Each Z represents a leading numeric character position; when that position contains a zero, the zero is replaced by a space character. Each Z is counted in the size of the item.

Each 0 represents a character position into which the numeral zero will be inserted and is counted in the size of the item.

Each 9 represents a character position that contains a numeral and is counted in the size of the item.

Each comma represents a character position into which a comma will be inserted and is counted in the size of the item.

The period represents a character position into which the period will be inserted and is counted in the size of the item. It also is used for alignment purposes.

The minus sign (-) represents a character position into which the editing sign control symbol will be inserted and is counted in the size of the item.

The plus sign (+) represents a character position into which the editing sign control symbol will be inserted and is counted in the size of the item.

Each asterisk represents a leading numeric character position into which the asterisk (*) will be inserted and is counted in the size of the item.

The currency symbol ($) represents a character position in to which the ($) is inserted and is counted in the size of the item.

The credit and debit symbols (CR) (DB) each represent two character positions into which they will be inserted and are counted in the size of the item.

9. The USAGE IS clause determines the format of numeric data items stored internally and externally. The default value is DISPLAY which represents ASCII format with the sign stored in the units position bit 7. A positive sign is a 0 bit and a negative sign is 1 bit. Thus a negative number prints as a lower case letter (-500 = 50p) unless it is moved to an edited field. COMPUTATIONAL-3 (COMP-3) directs the compiler to store digits two to the byte in packed decimal format with the sign stored in the right hand end 4 bits. A positive sign is 0000 and a negative sign is 0001. COMPUTATIONAL (COMP) directs the compiler to store values in binary Intel 8080 format with a maximum value of decimal 32767. No matter how the COMP picture is described 9 or 9999, the compiler always assigns 2 bytes for storage.

10. Binary data types should not be used in delimited files because of the possibility of duplicating the delimiter character.

11. When moving numeric values greater than 32767 to a binary data type the results are unspecified. For purposes of data conversion to binary the value 67.000 is greater than 32767 if the binary picture is 99V999.

12. Justified can only be used with elementry data items and cannot be used with numeric or edited picture items.

13. REDEFINES must not be used in Level 01 entries in the File Section. Use the Data Records clause and repeated level 01's for multiple records in the file section.

14. COMP & COMP-3 may be used at the group level.

EXAMPLE:

```
0047 01  HOURLY.
0048      02 PAY-TYPE PICTURE IS X.
0049      02 FIRST-NAME PICTURE IS X(20).
0050      02 LAST-NAME PICTURE X(20).
0051      02 SOC-SEC-NUM PIC 9(9) USAGE IS COMP-3.
0052      02 ITM1 PICTURE IS X.
0053      02 ITM11 REDEFINES ITM1 PIC 9.
0054      02 INCOME PIC S9(16)V99.
0055      02 TAXES OCCURS 10 TIMES PICTURE IS S9(10)V99.
0056 01  MONTHLY.
0057      05 FILLER PIC X.
0058      05 GRP-ITM.
0059      10 GRP-ITM2.
0060      15 GRP-AMT PIC 9(6)V99.
0061      15 GRP-AMT-1 PIC 9(6)V99.
0056 01  PRINT-LINE PICTURE IS X(132).
0081 WORKING-STORAGE SECTION.
0082 01  INVENTORY.
0083      02 PART-NUM PICTURE 9(5) USAGE IS COMP-3.
0084      02 QTY-IN-STOCK PIC 9(6) COMP-3.
0085      02 W-INDEX PICTURE 99 VALUE IS 01 COMP.
0086      02 W-ITM2 PIC X(5) VALUE "TEST1".
0087 01  A-TABLE.
0088      02 T1 PIC X(5) VALUE "FIRST".
0089      02 T2 PIC X(5) VALUE "SECND".
0090      02 T3 PIC X(5) VALUE "THIRD".
0091 01  B-TABLE REDEFINES A-TABLE.
0092      02 ORDER OCCURS 3 TIMES PICTURE X(5).
0093 01  EDIT.
0094      02 E-1  PICTURE $,$$$,$$$,$$$,$$$.99CR.
0095      02 E-2  PIC  99V999+.
0096      02 E-3  PIC ZZ,ZZZ,ZZZ.99-.
0097      02 E-4  PIC $,$$$,$$$.99DB.
0098*  by using the ACCEPT verb the next file name can be
0099*  changed at object time
0100 01  THE-FILE PICTURE X(14) VALUE "A:FILENAME.WRK".
0101 01  KEY3 PIC 9(7) COMP-3 VALUE 1.
0102 01  STA-1.
0103      02 KEY1 PIC X.
0104      02 KEY2 PIC X.
0105* maximum record or item size is 4095
0106 01 BIG-ITEM PIC X(4095).
```

COPY

COPY

FUNCTION: The COPY statement inserts text into the source program at compile time.

FORMAT:

COPY u:file-name .

RULES:

1. A COPY cannot occur within another COPY.

2. The disk unit (u:) is optional and if not present the default drive is used.

3. The COPY statement should be preceeded by a space and terminated by a period, normally, starting in column 7.

4. The file type is not part of the COPY statement but must be type CBL.

EXAMPLE:

```
0041 DATA DIVISION.
0042  COPY A:FILE6.
0055  COPY A:FILE7.
0105  COPY A:FILE8.
```

SECTION IV

PROCEDURE DIVISION

PROCEDURE DIVISION

FUNCTION: To set forth the procedures to solve a given problem.

FORMAT:

```
PROCEDURE DIVISION
          [USING data-name-1 [, data-name-2] ...].

[section-name SECTION [segment-number]].
paragraph-name.
    problem-solving statements.
paragraph-name.
    .
    .
    .
    problem-solving statements.
END PROGRAM program-name.
```

RULES:

1. The first entry in the PROCEDURE DIVISION must be a paragraph name, section-name or USING statement.

2. Each paragraph-name or section-name must be unique.

3. Each paragraph-name must be followed by a period.

4. Each problem-solving statement must be made up of reserved words, words previously described in a previous division, paragraph-names, figurative constants, numeric literals, non-numeric literals and/or punctuation marks.

EXAMPLE:

```
0100 PROCEDURE DIVISION.
0101 BEGIN.
0102     DISPLAY "HELLO".
0103     STOP RUN.
0104 END PROGRAM TEST1.
```

NOTE. The following pages are in alphabetical sequence by key word for easy reference.

ACCEPT **ACCEPT**

FUNCTION: To cause data to be made available to the specified data item via the console device.

FORMAT:

ACCEPT identifier

RULES:

1. The ACCEPT device is the console video typewriter.

2. Data is transferred from left to right until the receiving data item (identifier) is filled or until a carriage return is entered. The carriage return key is used to release the item and is not transferred to memory.

3. The delete key can be used to backspace if a mistake is made.

4. The backspace does not go past the beginning of the accept field.

5. In the CP/M mode using function 1 & 2 when the right end of a field is exceeded a "<" character notifies the user the last character was not entered into memory. This is done because CP/M automaticlly echo's the input character when it is keyed and it appears to the user as if it was processed internally when it was not. However, if the run time package is modified to use function 6 or direct BIOS then the characters exceeding the user field are not output to the screen.

6. See DISPLAY UNIT and the program CONFIG for details on setting up the CRT drivers.

7. The carriage return character is not echoed to the screen unless the CP/M function 1 & 2 mode is being used where CP/M automatically echo's it.

EXAMPLE:

```
0101 PROCEDURE DIVISION.
0102 BEGIN.
0103     ACCEPT EMPLOYEE-NAME (X1).
0104     ACCEPT TODAYS-DATE.
0105     DISPLAY "ENTER FILE NAME <D:FFFFFFFF.EEE>".
0105     ACCEPT THE-FILE-NAME.
0106*
0107*  clear the screen on a sol-20 next.
0108    DISPLAY ""0B"".
0109*  note screen-full can be 80*24=1920 size item.
0110    DISPLAY SCREEN-FULL.
0111*  set the cursor using a hexadecimal string.
0112    DISPLAY ""1B,01,3F"".
0113    ACCEPT INPUT-ITEM.
```

ADD **ADD**

FUNCTION: To add two numeric data items and to store the sum.

FORMAT:

```
        {literal-1   }       {literal-2   }
ADD     {identifier-1} [TO]  {identifier-2}

[GIVING identifier-3] [ROUNDED]

[, ON SIZE ERROR imperative-statement]
```

RULES:

1. Each ADD verb statement must contain an addend and an augend.

2. Figurative constants cannot be used.

3. Only numeric items and numeric literals can be used, except identifier-3 which can be an elementary numeric edited item.

4. The composite of operands must not contain more than 18 digits.

5. An identifier can only reference an elementary item.

6. Each operand can contain an operational sign and an implied decimal point.

7. Operands are aligned according to implied decimal points.

8. ROUNDED performs a test to see if right truncation will occur and, if it will, adjusts the result by adding 1 if the truncated digit is 5 or greater.

9. ON SIZE ERROR performs a test to see if overflow has occurred and, if it has, executes the imperative-statement.

EXAMPLE:

```
0150     ADD SALES-TAX TO TOTAL GIVING GRAND-TOTAL ROUNDED
0151         ON SIZE ERROR GO TO ERROR-ROUTINE.
```

ALTER **ALTER**

FUNCTION: To modify a predetermined sequence of operations.

FORMAT:

ALTER paragraph-name-1 TO PROCEED TO paragraph-name-2.

RULES:

1. Paragraph-name-1 must be the name of a paragraph which contains a single sentence consisting of:

GO TO paragraph-name.

2. The execution of the ALTER statement modifies the GO TO paragraph-name-1, so that subsequent executions of paragraph-name-1 transfer control to paragraph-name-2.

EXAMPLE:

```
0200 PARA-6. GO TO BEGIN.
0201 PARA-7.
0202     ALTER PARA-6 TO PROCEED TO END-OF-JOB.
0203     GO TO PARA-5.
0205 END-OF-JOB.
```

CALL **CALL**

FUNCTION: The CALL statement causes control to be transferred from one object program to another, within the run unit.

FORMAT:

```
     {literal-1   }
CALL {identifier-1}
          [USING data-name-1 [data-name-2]...]
```

RULES:
1. Literal-1 must be a nonnumeric literal.

2. Identifier-1 must be defined as an alphanumeric data item such that its value can be a program name.

3. The USING phrase is included in the CALL statement only if there is a USING phrase in the Procedure Division header of the called program and the number of operands in each USING phrase must be identical.

4. Each of the operands in the USING phrase must have been defined as a data item in the File Section or Working-Storage Section, and must have a level-number of 01 or 77.

5. The program whose name is specified by the value of literal-1 or identifier-1 is the called program; the program in which the call statement appears is the calling program.

6. The execution of a CALL statement causes control to pass to the called program.

7. A called program is in its initial state the first time it is called within a run unit and the first time it is called after a CANCEL to the called program. On all other entries into the called program, the state of the program remains unchanged from its state when last exited. This includes all data fields, the status and positioning of all files, and all alterable switch settings.

8. Called programs may contain call statements. However, a called program must not contain a CALL statement that

directly or indirectly calls the calling program.

9. The data-names, specified by the USING phrase of the CALL statement, indicate those data items available to a calling program that may be referred to in the called program. The order of appearance of the data-names in the USING phrase of the CALL statement and the USING phrase in the Procedure Division header is critical. Corresponding data-names refer to a single set of data which is available to the called and calling program. The correspondence is positional, not by name.

NEVADA COBOL details:

1. Called programs must be type .OBJ.

2. Each called program is dynamically loaded the first time and entered into a table in the run time package. Future calls go directly to the called program.

3. Up to 5 active called programs may be resident at any one time. At that point one will have to be CANCELed before any other can be loaded.

4. You can CALL another main program from the current program thus overlaying the first program. Since the working-storage section always begins at the same point in memory, those data-items not initialized with value statements will contain the information from the prior program. Be sure to CANCEL the program to remove it from the table as once the table is full and a program is called the job will terminate.

5. CALLed programs need not be COBOL programs. However, they must be type .OBJ and be ORGed correctly. The .OBJ file contains the machine language code for a program, the address at which the run time package is to load it, and the address at which execution of the loaded program is to begin. An .OBJ file consists of one or more segments that have the format:

#BYTES	DESCRIPTION
2	Number of code and data bytes in segment
2	Load address of code and data belonging to the segment.
Variable	Code and/or data.

The run time package will load each segment at the specified address until a starting address is encountered. A starting address is represented as load address with a zero byte count.

6. A program is supplied to convert CP/M HEX files to .OBJ format named CONVHEX.COM.

7. The run time package transfers control to the called program by means of an 8080 CALL instruction. The called program should return via the 8080 RET instruction. The called program should use its own stack not the COBOL stack.

8. Parameters are passed to the called program in the registers. H&L = parameter 1, D&E = parameter 2, B&C = either parameter 3 or the address of the left end of a list of parameter addresses if more than three parameters are passed. The parameters consist of 16-bit addresses pointing to the right end of each data-name.

9. In some cases it is possible to execute called programs without the calling program for testing when no data is being passed. Since the loading format is the same for all type .OBJ programs, you can A>RUN NEXTPROG

EXAMPLE:

```
0001    CALL "NEXTPROG" USING REC-1, REC-2.
0555    CALL NEXT-PROG USING REC-1, REC-2.
    *
    * also see complete programs at end of manual.
```

CANCEL CANCEL

FUNCTION: The CANCEL statement releases the memory areas occupied by the referred to program.

FORMAT:

```
              {literal-1    }
CANCEL        {identifier-1}
```

RULES:
1. Subsequent to the execution of a CANCEL statement, the program referred to therein ceases to have any logical relationship to the run unit in which the CANCEL statement appears. A subsequently executed CALL statement naming the same program will result in that program being initiated in its initial state. The memory areas associated with the named programs are released so as to be made available for disposition by the operating system.

2. A program named in the CANCEL statement must not refer to any program that has been called and has not yet executed an EXIT PROGRAM statement.

3. A logical relationship to a cancelled subprogram is established only by execution of a subsequent CALL statement.

4. A called program is cancelled either by being referred to as the operand of a CANCEL statement or by the termination of the run unit of which the program is a member.

5. No action is taken when a CANCEL statement is executed naming a program that has not been called in this unit or has been called and is at present cancelled. Control passes to the next statement.

EXAMPLE:

```
0001      CANCEL "LASTPROG".
0555      CANCEL LAST-PROG.
```

* also see complete programs at end of manual.

CLOSE **CLOSE**

FUNCTION: To terminate the processing of input and output files.

FORMAT:

CLOSE file-name

RULES:

1. A file must be opened before it can be closed.

2. If required, the CLOSE statement writes the final block with padding before closing the file.

EXAMPLE:

```
0300 END-OF-JOB.
0301       CLOSE NEW-PAYROLL-MASTER-FILE.
0302       CLOSE OLD-PAYROLL-MASTER-FILE.
0303       CLOSE LISTING.
```

COPY COPY

FUNCTION: The COPY statement inserts text into the source program at compile time.

FORMAT:

COPY u:file-name .

RULES:

1. A COPY cannot occur within another COPY.

2. The disk unit (u:) is optional and if not present the default drive will be used.

3. The COPY statement should be preceeded by a space and terminated by a period, normally, starting in column 7.

4. The file type is not part of the COPY statement but must be type CBL.

EXAMPLE:

```
0100 PROCEDURE DIVISION.
0101 PARAGRAPH-A.
0102  COPY A:FILEA.
2500 PARAGRAPH-B.
2501  COPY A:FILEB.
3500 PARAGRAPH-C.
3501  COPY B:FILEC.
```

DISPLAY **DISPLAY**

FUNCTION: To display data on the video typewriter.

FORMAT-1:

```
        {literal-1    }  {literal-2    }
DISPLAY {identifier-1} [{identifier-2}] ...
                       [WITH NO ADVANCING]
```

FORMAT-2:

```
                  {literal-3    }
DISPLAY UNIT      {identifier-3}.
```

RULES:

1. The DISPLAY device is the console video typewriter.

2. If the literal is a numeric literal, then it must not be signed as the sign would be displayed as a lower case letter.

3. A carriage return & line feed are executed before data transfer begins unless WITH NO ADVANCING is specified.

4. Data4is transferred from left to right until all of the data in literal or identifier-1 is transferred.

5. If data is longer than 64 or 80 characters as set by the CONFIG program, the video display will continue on the next line. In this way the entire screen can be filled with one DISPLAY statement.

6. Each literal may be any figurative constant, except ALL.

7. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

8. The DISPLAY statement causes the contents of each operand to be transferred to the hardware device in the order listed.

9. The DISPLAY UNIT literal changes the I-O driver at run time as follows:

"0X" skips CP/M and uses the BIOS driver.
"2X" uses CP/M function 1 & 2 drivers.
"6X" uses CP/M 2.X function 6 drivers.
X will allow any character to be input any other character in this position will allow only ASCII input. All of these changes are temporary.

10. To permanently change the run time package drivers read the instructions for the program CONFIG.

11. UNIT 0 or UNIT 6 must be used if sending or receiving characters other than ASCII, such as video control characters. This is because CP/M monitors function 1 and 2 and will not allow certain control characters to pass too and from the user.

EXAMPLE:

```
0350 ERROR-ROUTINE.
0351     DISPLAY ERROR-MESSAGE (ERROR-CODE).
0352     DISPLAY FIRST-NAME, LAST-NAME, "NAME ".
0359D    DISPLAY "DEBUG MODE ERROR ROUTINE".
0360     DISPLAY "CONTINUE ON SAME LINE" WITH NO ADVANCING.
0370* the next line clears the screen on a Sol-20 or VDM-1
0380     DISPLAY ""0B"".
0391* the next line clears the screen on Hazeltine-1520
0392     DISPLAY ""7E,1C"".
0390* each CRT is different but if you know the commands you
0391* can also set the cursor and display in reverse.
0391* the next line sets the I-O driver for BIOS any
0391* incoming character will be passed to user.
0392     DISPLAY UNIT "0X".
0500* the following sequence is a common debugging method.
0501 PARAGRAPH-A.
0502* line 0505 is a debugging line used when testing
0503* to let the programmer know that the paragraph has been
0504* executed
0505D    DISPLAY "PARAGRAPH-A".
```

DIVIDE **DIVIDE**

FUNCTION: To divide one numerical data item into another and set the value of an item equal to the quotient.

FORMAT:

DIVIDE {identifier-1} INTO {identifier-2}

[GIVING identifier-3]

[ROUNDED] [, ON SIZE ERROR imperative-statement]

RULES:

1. Each DIVIDE statement must contain a dividend and a divisor.

2. Each identifier must refer to an elementary numeric item, except the identifier-3 which may be an elementary numeric edited item.

3. The composite of operands must not contain more than 18 digits.

4. An identifier can only reference an elementary item.

5. Each operand can contain an operational sign and an implied decimal point.

6. Operands are aligned according to implied decimal points.

7. ROUNDED performs a test to see if right truncation will occur and, if it will, adjusts the result by adding 1 if the truncated digit is 5 or greater.

8. ON SIZE ERROR performs a test to see if overflow has occurred and, if it has, executes the imperative-statement.

EXAMPLE:

```
0400 CALC-1.
0401       DIVIDE HOURS INTO GROSS-PAY GIVING HOURLY-RATE
0402          ROUNDED ON SIZE ERROR GO TO ERR-2.
0403       DIVIDE HOURS INTO MILES.
```

END PROGRAM **END PROGRAM**

FUNCTION: To signal the physical end of the program.

FORMAT:

END PROGRAM program-name

RULES:

1. This entry must be the last physical statement in every source program.

EXAMPLE:

```
9999 END PROGRAM TEST1.
```

EXIT **EXIT**

FUNCTION: To furnish an end point for a series of procedures.

FORMAT-1:

EXIT.

FORMAT-2:

EXIT PROGRAM.

RULES:

1. The EXIT statement must appear in a sentence by itself, and be the only sentence in the paragraph.

2. An execution of an EXIT PROGRAM statement in a called program causes control to be passed to the calling program. Execution of an EXIT PROGRAM statement in a program which is not called behaves as if the statement were an EXIT statement.

EXAMPLE:

```
0500 PARA-END.
0501       EXIT.
0600 END-SUBPROGRAM.
0601       EXIT PROGRAM.
```

GO TO GO TO

FUNCTION: To depart from the normal sequence of procedures.

FORMAT-1:

GO TO procedure-name-1

FORMAT-2:

GO TO procedure-name-1, [procedure-name-2]...
DEPENDING ON identifier.

RULES:

1. The GO TO statement must be the last statement in a sequence.

2. Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.

3. When a paragraph is referenced by an ALTER statement, that paragraph can consist only of a paragraph header followed by a format-1 GO TO statement.

4. When a GO TO statement, represented by format-1 is executed, control is transferred to procedure-name-1 or to another procedure-name if the GO TO statement has been modified by an ALTER statement.

5. When a GO TO statement represented by format-2 is executed, control is transferred to procedure-name-1, procedure-name-2, etc., depending on the value of the identifier being 1, 2,..., n. If the value of the identifier is anything other than the positive or unsigned intergers 1, 2, ..., n, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

EXAMPLE:

```
0330      IF A-SWITCH IS EQUAL TO 1
0331         MOVE X-AMT TO Y-AMT
0332         GO TO A-SUBROUTINE.
0333      GO TO MAIN-PROGRAM.
0334 CASE-STATEMENT-PARA.
0335      GO TO A-PARA, B-PARA, C-PARA DEPENDING ON X1.
0336 ALTERED-PARA.
0337      GO TO FIRST-PARA.
```

IF **IF**

FUNCTION: The IF statement causes a condition to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

FORMAT-1:

```
                    {statement-1   } {ELSE statement-2   }
    IF {condition} {NEXT SENTENCE} {ELSE NEXT SENTENCE}
```

{condition}:

```
                                 { = < >       }
                                 {EQUAL TO     }
                                 {LESS THAN    } {literal     }
     identifier-1 IS [NOT] {GREATER THAN} {identifier-2}
```

{condition}:

```
                                      {NUMERIC   }
          identifier-3  IS [NOT] {ALPHABETIC}
```

FORMAT-2:

```
                    {OR }
       IF condition {AND} condition
```

RULES:

1. Statement-1 and statement-2 represent an imperative statement.

2. Non-numeric comparisons are made left to right using the ASCII collating sequence.

3. Numeric comparisons are made by aligning the decimal points and treating them as algebric quantities.

4. Identifier-3 must be a DISPLAY (ASCII) data type.

5. If the condition is true, statement-1 is executed if specified. If statement-1 contains a procedure branching statement, control is explicitly transferred in accordance with the rules of that statement. If statement-1 does not

contain a procedure branching statement, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

6. The ELSE NEXT SENTENCE phrase may be omitted if it immediately precedes the terminal period of the sentence.

7. If the condition is true and the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

8. If the condition is false, statement-1 or its surrogate NEXT SENTENCE is ignored, and statement-2, if specified, is executed. If statement-2 contains a procedure branching statement, control is explicitly transferred in accordance with the rules of that statement. If statement-2 does not contain a procedure branching statement, control passes to the next executable sentence. If the ELSE statement-2 is not specified, statement-1 is ignored and control passes to the next executable sentence.

9. If the condition is false, and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored, if specified, and control passes to the next executable sentence.

10. Two conditions can be combined by the logical operators AND and OR.

EXAMPLE:

```
0340      IF LAST-NAME IS NOT ALPHABETIC
0341         MOVE ERR-CODE TO MESG
0342         ADD 1 TO ERR-COUNT
0343         GO TO KEY-PUNCH-ERROR
0344       ELSE
0345         PERFORM A-PARA THRU B-PARA.
0346      IF HOURLY-RATE < 3.90 AND FRINGE-BENEFITS < 6000
0347         GO TO MIN-WAGE-ERROR.
0348      IF A = B
0349        OR = C
0350        OR = D
0351        OR X NOT > Y
0352          MOVE S TO W
0353        ELSE
0354          MOVE S TO AW.
```

INSPECT INSPECT

FUNCTION: The INSPECT statement provides the ability to tally, replace, or tally and replace occurences of single characters in a data item.

FORMAT-1

```
 INSPECT identifier-1 TALLYING
                      {ALL       {literal-1   }}
                      {LEADING   {identifier-3}}
   {identifier-2 FOR {{CHARACTERS              }
                     {AFTER }          {literal-2   }
                    [{BEFORE} INITIAL {identifier-4}]}...}...
```

FORMAT-2

```
 INSPECT identifier-1 REPLACING
                    {literal-4   }
  {CHARACTERS BY {identifier-6}
                           {AFTER }          {literal-5   }
                          [{BEFORE} INITIAL {identifier-7}
  {ALL       }
  {FIRST     }  {literal-3   }    {literal-4   }
  {LEADING   } {{identifier-5} BY {identifier-6}
                           {AFTER }          {literal-5   }
                          [{BEFORE} INITIAL {identifier-7}
```

FORMAT-3

```
 INSPECT identifier-1 TALLYING
                      {ALL       {literal-1   }}
                      {LEADING   {identifier-3}}
   {identifier-2 FOR {{CHARACTERS              }
                     {AFTER }          {literal-2   }
                    [{BEFORE} INITIAL {identifier-4}]}...}...

 REPLACING

                    {literal-4   }
  {CHARACTERS BY {identifier-6}
                           {AFTER }          {literal-5   }
                          [{BEFORE} INITIAL {identifier-7}
  {ALL       }
  {FIRST     }  {literal-3   }    {literal-4   }
  {LEADING   } {{identifier-5} BY {identifier-6}
                           {AFTER }          {literal-5   }
                          [{BEFORE} INITIAL {identifier-7}
```

RULES:

1. Identifier-1 must reference either a group item or any category of elementary item, described (either implicitly or explicitly) as usage is DISPLAY.

2. Identifier-3... identifier-n must reference either an elementary alphabetic, alphanumeric or numeric item described (either implicitly or explicitly) as usage is DISPLAY.

3. Each literal must be nonnumeric and may be any figurative constant, except ALL.

4. Literal-1, 2, 3, 4, 5 and the data items referenced by identifier-3, 4, 5, 6, and 7 must be one character in length.

FORMATS 1 and 3 only
5. Identifier-2 must reference an elementary numeric data item.

6. If either literal-1 or literal-2 is a figurative constant, the figurative constant refers to an implicit one character data item.

FORMATS 2 and 3 only
7. The size of the data referenced by literal-4 or identifier-6 must be equal to the size of the data referenced by literal-3 or identifier-5. When a figurative constant is used as literal-4, the size of the figurative constant is equal to the size of literal-3 or the size of the data item referenced by identifier-5.

8. When the CHARACTERS phrase is used, literal-4, literal-5 or the size of the data item referenced by identifier-6, identifier-7 must be one character in length.

9. When a figurative constant is used as literal-3, the data referenced by literal-4 or identifier-6 must be one character in length.

GENERAL RULES:
1. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing) begins at the leftmost character position of the data item referenced by identifier-1, regardless of its class, and proceeds from left to right to the rightmost character position as described in general rules 4 through 6.

2. For use in the INSPECT statement, the contents of the data item referenced by identifier-1, 3, 4, 5, 6 or 7 will be treated as follows:

a. If any of identifier-1, 3, 4, 5, 6 or 7 are described as alphanumeric, the INSPECT statement treats the contents of each such identifier as a character-string.

b. If any of identifier-1, 3, 4, 5, 6 or 7 are described as alphanumeric edited, numeric edited or unsigned numeric, the data item is inspected as though it had been redefined as alphanumeric and the INSPECT statement had been written to reference the redefined data item.

c. If and of the identifier-1, 3, 4, 5, 6 or 7 are described as signed numeric, the data item is inspected as though it had been moved to an unsigned numeric data item of the same length and then the rules in general rule 2b had been applied.

3. In general rules 4 through 11 all references to literal-1, 2, 3, 4 and 5 apply equally to the contents of the data item referenced by identifier-3, 4, 5, 6 and 7, respectively.

4. During inspection of the contents of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (formats 1 and 3) and/or each properly matched occurrence of literal-3 is replaced by literal-4 (formats 2 and 3).

5. The comparison operation to determine the occurrences of literal-1 to be tallied and/or occurrences of literal-3 to be replaced, occurs as follows:

a. The operands of the TALLYING and REPLACING phrases are considered in the order they are specified in the INSPECT statement from left to right. The first literal-1, literal-3 is compared to an equal number of contiguous characters, starting with the leftmost character position in the data item referenced by identifier-1. Literal-1, literal-3 and that portion of the contents of the data item referenced by identifier-1 match if, and only if, they are equal, character for character.

b. If no match occurs in the comparison of the first literal-1, literal-3, the comparison is repeated with each successive literal-1, literal-3, if any until either a match is found or there is no next successive literal-1, literal-3. When there is no next successive literal-1, literal-3, the character position in the data item

referenced by identifier-1 immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins again with the first literal-1, literal-3.

c. Whenever a match occurs, tallying and/or replacing takes place as described in general rules 8 through 10. The character position in the data item referenced by identifier-1 immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of the data item referenced by identifier-1, and the comparison cycle starts again with the first literal-1,literal-3.

d. The comparison operation continues until the rightmost character position of the data item referenced by identifier-1 has participated in a match or has been considered as the leftmost character position. When this occurs, inspection is terminated.

e. If the CHARACTERS phrase is specified, an implied one character operand participates in the cycle described in paragraphs 5a through 5d above, except than no comparison to the contents of the data item referenced by identifier-1 takes place. This implied character is considered always to match the leftmost character of the contents of the data item referenced by identifier-1 participating in the current comparison cycle.

6. The comparison operation defined in general rule 5 is affected by the BEFORE and AFTER phrases as follows:

a. If the BEFORE or AFTER phrase is not specified, literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates in the comparison operation as described in general rule 5.

b. If the BEFORE phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from its leftmost character position up to, but not including, the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 5 is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no

occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

c. If the AFTER phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase may participate only in those comparison cycles which invlove that portion of the contents of the data item referenced by identifier-1 from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1 and the rightmost character position of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 5 is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-1, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

FORMAT 1

7. The contents of the data item referenced by identifier-2 is not initialized by the execution of the INSPECT statement.

8. The rules for tallying are as follows:

a. If the ALL phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each occurrence of literal-1 matched within the contents of the data item referenced by identifier-1.

b. If the LEADING phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each contiguous (adjacent) occurrence of literal-1 matched within the contents of the data item referenced by identifier-1, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.

c. If the CHARACTERS phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each character matched, in the sense of general rule 5e, within the contents of the data item referenced by

identifier-1.

FORMAT 2
9. The required words ALL, LEADING, and FIRST are adjectives.

10. The rules for replacement are as follows:

a. When the CHARACTERS phrase is specified, each character matched, in the sense of general rule 5e, in the contents of the data item referenced by identifier-1 is replaced by literal-4.

b. When the adjective ALL is specified, each occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4.

c. When the adjective LEADING is specified, each contiguous occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4, provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-3 was eligible to participate.

d. When the adjective FIRST is specified, the leftmost occurrence of literal-3 matched within the contents of the data item referenced by identifier-1 is replaced by literal-4.

FORMAT 3
11. A format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written with one statement being a format 1 statement with TALLYING phrases identical to those specified in the format 3 statement, and the other statement being a format 2 statement with REPLACING phrases identical to those specified in the format 3 statement. The general rules given for matching and counting apply to the format 1 statement and the general rules given for matching and replacing apply to the format 2 statement.

EXAMPLES:
Following are six examples of the INSPECT statement:

```
INSPECT word TALLYING count FOR LEADING "L" BEFORE INITIAL
"A", count-1 FOR LEADING "A" BEFORE INITIAL "L".

Where word = LARGE, count = 1, count-1 = 0.
Where word = ANALYST, count =0, count-1 = 1.


INSPECT word TALLYING count FOR ALL "L", REPLACING LEADING
"A" BY "E" AFTER INITIAL "L".

Where word = CALLAR, count = 2, word = CALLAR.
Where word = SALAMI, count = 1, word = SALEMI.
Where word = LATTER, count = 1, word = LETTER.


INSPECT word REPLACING ALL "A" BY "G" BEFORE INITIAL "X".

Where word = ARXAX, word = GRXAX.
Where word = HANDAX, word = HGNDGX.


INSPECT word TALLYING count FOR CHARACTERS AFTER INITIAL "J"
REPLACING ALL "A" BY "B".

Where word = ADJECTIVE, count = 6, word = BDJECTIVE.
Where word = JACK, count = 3, word = JBCK.
Where word = JUJMAB, count = 5, word = JUJMBB.


INSPECT word REPLACING ALL "X" BY "Y"  "B" BY "Z"  "W" BY
"Q" AFTER INITIAL "R".

Where word = RXXBQWY, word = RYYZQQY.
Where word = YZACDWBR, word = RAQRYEZ.


INSPECT word REPLACING CHARACTERS BY "B" BEFORE INITIAL "A".

word before: 12 XZABCD
word after:  BBBBBABCD
```

MOVE **MOVE**

FUNCTION: To transfer data from one data area to another.

FORMAT:

```
      {literal-1   }
MOVE  {identifier-1} TO identifier-2 [identifier-3]...
```

RULES:

1. Identifier-1 and literal-1 represent the sending area and identifier-2 identifier-3, ..., represent the receiving area.

2. The data designated by the literal-1 or identifier-1 is moved first to identifier-2, then to identifier-3, ... The rules referring to identifier-2 also apply to the other receiving areas. Any subscripting associated with identifier-2, ..., is evaluated immediately before the data is moved to the receiving data item.

3. Any MOVE in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited. These categories are described in the PICTURE clause. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric. The figurative constant ZERO belongs to the category numeric. The figurative constant SPACE belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric.

The following rules apply to an elementary move between these categories:
a. The figurative constant SPACE, a numeric edited, alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.
b. A numeric literal, the figurative constant ZERO, a numeric data item or a numeric edited data item must not be moved to an alphabetic data item.
c. A non-integer numeric literal or a non-integer numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.
d. All other elementary moves are legal and are performed

according to the rules given in general rule 4.

4. Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any editing specified for the receiving data item:
a. When an alphnumeric edited or alphanumeric item is a receiving item, alignment and any necessary space filling takes place. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled. If the sending item is described as being signed numeric, the operational sign will not be moved; if the operational sign occupied a separate character position, that character will not be moved and the size of the sending item will be considered to be one less than its actual size (in terms of standard data format characters).
b. When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero-filling takes place as necessary, except where zeroes are replacing because of editing requirements.
1. When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. Conversion of the representation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.
2. When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.
3. When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.
c. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place as defined. If the size of the sending item is greater than the the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled.

5. Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area, except as noted in the OCCURS clause.

6. If literal-1 is SPACE, QUOTE or ZERO then identifier-2 is entirely filled with the figurative constant.

7. In a non-numeric move the data is moved left to right.

EXAMPLE:

```
0360 MAIN-MOVE-ROUTINE.
0361     MOVE SPACES TO PRINT-LINE.
0362     MOVE FIRST-NAME TO P-FIRST-NAME.
0363     MOVE LAST-NAME TO P-LAST-NAME.
0364     MOVE ORDER (W-INDEX) TO P-ORDER.
0365     MOVE ZEROS TO AMT-1, AMT-2, AMOUNT-3.
0366     MOVE SPACES TO FIRST-NAME  LAST-NAME.
```

MULTIPLY **MULTIPLY**

FUNCTION: To multiply numeric data items and set the value of an item equal to the result.

FORMAT:

```
                {literal-1   }     {literal-2   }
MULTIPLY        {identifier-1} BY  {identifier-2}

[GIVING identifier-3]     [ROUNDED]

[, ON SIZE ERROR imperative-statement]
```

RULES:

1. Each identifier must be a elementary numeric item, except identifier-3 which may be a elementary numeric edited item.

2. Each literal must be a numeric literal.

3. The resultant product must not contain more than 18 digits.

4. An identifier can only reference an elementary item.

5. Each operand can contain an operational sign and an implied decimal point.

6. Operands are aligned according to implied decimal points.

7. ROUNDED performs a test to see if right truncation will occur and, if it will, adjusts the result by adding 1 if the truncated digit is 5 or greater.

8. ON SIZE ERROR performs a test to see if overflow has occurred and, if it has, executes the imperative-statement.

EXAMPLE:

```
0399 CALCULATION-ROUTINE.
0400     MULTIPLY WAGE-RATE BY REGULAR-HRS GIVING
0401       GROSS-PAY ROUNDED ON SIZE ERROR GO TO P-ERR.
0402     MULTIPLY WAGE-RATE BY OVERTIME-HOURS.
```

OPEN **OPEN**

FUNCTION: To initiate the processing of both input and output files.

FORMAT:

```
          {I-O   }
          {INPUT }
OPEN      {OUTPUT} file-name
```

RULES:

1. A file must be opened before it can be read, written or closed.

2. The OPEN statement does not cause a data transfer to or from the file.

3. In the output SEQUENTIAL ACCESS mode, if the file does not exist it is created.

4. In the RANDOM ACCESS mode, the file must already exist.

5. The I-O (INPUT-OUTPUT) option applies to DISK files only.

EXAMPLE:

```
0700 BEGIN.
0701     OPEN OUTPUT NEW-PAYROLL-MASTER-FILE.
0702     OPEN INPUT OLD-PAYROLL-MASTER-FILE.
0703     OPEN OUTPUT LISTING.
```

PERFORM **PERFORM**

FUNCTION: To depart from the normal sequence of procedures in order to execute one statement, or a sequence of statements, and then return to the normal sequence.

FORMAT 1:

```
                                    {THROUGH}
      PERFORM procedure-name-1 [{THRU} procedure-name-2]
```

FORMAT 2:

```
PERFORM procedure-name-1 [{THRU} procedure-name-2]
                                  {integer-1   }
                                  {identifier-1} TIMES
```

FORMAT 3:

```
PERFORM procedure-name-1 [{THRU} procedure-name-2]
                                         {OR }
                  UNTIL condition-1 {AND} condition-2
```

RULES:

1. Each identifier represents a numeric elementary item described in the Data Division. In format 2, identifier-1 must be described as a numeric integer.

2. The words THRU and THROUGH are equivalent.

3. When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. This transfer of control occurs only once for each execution of a PERFORM statement. For those cases where a transfer of control to the named procedure does take place, an implicit transfer of control to the next executable statement following the PERFORM statement is established as follows:
a. If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return is after the last statement of procedure-name-1.
b. If procedure-name-1 is a section-name and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in

procedure-name-1.
c. If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.
d. If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

4. There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. In particular, GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement, to which all of these paths must lead.

5. If control passes to these procedures by means other than a PERFORM statement, control will pass through the last statement of the procedure to the next executable statement as if no PERFORM statement mentioned these procedures.

6. The PERFORM statements operate as follows with rule 5 above applying to all formats:
a. Format 1 is the basic PERFORM statement. A procedure referenced by this type of PERFORM statement is executed once and then control passes to the next executable statement following the PERFORM statement.
b. Format 2 is the PERFORM...TIMES. The procedures are performed the number of times specified by integer-1 or by the initial value of the data item referenced by identifier-1 for that execution. If, at the time of execution of a PERFORM statement, the value of the data item referenced by identifier-1 is equal to zero or is negative, control passes to the next executable statement following the PERFORM statement. Following the execution of the procedures the specified number of times, control is transferred to the next executable statement following the PERFORM statement.

During execution of the PERFORM statement, references to identifier-1 cannot alter the number of times the procedures are to be executed from that which was indicated by the initial value of identifier-1.
c. Format 3 is the PERFORM...UNTIL. The specified procedures are performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the next executable statement after the PERFORM statement. If the condition is true when

the PERFORM statement is entered, no transfer to procedure-name-1 takes place, and control is passed to the next executable statement following the PERFORM statement.
7. If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements may not have a common exit.

8. A PERFORM statement that appears in a section that is not in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:
a. Sections and/or paragraphs wholly contained in one or more non-independent segments.
b. Sections and/or paragraphs wholly contained in a single independent segment.

9. A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:
a. Sections and/or paragraphs wholly contained in one or more non-independent segments.
b. Sections and/or paragraphs wholly contained in the same independent segment as the PERFORM statement.

EXAMPLE:

```
0750       PERFORM CALCULATE-PAY THRU PARA-END.
0751       PERFORM MAIN-PROGRAM.
0791       PERFORM CHECK-ROUTINE 5 TIMES.
0799       PERFORM TEST-ROUTINE UNTIL CODE-1 > T-CODE.
0800       PERFORM PARA-1 THRU PARA-2
0801         UNTIL A = B OR X=Y AND Z=W.
```

READ **READ**

FUNCTION: To make available the next logical record from an open file.

FORMAT:

```
                              {AT END     }
READ file-name RECORD         {INVALID KEY} imperative-statement
```

RULES:

1. A file must be OPENed before it can be read.

2. The AT END statement must be used for SEQUENTIAL files and is executed at the end of the file.

3. The INVALID KEY statement must be used with RANDOM files and if executed the data in the user area is unspecified.

4. The number of the requested record in a RANDOM file must be placed in the RELATIVE KEY before the READ statement is executed.

5. When reading variable length delimited files the record area should be cleared to spaces before each read because the data in the user record area to the right of the last valid character of the input item is unspecified i.e., what ever data was there from before the read will be there.

6. When reading variable length delimited files the TAB (09H) characters created by some text editors are not expanded to avoid conflict with packed decimal (COMP-3) data type. If tab characters are used they can be expanded by CP/M's PIP command using the "T" option before processing by COBOL programs.

EXAMPLE:

```
0800 READ-ROUTINE.
0801     MOVE SPACES TO PAYROLL-RECORD.
0802     READ OLD-PAYROLL-MASTER-FILE
0803       AT END GO TO OLD-EOJ-ROUTINE.
0900 READ-RANDOM.
    * if you wanted record 100 in a random file
0901     MOVE 100 TO KEY3-RECORD-NUMBER.
0902     READ IN-RANDOM-FILE
0903          INVALID KEY DISPLAY "INVALID KEY".
```

REWRITE **REWRITE**

FUNCTION: To replace a record existing in a disk file.

FORMAT:

REWRITE record-name [INVALID KEY imperative-statement]

RULES:

1. The file must have been opened in the I-O mode.

2. The record-name must be the name of a logical record in the FILE SECTION of the DATA DIVISION.

3. The REWRITE statement must have been preceeded by a successful READ statement in the SEQUENTIAL ACCESS MODE as it is this logical record that is replaced.

4. The INVALID KEY clause must be used for RANDOM files.

5. For files accessed in RANDOM access mode the record logically replaces the record specified by the contents of the RELATIVE KEY data item associated with the file.

EXAMPLE:

```
    * in-file is the file name and in-rec is a record name
    * for the file
0097 SEQ-REWRITE.
0098     READ IN-FILE RECORD AT END GO TO EOJ.
0099     MOVE NEW-DATA TO IN-REC.
0100     REWRITE IN-REC.
0200 RANDOM-REWRITE.
0201     MOVE 100 TO KEY-REL.
0202     REWRITE NEW-REC INVALID KEY GO TO ERROR.
```

STOP **STOP**

FUNCTION: To cause permanent or temporary suspension of the execution of the object program.

FORMAT:

```
          {literal}
STOP      {RUN    }
```

RULES:

1. All files should be closed before a STOP RUN statement is issued.

2. The STOP RUN statement must be the last statement executed in the program as the operating system takes control after execution.

3. The literal is displayed on the console device and waits for a code followed by a carriage return to be entered as follows:

C <CR>= continue
E <CR>= exit to operating system.

EXAMPLE:

```
0900 END-OF-JOB. STOP RUN.
0500 ERR. STOP "SIZE ERROR ENTER C TO CONTINUE".
```

SUBTRACT **SUBTRACT**

FUNCTION: To subtract one numeric data item from another and set the value of an item equal to the result.

FORMAT:

```
                {literal-1   }       {literal-2   }
SUBTRACT        {identifier-1} FROM  {identifier-2}

[GIVING identifier-3] [ROUNDED]

[, ON SIZE ERROR imperative-statement]
```

RULES:

1. Each identifier must refer to a elementary numeric item, except identifier-3 which may refer to a elementary numeric edited item.

2. The composite of operands must not contain more than 18 digits.

3. An identifier can only reference an elementary item.

4. Each operand can contain an operational sign and an implied decimal point.

5. Operands are aligned according to implied decimal points.

6. ROUNDED performs a test to see if right truncation will occur and, if it will, adjusts the result by adding 1 if the truncated digit is 5 or greater.

7. ON SIZE ERROR performs a test to see if overflow has occurred and, if it has, executes the imperative-statement.

EXAMPLE:

```
0870     SUBTRACT TAXES FROM GROSS-PAY GIVING NET-PAY
0871       ROUNDED ON SIZE ERROR GO TO TAX-ERR-ROUTINE.
```

WRITE **WRITE**

FUNCTION: To release a record to an output file. To allow for vertical positioning if the output device is a printer.

FORMAT:

```
                                          {PAGE          }
                                          {              }
  WRITE record-name [BEFORE ADVANCING     {integer LINE  }]
                                          {        LINES }

      WRITE record-name [INVALID KEY imperative-statement]
```

RULES:

1. The record-name must be the name of a logical record in the FILE SECTION of the DATA DIVISION.

2. The reserved word PAGE issues a standard form feed (0CH) control character to the device driver.

3. Integer LINES issues the specified number of carriage return line feeds.

4. The INVALID KEY clause must be used for RANDOM files.

5. The requested record number must be placed in the RELATIVE KEY before writing to a RANDOM file.

EXAMPLE:

```
0900 P-ROUTINE.
0901     WRITE PRINT-LINE BEFORE ADVANCING 2 LINES.
0902     MOVE SPACES TO PRINT-LINE.
0903     WRITE PRINT-LINE BEFORE ADVANCING PAGE.
1000 WRITE-RANDOM.
1001     MOVE 1000 TO KEY3.
1002     WRITE D-RANDOM-OUT
1003           INVALID KEY DISPLAY "INVALID KEY".
1050 SEQ-WRITE.
1051     WRITE D-REC.
```

SECTION V

INTRODUCTION

COBOL (Common Business Oriented Language) is an easy to learn programming language that has been used for business applications since the early 1960s. Because COBOL uses simple English statements, programmers need not be concerned with the unique details of the computer and can therefore concentrate their energies on solving particular business problems. NEVADA COBOL is an up-to-date subset designed for small businesses using microprocessors.

A few years ago most programmers thought in terms of creating a program as if they were creating a work of art, which -like a painting- could never be changed. Today, programs are engineered and built the way a construction company designs and builds a home. In both approaches the objective is the completion of the program, but the "building a program" concept suggests an element of productivity and modularity.

In order to start any building project one must first know what the objectives are. In the case of writing a program the objective is to produce an efficient, easy to maintain, easy to change , easy to understand, easy to read program that solves the problem.

SIMPLICITY

In our opinion SIMPLE IS BETTER. We often see COBOL, FORTRAN, BASIC, and other languages so complicated with nested statements and abbreviated labels, that the original programmer can't understand the program six months later. In the business world it is important that the program be easily understood, because programs and businesses are always changing. They are never complete. They are conceived and born, they grow and change, and finally pass on. It is important that the program doesn't pass on before its time. By writing programs that are easily transported to other equipment and easy to change we can increase their longevity, thus increasing their value. So write and test your COBOL programs on a microprocessor and then run them in production on a giant machine when the time comes. Today, the greatest cost of a computer system is not the hardware, it's the software! That's because software is 95% labor. So make your labor efficient by writing simple programs to solve complex problems.

GLOSSARY

INTRODUCTION

The terms that follow are defined in accordance with their meaning as used in this document describing NEVADA COBOL and may not have the same meaning for other languages.

These definitions are also intended to be either reference material or introductory material to be reviewed prior to reading the detailed language specifications. For this reason, these definitions are, in most instances, brief and do not include detailed syntactical rules. Also, some terms defined here are not yet implemented in NEVADA COBOL.

DEFINITIONS

Abbreviated combined relation condition - The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

Access Mode - The manner in which records are to be operated upon within a file.

Actual Decimal Point - The physical representation, using either of the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

Alphabetic Character - A character that belongs to the following set of letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, and the space.

Alphanumeric character - Any character in the computer's character set.

Alternate Record Key - A key, other than the prime record key, whose contents identify a record within an indexed file.

Arithmetic Expression - An arithemetic expression can be an identifier or a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

Arithmetic Operator - A single character, or a fixed two-character combination, that belongs to the following set:

character	meaning
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

Ascending Key - A key upon the values of which data is ordered starting with the lowest value of key up to the highest value of key in accordance with the rules for comparing data items.

Assumed Decimal Point - A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

At End Condition - A condition caused:

1. During the execution of a READ statement for a sequentially accessed file.
2. During the execution of a RETURN statement, when no next logical record exists for the associated sort or merge file.

Block - A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either continued within the block or that overlap the block. The term is synonymous with physical record.

Called Program - A program which is the object of a CALL statement.

Calling Program - A program which executes a CALL to another program.

Character - A basic indivisible unit of the language.

Character Position - A character position is the amount of physical storage required to store a single standard data format character described as usage is DISPLAY.

Character-string - A sequence of contiguous characters which form a COBOL word, a literal, a PICTURE character-string, or a comment-entry.

Class condition - The proposition, for which a truth value can be determined, that the content of an item is wholly

alphabetic or is wholly numeric.

Clause - A clause is an ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

COBOL Character Set - The complete COBOL character set consists of the 51 characters listed below:

Character	Meaning
0,1,....9	digit
A,B,....Z	letter
	space (blank)
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

COBOL Word - (See Word)

Collating Sequence - The sequence in which the characters that are acceptable in a computer are ordered for purposes of sorting, merging, and comparing.

Column - A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

Combined Condition - A condition that is the result of connecting two or more conditions with the 'AND' or the 'OR' logical operator.

Comment-Entry - An entry in the Identification Division that may be any combination of characters from the COBOL character set.

Comment Line - A source program line represented by an asterisk in the indicator area of the line and any character from the computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a stroke (/) in the indicator area of the line and any

characters from the computer's character set in area A and area B of that line causes page ejection prior to printing the comment.

Compile time - The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

Compiler Directing Statement - A statement, beginning with a compiler directing verb, that causes the compiler to take specific action during compilation.

Computer-Name - A system-name that identifies the computer upon which the program is to be compiled or run.

Condition - A status of a program at execution time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2,...) appears in these language specifications in or in reference to 'condition' (condition-1, condition-2, ...) of a gereral format, it is a conditional expression consisting of either a simple condition or a combined condition consisting of the syntactically correct combination of simple conditions, logical operators,and parentheses, for which a truth value can be determined.

Condition-Name - A user-defined word assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable may possess.

Condition-Name Condition - The proposition, for which truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

Conditional Expression - A simple condition or a complex condition specified in an IF, or PERFORM statement.

Conditional Statement - A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

Conditional Variable - A data item one or more values of which has a condition-name assigned to it.

Configuration Section - A section of the Environment Division that describes overall specifications of source and object computers.

Connective - A reserved word that is used to:

1. Associate a data-name, paragraph-name,

condition-name, or text-name with its qualifier.
2. Link two or more operands written in a series.
3. Form conditions.

Contiguous Item - Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

Counter - A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

Currency Sign - A character '$' of the COBOL character set.

Currency Symbol - The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present in a COBOL source program, the currency symbol is identical to the currency sign.

Current Record - The record which is available in the record area associated with the file.

Current Record Pointer - A conceptual entity that is used in the selection of the next record.

Data Clause - A clause that appears in a data description entry in the Data Division and provides information describing a particular attribute of a data item.

Data Description Entry - An entry in the Data Division that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

Data Item - A character or a set of contiguous characters (excluding in either case literals) defined as a unit of data by the COBOL program.

Data-Name - A user-defined word that names a data item described in a data description entry in the Data Division. When used in the general formats, 'data-name' represents a word which can neither be subscripted, nor indexed unless specifically permitted by the rules for that format.

Debugging Line - A debugging line is any line with 'D' in the indicator area of the line.

Declaratives - A set of one or more special purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the key word

DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one or more associated paragraphs.

Declarative-Sentence - A compiler-directing sentence consisting of a single USE statement terminated by the separator period.

Delimiter - A character or a sequence of contiguous characters that identify the end of a string of characters and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

Descending Key - A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

Digit Position - A digit position is the amount of physical storage required to store a single digit. This amount may vary depending on the usage of the data item describing the digit position.

Division - A set of zero, one or more sections of paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. There are four (4) divisions in a COBOL program: Identification, Environment, Data, and Procedure.

Division Header - A combination of words followed by a period and a space that indicates that beginning of a division. The division headers are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION [USING data-name-1 ...].
```

Dynamic Access - An access mode in which specific logical records can be obtained from or placed into a mass storage file in a non-sequential manner (see Random Access) and obtained from a file in a sequential manner (see Sequential Access), during the scope of the same OPEN statement.

Editing Character - A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
$	currency sign
,	comma (decimal point)
.	period (decimal point)
/	stroke (virgule, slash)

Elementary Item - A data item that is described as not being further logically subdivided.

End of Procedure Division - The physical position in a COBOL source program after which no further procedures appear.

Entry - Any descriptive set of consecutive clauses terminated by a period and written in the Identification Division, Environment Division, or Data Division of a COBOL source program.

Environment Clause - A clause that appears as part of an Environment Division entry.

Execution Time - (See Object Time).

Extended Mode - The state of a file after execution of an OPEN statement, with the EXTEND phrase specified, for that file and before the execution of a CLOSE statement for that file.

Figurative Constant - A compiler generated value referenced through the use of certain reserved words.

File - A collection of records.

File Clause - A clause that appears as part of a File description (FD).

FILE-CONTROL - The name of an Environment Division paragraph in which the data files for a given source program are declared.

File Description Entry - An entry in the File Section of the Data Division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

File-Name - A user-defined word that means a file described in a file description entry or a sort-merge file description entry within the File Section of the Data Division.

File Organization - The permanent logical file structure established at the time that a file is created.

File Section - The section of the Data Division that contains file description entries and sort-merge file description entries together with their associated record descriptions.

Format - A specific arrangement of a set of data.

Group Item - A named contiguous set of elementary or group items.

High Order End - The leftmost character of a string of characters.

I-O-CONTROL - The name of an Environment Division paragraph in which object program requirements for specific input-output techniques, rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

I-O-MODE - The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement for that file.

Identifier - A data-name, followed as required, by the syntactically correct combination of qualifiers, subscripts, and indices necessary to make unique reference to a data item.

Imperative Statement - A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements.

Index - A computer storage position or reqister, the contents of which represent the identification of a particular element in a table.

Index Data Item - A data item in which the value associated with an index-name can be stored in a form specified by the implementor.

Index-Name - A user-defined word that names an index associated with a specific table.

Indexed Data-Name - An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

Indexed File - A file with indexed organization.

Indexed Organization - The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

Input File - A file that is opened in the input mode.

Input Mode - The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement for that file.

Input-Output File - A file that is opened in the I-O mode.

Input-Output Section - The section of the Environment Division that names the files and the external media required by an object program and which provides information required for transmission and handling of data during execution of the object program.

Integer - A numeric literal or an numeric data item that does not include any character positions to the right of the assumed decimal point. Where the term 'integer' appears in general formats, integer must not be a numeric data item, and must not be signed, nor zero unless explicitly allowed by the rules of that format.

Invalid Key Condition - A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

Key - A data item which identifies the location of a record, or a set of data items which serve to identify the ordering of data.

Key of Reference - The key, either prime or alternate, currently being used to access records within and indexed file.

Key Word - A reserved word whose presence is required when the format in which the word appears is used in a source program.

Language-Name - A system-name that specifies a particular programming language.

Level Indicator - Two alphabetic characters that identify a

specific type of file or a position in hierarchy.

Level-Number - A user-defined word which indicates the position of a data item in the hierarchical structure of a logical record or which indicates special properties of a data description entry. A level-number is expressed as a one or two digit number. Level-numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77, and 88 identify special properties of a data description entry.

Library-Name - A user-defined word that names a COBOL library that is to be used by the compiler for a given source program compilation.

Library Text - A sequence of character-strings and/or separators in a COBOL library.

Line Number - An integer that denotes the vertical position of a line on a page.

Linkage Section - The section in the Data Division of the called program that describes data items available from the calling program. These data items may be referred to by both the calling and called program.

Literal - A character-string whose value is implied by the ordered set of characters comprising the string.

Logical Operator - One of the reserved words AND, OR or NOT. In the formation of a condition, both or either of AND and OR can be used as logical connectives. NOT can be used for logical negation.

Logical Record - The most inclusive data item. The level-number for a record is 01.

Low Order End - The rightmost character of a string of characters.

Mass Storage - A storage medium on which data may be organized and maintained in both a sequential and nonsequential manner.

Mass Storage File - A collection of records that is assigned to a mass storage medium.

Mnemonic-Name - A user-defined word that is associated in the Environment Division with a specified implementor-name.

Native Character Set - The implementor-defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

Native Collating Sequence - The implementor-defined

collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

Negated Simple Condition - The 'NOT' logical operator immediately followed by a simple condition.

Next Executable Sentence - The next sentence to which control will be transferred after execution of the current statement is complete.

Next Executable Statement - The next statement to which control will be transferred after execution of the current statement is complete.

Next Record - The record which logically follows the current record of a file.

Noncontiguous Items - Elementary data items, in the Working-Storage and Linkage Section, which bear no hierarchic relationship to other data items.

Nonnumeric Item - A data item whose description permits its contents to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

Nonnumeric Literal - A character-string bounded by quotation marks. The string of characters may include any character in the computer's character set. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.

Numeric Character - A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Numeric Item - A data item whose description restricts its contents to a value represented by characters chosen from the digits '0' through '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign.

Numeric Literal - A literal composed of one or more numeric characters that also contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be te leftmost character.

OBJECT-COMPUTER - The name of an Environment Division paragraph in which the computer environment, within which the object program is executed, is described.

Object of Entry - A set of operands and reserved words, within a Data Division entry, that immediately follows the subject of the entry.

Object Program - A set or group of executable machine language instructions and other material designed to

interact with data to provide problem solutions. In this context, an object program is generally the machine language result of the operation of a COBOL compiler on a source program. Where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'object program'.

Object Time - The time at which an object program is executed.

Open Mode - The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O or EXTEND.

Operand - Whereas the general definition of operand is 'that component which is operated upon', for the purposes of this publication, any lowercase word (words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

Operational Sign - An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

Optional Word - A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

Output File - A file that is opened in either the output mode or extend mode.

Output Mode - The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified for that file and before the execution of a CLOSE statement for that file.

Page - A vertical division of a report representing a physical separation of report data, the separation being based on internal reporting requirements and/or external characteristics of the reporting medium.

Paragraph - In the Procedure Division, a paragraph-name followed by a period and a space and by zero, one, or more sentences. In the Identification and Environment Divisions, a paragraph header followed by zero, one, or more entries.

Paragraph Header - A reserved word, followed by a period and a space that indicates the beginning of a paragraph in the Identification and Environment Divisions. The permissible headers are:

In the Identification Division:

```
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

In the Enviroment Division:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O-CONTROL.
```

Paragraph-Name - A user-defined word that identifies and begins a paragraph in the Procedure Division.

Phrase - A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

Prime Record Key - A key whose contents uniquely identify a record within an indexed file.

Procedure - A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

Procedure-Name - A user-defined word which is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name or a section-name.

Program-Name - A user-defined word that identifies a COBOL source program.

Punctuation Character - A character that belongs to the following set:

Character	Meaning
,	comma
;	semicolon
.	period
"	quotation mark
(	left parenthesis
)	right parenthesis
=	equal sign

Random Access - An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from or placed into a relative or indexed file.

Record - (See Logical Record).

Record Area - A storage area allocated for the purpose of processing the record described in a record description entry in the File Section.

Record Description Entry - The total set of data description entries associated with a particular record.

Record Key - A key, either the prime record key or an alternate record key, whose contents identify a record within an indexed file.

Record-Name - A user-defined word that names a record described in a record description entry in the Data Division.

Reference Format - A format that provides a standard method for describing COBOL source programs.

Relation Character - A character that belongs to the following set:

Character	Meaning
>	greater than
<	less than
=	equal to

Relation Condition - The proposition, for which a truth value can be determined, that the value of an arithmetic expression or data item has a specific relationship to the value of another arithmetic expression or data item.

Relational Operator - A reserved word, a relation character, or group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meaning are:

Relational operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater

```
IS [NOT] LESS THAN
IS [NOT] <                          Less than or not less than

IS [NOT] EQUAL TO
IS [NOT] =                          Equal to or not equal to
```

Relative File - A file with relative organization.

Relative Key - A key whose contents identify a logical record in a relative file.

Relative Organization - The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

Reserved Word - A COBOL word specified in the list of words which may be used in COBOL source programs, but which must not appear in the programs as user-defined words or system-names.

Routine-Name - A user-defined word that identifies a procedure written in a language other than COBOL.

Section - A set of zero, one, or more paragraphs or entries, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

Section Header - A combination of words followed by a period and a space that indicates the beginning of a section in the Environment, Data and Procedure Division.

In the Environment and Data Divisions, a section header is composed of reserved words followed by a period and a space. The permissible section headers are:

```
In the Environment Division:
        CONFIGURATION SECTION.
        INPUT-OUTPUT SECTION.

In the Data Division:
        FILE SECTION.
        WORKING-STORAGE SECTION.
        LINKAGE SECTION.
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a segment-number (optional), followed by a period and a space.

Section-Name - A user-defined word which names a section in the Procedure Division.

Segment-Number - A user-defined word which classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only characters

'0', '1', ..., '9'. A segment-number may be expressed either as one or two digit number.

Sentence - A sequence of one or more statements, the last of which is terminated by a period followed by a space.

Separator - A punctuation character used to delimit character-strings.

Sequential Access - An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

Sequential File - A file with sequential organization.

Sequential Organization - The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

Sign Condition - The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

Simple Condition - Any single condition chosen from the set:
- relation condition
- class condition
- condition-name condition
- sign condition

SOURCE-COMPUTER - The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

Source Program - Although it is recognized that a source program may be represented by other forms and symbols, in this document it always refers to a syntactically correct set of COBOL statements beginning with an Identification Division and ending with the end of the Procedure Division. In contexts where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'source program'.

Special Character - A character that belongs to the following set:

Character	Meaning
+	plus sign
-	minus sign
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)

"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

Special-Character Word - A reserved word which is an arithmetic operator or a relation character.

SPECIAL-NAMES - The name of an Environment Division paragraph in which implementor-names are related to user specified mnemonic-names.

Special Registers - Compiler generated storage areas whose primary use is to store information produced in conjunction with the user of specific COBOL features.

Standard Data Format - The concept used in describing the characteristics of data in a COBOL Data Division under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

Statement - A syntactically valid combination of words and symbols written in the Procedure Division beginning with a verb.

Subject of Entry - An operand or reserved word that appears immediately following the level indicator or the level-number in a Data Division entry.

Subprogram - (See Called Program).

Subscript - An integer whose value identifies a particular element in a table.

Subscripted Data-Name - An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

System-Name - A COBOL word which is used to communicate with the operating environment.

Table - A set of logically consecutive items of data that are defined in the Data Division by means of the OCCURS clause.

Table Element - A data item that belongs to the set of repeated items comprising a table.

Text-Name - A user-defined word which identifies library text.

Text-Word - Any character-string or separator, except space, in a COBOL library.

Truth Value - The representation of the result of the evaluation of a condition in terms of one of two values
true
false

Unary Operator - A plus (+) or a minus (-) sign, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression of +1 or -1 respectively.

User-Defined Word - A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

Variable - A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

Verb - A word that expresses an action to be taken by a COBOL compiler or object program.

Word - A character-string of not more than 30 characters which forms a user-defined word, a system-name, or a reserved word.

Working-Storage Section - The section of the Data Division that describes working storage data items, composed either of noncontiguous items or of working storage records or of both.

77-level-Description-Entry - A data description entry that describes a noncontiguous data item with the level-number 77.

LANGUAGE CONCEPTS

CHARACTER SET

The standard (ANSI-1974) COBOL character set consists of the following 51 characters:

0,1,...,9	digit
A,B,...,Z	letter
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

USER-DEFINED WORDS

A word contains not more than 30 characters from the set {A-Z,0-9,and -}. A user-defined word cannot begin or end with a (-) hyphen and must contain at least one alphabetic character.

PUNCTUATION

The period, comma, or semicolon must immediately follow the last character of a word and be followed by a space. The comma and the semicolon are interchangeable. The left parenthesis must not be followed by a space and the right parenthesis must not be preceeded by a space.

EXAMPLE:

```
1234      MOVE MONEY (10) TO SAVING-AND-LOAN.
```

LITERALS

NUMERIC

1. A numeric literal must contain from 1 to 18 characters.
2. A numeric literal cannot be bounded by quotation marks.
3. A numeric literal can contain a minus (-) sign as its left most character. If there is no minus sign the literal is assumed to be positive.
4. A decimal point (.) can appear within a numeric literal. If there is no decimal point the literal is assumed to be an integer. The decimal point cannot be to the immediate left of a minus sign or as the right most character in the literal.

NON-NUMERIC

1. A non-numeric literal must be bounded by quotation marks ("). The word quote cannot be used.
2. A non-numeric literal must contain from 1-120 characters.
3. Each imbedded pair of quotation marks creates one quotation mark in the literal.
4. A second set of quotation marks (") can be used to bound hexadecimal values. Each hexadeciamal value can be separated by a comma. Hexadecimal characters are from the set 0-9 and A-F.

EXAMPLE:

```
    * note the following 2 lines would display ABC
0051            DISPLAY "ABC".
    * the following is a hexadecimal literal for ABC
0050 GRAPHICS. DISPLAY ""41,42,43"".
    * the following line would display a single quotation
    * mark because of the imbedded pair of quotation marks.
0052            DISPLAY """".
0053            DISPLAY "LONG LINE CONTINUES TO NEXT LINE
0054-     " QUOTE IN COL 10 & - IN COL 5 IS NECESSARY".
    * numeric literals
0060 MATH.      ADD 1 TO TOTAL-ITEMS.
0061            ADD 3.75 TO AMT-SAVED.
```

FIGURATIVE CONSTANTS

1. The reserved words ZERO, ZEROS or ZEROES represents one or more occurrences of the character zero (0).

2. The reserved words SPACE or SPACES represents one or more occurrences of the character space (blank).

3. The reserved words QUOTE or QUOTES represents one or more occurrences of the character quote (").

4. The reserved words HIGH-VALUE or HIGH-VALUES represents one or more occurrences of the character FF Hexadecimal.

5. The reserved words LOW-VALUE or LOW-VALUES represents one or more occurrences of the character 00 Hexadecimal.

6. The reserved word ALL "literal" represents one or more occurrences of the single non-numeric literal character.

EXAMPLE:

```
0001      MOVE ALL "X" TO CUSTOMER-NAME.
0002      IF CUSTOMER-NAME IS EQUAL TO ALL "X"
0003        GO TO PRT-ALIGNMENT.
0004      MOVE HIGH-VALUES TO OUT-RECORD.
```

SUBSCRIPTING

A subscript is an integer whose value determines the ordinal position of an item in a table. This value can cross record boundaries (4095) for large tables (30K - 40K+) in working-storage by having a series of tables and referencing the first one with a subscript value which points to an item in the second, third, ... table. However, if the subscript value is such that it crosses a record boundary and no table follows, then there is no error indication and the results are unspecified.

EXAMPLE:

```
0001 WORKING-STORAGE.
0002 01 TABLE.
0003    02 FILLER PIC X(9) VALUE "JANUARY  ".
0004    02 FILLER PIC X(9) VALUE "FEBRUARY ".
0005    02 FILLER PIC X(9) VALUE "MARCH    ".
0006    02 FILLER PIC X(9) VALUE "APRIL    ".
0007    02 FILLER PIC X(9) VALUE "MAY      ".
0008    02 FILLER PIC X(9) VALUE "JUNE     ".
0009    02 FILLER PIC X(9) VALUE "JULY     ".
0010    02 FILLER PIC X(9) VALUE "AUGUST   ".
0011    02 FILLER PIC X(9) VALUE "SEPTEMBER".
0012    02 FILLER PIC X(9) VALUE "OCTOBER  ".
0013    02 FILLER PIC X(9) VALUE "NOVEMBER ".
0014    02 FILLER PIC X(9) VALUE "DECEMBER ".
0015 01 M-TBL REDEFINES TABLE.
0016    02 MONTH OCCURS 12 TIMES PIC IS X(9).
0017 PROCEDURE DIVISION.
0018 DATE-PARA.
0019    MOVE MONTH (MONTH-NO) TO PRT-MONTH-NAME.
0020*
0021*  other examples.
0022*
1234     MOVE ITEM TO TABLE (7).
1235     MOVE TABLE (7) TO PRINT-ITEM-SEVEN.
1236     MOVE 007 TO INDEX-1.
1237     MOVE TABLE (INDEX-1) TO PRINT-ITEM-SEVEN.
1238     MOVE ZEROS TO TABLE (3000).
1239     MOVE SPACES TO PRINT-LINE.
1240* if both BIN-1 and X1 are binary data types then at run
1240*    time the math is 20 times faster than decimal.
1241     ADD BIN-1 TO X1.
1242     IF ITEM (X1) IS EQUAL TO SPEED GO TO FAST.
1243     MOVE ALL "A" TO PRINT-LINE.
```

SYMBOLS AND CONVENTIONS

The symbology used in this manual is the standard COBOL symbology and the carriage return symbol.

1. Lower-case characters represent data to be supplied by the programmer.

2. Underlined UPPER-CASE characters are key words that must be used.

3. Upper-case characters that are not underlined are optional reserved words.

4. Braces {} indicate that a choice must be made.

5. Square brackets [] indicate optional information.

6. Ellipsis (...), three consecutive periods, indicate that preceeding clauses can be repeated.

7. The carriage return is indicated by <CR>.

SECTION VI

OPERATING PROCEDURES

HARDWARE REQUIRED

1. 8080/Z80/8085 microprocessor.
2. A minimun of 32K of RAM for the compiler.
3. Any disk drive.
4. Video display and keyboard.

SOFTWARE REQUIREMENTS

Digital Research Corp's CP/M operating system. CP/M is a trademark of Digital Research Corp. Any text editor that terminates a line with a carriage return line feed.

FILES ON THE DISTRIBUTION DISKETTE

CC.COM is the COBOL COMPILER and is always on the default drive at compile time.

W4.COM is a random file and is always on the default drive at compile time.

W5.CBL is the error message file and is always on the default drive at compile time. This file is a standard text file that may be changed by the user.

RUN.COM is the run time loader/subroutine code and can be on any drive. It is only used at run time.

ERRORS.COM displays the error report from the default drive. This program is used to re-display the error report from the last compile but is not required for compiling.

RENUMBER.CBL is a COBOL source code program that must be compiled before it can be used. It renumbers COBOL source programs.

CONFIG.CBL is a COBOL source code program that must be compiled before it can be used. It will configure the CRT for line length, BIOS and the delete character, etc.

CONVHEX.COM converts (.HEX) files to (.OBJ) files. This program is executed as follows:

```
CONVHEX file-name[.HEX].
```

The program will create the output file if necessary with the same file-name and type (.OBJ).

FILES NOT DISTRIBUTED BUT WILL BE CREATED AT COMPILE TIME.

W1.WRK is a work file and will be created on the default drive or the assigned drive at

compile time.
W3.WRK is the error work file and will be created on the default drive at compile time.

GETTING STARTED

NEVADA COBOL is distributed on a DATA DISK without the CP/M operating system, so, first, prepare a CP/M system's diskette for use as your NEVADA COBOL operations diskette. On 5 1/4 inch diskettes you may have to remove (use the CP/M ERA command) most of the files in order to make room for the COBOL files.

Then insert the newly created CP/M diskette in disk drive A, and insert the NEVADA COBOL diskette in drive B and type (ctl-c) to initialize CP/M. Now copy all the files from the COBOL diskette onto the CP/M diskette.

```
PIP A:=B:*.*[VO]
```

If you get a write error message from CP/M during the PIP operation it usually means the disk is full and you should erase more files from the operational diskette.

At this point, put the NEVADA COBOL diskette in a safe place. You will not need it unless something happens to your operations diskette. By the way, back up your operations diskette with a copy each week! If your system malfunctions you can then pat yourself on the back for having a safe back up copy of your work.

Now, boot up the newly created NEVADA COBOL operations diskette. Notice that CP/M display's the amount of memory for which this version of CP/M has been specialized. Compiling and executing of COBOL programs should be done on the same CP/M version or one of larger memory unless your COBOL programs are given an upper address limit. Also, do not write protect this operations diskette because during a compile data will be written onto it.

BUILDING A PROGRAM

The first step is to create a COBOL source program file. This can be done by using a text editor like NEVADA EDIT and copying an existing COBOL program file like RENUMBER.CBL and creating a new program file like MYPROG.CBL. Then modify MYPROG.CBL as required. This saves keying time as well as avoiding the possibility of misspelling required key words. Each line must be terminated with a carriage return line feed. This is automatically done with NEVADA EDIT or ED (text editor that comes with CP/M).

You can use the RENUMBER program to automatically number or renumber the source program. All source programs must be type CBL.

COBOL CODING FORMAT
optional

	ANSI-1974 column	NEVADA COBOL column
Sequence number	1-6	1-4
Indicator area	7	5
A-field	8-11	6-9
B-field	12-72	10-70

1. The user can use either format as the compiler looks at the first line of the source program and determines either 4 position or 6 position line numbers are used. When converting ANSI programs to NEVADA COBOL (or vice versa) adjust the sequence number by two positions and the other columns will align properly. We felt that 9999 sequence numbers would be enough for microprocessors and would also be compatible with other microprocessor software (ie. RENUMBER, edits, etc.).

2. The indicator area can contain only the following:
   - `*` which indicates a comments line.
   - `/` which indicates a comments line after head of forms.
   - SPACE which indicates a standard COBOL statement line.
   - `D` which indicates a Debugging line.
   - `-` which indicates a continuation line. When a non-numeric literal is continued a quotation mark (") must also appear in column 10.

All other characters are flagged by the compiler and are treated as comment lines.

3. Sequential line numbers are required because all errors are referenced by line number.

4. Each line must be terminated by a carriage return line feed (0D0A hex).

EXAMPLE:
```
Columns
12345678901234567890123456789O
0001 IDENTIFICATION DIVISION.
0002*  this is a comment line. the next line is blank
0003*
```

COMPILING A PROGRAM

To compile a program simply type CC file-name. A copyright message will appear until the error report is displayed or until the successful compile message is displayed. Using the error report line number/message correct the source and recompile if necessary. The compile can be interrupted (aborted) by pressing the CONTROL-C keys. In the following examples the CP/M operating system gives the prompt A> and the user types in the rest of the line.

```
A>CC RENUMBER<CR>
```

in the above case, the source file RENUMBER.CBL is on the default drive. The object code file will be created if necessary on the default drive with the file-name of RENUMBER.OBJ. The work file W1 will be created if necessary on the default drive.

A>CC SOURCE.BBB<CR>

in the above example, all assignable files are on disk drive B. The type field is used for drive assignment. The first position is for the source file, the second position is for the object file and the third position is for W1 a large work file. All source files must be type '.CBL'.

A>CC CONFIG.ABB<CR>

in the above case, the source file CONFIG.CBL is on drive (A) and the object file CONFIG.OBJ will be created if necessary on drive (B) as will the work file W1.WRK.

Warning: If you forget and type CC file-name.CBL you will get a CP/M BDOS Select ERROR. Because the compiler will look for drive C: or L: in error.

EXECUTING A PROGRAM

Once the object file has been produced, the program can be executed by simply typing RUN file-name. The run time package is called RUN and lives in memory locations 100H to 2EFFH. It contains a special loader and all the required run time subroutines. Execution of the program can be interrupted (aborted) by pressing the CONTROL C keys.

A>RUN u:OBJECT<CR>
in the above case, RUN is on the current logged-in drive A and OBJECT is on some other disk drive u:

B>RUN A:PAYROLL<CR>
in the above case RUN.COM is on the current logged-in drive (B) and PAYROLL.OBJ is on drive (A).

A>RUN RENUMBER<CR>
where RUN.COM and the object program RENUMBER.OBJ are both on the current logged-in disk drive (A). The program RENUMBER is used to renumber the first four positions of COBOL source code programs. After loading it asks the question:
ENTER FILE NAME A:FILENAME.TYP
at this point the user enters his program-name.
A:CONFIG.CBL
the program then renumbers the requested file-name and when complete displays:

RENUMBERING COMPLETE

In some cases the renumber program issues error messages. If a input line is all spaces (blank) or if any of the first four positions contain a tab character (09H) the user is notified that the line has been skipped. This is because the renumber program uses the rewrite statement which cannot expand the input.

A>RUN CONFIG
where RUN.COM and CONFIG.OBJ are both on the current logged-in disk drive (A). The program CONFIG is used to specialize the RUN time package and asks the following questions:

```
ENTER SCREEN INFORMATION
ENTER 2-DIGIT HEXADECIMAL CODE FOR DELETE-KEY
        this is usually 08 the CP/M standard Ctl-H
        on the apple the backarrow is 15H
ENTER 2-DIGIT HEXADECIMAL CODE SENT TO BACKSPACE CURSOR
        this is usually 08
IS THE BACKSPACE PRECEEDED WITH AN ESCAPE CHARACTER (Y/N)?
        this is usually N
ENTER # OF CHARACTERS ACROSS SCREEN
        this is usually 64 or 80
ENTER # OF LINES PER SCREEN PAGE
        this is usually 16 or 24
DOES YOUR BIOS ISSUE A CR/LF
AT THE END OF EACH LINE (Y/N)?
        this is usually Y
DOES YOUR PRINTER REQUIRE A LINE FEED (Y/N)?
        this is usually Y
DO YOU WANT TO USE CPM FUNCTION 1 & 2
CONSOLE I-O (Y/N)?
        if N other info will be displayed
DOES YOUR CPM BACKSPACE AND BLANK THE
DELETED CHARACTER (Y/N)?
        this is usually N
DO YOU WANT TO ACCEPT ANY HEX CHARACTER
OR ONLY DISPLAY ASCII (H/A)?
        this is usually A
EOJ CONFIG RETURNING TO CPM
```

Once the CONFIG program has been run and you are satisfied with the way the ACCEPT/DISPLAY functions, the programs are no longer needed on the operational diskette and may be removed as follows:

```
A>ERA CONFIG.*
```

LISTING A PROGRAM

To list a NEVADA COBOL source code program use the CP/M TYPE command and if you have a printer use (ctl-p).

```
A>TYPE RENUMBER.CBL[CTL-P]<CR>
A>TYPE CONFIG.CBL<CR>
A>TYPE W5.CBL<CR>
```

SECTION VII

ERROR CODES AND MESSAGES

COMPILER ERROR MESSAGES

During compilation all error codes are output to a disk work file (W3.WRK). At the end of each COBOL Division the compiler checks for any fatal errors and terminates the compile if any have been found. At the end of compilation a report is displayed and is available for redisplay using the program ERRORS if needed:

A>ERRORS<CR>.

Using the CP/M feature control-p, the error messages can also be sent to the printer as they are displayed. Also, control-s can be used to stop and start the report.

All of the compiler error messages are contained on a file named W5.CBL and can be changed by the user. For example, you may want to have your error messages displayed in German or some other language. These messages can be more than one line and upper-case or lower-case. See error code number 003 below for an example.

The LVL (level) codes are F for Fatal (no object code generated) and W for Warning Possible Error. Also, not shown below each line is preceeded by the source programs actual line number.

```
SEQ     ERR
NO. COL     LVL TEXT
9999 70 001  F  SYNTAX ERROR
        002  F  NOT A COBOL WORD
        003  F  syntax error or period
                missing from prior line
        004  F  FILE NOT SELECTED IN THE I-O SECTION
dataname005  F  OCCURS LIMITED TO ONE LEVEL
dataname006  F  SUBSCRIPTED ITEMS CANNOT BE REDEFINED
dataname007  F  PICTURE ITEMS MUST BE ELEMENTARY
dataname008  F  EDITED PICTURE CONTAINS ILLEGAL COMBINATIONS
dataname009  F  MAX RECORD LENGTH OF 4095 EXCEEDED
dataname010  F  ELEMENTARY ITEM DOES NOT HAVE PICTURE CLAUSE
dataname011  F  ILLEGAL REDEFINES DUE TO INCORRECT REFERENCE
        012  F  SUBSCRIPT ERROR
        013  F  ILLEGAL COMBINATION OF CHARACTERS IN PICTURE
        014  F  DUPLICATION OF PREVIOUS NAME IS ILLEGAL
        015  F  ENVIRONMENT DIVISION MISSING
        016  F  FD MUST CONTAIN A LABEL RECORD CLAUSE
        017  F  VALUE OF FILE-ID MISSING
        018  F  SUBSCRIPT LITERAL CONTAINS ILLEGAL VALUE
        019  F  USAGE CONFLICT
```

```
         020  F  OCCURS CLAUSE IS ILLEGAL AT 01 LEVEL
         021  F  VALUE IS ILLEGAL WITH OCCURS CLAUSE
         022  F  VALUE IS ILLEGAL FOR REDEFINED ITEMS
         023  F  ILLEGAL CHARACTER IN WORD
dataname024  F  MUST HAVE RELATIVE KEY
dataname025  F  MUST BE IN WORKING-STORAGE
         026  F  KEY NOT ELEMENTARY
dataname027  F  RELATIVE KEY MUST BE PIC 9(7)
paraname028  F  PARAGRAPH NAME IS NOT DEFINED
paraname029  F  PARAGRAPH NAME IS NOT ALTERABLE
         030  F  TOO MANY FILES SELECTED
         031  F  NEED MORE MEMORY OR REDUCE SIZE OF LABELS
         032  F  CORRECT ALL ERRORS AND RECOMPILE
         033  F  MISSING DIVISION STATEMENT
         034  F  TOO MANY PARAGRAPH NAMES
         035  F  TOO MANY FORWARD REFERENCES
         036  F  NOTIFY ELLIS COMPUTING
         037  F  01-10 AND 77 LEVELS ONLY
dataname038  F  IS NOT DEFINED
         039  F  AREA B MUST START WITH " ON CONTINUED LITERAL
         040  F  ILLEGAL HEXADECIMAL CHARACTER
         041  F  ILLEGAL FILE-ID "U:FILE"
         042  F  ASCII (DISPLAY) DATA TYPE REQUIRED
         043  F  RANDOM FILES MUST USE INVALID KEY CLAUSE
         044  F  RESERVED WORD NOT YET IMPLEMENTED
         045  F  VALUE/PICTURE SIGN ERROR
         046  F  COPY CANNOT ALSO COPY
         047  F  COPY FILE NAME TOO LONG
         048  F  COULD NOT FIND REDEFINED ITEM NAME
         049  F  LITERAL OVER 120 CHARACTERS LONG
         050  W  LITERAL TRUNCATED RIGHT END
         051  W  MORE THAN 30 CHARACTERS IN A WORD
         052  F  LITERAL LARGER THAN PICTURE
         053  W  REDEFINED AREA ADJUSTED
         054  W  EDITED PICTURE MODIFIED
         055  W  TWO RECORDS IN A FILE HAVE DIFFERENT SIZES
         056  W  COLUMN 5 OR 7 TREATED AS COMMENTS
         057  W  LINE NUMBER OUT OF SEQUENCE
         058  W  RANDOM FILE CANNOT BE DELIMITED
         059  W  PERIOD IS MISSING AFTER PREVIOUS WORD
         060  F  DECIMAL POINT SIZES DIFFERENT
         061  W  PRINTER CANNOT BE DELIMITED
         062  F  VALUE EXCEEDS 5 DIGITS FOR COMP
         063  F  ILLEGAL VALUE FOR COMP
         064  F  ILLEGAL CURRENCY SIGN
         065  F  COPY FILE-NAME MISSING
         066  W  ALL LITERAL LIMITED TO 1 BYTE
         067  F  ZERO MISSING IN BLANK WHEN ZERO
         068  F  BLANK WHEN ZERO NOT ALLOWED AT GROUP LEVEL
         069  F  BLANK WHEN ZERO MUST BE ASCII DISPLAY
         070  F  BLANK WHEN ZERO FOR NUMERIC ONLY
         071  F  JUSTIFIED MUST BE ELEMENTARY DATA ITEM
         072  F  JUSTIFIED CANNOT BE NUMERIC OR EDITED
         073  W  ADDRESS EXCEEDS CURRENT CPM BASE ADDRESS
         074  F  MORE THAN 255 LINKAGE ITEMS
         075  F  USING WITH NO LINKAGE SECTION
         076  F  IF/UNTIL NESTED CONDITIONAL ARE ILLEGAL
         077  F  RESERVED WORD "SENTENCE" IS MISSING
```

```
078  F  ONLY PRINTER FILES CAN HAVE OMITTED
079  F  MORE THAN ONE LABEL RECORDS CLAUSE
080     SUCCESSFUL COMPILE MEMORY AVAILABLE
081  F  MEMORY OVERFLOW REDUCE PROGRAM SIZE
        look at MEMORY size clause under
        OBJECT-COMPUTER.
082  F  ADVANCING FOR PRINTER FILES ONLY
083  F  REDEFINES AT 01 LEVEL IN FILE SECTION
        IS ILLEGAL
084  F  COMP AND COMP-3 CANNOT CONTAIN EDIT SYMBOLS
085  F  PROGRAM NAME
086     USER  LINE   ERR
087     LINE  NO         LVL TEXT
088     ENTER <CR> FOR NEXT LINE
089     ERROR MESSAGE NEXT LINE:
```

RUN TIME AND COMPILE TIME ERROR MESSAGES

The runtime package will display the unit and file-name following the error codes. The following codes are also used in the STATUS keys when specified.

```
90 No additional information
91 Error in extending the file
92 end of disk data - disk is full
93 file not open
94 no more directory space - disk is full
95 file cannot be found
96 file already open
97 reading unwritten data in random access
98 rewrite without prior read in I-O MODE
99 reading an output file or writing to an input file

100 ERROR MESSAGE NOT IN TABLE
    Please send the code to Ellis Computing.
101 SUBSCRIPT ERROR value exceeds 65K.
102 BOUNDARY ERROR program fell through last paragraph.
```

Note: The RUN time package must be the one distributed with the current version of the compiler. Older version will not work. Also older versions of the object code programs will not work with current RUN time packages and must be recompiled.

SECTION VIII

ANSI-1974 COBOL RESERVED WORDS

X = implemented in NEVADA COBOL

ACCEPT	X
ACCESS	X
ADD	X
ADVANCING	X
AFTER	X
ALL	X
ALPHABETIC	X
ALSO	
ALTER	X
ALTERNATE	
AND	X
ARE	X
AREA	X
AREAS	
ASCENDING	
ASSIGN	X
AT	X
AUTHOR	X
BEFORE	X
BLANK	
BLOCK	X
BOTTOM	
BY	X
CALL	X
CANCEL	X
CD	
CF	
CH	
CHARACTER	
CHARACTERS	X
CLOCK-UNITS	
CLOSE	X
COBOL	
CODE	
CODE-SET	
COLLATING	X
COLUMN	
COMMA	X
COMMUNICATION	
COMP	X
COMPUTATIONAL	X
COMPUTE	
CONFIGURATION	X
CONTAINS	X
CONTROL	
CONTROLS	
COPY	X
CORR	
CORRESPONDING	

COUNT	
CURRENCY	X
DATA	X
DATE	
DATE-COMPILED	X
DATE-WRITTEN	X
DAY	
DE	
DEBUG-CONTENTS	
DEBUG-ITEM	
DEBUG-LINE	
DEBUG-NAME	
DEBUG-SUB-1	
DEBUG-SUB-2	
DEBUG-SUB-3	
DEBUGGING	X
DECIMAL-POINT	X
DECLARATIVES	
DELETE	
DELIMITED	
DELIMITER	X
DEPENDING	X
DESCENDING	
DESTINATION	
DETAIL	
DISABLE	
DISPLAY	X
DIVIDE	X
DIVISION	X
DOWN	
DUPLICATES	
DYNAMIC	
EGI	
ELSE	X
EMI	
ENABLE	
END	X
END-OF-PAGE	
ENTER	
ENVIRONMENT	X
EOP	
EQUAL	X
ERROR	X
ESI	
EVERY	
EXCEPTION	
EXIT	X
EXTEND	
FD	X
FILE	X
FILE-CONTROL	X
FILLER	X
FINAL	
FIRST	X
FOOTING	

FOR	X
FROM	X
GENERATE	
GIVING	X
GO	X
GREATER	X
GROUP	
HEADING	
HIGH-VALUE	X
HIGH-VALUES	X
I-O	X
I-O-CONTROL	X
IDENTIFICATION	X
IF	X
IN	
INDEX	
INDEXED	
INDICATE	
INITIAL	X
INITIATE	
INPUT	X
INPUT-OUTPUT	X
INSPECT	X
INSTALLATION	X
INTO	X
INVALID	X
IS	X
JUST	X
JUSTIFIED	X
KEY	X
LABEL	X
LAST	
LEADING	X
LEFT	X
LENGTH	
LESS	X
LIMIT	
LIMITS	
LINAGE	
LINAGE-COUNTER	
LINE	X
LINE-COUNTER	
LINES	X
LINKAGE	X
LOCK	
LOW-VALUE	X
LOW-VALUES	X
MEMORY	X
MERGE	
MESSAGE	
MODE	X
MODULES	X

MOVE	X
MULTIPLE	
MULTIPLY	X
NATIVE	
NEGATIVE	
NEXT	X
NO	X
NOT	X
NUMBER	
NUMERIC	X
OBJECT-COMPUTER	X
OCCURS	X
OF	X
OFF	X
OMITTED	X
ON	X
OPEN	X
OPTIONAL	
OR	X
ORGANIZATION	X
OUTPUT	X
OVERFLOW	
PAGE	X
PAGE-COUNTER	
PERFORM	X
PF	
PH	
PIC	X
PICTURE	X
PLUS	
POINTER	
POSITION	
POSITIVE	
PRINTING	
PROCEDURE	X
PROCEDURES	
PROCEED	X
PROGRAM	X
PROGRAM-ID	X
QUEUE	
QUOTE	X
QUOTES	X
RANDOM	X
RD	
READ	X
RECEIVE	
RECORD	X
RECORDS	X
REDEFINES	X
REEL	
REFERENCES	
RELATIVE	X
RELEASE	

REMAINDER	
REMOVAL	
RENAMES	
REPLACING	X
REPORT	
REPORTING	
REPORTS	
RERUN	
RESERVE	
RESET	
RETURN	
REVERSED	
REWIND	
REWRITE	X
RF	
RH	
RIGHT	X
ROUNDED	X
RUN	X
SAME	X
SD	
SEARCH	
SECTION	X
SECURITY	X
SEGMENT	
SEGMENT-LIMIT	
SELECT	X
SEND	
SENTENCE	X
SEPARATE	
SEQUENCE	X
SEQUENTIAL	X
SET	
SIGN	X
SIZE	X
SORT	
SORT-MERGE	
SOURCE	
SOURCE-COMPUTER	X
SPACE	X
SPACES	X
SPECIAL-NAMES	X
STANDARD	X
STANDARD-1	
START	
STATUS	X
STOP	X
STRING	
SUB-QUEUE-1	
SUB-QUEUE-2	
SUB-QUEUE-3	
SUBTRACT	X
SUM	
SUPPRESS	
SYMBOLIC	
SYNC	X
SYNCHRONIZED	X

TABLE	
TALLYING	X
TAPE	
TERMINAL	
TERMINATE	
TEXT	
THAN	X
THROUGH	X
THRU	X
TIME	
TIMES	X
TO	X
TOP	
TRAILING	
TYPE	
UNIT	
UNSTRING	
UNTIL	X
UP	
UPON	
USAGE	X
USE	
USING	X
VALUE	X
VALUES	
VARYING	
WHEN	
WITH	X
WORDS	X
WORKING-STORAGE	X
WRITE	X
ZERO	X
ZEROES	X
ZEROS	X
+	
-	
*	
/	
**	
>	X
<	X
=	X

NEVADA COBOL RESERVED WORDS (NOT ANSI-1974)

ASCII	X
BEGINNING	X
COMP-3	X
COMPUTATIONAL-3	X
DISK	X
ENDING	X
FILE-ID	X
PRINTER	X

SECTION IX

A COBOL PRIMER

INTRODUCTION

The NEVADA COBOL system consists of three elements: a language, a compiler and a run time package. The language is used by the programmer in writing the source program. The compiler is a program that processes programmer-written COBOL sentences and produces (compiles) an object program (a program that is in machine language). The run time package loads the object program and also contains subroutines used by the object program.

ANALYSIS & DESIGN

This topic is too large to be covered here. The reader should consult one of the many books on this subject. In general, to prepare a business problem for solution, the first step is to thoroughly describe the functions to be performed and the objectives to be accomplished. One of the most important parts of this analysis is the description of the format of the input and output data. Special forms are usually used. The form used for printers is called a printer layout form and the form used for other files, such as Disk or Tape, is called a Record layout form.

THE PROGRAM

After a job has been defined the program can be written. The COBOL language was created specifically to facilitate the processing of data generated by business and industry.

Every COBOL program consists of four separate divisions, or parts:

1. IDENTIFICATION DIVISION	Identifies the program.
2. ENVIRONMENT DIVISION	Describes the computer used and the hardware available.
3. DATA DIVISION	Describes the files, record layouts and storage areas.
4. PROCEDURE DIVISION	States the program logic.

The first division is the **IDENTIFICATION DIVISION.** It provides all of the necessary documentation for the program such as the program name, the programmer's name, the system or application to which the program belongs, the security restrictions on the use of the program, the dates on which the program was written and compiled. Look in the manual for the IDENTIFICATION DIVISION Chapter for an example.

The format for the second division, the **ENVIRONMENT**

DIVISION, is the same as that of the IDENTIFICATION DIVISION. In addition to fixed paragraph-names, there are also fixed section-names. These section-names, like the division-name, appear as single entries on individual lines. Each section contains one or more paragraphs. The last character of the section-name is followed by a space, the word SECTION, and a period.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. The computer that will be used in compiling
                 the source program  (8080-CPU).
OBJECT-COMPUTER. The computer that will be used in executing
                 the compiled object program  (8080-CPU).
INPUT-OUTPUT SECTION.
FILE-CONTROL.    In this paragraph the programmer supplies
                 the computer with two facts concerning each
                 data file used in the program:
                 1. The name of the file - a unique thirty-
                 character name created by the programmer.
                 This name will be used in other divisions
                 of the program whenever referencing the
                 file.
                 2. The hardware input/output device to
                 which the file is assigned
                          Disk
                          Printer
```

The format which must be supplied for each input or output data file is

```
SELECT (file-name) ASSIGN TO (hardware-device)
```

The third division is the **DATA DIVISION.** Data processed by a COBOL program fall into four categories:

1. Data read from input files or written on output files. Memory areas must be reserved into which the data can be read or from which the data can be written.
2. Data which is developed internally, such as totals, and placed into intermediate storage areas.
3. Stored constant values (literals) which are required by the program.
4. Data from other COBOL programs.

To define these area, the DATA DIVISION consists of three sections. The FILE SECTION in which files and the input/output areas are described. For each file named in the FILE-CONTROL paragraph the file name, the record name, the layout of the record and the name, location, size, and format of each field. The WORKING-STORAGE SECTION in which the intermediate storage areas and constant values are defined. The size, format, and content of every counter, storage area, or constant value used by the program. And, the LINKAGE SECTION for accessing information from other programs.

The fourth and last division is the **PROCEDURE DIVISION.** It

is divided into programmer-created paragraphs and sections, containing all of the procedure statements which constitute the steps to be followed in processing the data.

TESTING

Most source programs contain clerical errors the first time they are run, so the first output of a compilation run is usually a listing of errors. The COBOL compiler will diagnose each clerical error it finds and display what are called diagnostics. There are two types of diagnostics - warnings and fatals. Warnings will not prevent compilation of the source program, but fatals will prevent compilation.

Getting the errors out of a source program is called "debugging". If there is a source program error, the program will have to be debugged and compiled again. This cycle is repeated until compilation is successful and an error free object program is produced.

The compiler cannot detect most logical errors. Thus it is possible to produce an object program file that will not solve the problem correctly. For example, suppose "ADD" has been written when the task called for a "SUBTRACT" instruction. The "ADD" instruction will compile, but the resulting object program will not solve the problem correctly. This means that each program must be tested with known data so the results can be verified. Once this is done the program can be run and the business problem "solved".

WORDS

One of the advantages of COBOL is achieved through the standardization of words.

```
ADD FIRST-NUMBER TO SECOND-NUMBER GIVING RESULT-OF-ADDITION.
```

You can see from this example COBOL is relatively easy to learn because, for the most part, it uses words from the English language. People can read the source listing and tell what the program does, what hardware it needs, etc. They know who to go to with questions about the program, and how current the program is. We call this feature documentation and it's one of the most important advantages of COBOL. Many of the entries in COBOL are not required, they are optional. However, they should generally be included in the source program because they add to the documentation.

COBOL words can be broken down into two major categories - reserved words and nonreserved words. A COBOL programmer must not use reserved words for file-names, data-names, etc. Nonreserved words must be used.

```
   MOVE CUSTOMERS-NAME TO PRINT-CUSTOMER-NAME.
```

The above is an excerpt from a COBOL source program. The

reserved words are underlined. The remaining words were made up by the programmer.

Now look in the manual at the list of COBOL reserved words. When writing a COBOL source program, if you are not sure about a data-name you supply, you would first check the list of COBOL reserved words because these words cannot be used.

COBOL words may be constructed from a limited set of characters, following certain rules. COBOL words may contain as many as 30 characters. These characters may be any of the 26 letters, 10 numeric characters, and the "-" or hyphen. Hyphens may be used in COBOL words anywhere except at the beginning or the end of a word.

DCROAB-1 and BALANCE-DATA can both be used as COBOL words since they contain only letters of the alphabet, numeric characters, and hyphens and have no more than 30 characters. A1042 can be a COBOL word, but A10.42 cannot, because A10.42 contains a decimal point (period), which is not an allowable character in a COBOL word.

A word may be ended by a space, or by a period, right parenthesis, comma, or semicolon. Therefore, spaces in the middle of COBOL words are not permissible. TAX-RATE is a legal, nonreserved COBOL word, but TAX RATE is not.

File-names, record-names, and individual item-names must all conform to the rules for the construction of data-names. A data-name must include at least one alphabetic character, must not contain any spaces and must not exceed 30 characters. A data-name may not begin or end with a hyphen, although this character may be contained within a data-name. It is usually convenient to use data-names which suggest the data content, however, it is important that a data-name is not spelled the same as a word on the list of reserved words.

If you don't want to specify a data-name, you can use the word FILLER. Data-name is used to specify the name of the data being described. FILLER is used to specify an unused portion of the logical record. If you want to reference a portion of your record, you cannot call it FILLER. MOVE FILLER TO NAME-FIELD would be an illegal entry because you may never directly reference FILLER. The only way to reference FILLER data is through a group item name.

COBOL language is compatible because it can be compiled by any computer which has a COBOL compiler. Reserved words provide COBOL with this quality of compatibility.

PUNCTUATION

In English, punctuation is used to separate words, clauses and sentences, so that the meaning is clearly understood. In COBOL, for the same purpose, specific punctuation marks are used. Punctuation rules for the four COBOL divisions are

specified in the manual along with the division entries. The words, IDENTIFICATION DIVISION, must be followed by a period. The IDENTIFICATION DIVISION paragraphs may consist of any statement or sentence, but each one must be ended with a period.

Incorrect punctuation can cause fatal diagnostics at compile time. Some punctuation errors will cause warnings to be reported, but others will stop compilation and cause a fatal diagnostic. A frequent diagnostic is "SYNTAX ERROR". "SYNTAX" refers to the rules governing the structure of a language. Many clerical errors that cannot be otherwise analyzed will result in a diagnostic of SYNTAX ERROR. A file-name in the DATA DIVISION must match exactly the same file-name in the ENVIRONMENT DIVISION and the PROCEDURE DIVISION. If it does not match, at compilation time you will receive a fatal diagnostic of the error.

FORMAT

Look in the manual for the paragraph on Symbols and Conventions. Just as COBOL words are standardized for use in source programs, program format is also standardized. Optional entries are shown in square brackets. However, if you decide to use an option, then you must use the complete format in the brackets.

[; BLOCK CONTAINS integer-1 CHARACTERS]

If you use the above optional clause, you must use the words BLOCK and CHARACTERS. You may use the optional word CONTAINS. Note that programmer-supplied information is always required so you must supply integer-1.

THE DATA DIVISION

The purpose of the DATA DIVISION is to describe the data upon which the processing (as stated in the PROCEDURE DIVISION) will be performed.

DATA DIVISION.
FILE SECTION.
Every file selected in the FILE-CONTROL paragraph of the INPUT-OUTPUT SECTION must have two sets of entries in the FILE SECTION.

1. File Description (FD)
   a. Name of the file (same name as in FILE-CONTROL).
   b. LABEL RECORDS ARE STANDARD or OMITTED.
      VALUE OF FILE-ID IS "value " or data-name.
   c. The programmer-created name assigned to a record of the file.
      DATA RECORD IS (record-name).
      the programmer must supply a unique record-name.

2. Record description (01)
The record description entries are located just below the file description. It is the record description which tells the compiler how to set up a record area for each file in which input records can be stored and processed and output records assembled and written. It provides the compiler with the format, or PICTURE, of one record of the file. Each entry begins with a level number followed by at least one space, the name of the data item, and a sequence of independent clauses descriptive of the item. The last clause is terminated by a period and space.

Position -
The record is always described from left to right, i.e., from print position 1 to print position 132 for print files. Every column or print position must be accounted for.

Level -
The first step in preparing a record description entry is the assignment of level-numbers. Level-numbers are used to show the hierarchy of data within a logical record. There may be forty-nine different levels specified for a record, numbered 01 through 49.

The name assigned to the entire record always has the highest level (01). Major divisions (fields) within the record are assigned the next lowest level (02). Subfields within these major divisions are assigned still lower levels (03,04,05 etc.). To show that an elementary item (or field) belongs to a group item, you assign it a higher level-number than the group item.

```
    02 CUST-CODE
    02 CUST-NAME
       03 LAST-NAME
       03 FIRST-NAME
    02 CUST-ADDRESS
```

Notice that you signify the end of a group item by using a level-number item equal to or less than the level-number of the group item.

In the FD entry, the letters "FD" immediately precede the file-name. They tell the compiler (and the user) that the next word is the name of the file. In a record description entry, the level number "01" immediately precedes the record-name. It tells the compiler (and the user) that the next word is the record-name.

Name -
Every field within the record is assigned a unique name of up to thirty characters. This name is used to reference the field in PROCEDURE DIVISION statements. Any unreferenced fields can be assigned the name FILLER.

Following the level-number and data-name are a series of independent clauses.

Format -
Each field must be described as to size, type (numeric, alphabetic, or alphanumeric), location of the actual or assumed decimal point, and any editing desired. This description is given in the form of a PICTURE, using special PICTURE symbols

9 - One numeric field position
A - One alphabetic field position
X - One alphanumeric field position; also used to describe FILLER positions.

Assumed decimal point location -
V - Location of imaginary decimal point; does not occupy an actual field position and is not counted in determining the size of the field.

Editing characters (used to indicate editing on fields contained in print record lines) -
. - Actual decimal point to be inserted in the field where shown.
, - Actual comma to be inserted in the field where shown.
Z - Indicates that all leading zeros and commas are to be suppressed before printing this field.

Abbreviations - Instead of having to write out a whole string of identical character symbols in describing a PICTURE, COBOL allows you to write the symbol once, followed by a set of

parentheses within which the number of repetitions of that character symbol is indicated. The number of characters in a PICTURE clause is limited to 30. PICTURE abbreviations such as),(and S are counted in the size of the PICTURE clause.

A seven-position numeric field

9999999 or 9(7)

An eight-position numeric field with two cent positions

999999V99 or 9(6)V99

A twenty-position alphanumeric field

XXXXXXXXXXXXXXXXXXX or X(20)

Using the PICTURE clause we can also insert a dollar sign, commas, and a decimal point in a numeric value. Below are PICTUREs of input fields, which will be MOVEd to output printer fields, where we want them to be edited as monetary values.

	input	output
1.	PICTURE IS 9(4)	PICTURE IS $9,999
2.	PICTURE 9V99	PICTURE IS $9.99
3.	PICTURE 9(8)V99	PICTURE IS $99,999,999.99

Note in the examples that each PICTURE has enough 9's to hold the entire incoming value, in addition to the insertion characters. When editing a numeric value, you must be sure to have enough 9's to hold the entire value (unless you want to truncate or round it off).

Numeric data to be printed out, especially monetary values, will often have insertion characters put in. Other data, such as input data, or output disk data, will rarely have insertion characters, but will often have operational characters (such as S and V).

Nonintegral input data will usually be described with an assumed decimal point. If data is to be used in arithmetic operations, it may not have an actual decimal point. It must have an assumed decimal point, if any decimal point at all. The PICTURE symbol for an assumed decimal point is V. An actual decimal point cannot be specified without using a PICTURE clause.

The only time a "+" will be printed out is if you specify "+" in the PICTURE and the value stored is positive. If "-", "CR", or "DB" are specified in a PICTURE, and the value received is positive, spaces (blanks) will be printed out.

In some programs, a negative value will represent a credit (such as returned items on a charge account). In others, they will represent a debit (such as accounts due and

payable to wholesale suppliers). In the first case, you would use the characters CR in the receiving field; in the second case DB.

Unless otherwise specified, all data is considered positive. To change this, an input data field must contain a standard operational sign, and an output field must contain either a + - CR DB or a standard operational sign.

```
X-FLD PICTURE IS 9(3)V99.        Y-FLD PICTURE IS +9(3).99.
```

If you MOVEd X-FLD to Y-FLD, the report sign in Y-FLD will always be "+" because X-FLD will always be considered positive.

```
X-FLD PICTURE IS S9(3)V99.       Y-FLD PICTURE IS +9(3).99.
```

The PICTURE of X-FLD has been changed; the S stands for a standard operational sign. Now the report sign in Y-FLD will vary according to whether the value in X-FLD is positive or negative.

Floating Characters
The following symbols may be used as floating zero suppression characters + - $. The other two (nonfloating) zero suppression characters are * and Z.

If the value of (-)003v19 were to be moved to a receiving field described by the PICTURE $$$$.99DB, the appearance of the receiving field would be $3.19DB

When a floating dollar sign is given, it represents a 9 in all but the leftmost position. That is, $$$$.99 will hold 5 numeric digits.

```
    Sending field     45612v34
    Receiving field   $$$$$.99
    value             $5612.34
```

In the above example, the most significant digit of the value the 4, was truncated to make room for the dollar sign. To prevent that from happening, the receiving field should have been described as PICTURE IS $$$$$$.99

OCCURS clause

When describing data which is repeated, the use of the OCCURS clause eliminates the need for separate entries. Whenever the data-name which is the subject of an OCCURS clause is used as an operand, it must be subscripted.

```
    IF SEAT-AVAIL-CODE (39) = "Y" . . .
```

The example above indicates that reference is being made to seat 39(the 39th occurence of this entry). (39) is a subscript.

The reference format rules for parentheses are summarized as

follows:

1. A left parenthesis must be preceded by a space; a right parenthesis must be followed by a space.
2. A left parenthesis must not be followed by a space; a right parenthesis must not be preceded by a space.

The subscript must be, or represent, an integer. The subscript may be a literal or a data-name. If the subscript is a data-name, the value stored in the data-name field must be an integer.

A 32 character entry may not occur 128 times in any record because it would require more than 4,095 characters of computer memory. But you can get around this limitation by setting up several records of up to 4,095 each in a row and then subscripting the first one accessing the information in all of them.

REDEFINES clause

It is often advantageous to be able to call memory areas by more than one name. The REDEFINES clause allows us to give a name to a field which crosses from one elementary item into the next. Note that, in the FILE SECTION, REDEFINES may not be used on the 01 level. To redefine an entire record, you need only name the new record in the DATA RECORDS clause (this implicitly redefines it).

The format of the REDEFINES clause shows nothing may come between data-name-1 and the REDEFINES clause.

A file (whether tape, disk, or printer) is simply a collection of records. For example, a printer file would be a long sheet of paper, on which each line is a record. Normally, a printer prints 132 characters per line. Therefore most printer records would contain 132 characters. Disk files have different physical limitations. In NEVADA COBOL, a logical disk record is limited to 4095 characters. However, it is not likely that you will be using a record that large.

The FILE SECTION of the DATA DIVISION is composed of one or more file description entries. If three files are used by the program, there will be 3 FD entries in the DATA DIVISION. Files contain records which in turn contain data.

The files described so far have each had one type of data record. This is not always the case. Files may have more than one type of data record. Find the DATA RECORDS clause under FILE DESCRIPTION in the manual. The format shows that the programmer can supply record-name-1, record-name-2 . . .

```
    FD DISK-FILE . . .
          DATA RECORDS ARE CONTROL-REC, INFO-REC,
SPECIAL-REC.
```

The DISK-FILE file contains 3 types of data records. Each type has been assigned a record-name. A file description entry must be followed by one 01 level entry (and its associated 02,03,etc level entries) for each type of data record only. One of them can be stored in memory at a time. In order to store a record of different sizes, the computer would have to reserve enough area for the larger of the two types of records. The computer cannot distinguish whether the record currently stored in memory is in INFO-REC or CONTROL-REC format. It is up to the programmer via programming to determine which record is in memory.

WORKING-STORAGE SECTION

A second part of the DATA DIVISION is called the WORKING-STORAGE SECTION. It is in this section that we set up all of the counters and stored values as we find they are needed while coding the application. The format for each entry is very similar to the format for defining any of the data fields in the FILE SECTION. Every elementary WORKING-STORAGE item must be described by a PICTURE clause. In addition to the level-number, data-name, and format, the programmer can also specify that the field or area be set to some initial value. This initial value is expressed in a VALUE IS clause following the PICTURE format. The value itself is written as either a numeric or a non-numeric literal. Values for numeric fields (those defined by 9 and V symbols) are coded as numeric literals

```
  100
  -65
 2.50
```

Values for non-numeric fields (those defined by A or X symbols) must be coded as non-numeric literals. A non-numeric literal is always enclosed in quotation marks.

```
"   CUSTOMER NAME "
"PROFITS FOR 1981"
```

Every space within the quotation marks is counted as a position. The maximum length of any single non-numeric literal is 120 character positions.

Output data often is edited and cannot be used as arithmetic operands. Thus, until it is time to actually write this data, it is often handy to store it in an unedited field. This field would be located in the WORKING-STORAGE SECTION.

While the WORKING-STORAGE SECTION does not contain files, it may contain logical records and therefore be defined by record description entries.

Often, WORKING-STORAGE fields bear no relationship to each other, and are called noncontiguous (not adjacent) fields. However, sometimes it is handy to have group items (so you can MOVE groups, etc). These items are called contiguous (adjacent) fields. A special level number, 77, is used to

define items which are not part of a group item. That is, level number 77 indicates a noncontiguous item. Each separate, noncontiguous Record Description must begin with level-number 77 and include a data-name and a PICTURE clause.

PROCEDURE DIVISION

In this last division of the COBOL program, the processing steps are coded as an English-language statement according to the rules and conventions of the COBOL language. Verb statements are combined to form sentences, which are the basic units in this division. Sentences, one or more, are grouped according to procedure to form paragraphs. The paragraphs in turn are combined to form sections. As in other COBOL divisions, paragraphs and sections must be named.

For example, a series of statements required to print the title and column headings at the top of each page might be grouped together under the paragraph-name TOP-OF-PAGE. If, at some point in the program, the programmer wishes to execute this routine, he simply writes the statement PERFORM TOP-OF-PAGE.

In English, words which describe action are called verbs. In COBOL, words which describe procedures are also called verbs.

Example

```
OPEN INPUT DATA-FILE.
READ DATA-FILE AT END CLOSE DATA-FILE, STOP RUN.
ADD FIRST-NUMBER TO SECOND-NUMBER GIVING
   GIVING RESULT-OF-ADDITION.
```

In this example, OPEN, READ and ADD are examples of verbs which specify the procedures to be followed in processing the data.

The number of different kinds of statements that can be written depends upon the number of COBOL verbs (ADD, GO TO, IF, MOVE, READ, etc.) which the compiler can handle. Every procedural statement must begin with one of these verbs, and each verb has its own unique statement format which must be followed. Only one procedural statement should be written on each line.

Most verb statements contain a verb followed by one or more operands. The word which describes the processing is the verb, while the name of the data to be processed is indicated by the operand.

```
SUBTRACT TAXES FROM GROSS-PAY.
```

In the example above, SUBTRACT is the verb, TAXES and GROSS-PAY are its operands.

The minimum form of a statement consists of a verb and its operands.

```
    MOVE CUST-NAME TO PRT-CUST-NAME; GO TO COMPUTE-BALANCE.
```

Larger statements may consist of two or more minimum statements. There are a total of 3 statements represented by the above entry (two individual + one overall). The individual statements have been separated by the punctuation mark semicolon (;). No punctuation is ever required between statements, it is optional. However, to make the statement easier to read, you could use a separator. The allowable separators are ;(semicolon) ,(comma).

SENTENCES

A sentence consists of a sequence of one or more statements, the last of which is terminated by a period. In COBOL, sentences describe the procedure that is to be accomplished. The sentences are written successively, according to the rules of the reference format. The order in which the sentences are written is important because the sequence of the sentences establishes the order in which the procedures are executed.

```
    PROCEDURE DIVISION.
    BEGINING.
            OPEN INPUT CUSTOMERS-TRANSACTIONS;
            OPEN OUTPUT CUSTOMER-STATEMENT.
            MOVE SPACES TO HEADING.
            READ CUSTOMERS-TRANSACTIONS; AT END STOP RUN.
```

Sentences executing a particular procedure are grouped to form a paragraph. In the example above, there are 3 sentences in the paragraph named BEGINING.

```
0001 PRINT-TRANSACTION.
0002      MOVE AMOUNT TO AMOUNT-FIELD.
0003      WRITE HEADING.
0004 DISK-FILE-READ.
0005      READ CUSTOMERS-TRANSACTIONS;
0006        AT END GO TO TRANSFER-DISK.
0007      GO TO ACCT-NO-MATCH.
```

Paragraph-names must begin under position A (see COBOL Coding Format in the manual). In the example above, there are 2 paragraphs named PRINT-TRANSACTION and DISK-FILE-READ. Beginning under position B are the sentences which form each paragraph.

Paragraph-names are used so that one procedure can reference another by its name. Normally sentences are executed in the order of their appearance in the source program, however, you can change this by referring to a paragraph-name. You can jump from one procedure to another by saying GO TO paragraph-name.

Statements consist of a verb and its operands or several statements. Sentences consist of one or more such statements. Paragraphs consist of one or more sentences. Sections consist of one or more paragraphs.

THE CATEGORIES OF VERBS

All the verbs which may be used in the PROCEDURE DIVISION fall into one of three functional categories -- imperative, conditional, or compiler-directing.

ALTER is an imperative verb. IF is a conditional verb. COPY is a compiler-directing verb. Most of the verbs are imperative verbs. A verb which gives a direct processing instruction, such as ADD or READ, is called an imperative verb.

A verb which tests a condition, such as A = B, is called a conditional verb. The conditional verb IF creates conditional statements, and cannot appear in imperative statements.

A verb which instructs the compiler, at compilation time, rather than instructing the central processor during data processing, is called a compiler-directing verb. Statements using EXIT or COPY are compiler-directing statements.

The statement ADD A TO B ... is imperative. But the statement ADD A TO B; ON SIZE ERROR GO TO X-PROCEDURE is conditional. The statement READ FILE-A; AT END GO TO CLOSING-ROUTINE ... is conditional. Any statement starting with IF is conditional.

```
IF A = B GO TO ADD-ROUTINE ...
```

Basically, GO TO ADD-ROUTINE is an imperative statement. However, the statement shown is a conditional statement. Remember that statements may contain statements. A sentence is a statement or group of statements that are terminated by a period.

THE INPUT-OUTPUT VERBS

The transfer of data between peripheral device and memory can be a complicated subject. COBOL simplifies the matter by letting the compiler select all the ports, channels, appropriate devices, memory storage areas, etc. All the COBOL programmer need do is give the appropriate input or output instruction.

In the manual turn to the page devoted to the OPEN verb. (The verbs are arranged in alphabetical order.) The function of the verb is to initiate the processing of input and output files. Before an input file can be READ, certain standardized procedures may take place. The OPEN verb generates these standardized procedures.

Now turn to the CLOSE verb in your manual. The CLOSE verb is just the opposite of the OPEN verb. The OPEN verb initiates the processing of input and output files. The CLOSE verb terminates the processing of input and output files. Many times after you have CLOSED the files, you would STOP RUN. Turn to the STOP verb in your manual. The use of the STOP verb is to halt the object program either permanently or temporarily.

Now turn to the READ verb in your manual. To transfer a data record from an input file to memory storage, you use a READ instruction. A file may have many, many records, but only one at a time is made available. Every record in the file is stored in the same memory area as the last record was stored. A READ instruction does not destroy the previous record in the file, but it does destroy the previous record in memory. Since most programs loop back to a READ instruction, provisions must be made in the READ instruction for running out of records in the file. This is done in the phrase AT END any imperative statement. The phrase AT END any imperative statement is required. This means that, even though you know the AT END will not be taken, you must include it.

```
READ DISK-IN; AT END IF TAX-INCOME IS GREATER THAN 8400
GO TO COMPUTE-WAGE-TAX.
```

The above sentence would cause a diagnostic at compilation time because the AT END branch is followed by a conditional statement when it should be followed by an imperative statement.

Now turn to the WRITE verb in your manual. The READ verb makes one record available. The write verb releases one record to an output device. To make the ACCOUNT-COPY record available to an output device, you would enter:

```
WRITE ACCOUNT-COPY.
```

Compare the READ and WRITE verbs. Notice that you READ a file-name but you WRITE a record-name.

MOVE VERB

An often used procedure in a source program is the transferal of data from one area in memory to another. For example, data can be transferred from memory area reserved for the input file to a memory area reserved for the output file. To execute this type of procedure, the MOVE verb is used.

Look at the MOVE verb format in the manual. Every MOVE verb references one "sending field" and at least one "receiving field". If you want to transfer the data stored in the field named CUSTOMERS-ACCOUNT to the data area ACCOUNT-OF-CUSTOMER, you write:

```
MOVE CUSTOMERS-ACCOUNT TO ACCOUNT-OF-CUSTOMER.
```

If you want to transfer the data in NEW-BALANCE to three different data areas, named CUSTOMER-BALANCE, CLOSING-BALANCE and AUDIT-BALANCE, you would make the following entry;

```
MOVE NEW-BALANCE TO CUSTOMER-BALANCE, CLOSING-BALANCE,
     AUDIT-BALANCE.
```

When data is MOVEd from a memory area, it is not removed from that area. Therefore, after the MOVE is executed, the data appears in at least two memory areas. The sending field and each receiving field. In the verb format for MOVE, you may use a literal. A literal is an item of data, the value of which is identical to those characters comprising the literal. As you know, a data-name refers to a memory area where a value is stored. Thus, the value of the data-name ACCOUNT-NUMBER may be 50321, or 27993, or any other number. However, the value of the literal 34729 can only be the number 34729. Usually, the programmer does not know the values he is dealing with, so he uses data-names and lets the computer find the values. If, however, he wishes to specify the exact characters to be processed, he uses a literal. There are two types of literals, nonnumeric and numeric. "1984" is a nonnumeric literal, while 1984 is a numeric literal.

Since data-names must contain at least 1 alphabetic character, numeric literals do not resemble data-names. That is 40325 must be a literal because it does not contain an alphabetic character.

Nonnumeric literals may tend to look like data-names, since they may be comprised of alphabetic characters. Therefore, to distinguish between nonnumeric literals and data-names, we put nonnumeric literals in quotation marks.

COST OF ITEM is an illegal data-name because it contains spaces. However, "COST OF ITEM" is acceptable as a nonnumeric literal. While nonnumeric literals may contain

any allowable character, numeric literals may never contain alphabetic characters. A numeric literal may contain a (+) or (-) sign as its leftmost character. If no sign is used, the literal is assumed to be positive. If a numeric literal does not contain a decimal point, it is assumed to an integer.

While nonnumeric literals may contain up to 120 characters, numeric literals contain no more than 18 characters.

The data to be transferred by a MOVE statement may be specified by a **figurative constant.** A figurative constant is a value that has been assigned a fixed data-name. No matter how many characters the receiving field contains, it will be filled with the value represented by the figurative constant. If we want to clear the memory area reserved for CUSTOMER-NAME, this mean we want to replace any data there with spaces, we would write:

```
MOVE SPACES TO CUSTOMER-NAME.
```

If we wanted to fill the CUSTOMERS-ADDRESS field will X's. We would write:

```
 MOVE ALL "X" TO CUSTOMER-ADDRESS.
```

If both the sending field and the receiving field are numeric, you would expect the MOVE to be numeric. If both the sending field and the receiving field are nonnumeric, the MOVE will be nonnumeric. However, if either the sending field or the receiving field is nonnumeric, the MOVE is nonnumeric.

Sometimes the sending field and the receiving field are not the same size. The results of such a MOVE depend on the description of the data you are moving. In the example below, the MOVE is a nonnumeric MOVE. The extra spaces are on the right side of the receiving field.

```
     [JOHN JONES]          [JOHN JONES      ]
```

When a receiving field is longer than the sending field, and the MOVE is nonnumeric, the righthand side of the receiving field is filled with spaces (blanks).

The coding generated for **numeric moves** is such as to move characters from the sending to the receiving field lining up the decimal point. Every number has a decimal point. If it doesn't show, it is to the right of the rightmost (least significant) digit.

When we use the symbol "V", it means the decimal point is "implied". (i.e., it is not actually present in the data field but the computer can align data by it.)

Some examples of decimal point alignment are shown below. As you can see, the computer can align real decimal points with other real decimal points, and implied decimal points with either implied or real decimal points.

```
[123.45]      [0123.450]
[123v45]      [0123.450]
[123v45]      [0123v4500]
```

A numeric MOVE is a 3-step process
1. Align the decimal points.
2. MOVE the digits.
3. Fill in zeros.

On the prior unequal MOVEs the receiving field has been larger than the sending field. However, if the sending field is larger than the receiving field "Truncation" (the chopping off of extra characters) occurs.

```
[WILLIAM K. SMITH]   [WILLIAM K. SM]
```

The above example shows that for nonnumeric MOVEs, the extra characters will be truncated on the right side.

```
[1234.56]     [34.5]
```

For numeric or edited MOVEs, the extra characters may be truncated on both sides, depending on the decimal point alignments.

Subdivisions of a field are very handy in COBOL, because sometimes we may want to use the entire field and sometimes just part of a field. You may MOVE either group items or elementary items. Often, group items are set up for the very purpose of moving; i. e., we may want to process the customer's name individually. However, we know that we need to transfer all the data to the output master file. So we created a group name, CUSTOMERS-RECORD and, rather than writing 3 individual MOVE verbs, we can say

```
MOVE CUSTOMERS-RECORD TO...
```

In every file, the largest group item will be the record name. In an updating procedure, the input record and the output record will have exactly the same format. You can transfer the entire record from the input memory area to the output memory area by saying:

```
  MOVE input-record-name TO output-record-name.
```

All group MOVEs are nonnumeric.

THE ARITHMETIC VERBS

Most of the arithmetic functions of a business application can be handled by addition, subtraction, multiplication, and/or division. So, COBOL includes four arithmetic verbs ADD, SUBTRACT, MULTIPLY and DIVIDE.

Suppose we wanted to add 5 to the field named PRINTER-LINE-COUNT. We would write:

```
ADD 5 TO PRINTER-LINE-COUNT.
```

Look in the manual under ADD. Note that you may combine TO and GIVING. If you do, the sum is stored in the data-name following the word GIVING.

```
        INPUT                          OUTPUT
  WAGE-TAX [3029v73]              DEDUCTIONS [9999v99]
      FICA [7637v69]
```

The fields on the left represent fields in memory used as operands. The field on the right represents a field in memory used as a receiving field. These fields are referable by the data-names next to them, although the data-names do not actually appear in memory. The fields are set up by data description entries in the DATA DIVISION. The "v" stands for the location of the decimal point. Add WAGE-TAX and FICA together. The sum is 10667v42. You cannot store the sum in DEDUCTIONS because it's too long and you will lose the 1. This is called a SIZE ERROR. Whenever arithmetic processes take place there is the possibility of a SIZE ERROR.

```
    PRINCIPLE [0005023v7]          INTEREST [9999v99]
     INT-RATE      [v042]
```

If we were to multiply PRINCIPLE by INT-RATE and store the sum in INTEREST, a SIZE ERROR would occur. You could not store the righthand digits.

Truncation of leftmost and/or rightmost digits of the sum can occur during the storage of the sum according to the size of the receiving field. The normal rules for truncation are:
a. When the receiving field contains fewer integer places than the sum, truncation of the leftmost digits occurs.
b. When the receiving field contains fewer decimal places than the sum, the truncation of rightmost digits occur.

As you can see, if a SIZE ERROR occurs, the sum will be stored but truncation will occur. The SIZE ERROR option is to detect left digit truncation. The programmer must specify how to store the sums and must supply any imperative statement. Normally, more than one step is involved in preventing left truncation. Therefore, rather than specifying the procedures to be taken as part of the ADD sentence, you would branch to a subroutine.

The ROUNDED option is to prevent right digit truncation. The ROUNDED option automatically takes care of storing the sum.

Editing of Output Data

Numeric data which contains no extraneous symbols on input (ie, no real decimal points, commas, dollar signs, etc) may be edited on output. That is, symbols such as commas and

dollar signs may be added to the data.

```
      INPUT                          OUTPUT
CUST-BAL [     v]              NEW-BAL [$    .  ]
 DEPOSITS [    v ]
```

To edit data, the editing symbols are stored in the receiving fields. In the NEW-BAL receiving field a dollar sign and a decimal point are stored.

NUMERIC LITERALS

```
    ADD "29" TO LINE-COUNT.
```

This is not legal. "29" is a nonnumeric literal; only numeric literals may be used.

```
    ADD 14.21 TO X-FIELD.
```

This is legal. (14.21 is a numeric literal; the decimal point will be aligned with the implied decimal point of X-FIELD before addition takes place.)

Turn now to the **SUBTRACT** verb in your manual. As you can see, the SUBTRACT verb format is similar to the ADD verb format. Whereas, in the ADD verb, TO is optional, in the SUBTRACT verb, FROM is required.

```
    GROSS-PAY  -  DEDUCTIONS = NET-PAY
```

To accomplish the above, you would enter SUBTRACT DEDUCTIONS FROM GROSS-PAY GIVING NET-PAY.

Now turn to the **MULTIPLY** verb. Again, the format is very similar to the ADD and SUBTRACT verbs. Like the SUBTRACT verb FROM, the word BY is required. Note that only two operands may be used, the multiplier (data-name-1) and the multiplicand (data-name-2).

In the multiplication process, MULTIPLY A BY B GIVING C and MULTIPLY B BY A GIVING C will produce same results. The rightmost operand must be a data-name, since it will be the receiving field. Therefore, rather than saying MULTIPLY LINE-COUNT BY 2 you should say MULTIPLY 2 BY LINE-COUNT.

MULTIPLY LINE-COUNT BY 2 GIVING TALLY-LINES is permissible because the rightmost operand (the receiving field) is TALLY-LINES, which is a data-name.

Turn to the **DIVIDE** verb. The format is the same. In DIVIDE processes, the decimal places can often go on and on. Therefore, the number of places computed is not determined by the operands, but the receiving field. If the receiving field contains three decimal places, the DIVIDE process will generate three decimal digits.

Since enough digits are generated to fill the right hand portion of the field, there is no right truncation in the

division process (although the effect is the same as right truncation). Division simply ceases at the appropriate time. However, left truncation can occur.

Anything divided by zero = infinity. You cannot DIVIDE by 0. Division by zero is regarded as a SIZE ERROR. If even a remote possibility exists, you must specify an ON SIZE ERROR option.

Recall that ROUNDED is used to prevent right truncation. Since no right truncation takes place in the DIVIDE process, the rules for ROUNDED option are different. ROUNDED will cause two more right hand digits to be generated. The two extra right hand digits created by the ROUNDED option are then rounded off, as normal.

THE SEQUENCE CONTROL VERBS

The normal sequence of executing a COBOL program is to start with the first statement after the words PROCEDURE DIVISION, execute that, pass control to the second statement, execute that, pass control to the third statement, etc. Sometimes you may wish to instruct the computer not to execute the statements in this "straight-line" sequence. In this case, depending on whether you want to permanently depart or only temporarily depart from the "straight-line" sequence, you may use the GO TO or the PERFORM verb.

Turn to the **GO TO** page of the manual. In a source program, procedural statements are written in the order in which the programmer decided they should be executed. To depart from this normal sequence of procedures, you may use the GO TO verb. The GO TO verb format consists of the words GO TO plus the desired procedure-name. To write the GO TO entry using the procedure-name BEGIN. You would write GO TO BEGIN. When this statement is encountered, the entire PROCEDURE DIVISION is scanned for a paragraph or a section named BEGIN. Control is transferred to that paragraph or section.

Another sequence control verb is PERFORM. Whereas GO TO permanently transfers control, PERFORM only temporarily transfers control.

A subroutine must have one, and only one, EXIT point. If it logically has more than one, then you must change it so that it has only one exit point.

```
    SUMMARY SECTION.
     A-PARA.
            MOVE IN-REC TO OUT-REC.
            WRITE OUT-REC.
            IF FLAG = 1 GO TO B-PARA.
            WRITE OUT-REC.
     B-PARA.  EXIT.
```

The statement says IF FLAG = 1 GO TO B-PARA.

B-PARA of SUMMARY subroutine contains one instruction EXIT.

In many cases where a procedure has more than one path, the last statement will be the verb EXIT.

Since EXIT must be in a paragraph by itself, with a paragraph-name, the other procedures to be PERFORMed must be contained in at least one other paragraph. Since the PERFORM statement must reference the EXIT verb as well as the procedures to be PERFORMed, the PERFORM statement can reference a section-name. The section referenced will contain the procedural paragraph's plus the EXIT paragraph. Or another way would be to PERFORM ... THRU. In this case:

PERFORM A-PARA THRU B-PARA.

A command which will cause the ADD-DAILY-HOURS through the COMPUTE-DAILY-WAGE paragraphs to be executed five times in a row is written:

```
PERFORM ADD-DAILY-WAGE THRU COMPUTE-DAILY-WAGE 5 TIMES.
```

THE IF VERB

At many points in a program it is important to take different courses of action based on the results of a test. For example, in payroll computations, employees with over a certain amount of year-to-date earnings do not get charged social security tax, those under do. Some employees may have Blue Cross, Blue Shield, savings bonds, and other employee benefits charged. Others may not. Each of these would need a special test made to see if the deductions should be made.

Turn to the IF verb in your manual. Read the function. Remember that conditional statements have two forms the second form is represented by ON SIZE ERROR or AT END. IF represents the 1st form. Read the format of the IF statement. The first programmer- supplied information is condition. Look at the format for the entire IF statement. Following the condition the programmer either supplies statement-1 or writes the key words NEXT SENTENCE.

```
IF LINE-COUNT IS NOT LESS THAN 40 PERFORM PAGE-ROUTINE
 ELSE PERFORM SPACING-ROUTINE.
 WRITE HEADING-1.
```

The LINE-COUNT is tested and if 39 or less, the SPACING-ROUTINE is performed, and then the instruction WRITE HEADING-1 is processed. In fact, no matter what the results of the test, control is returned to the next sentence, WRITE HEADING-1.

Look at your manual. In order to pass control directly to the next sentence by statement-2, you would use the key words NEXT SENTENCE. You can also transfer control directly to the next sentence by statement-1. In this case, statement-1 would say NEXT SENTENCE. NEXT SENTENCE can be substituted for either statement-1 or statement-2. If

substituted for statement-2, the whole clause ELSE NEXT SENTENCE can be omitted.

```
IF AMOUNT IS POSITIVE PERFORM DEPOSIT-ROUTINE.
WRITE HEADING-1.
```

In the above conditional sentence, the compiler will assume that ELSE NEXT SENTENCE is missing. Therefore, if AMOUNT is not positive, control will be sent to the statement WRITE HEADING-1.

CONCLUSION

This has been a brief introduction to the COBOL language. Perhaps with the example programs that follow, it will be enough to get you started. The interested reader should consult one of the many COBOL self-teaching books available at most book stores. Good luck!

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T6WF.
0004* THIS PROGRAM CREATES A FILE OF FIXED LENGTH RECORDS AND
0004* IF THE RECORD SIZES ARE CHANGED TO YOUR NEEDS CAN BE
0004* USED TO CREATE THE SPACE NEEDED (ALLOCATE) FOR A RANDOM
0004* FILE.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS SEQUENTIAL
0015       ACCESS MODE IS SEQUENTIAL.
0016 DATA DIVISION.
0017 FILE SECTION.
0018 FD FILE1
0019     LABEL RECORDS ARE STANDARD
0020     VALUE OF FILE-ID IS OUT-FILE-NAME
0021     BLOCK CONTAINS 1 RECORD
0022     DATA RECORDS ARE O-RECORD.
0023 01 O-RECORD.
0024     02 SEQ PIC 9999.
0025     02 REC1 PIC IS X(156).
0026     02 SEQ2 PIC 9999.
0027 WORKING-STORAGE SECTION.
0028 01 OUT-FILE-NAME PIC X(14)
0029         VALUE "A:TESTF.WRK".
0030 01 X1 PIC 9999
0031         VALUE 0001.
0032 PROCEDURE DIVISION.
0033 BEGIN.
0034     DISPLAY "ENTER OUTPUT FILE NAME ".
0035     DISPLAY OUT-FILE-NAME WITH NO ADVANCING.
0036* to accept and use the file-name just displayed you can
0036* hit the <CR> key.  see # 2 under accept.
0036     ACCEPT OUT-FILE-NAME.
0037     OPEN OUTPUT FILE1.
0038     MOVE SPACES TO O-RECORD.
0039 BEGIN2.
0040     MOVE X1 TO SEQ.
0041     MOVE X1 TO SEQ2.
0042     ADD 1 TO X1.
0043     DISPLAY O-RECORD.
0044     WRITE O-RECORD.
0045     IF X1 IS = TO 201
0046       GO TO EOJ.
0047     GO TO BEGIN2.
0048 EOJ.
0049     CLOSE FILE1.
0050     STOP RUN.
0051 END PROGRAM T6WF.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T6RF.
0004* THIS PROGRAM READS A FIXED LENGTH FILE SEQUENTIALLY
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS SEQUENTIAL
0015       ACCESS MODE IS SEQUENTIAL
0016       FILE STATUS IS STATUS-KEY.
0017 DATA DIVISION.
0018 FILE SECTION.
0019 FD FILE1
0020     LABEL RECORDS ARE STANDARD
0021     VALUE OF FILE-ID IS NAME-OF-FILE
0022     BLOCK CONTAINS 1 RECORD
0023     DATA RECORDS ARE I-RECORD.
0024 01 I-RECORD.
0025     02 SEQ PIC 9999.
0026     02 REC1 PIC IS X(160).
0027 WORKING-STORAGE SECTION.
0028 01 STATUS-KEY PIC XX.
0029 01 NAME-OF-FILE PIC X(14)
0030         VALUE "A:TESTF.WRK".
0031 PROCEDURE DIVISION.
0032 BEGIN.
0033     DISPLAY "ENTER INPUT FILE NAME ".
0034     DISPLAY NAME-OF-FILE WITH NO ADVANCING.
0035     ACCEPT NAME-OF-FILE
0036     OPEN INPUT FILE1.
0037 BEGIN2.
0038     MOVE SPACE TO I-RECORD.
0039     MOVE SPACE TO STATUS-KEY.
0040     READ FILE1
0041       AT END
0042       GO TO EOJ.
0043     DISPLAY I-RECORD.
0044     DISPLAY STATUS-KEY.
0045     GO TO BEGIN2.
0046 EOJ.
0047     DISPLAY STATUS-KEY.
0048     CLOSE FILE1.
0049     DISPLAY STATUS-KEY.
0050     STOP RUN.
0051 END PROGRAM T6RF.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T6IOF.
0004* THIS PROGRAM READS THEN REWRITES FIXED LENGTH RECORDS.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS SEQUENTIAL
0015       ACCESS MODE IS SEQUENTIAL.
0016 DATA DIVISION.
0017 FILE SECTION.
0018 FD FILE1
0019     LABEL RECORDS ARE STANDARD
0020     VALUE OF FILE-ID IS IN-OUT-FILE
0021     DATA RECORDS ARE I-O-RECORD.
0022 01 I-O-RECORD.
0023     02 SEQ PIC 9999.
0024     02 REC1 PIC IS X(160).
0025 WORKING-STORAGE SECTION.
0026 01 IN-OUT-FILE PIC X(14)
0027         VALUE "A:TESTF.WRK".
0028 01 X1 PIC 9999
0029         VALUE 1001.
0030 PROCEDURE DIVISION.
0031 BEGIN.
0032     DISPLAY "ENTER FILE NAME ".
0033     DISPLAY IN-OUT-FILE WITH NO ADVANCING.
0034     ACCEPT IN-OUT-FILE.
0035     OPEN I-O FILE1.
0036     MOVE SPACE TO I-O-RECORD.
0037 BEGIN2.
0038     READ FILE1
0039       AT END
0040       GO TO EOJ.
0041     DISPLAY SEQ.
0042     DISPLAY " IN" WITH NO ADVANCING.
0043     MOVE X1 TO SEQ.
0044     ADD 1 TO X1.
0045     DISPLAY SEQ.
0046     REWRITE I-O-RECORD.
0047     DISPLAY " OUT" WITH NO ADVANCING.
0048     GO TO BEGIN2.
0049 EOJ.
0050     CLOSE FILE1.
0051     STOP RUN.
0052 END PROGRAM T6IOF.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T6WD.
0004*  THIS PROGRAM CREATES A FILE OF VARIABLE LENGTH
0005*  (DELIMITED) RECORDS.  Most text editors create
    *  this type of file.  Each record ends with a carriage
    *  return and line feed.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS SEQUENTIAL
0015       ACCESS MODE IS SEQUENTIAL
    * the next statement tells the compiler each record is to
    * be delimited (separated) by or ended with a carriage
    * return and line feed.
0016       RECORD DELIMITER IS STANDARD.
0017 DATA DIVISION.
0018 FILE SECTION.
0019 FD FILE1
0020     LABEL RECORDS ARE STANDARD
0021     VALUE OF FILE-ID IS OUT-FILE
0022     DATA RECORDS ARE O-RECORD.
0023 01 O-RECORD.
0024     02 SEQ PIC 9999.
0025     02 REC1 PIC IS X(156).
0026     02 SEQ2 PIC 9999.
0027 WORKING-STORAGE SECTION.
0028 01 OUT-FILE PIC X(14)
0029         VALUE IS "A:TESTB.WRK".
0030 01 X1 PIC 9999
0031         VALUE 0001.
0032 01 PAD.
0033     02 FILLER PIC X(30)
0034         VALUE SPACE.
0035     02 FILLER PIC X(30)
0036         VALUE SPACE.
0037     02 FILLER PIC X(30)
0038         VALUE SPACE.
0039     02 FILLER PIC X(30)
0040         VALUE SPACE.
0041     02 FILLER PIC X(30)
0042         VALUE SPACE.
0043     02 FILLER PIC X(05)
0044         VALUE "AAAAA".
0045 PROCEDURE DIVISION.
0046 BEGIN.
0047     DISPLAY "ENTER OUTPUT FILE NAME ".
0048     DISPLAY OUT-FILE WITH NO ADVANCING.
0049     ACCEPT OUT-FILE.
0050     MOVE SPACES TO O-RECORD.
0051     OPEN OUTPUT FILE1.
```

```
0052      DISPLAY "OPEN".
0053      MOVE PAD TO REC1.
0054 BEGIN2.
0055      MOVE X1 TO SEQ.
0056      MOVE X1 TO SEQ2.
0057      ADD 1 TO X1.
0058      DISPLAY O-RECORD.
0059      WRITE O-RECORD.
0060      IF X1 = 011
0061        GO TO EOJ.
0062      GO TO BEGIN2.
0063 EOJ.
0064      CLOSE FILE1.
0065      STOP RUN.
0066 END PROGRAM T6WD.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T6RD.
0004*  THIS PROGRAM READS A VARIABLE LENGTH (DELIMITED) FILE.
    *  the kind of file created by most text editors.  Each
    *  record in the file is terminated with a carriage
    *  return and line feed.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS SEQUENTIAL
0015       ACCESS MODE IS SEQUENTIAL
    * the next statement tells the compiler the records will
    * end with a carriage return and line feed.
0016       RECORD DELIMITER IS STANDARD.
0017 DATA DIVISION.
0018 FILE SECTION.
0019 FD FILE1
0020     LABEL RECORDS ARE STANDARD
0021     VALUE OF FILE-ID IS IN-FILE
0022     DATA RECORDS ARE I-RECORD.
0023 01 I-RECORD.
0024     02 SEQ PIC 9999.
0025     02 REC1 PIC IS X(160).
0026 WORKING-STORAGE SECTION.
0027 01 IN-FILE PIC X(14)
0028         VALUE "A:TESTB.WRK".
0029 PROCEDURE DIVISION.
0030 BEGIN.
0031     DISPLAY "ENTER INPUT FILE NAME ".
0032     DISPLAY IN-FILE WITH NO ADVANCING.
0033     ACCEPT IN-FILE.
0034     OPEN INPUT FILE1.
0035 BEGIN2.
0036* the next statement is necessary because the delimited
0036* read only transfers data into the record area and if
0036* short the data from prior reads will be in the record
0036* area on the right end.
0036     MOVE SPACE TO I-RECORD.
0037     READ FILE1
0038       AT END
0039       GO TO EOJ.
0040     DISPLAY I-RECORD.
0041     GO TO BEGIN2.
0042 EOJ.
0043     CLOSE FILE1.
0044     STOP RUN.
0045 END PROGRAM T6RD.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T6IOD.
0004* THIS PROGRAM READS THEN REWRITES VARIABLE LENGTH RECORDS.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS SEQUENTIAL
0015       ACCESS MODE IS SEQUENTIAL
0016       RECORD DELIMITER IS STANDARD.
0017 DATA DIVISION.
0018 FILE SECTION.
0019 FD FILE1
0020     LABEL RECORDS ARE STANDARD
0021     VALUE OF FILE-ID IS I-O-FILE-NAME
0022     DATA RECORD IS A-RECORD.
0023 01 A-RECORD.
0024     02 SEQ PIC 9999.
0025     02 REC1 PIC IS X(160).
0026 WORKING-STORAGE SECTION.
0027 01 X1 PIC 9999
0028         VALUE 2001.
0029 01 I-O-FILE-NAME PIC X(14)
0030         VALUE IS "A:TESTB.WRK".
0031 PROCEDURE DIVISION.
0032 BEGIN.
0033     DISPLAY "ENTER I-O FILE NAME ".
0034     DISPLAY I-O-FILE-NAME WITH NO ADVANCING.
0035     ACCEPT I-O-FILE-NAME.
0036     OPEN I-O FILE1.
0037 BEGIN2.
0038     MOVE SPACE TO A-RECORD.
0039     READ FILE1
0040       AT END
0041       GO TO EOJ.
0042     MOVE X1 TO SEQ.
0043     ADD 1 TO X1.
0044     DISPLAY SEQ.
0045     REWRITE A-RECORD.
0046     GO TO BEGIN2.
0047 EOJ.
0048     CLOSE FILE1.
0049     DISPLAY "RENUMBERING COMPLETE".
0050     STOP RUN.
0051 END PROGRAM T6IOD.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID. TST-PRT.
    * this sample program reads in a variable length file and
    * outputs it to the printer.
0003 ENVIRONMENT DIVISION.
0004 CONFIGURATION SECTION.
0005 SOURCE-COMPUTER. 8080-CPU.
0006 OBJECT-COMPUTER. 8080-CPU.
0008 INPUT-OUTPUT SECTION.
0009 FILE-CONTROL.
0010     SELECT FILE1 ASSIGN TO DISK
0011     RECORD DELIMITER IS STANDARD.
    * the next line is for printers and/or printer-files.
0012     SELECT FILE2  ASSIGN TO PRINTER.
0013 DATA DIVISION.
0014 FILE SECTION.
0015 FD  FILE1
0016     LABEL RECORDS ARE STANDARD
0017     VALUE OF FILE-ID IS IN-FILE1-NAME
0018     DATA RECORD IS TESTB.
0019 01  TESTB PIC X(80).
0020 FD  FILE2
0021     LABEL RECORDS ARE STANDARD
0022     VALUE OF FILE-ID IS OUT-FILE2-NAME
0023     DATA RECORD IS PRINT-LINE.
0024 01  PRINT-LINE PICTURE IS X(132).
0025 WORKING-STORAGE SECTION.
    * the input file-name can be a cobol source file to be
    * listed on the printer. this file-name can be changed at
    * run time see lines 0030-0032.
0026 01 IN-FILE1-NAME PIC X(14) VALUE "A:T01.CBL".
    * in line 0027  "printer" is the key word to send output
    * to the physical printer.
    * any other file-name sends output to the named disk file.
    * this option of either printing or sending output to the
    * printer can be made at run time. see lines 0033-0035.
0027 01 OUT-FILE2-NAME PIC X(14) VALUE "PRINTER".
0028 PROCEDURE DIVISION.
0029 BEGIN.
0030     DISPLAY "ENTER INPUT FILE ".
0031     DISPLAY IN-FILE1-NAME WITH NO ADVANCING.
0032     ACCEPT IN-FILE1-NAME.
0033     DISPLAY "ENTER PRINTER FILE ".
0034     DISPLAY OUT-FILE2-NAME WITH NO ADVANCING.
    * no need to re-enter the word "printer" just hit <cr>
0035     ACCEPT OUT-FILE2-NAME.
0036     OPEN INPUT FILE1.
0037     OPEN OUTPUT FILE2.
0038     MOVE SPACES TO PRINT-LINE.
0039 PARA-3.
0040     MOVE SPACES TO TESTB.
0041     READ FILE1 AT END GO TO EOJ.
0042     MOVE TESTB TO PRINT-LINE.
0043     WRITE PRINT-LINE BEFORE ADVANCING 1 LINE.
0044     GO TO PARA-3.
0045 EOJ.
0046     MOVE SPACES TO PRINT-LINE.
```

```
0047      WRITE PRINT-LINE BEFORE ADVANCING PAGE.
0048      CLOSE FILE1.
0049      CLOSE FILE2.
0050      STOP RUN.
0051 END PROGRAM TST-PRT.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T8WR.
0004* THIS PROGRAM WRITES RANDOM FIXED LENGTH RECORDS TO A
0004* FILE THAT HAS BEEN CREATED USING A SEQUENTIAL FIXED
0004* LENGTH WRITE PROGRAM TO ALLOCATE THE REQUIRED FILE
0004* SPACE.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS
0015       RELATIVE
0016       ACCESS MODE IS RANDOM
0017       RELATIVE KEY IS KEY-1.
0018 DATA DIVISION.
0019 FILE SECTION.
0020 FD FILE1
0021     LABEL RECORDS ARE STANDARD
0022     VALUE OF FILE-ID IS OUT-FILE
0023     DATA RECORDS ARE O-RECORD.
0024 01 O-RECORD.
0025     02 SEQ PIC 9999.
0026     02 REC1 PIC IS X(160).
0027 WORKING-STORAGE SECTION.
0028 01 OUT-FILE PIC X(14)
0029         VALUE "A:TESTF.WRK".
0030 01 KEY-1 PIC 9(7) COMP-3.
0031 01 XX-KEY PIC 9(4) VALUE 1.
0032 PROCEDURE DIVISION.
0033 BEGIN.
0034     DISPLAY "ENTER OUTPUT FILE NAME ".
0035     DISPLAY OUT-FILE WITH NO ADVANCING.
0036     ACCEPT OUT-FILE.
0037     OPEN OUTPUT FILE1.
0038 BEGIN2.
0039     MOVE SPACE TO O-RECORD.
0040     MOVE 0001 TO XX-KEY.
0041     DISPLAY "ENTER RECORD NUMBER 0001 ".
0042     ACCEPT XX-KEY.
0043     IF XX-KEY IS NOT NUMERIC
0044       GO TO BEGIN2.
0045     IF XX-KEY = 9999
0046       GO TO EOJ.
0047     MOVE XX-KEY TO KEY-1.
0048     MOVE XX-KEY TO SEQ.
0049     DISPLAY "ENTER DATA FOR RECORD ".
0050     ACCEPT REC1.
0051     WRITE O-RECORD
```

```
0052         INVALID KEY
0053         DISPLAY "INVALID KEY" GO TO BEGIN2.
0054       DISPLAY O-RECORD.
0055       GO TO BEGIN2.
0056 EOJ.
0057       CLOSE FILE1.
0058       DISPLAY "EOJ".
0059       STOP RUN.
0060 END PROGRAM T8WR.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T8RR.
0004*  THIS PROGRAM READS RANDOM FIXED LENGTH RECORDS.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS
0015       RELATIVE
0016       ACCESS MODE IS RANDOM
0017       RELATIVE KEY IS KEY-1.
0018 DATA DIVISION.
0019 FILE SECTION.
0020 FD FILE1
0021     LABEL RECORDS ARE STANDARD
0022     VALUE OF FILE-ID IS IN-FILE
0023     DATA RECORDS ARE I-RECORD.
0024 01 I-RECORD.
0025     02 PART-NUMBER PIC 9999.
0026     02 ITEM-DESCRIPTION PIC IS X(160).
0027 WORKING-STORAGE SECTION.
0028 01 IN-FILE PIC X(14)
0029         VALUE "A:TESTF.WRK".
0030 01 KEY-1 PIC 9(7) COMP-3.
0031 01 XX-KEY PIC 9(4).
0032 PROCEDURE DIVISION.
0033 BEGIN.
0034     DISPLAY "ENTER INPUT FILE NAME".
0035     DISPLAY IN-FILE WITH NO ADVANCING.
0036     ACCEPT IN-FILE.
0037     OPEN INPUT FILE1.
0038     DISPLAY "OPEN".
0039 BEGIN2.
0040     MOVE SPACE TO I-RECORD.
0041     MOVE 0001 TO XX-KEY.
0042     DISPLAY "ENTER RECORD NUMBER 0001 "
0043     ACCEPT XX-KEY.
0044     IF XX-KEY NOT NUMERIC
0045       GO TO BEGIN2.
0046     IF XX-KEY = 9999
0047       GO TO EOJ.
0048     MOVE XX-KEY TO KEY-1.
0049     READ FILE1
0050       INVALID KEY
0051       DISPLAY "INVALID KEY" GO TO BEGIN2.
0051* don't display on invalid key as data is unspecified.
0052     DISPLAY I-RECORD.
0053     GO TO BEGIN2.
0054 EOJ.
```

```
0055      CLOSE FILE1.
0056      DISPLAY "EOJ".
0057      STOP RUN.
0058 END PROGRAM T8RR.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T8IOR.
0004* THIS PROGRAM READS THEN REWRITES FIXED LENGTH RECORDS '
0005* IN RANDOM MODE.
0006 ENVIRONMENT DIVISION.
0007 CONFIGURATION SECTION.
0008 SOURCE-COMPUTER.
0009     8080-CPU.
0010 OBJECT-COMPUTER.
0011     8080-CPU.
0012 INPUT-OUTPUT SECTION.
0013 FILE-CONTROL.
0014     SELECT FILE1 ASSIGN TO DISK
0015       ORGANIZATION IS
0016       RELATIVE
0017       ACCESS MODE IS RANDOM
0018       RELATIVE KEY IS KEY-1.
0019 DATA DIVISION.
0020 FILE SECTION.
0021 FD FILE1
0022     LABEL RECORDS ARE STANDARD
0023     VALUE OF FILE-ID IS I-O-FILE
0024     BLOCK CONTAINS 1 RECORD
0025     DATA RECORDS ARE A-RECORD.
0026 01 A-RECORD.
0027     02 SEQ PIC 9999.
0028     02 REC1 PIC IS X(160).
0029 WORKING-STORAGE SECTION.
0030 01 I-O-FILE PIC X(14)
0031         VALUE "A:TESTF.WRK".
0032 01 KEY-1 PIC 9(7) COMP-3.
0033 01 XX-KEY PIC 9(4)
0034         VALUE 1.
0035 PROCEDURE DIVISION.
0036 BEGIN.
0037     DISPLAY "ENTER I-O FILE NAME "
0038     DISPLAY I-O-FILE WITH NO ADVANCING.
0039     ACCEPT I-O-FILE.
0040     OPEN I-O FILE1.
0041 BEGIN2.
0042     MOVE SPACE TO A-RECORD.
0043     MOVE 1 TO XX-KEY.
0044     DISPLAY "ENTER RECORD NUMBER 0001 ".
0045     ACCEPT XX-KEY.
0046     IF XX-KEY IS NOT NUMERIC
0047       GO TO BEGIN2.
0048     IF XX-KEY = 9999
0049       GO TO EOJ.
0050     MOVE XX-KEY TO KEY-1.
0051     READ FILE1
0052       INVALID KEY
0053       DISPLAY "READ INVALID KEY" GO TO BEGIN2.
0054     DISPLAY A-RECORD.
0055     DISPLAY "ENTER NEW DATA ".
0056     ACCEPT REC1.
```

```
0057      REWRITE A-RECORD
0058        INVALID KEY
0059        DISPLAY "REWRITE INVALID KEY".
0060      DISPLAY A-RECORD.
0061      GO TO BEGIN2.
0062  EOJ.
0063      CLOSE FILE1.
0064      DISPLAY "EOJ".
0065      STOP RUN.
0066  END PROGRAM T8IOR.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T20.
0004*  THIS PROGRAM CALLS PROGRAM T20A WHICH IN TURN CALLS
0005*  PROGRAM T20B.
0006 ENVIRONMENT DIVISION.
0007 CONFIGURATION SECTION.
0008 SOURCE-COMPUTER.
0009     8080-CPU.
0010 OBJECT-COMPUTER.
    * the following memory statement is necessary for memory
    * mapping as it marks the upper boundary address (16383).
    * the data from this program loads from the bottom-up
    * and from the top-down.  free space, if any, is somewhere
    * between the top address and the starting address.
0011     8080-CPU MEMORY SIZE 16383 CHARACTERS.
0012 DATA DIVISION.
0013 WORKING-STORAGE SECTION.
0014 01 M1.
0015     02 M1-2.
0016     03 M1-3 PIC XXX.
0017     02 M1-4 PIC 99.
0018     02 M1-5 PIC 99V99 COMP VALUE 11.11.
0019     02 M1-6 PIC 999999V99 COMP-3 VALUE 012345.78.
0020     02 M1-7 PIC $99,999.99.
0021 01 M2 PIC S9V9999 VALUE 0.6143.
0022 01 M3 PIC X(10) VALUE "A:T20A".
0023 01 M4 PIC X (120).
0024 01 M5 PIC X(20) JUSTIFIED.
0025 PROCEDURE DIVISION.
0026 BEGIN.
0027     DISPLAY "START T20".
0028     MOVE ALL "A" TO M4.
0029     CALL "T20A" USING M1, M2, M3, M4, M5.
0030     DISPLAY "EOJ-T20".
0031     STOP RUN.
0032 END PROGRAM T20.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T20A.
0004*  THIS PROGRAM IS CALLED BY T20 AND IN TURN CALLS
0005*   PROGRAM T20B.
0006 ENVIRONMENT DIVISION.
0007 CONFIGURATION SECTION.
0008 SOURCE-COMPUTER.
0009     8080-CPU.
0010 OBJECT-COMPUTER.
    * the following memory statement is necessary. it must be
    * at least 1 byte higher than the previous programs ending
    * address (16383+1=16384) in this example.
```

```
0011     8080-CPU MEMORY BEGINNING 16384 ENDING 20000.
0012 DATA DIVISION.
0013 WORKING-STORAGE SECTION.
0014 01 L3 PIC X(10) VALUE "A:T20A".
0015 LINKAGE SECTION.
0016 01 M1.
0017     02 M1-2.
0018     03 M1-3 PIC XXX.
0019     02 M1-4 PIC 99.
0020     02 M1-5 PIC 99V99 COMP.
0021     02 M1-6 PIC 999999V99 COMP-3.
0022     02 M1-7 PIC $99,999.99.
0023 01 M2 PIC S9V9999.
0024 77 M3 PIC X(10).
0025 77 M4 PIC X(120).
0026 77 M5 PIC X(20) JUSTIFIED.
0027 PROCEDURE DIVISION
0028* no period after the word division when using using
0029         USING M1, M2, M3, M4, M5.
0030 BEGIN.
0031     DISPLAY "THIS IS T20A".
0032     DISPLAY M3.
0033     DISPLAY M4.
0034     CALL "T20B" USING L3.
0035     CANCEL "T20B".
0036 EOJ1.
0036     EXIT PROGRAM.
0037 EOJ.
0038     STOP RUN.
0039 END PROGRAM T20A
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T20B.
0004*  THIS PROGRAM IS CALLED BY T20A AND EXITS BACK TO IT.
0005*  NOTE HOW THE MEMORY IS ALLOCATED.
0006 ENVIRONMENT DIVISION.
0007 CONFIGURATION SECTION.
0008 SOURCE-COMPUTER.
0009     8080-CPU.
0010 OBJECT-COMPUTER.
    * the following memory statement is necessary to control
    * the memory mapping of this third program module. it
    * starts at address 20001 just one byte higher than the
    * previous programs ending address.
0011     8080-CPU MEMORY BEGINNING 20001 ENDING 24000.
0012 DATA DIVISION.
0013 FILE SECTION.
0014 WORKING-STORAGE SECTION.
0015 01 L1 PIC X(10) VALUE SPACE.
0016 LINKAGE SECTION.
0017 01 L3 PIC X(10).
0018 PROCEDURE DIVISION
```

```
0019     USING L3.
0020 BEGIN.
0021     DISPLAY "THIS IS T20-B".
0022     DISPLAY L3.
0023 EOJ1.
0024     EXIT PROGRAM.
0025 EOJ.
0026     STOP RUN.
0027 END PROGRAM T20B
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     TSUBMIT.
0004*  THIS PROGRAM CHAINS TO EXECUTE THE NEXT PROGRAM USING
0005*  CP/M'S SUBMIT WHEN THE NEXT PROGRAM IS NOT TYPE (.OBJ)
0006 ENVIRONMENT DIVISION.
0007 CONFIGURATION SECTION.
0008 SOURCE-COMPUTER.
0009     8080-CPU.
0010 OBJECT-COMPUTER.
0011     8080-CPU.
0012 INPUT-OUTPUT SECTION.
0013 FILE-CONTROL.
0014     SELECT FILE1 ASSIGN TO DISK
0015       RECORD DELIMITER IS STANDARD.
0016 DATA DIVISION.
0017 FILE SECTION.
0018 FD FILE1
0019     LABEL RECORDS ARE STANDARD
0020     VALUE OF FILE-ID IS "A:$$$.SUB"
0021     DATA RECORDS ARE NEXT-PROGRAM.
0022 01 NEXT-PROGRAM PIC X(16).
0023 WORKING-STORAGE SECTION.
0024 01 W-NEXT-PROGRAM.
0025     02 NAME-SIZE PIC X VALUE ""07"".
0026     02 NAME PIC X(7) VALUE "ED TEXT".
0027     02 STOPPER PIC 99 COMP VALUE ZERO.
0028 PROCEDURE DIVISION.
0029 BEGIN.
0030     OPEN OUTPUT FILE1.
0031     MOVE W-NEXT-PROGRAM TO NEXT-PROGRAM.
0032     WRITE NEXT-PROGRAM.
0033     CLOSE FILE1.
0034     STOP RUN.
0035 END PROGRAM TSUBMIT.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID. TRANSFER.
    * this program calls an assembly language program called
    * "trans". it is used to transfer files from CP/M to
    * PTDOS a unix like operating system.
0003 ENVIRONMENT DIVISION.
0004 CONFIGURATION SECTION.
0005 SOURCE-COMPUTER. 8080-CPU.
0006 OBJECT-COMPUTER. 8080-CPU
    * the following is the actual ending address for this
    * program. the assembly language program is orged just
    * after it.
0007     MEMORY SIZE 16383 CHARACTERS.
0008 INPUT-OUTPUT SECTION.
0009 FILE-CONTROL.
0010     SELECT FILE1  ASSIGN TO INPUT DISK
0011        ORGANIZATION IS SEQUENTIAL
0012        ACCESS MODE IS SEQUENTIAL.
0013 DATA DIVISION.
0014 FILE SECTION.
0015 FD  FILE1
0016     LABEL RECORDS ARE STANDARD
0017     VALUE OF FILE-ID IS IN-FILE-NAME
0018     BLOCK CONTAINS 1 RECORD
0019     DATA RECORDS ARE TESTA.
0020 01  TESTA.
0021     02 REC1 PICTURE IS X(256).
0022 WORKING-STORAGE SECTION.
0023 01 ANSWER PIC X VALUE "Y".
0024 01  IN-FILE-NAME PIC X(14) VALUE "A:TXX.CBL     ".
0025 01  OUT-FILE-NAME PIC X(10) VALUE "TXX/1     ".
0026 01  TRANSFER-TYPE PIC 9 VALUE 1.
0027 01  TRANSFER-FUNCTION PIC X VALUE "1".
0028 01  TRANSFER-ERROR PIC XX VALUE "00".
0029 PROCEDURE DIVISION.
0030 BEGIN.
0031     DISPLAY "ENTER INPUT CP/M FILE NAME " IN-FILE-NAME.
0032     ACCEPT IN-FILE-NAME.
0033     OPEN INPUT FILE1.
0034     DISPLAY "ENTER OUTPUT PTDOS FILE NAME " OUT-FILE-NAME.
0035     ACCEPT OUT-FILE-NAME.
0036     DISPLAY "ENTER FILE TRANSFER TYPE".
0037     DISPLAY "1=FIXED   2=CRLF-CR (1/2)? ".
0038     ACCEPT TRANSFER-TYPE.
0039     MOVE 1 TO TRANSFER-FUNCTION.
0040     CALL "TRANS" USING OUT-FILE-NAME
0041       TRANSFER-TYPE TRANSFER-FUNCTION TRANSFER-ERROR
0042       TESTA.
0043     IF TRANSFER-ERROR NOT EQUAL "00"
0044       DISPLAY "PTDOS OPEN ERROR " TRANSFER-ERROR
0045       STOP RUN.
0046 BEGIN2.
0047     MOVE SPACE TO TESTA.
0048     READ FILE1 AT END GO TO EOJ.
0049     MOVE 3 TO TRANSFER-FUNCTION.
```

```
0050      CALL "TRANS" USING OUT-FILE-NAME
0051        TRANSFER-TYPE TRANSFER-FUNCTION TRANSFER-ERROR
0052        TESTA.
0053      IF TRANSFER-ERROR = "00" GO TO BEGIN2.
0054      DISPLAY "PTDOS WRITE ERROR".
0055      STOP RUN.
0056 EOJ.
0057      CLOSE FILE1.
0058      MOVE 2 TO TRANSFER-FUNCTION.
0059      CALL "TRANS" USING OUT-FILE-NAME
0060        TRANSFER-TYPE TRANSFER-FUNCTION TRANSFER-ERROR
0061        TESTA.
0062      DISPLAY "ANOTHER FILE (Y/N)? "
0063      ACCEPT ANSWER.
0064      IF ANSWER = "Y" GO TO BEGIN.
0065      STOP RUN.
0066 END PROGRAM TRANSFER.


0001 ;  THIS PROGRAM IS "TRANS"
0002 ;  IT IS AN ASSEMBLY LANGUAGE PROGRAM THAT IS CALLED
0003 ;  BY THE PRIOR COBOL PROGRAM NAMED TRANSFER.
0004 ;  IT TRANSFERS CP/M FILES TO PTDOS A UNIX LIKE OPERATING
0005 ;  SYSTEM.
0006 ;  IT IS AN EXAMPLE OF AN ASSEMBLY LANGUAGE CALLED
     ;  PROGRAM
0007 ;  after this program is assembled the .HEX file must be
     ;  converted to an .OBJ file.  use the program called
     ;  CONVHEX to do the conversion.
0008 RELOC EQU 0  ;4200H FOR TRS-80
0009 ;  SET UP AS FOLLOWS
0010 ;  BO              LOAD PTDOS
0011 ; *S               GO TO SOLOS
0012 ;  BO              LOAD CP/M FROM LIFEBOAT 32K
0013 ;
0014  COPY PTDEFS  ;THIS FILE CONTAINS THE PTDOS DEFINITIONS
0015  ORG 16384+RELOC
0016  XEQ START  ;necessary for ptdos assembler
0017 START EQU $  ;ENTRY FROM COBOL PROGRAM
0018  SHLD SAV1  ;OUT-FILE-NAME
0019  LXI H,0
0020  DAD SP
0021  SHLD SAVSP
0022  LXI SP,STACK  ;SET UP THE STACK
0023  CALL GETP
0024  LHLD SAV3  ;TRANSFER-FUNCTION
0025  MOV A,M  ;GET CODE
0026  CPI '1'  ;OPEN?
0027  JZ OPEN
0028  CPI '2'  ;CLOSE?
0029  JZ CLOSE
0030  CPI '3'  ;WRITE?
0031  JZ WRITE
0032 ; ERROR TRANSFER FUNCTION NOT 1,2,3
0033 ERRT LXI D,3232H  ;22
0034 EXIT EQU $
```

```
0035  LHLD SAV4   ;TRANSFER-ERROR
0036  MOV M,D
0037  DCX H
0038  MOV M,E
0039  LHLD SAVSP
0040  SPHL
0041  RET
0042 GETP EQU $
0043  XCHG
0044  SHLD SAV2   ;TRANSFER-TYPE
0045  PUSH B
0046  POP H   ;POINTS TO TABLE OF ADDRESS LEFT END
0047  MOV E,M
0048  INX H
0049  MOV D,M
0050  XCHG
0051  SHLD SAV3   ;TRANSFER-FUNCTION
0052  XCHG
0053  INX H
0054  MOV E,M
0055  INX H
0056  MOV D,M
0057  XCHG
0058  SHLD SAV4   ;TRANSFER-ERROR
0059  XCHG
0060  INX H
0061  MOV E,M
0062  INX H
0063  MOV D,M
0064  LXI H,255
0065  MOV A,E
0066  SUB L
0067  MOV L,A
0068  MOV A,D
0069  SBB H
0070  MOV H,A
0071  SHLD SAV5   ;LEFT END OF RECORD TO BE OUTPUT
0072  RET
0073 OPEN EQU $
0074  LHLD SAV1   ;OUT-FILE-NAME  RIGHT END
0075  LXI D,ONAME+9
0076  MVI C,10
0077 OP1 EQU $
0078  MOV A,M
0079  STAX D
0080  DCX H
0081  DCX D
0082  DCR C
0083  JNZ OP1
0084 ;  the next 9 lines is a ptdos open function
0085  MVI A,40H   ;OPEN CREATE IF NECESSARY
0086  LXI D,OBUFF
0087  LXI H,ONAME
0088  CALL PSCAN
0089  JC ERROR
0090  JZ ERROR
```

```
0091  MOV A,E  ;FILE NUMBER
0092  CPI 255  ;-1 for cpm
0093  JZ ERROR
0094  STA OFILENUMBER  ;ptdos uses file numbers
0095  LXI D,3030H  ;GOOD EXIT for the cobol program
0096  JMP EXIT
0097 ERROR EQU $
0098  MOV D,E
0099  MVI E,'9'
0100  JMP EXIT
0101 CLOSE EQU $  ;ptdos close function follows
0101  LDA OFILENUMBER
0101  CALL SYS
0101  DB EOFOP  ;END FILE
0101  JMP ERROR
0102  LDA OFILENUMBER
0103  CALL SYS
0104  DB CLOOP
0105  JMP $  ;NO ERRORS RETURNED ON CLOSE
0106  LXI D,3030H  ;good close message for the cobol program
0107  JMP EXIT
0108 WRITE EQU $
0109  LHLD SAV2  ;TRANSFER-TYPE
0110  MOV A,M
0111  CPI '2'  ;DROP THE LF'S
0112  JZ WT2
0113  CPI '1'
0114  JNZ ERRT  ;ERROR TRANSFER-TYPE CODE
0115  LHLD SAV5  ;LEFT-END
0116  XCHG
0117  LXI B,256
0118 WT1 EQU $  ;ptdos write function follows
0119  LDA OFILENUMBER
0120  CALL SYS
0121  DB WBLOP  ;WRITE BLOCK
0122  JMP ERROR
0123  LXI D,3030H  ;GOOD WRITE for cobol program
0124  JMP EXIT
0125 WT2 EQU $  ;DROP THE LF'S
0126  LHLD SAV5
0127  LXI D,BUFF2
0128  LXI B,256
0129 WT2A EQU $
0130  MOV A,M
0131  CPI 0AH  ;LF
0132  JZ WT2B
0133  CPI 1AH  ;CP/M'S EOF FOR ASCII FILES
0134  JZ WT2C
0135  STAX D
0136  INX D
0137 WT2B EQU $
0138  INX H
0139  DCX B
0140  MOV A,C
0141  ORA B
0142  JNZ WT2A
```

```
0143 WT2C EQU $
0144  LXI H,BUFF2
0145  MOV A,E
0146  SUB L
0147  MOV E,A
0148  MOV A,D
0149  SBB H
0150  MOV D,A
0151  PUSH D
0152  POP B  ;SIZE OF THIS WRITE FOR PTDOS
0153  XCHG
0154  JMP WT1
0155 SAV1 DW 0  ;OUT-FILE-NAME
0156 SAV2 DW 0  ;TRANSFER-TYPE
0157 SAV3 DW 0  ;TRANSFER-FUNCTION
0158 SAV4 DW 0  ;TRANSFER-ERROR
0159 SAV5 DW 0  ;OUTPUT RECORD
0160 SAVSP DW 0  ;STACK POINTER
0161 ; all that follows is for ptdos
0162  DB '.'+80H
0163  DW 04C0H
0164  DB 0
0165 OBUFF DS 20
0166  DS 20
0167 STACK DW 0
0168 ONAME DS 10
0169  DB 0
0170 OFILENUMBER DB 0
0171 BUFF2 DS 256
0172 LAST DB 0
0173  END START  ;necessary for cpm assembler
```

LIST OF REFERENCES

Ashley, Ruth, Structured COBOL self-teaching guide, Wiley, 1980.

Bauer, F.L, & Eickel J., Compiler Construction, Springer-Verlag, 1976.

Chumra, L., & Ledgard, H., COBOL WITH STYLE: Programming Proverbs, Hayden, 1976.

ELLIS COMPUTING, NEVADA COBOL Application Packages Book1, ELLIS COMPUTING, 1980.

ELLIS COMPUTING, NEVADA EDIT, ELLIS COMPUTING, 1982.

ELLIS COMPUTING, NEVADA SORT, ELLIS COMPUTING, 1982.

ELLIS COMPUTING, NEVADA FORTRAN, ELLIS COMPUTING, 1982.

Gries, D., Compiler Construction for Digital Computers, Wiley, 1971.

Hogan, T., CPM Users guide, Osborne, 1981.

Kunth, D., The Art of Computer Programming, Addison Wesley, 1973.

McCracken, D.M., A Simplified Guide to Structured COBOL Programming, Wiley, 1976.

Parkin Andrew, COBOL for students, Edward Arnold, 1975.

Starkweather, J., NEVADA PILOT, ELLIS COMPUTING, 1981.

FIPS Pub 21-1, National Technical Information Service, 425 13th Street, NW, Washington,DC 20004. Includes the ANSI-74 standards manual.

Receiver General of Canada, Journal of Developement, Dept of Supply & Services, Material Data Management Center, 4/B-1 Place du Portage, Phase III, 11 Laurier Street, Hull, Quebec, Canada K1A OS5

X3J4 Chairman, CBEMA, Suite 1200, 1828 L Street, NW, Washington, DC 20036 Ask for latest COBOL report.

Ellis Computing
3917 Noriega Street, San Francisco, CA 94122

SOFTWARE LICENSE AGREEMENT

IMPORTANT: All Ellis Computing programs are sold only on the condition that the purchaser agrees to the following License.

ELLIS COMPUTING agrees to grant and the Customer agrees to accept on the following terms and conditions nontransferable and nonexclusive Licenses to use the software program(s) (Licensed Programs) herein delivered with this Agreement.

TERM:

This Agreement is effective from the date of receipt of the above-referenced program(s) and shall remain in force until terminated by the Customer upon one month's prior written notice, or by Ellis Computing as provided below.

Any License under this Agreement may be discontinued by the Customer at any time upon one month's prior written notice. Ellis Computing may discontinue any License or terminate this Agreement if the Customer fails to comply with any of the terms and conditions of this Agreement.

LICENSE:

Each program License granted under this Agreement authorizes the Customer to use the Licensed Program in any machine readable form on any single computer system (referred to as System). A separate license is required for each System on which the Licensed Program will be used.

This Agreement and any of the Licenses, programs or materials to which it applies may not be assigned, sublicensed or otherwise transferred by the Customer without prior written consent from Ellis Computing. No right to print or copy, in whole or in part, the Licensed Programs is granted except as hereinafter expressly provided.

PERMISSION TO COPY OR MODIFY LICENSED PROGRAMS:

The customer shall not copy, in whole or in part, any Licensed Programs which are provided by Ellis Computing in printed form under this Agreement. Additional copies of printed materials may be acquired from Ellis Computing.

The NEVADA COBOL COMPILER Licensed Programs which are provided by Ellis Computing in machine readable form may be copied, in whole or in part, in machine readable form in sufficient number for use by the Customer with the designated System, for back-up purposes, or for archive purposes. The original, and any copies of the Licensed Programs, in whole or in part, which are made by the Customer shall be the property of Ellis Computing. This does not imply that Ellis Computing owns the media on which the Licensed Programs are recorded.

The NEVADA COBOL Licensed Program called "RUN" which is provided by Ellis Computing may be distributed to third parties.

The Customer agrees to reproduce and include the copyright notice of Ellis Computing on all copies, in whole or in part, in any form, including partial copies of modifications, of Licensed Programs made hereunder.

PROTECTION AND SECURITY:

The Customer agrees not to provide or otherwise make available the NEVADA COBOL COMPILER Program including but not limited to program listings, object code and source code, in any form, to any person other than Customer or Ellis Computing employees, without prior written consent from Ellis Computing, except with the Customer's permission for purposes specifically related to the Customer's use of the Licensed Program.

DISCLAIMER OF WARRANTY:

Ellis Computing makes no warranties with respect to the Licensed Programs.

LIMITATION OF LIABILITY:

THE FOREGOING WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL ELLIS COMPUTING BE LIABLE FOR CONSEQUENTIAL DAMAGES EVEN IF ELLIS COMPUTING HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

GENERAL:

If any of the provisions, or portions thereof, of this Agreement are invalid under any applicable statute or rule of law, they are to that extent to be deemed omitted. This is the complete and exclusive statement of the Agreement between the parties which supercedes all proposals, oral or written, and all other communications between the parties relating to the subject matter of this Agreement.
This Agreement will be governed by the laws of the State of California.

CORRECTIONS AND SUGGESTIONS

All suggestions and problems must be reported in writing. Please include sample source listings if possible.

ERRORS IN MANUAL:

SUGGESTIONS FOR IMPROVEMENTS TO MANUAL:

ERRORS IN COMPILER: For bugs we need to know the machine configuration, CP/M version # (2.2 vs 1.4), Nevada COBOL version # and a complete listing of the source program.

SUGGESTIONS FOR IMPROVEMENT TO COMPILER:

MAIL TO: Ellis Computing
3917 Noriega Street
San Francisco,CA 94122

FROM: NAME_____ DATE_____

ADDRESS_____

CITY,STATE,ZIP_____

COUNTRY_____

PHONE NUMBER_____

VERSION _____ SERIAL #_____

If you wish a reply include a self-addressed postage-paid envelope. Thank you.

DISCLAIMER

All Ellis Computing computer programs are distributed on an "AS IS" basis without warranty.

ELLIS COMPUTING makes no warranties, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. In no event will ELLIS COMPUTING be liable for consequential damages even if ELLIS COMPUTING has been advised of the possibility of such damages.

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES