NEVADA COBOL
for the Commodore 64
NEVADA COBOL
for the Commodore 64
Programmers’ Reference Manual

Copyright © 1983 Commodore Electronics Limited

Commodore Electronics Ltd.
1200 Wilson Drive
West Chester, PA 19380
COPYRIGHT

Copyright, 1983 by Ellis Computing and Commodore Electronics, Ltd. The distribution and sale of this product are intended for the use of the original purchaser only. Lawful users of this program are hereby licenced only to read the program, from its medium into memory of a computer, solely for the purpose of executing the program. Duplicating, copying, selling or otherwise distributing this product is a violation of the law.

This manual is copyrighted and all rights are reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine readable form without prior consent, in writing, from Ellis Computing and Commodore Electronics Ltd.

DISCLAIMER

ELLIS COMPUTING AND COMMODORE ELECTRONICS LTD. ("COMMODORE") MAKE NO WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THE PROGRAM DESCRIBED HEREIN, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. THIS PROGRAM IS SOLD "AS IS". THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE IS WITH THE BUYER. SHOULD THE PROGRAM PROVE DEFECTIVE FOLLOWING ITS PURCHASE, THE BUYER (AND NOT THE CREATOR OF THE PROGRAM, ELLIS COMPUTING, COMMODORE, THEIR DISTRIBUTORS OR THEIR RETAILERS) ASSUMES THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION AND ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES. IN NO EVENT WILL ELLIS COMPUTING AND/OR COMMODORE BE LIABLE FOR DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE PROGRAM EVEN IF IT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME LAWS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITIES FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY.

TRADEMARKS

NEVADA COBOL™, NEVADA FORTRAN™, NEVADA PILOT™, NEVADA EDIT™ and Ellis Computing™ are trademarks of Ellis Computing. CP/M is a registered trademark of Digital Research Corporation.
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


have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.
We hope you enjoy using NEVADA COBOL™ for the Commodore 64. NEVADA COBOL for the Commodore 64 runs under CP/M® 2.2 Operating System. Here are several other Commodore software packages which you should know about:

NEVADA FORTRAN™
This is an 8080/8085/Z80 version of FORTRAN IV. The compiler works from disk (also using the assembler) to produce machine code that executes at maximum CPU speed. First, a source program is entered as FORTRAN IV program statements and compiled to produce assembly code. Next, any errors must be corrected. Then, intermediate code is assembled into 8080 object code which is now ready for execution under CP/M.

EASY SCRIPT 64
This is a powerful word processor with table producing capabilities, comprehensive printer controls, easy update facilities, easy document handling, the ability to interact with EASY SPELL 64, and more.

THE WORD MACHINE and THE NAME MACHINE
This is an easy-to-learn and easy-to-use wordprocessing package. Perfect for letters, address lists, memos, and notes, these programs let you overtype, insert, and delete text; personalize form letters; and print in draft, formal, or informal formats.

EASY SPELL 64
Easy Spell 64 features the following: the automatic correction of spelling errors, the ability to count the number of words in your manuscript and interact with Easy Script 64, and a built-in 20,000 word dictionary that lets you add words not already stored there.

EASY MAIL 64
With Easy Mail 64, you can easily manage your address files. Label printing is also simplified with Easy Mail's ability to search for specific fields/categories. The program's features include entry, change, or deletion of a file by name or number; the capability to print one or two abreast labels; a HELP screen; and the ability to print a complete printout of all the data in each of your records.

EASY CALC 64
Easy Calc 64 is an easy-to-use electronic spread sheet which features editing functions and HELP screens. With Easy Calc 64, you can also print bar charts and individually formatted tables.

THE MANAGER
The Manager is a general data base for handing your files.

NEVADA COBOL and NEVADA FORTRAN are trademarks of Ellis Computing CP/M is a registered trademark of Digital Research, Inc.
THE COMMODORE 64 MACRO ASSEMBLER DEVELOPMENT SYSTEM
This package is designed for experienced Assembly language programmers. Everything you need to create, assemble, load, and execute 6500 series Assembly language code is included.

SCREEN EDITOR
The Screen Editor helps you design software by letting you create and edit your own screens. This programming tool is for users with some computer experience.

SUPER EXPANDER 64
This cartridge is a powerful extension of the BASIC language which gives you the commands needed to easily access and implement Commodore's graphics, music, and sound capabilities. You will be amazed at how quickly and easily you can plot points and lines; draw arcs, circles, ellipses, rectangles, triangles, octagons; paint shapes with specified colors; read game paddle and joystick locations; create music and sound; display text; split screens to display both text and graphics; and program the function keys.

THE EASY FINANCE SERIES
Commodore is proud to announce an entire series of EASY FINANCE software packages which may solve many of your business and personal needs. The EASY FINANCE series is called “easy” because all of the programs are simple to operate and require no programming experience. Here is a brief description of each:

EASY FINANCE I — LOANS
LOANS shows you how to make the most out of your hard-earned money by calculating 12 different loan concepts for you. Principal, regular payment, last payment, and remaining balance are just some of the functions EASY FINANCE I can determine.

EASY FINANCE II — INVESTMENTS
INVESTMENTS helps you make the right financial decisions by showing you how to make the most out of 16 investment concepts. Functions such as future investment value, initial investment, and internal rate of return can be calculated.

EASY FINANCE III — ADVANCED INVESTMENTS
ADVANCED INVESTMENTS is an advanced version of EASY FINANCE II. It shows you how to make the most out of 16 more investment concepts. Financial terms are clarified and functions such as discount commercial paper, financial management rate of return, and financial leverage and earnings per share are included.

EASY FINANCE IV — BUSINESS MANAGEMENT
This is a business management package that shows managers how to make the right decisions about production, inventory, control, compensation, and much more. Lease purchase analysis, depreciation switch, and optimal order quantity are some of the 21 functions this program can calculate for you.
EASY FINANCE V — STATISTICS
STATISTICS shows you how to make the most out of statistics. This includes payoff matrix analysis, regression analysis forecasting, and apportionment by ratios.

Please contact your local Commodore dealer for additional information on other software available for your Commodore computer.

Thank you for owning a Commodore computer. Now that you are a member of the Commodore family, maybe you'd like to expand your computer's family. Here is a list of additional hardware which is compatible with your Commodore computer:

1525 Printer
This printer is an 80-column, dot-matrix, impact printer for creating printouts and hard-copies from your VIC 20 or Commodore 64. The printer features 30 characters per second print speed and prints graphics and text characters.

1526 Printer
This bi-directional, 80 column, dot-matrix, impact printer is excellent for creating printouts and hardcopies from your computer. The printer features programmable line spacing and a print format interpreter.

1520 Plotter/Printer
This is a four color, high resolution plotter that connects directly to your VIC 20 or Commodore 64 computer. With the 1520 Plotter/Printer you can plot on a piece of paper, the unique color graphics that you have created on your screen.

Commodore Speech Module
The speech module cartridge comes with a built-in vocabulary of 234 words which are easily programmed into sentences. The module "talks" in a pleasant female or male voice . . . it can generate other types of voices with special vocabularies geared to each software package. The speech module works with disk, tape, and also has a slot for accepting plug-in cartridges.

1701/1702 Monitor
This full color monitor is compatible with the VIC 20, Commodore 64, and other computers. The 1701/1702 Monitor features high quality resolution video and a built-in speaker with audio amplifier.
**1530 DATASSETTE™**
The 1530 DATASSETTE is a low cost, highly reliable way to store and retrieve programs and data. It features keys for Play, Record, Fast-Forward, Rewind, and Stop. The 1530 DATASSETTE uses standard audio cassette tapes and allows naming of programs and files, verification of programs, and programmable end of tape marker sensing.

**Joystick and Paddles**
Controls for games and entertainment.

**Modem**
The 1600 Modem telephone interface lets you communicate with other computer systems over your telephone line! The modem package includes cassette-tape terminal software, a free password and one-hour subscription to the CompuServe System™* and software controls for duplex, baud rate, and parity. There is also an optional adapter available for non-modular phones. The 1650 Automatic Modem features all of the above plus automatic answer and automatic dial.

**PET 64**
This unique machine combines many of the Commodore 64 features with the capabilities of the Commodore PET. However, sprites, color, and sound are not featured on this machine.

**SX-64/DX-64 Portable Color Computers**
These new computers are Commodore 64’s in a convenient portable style. The model SX-64 (single disk drive) and DX-64 (double disk drive) are excellent investments for business people, as well as affordable for today’s students.

DATASSETTE is a trademark of Commodore Electronics, Ltd.
PREFACE

This reference manual assumes you already have the knowledge to program in COBOL and have read the Commodore 64 CP/M Operating System User’s Guide. An additional list of supplementary materials can be found in the back of this book.

This manual is not a tutorial and will not teach you “how to” program in COBOL. However, for the experienced COBOL programmer who is already familiar with the CP/M Operating System, this manual provides the necessary tools for using NEVADA COBOL on your Commodore 64. The manual includes:

• General concepts of COBOL programming
• Details on using COBOL
• A list of Reserved Words
• A description of the four Divisions of a COBOL program
• Sample Programs
• Error Codes and Messages
• A Glossary of Terms

We hope you enjoy using NEVADA COBOL on your Commodore 64.

USER CONVENTIONS

It is recommended that you familiarize yourself with the Commodore keyboard. Here is a brief discussion of certain keys and symbols, and their respective use in the NEVADA COBOL manual. This will also help you interpret the syntax of the commands, including their optional features.

{ } Braces indicate that a choice must be made

[ ] Square brackets indicate optional information that may be omitted

... Several consecutive periods, “ellipses”, specify that the preceding clauses can be repeated.

<CR> To continue on after a line of input, press the RETURN key.

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

Emboldened UPPER-CASE characters are key words that must be used

Upper-case characters that are not emboldened are optional reserved words
The following ANSI-1974 COBOL Reserved Words can be used with NEVADA COBOL for the Commodore 64.

<table>
<thead>
<tr>
<th>Reserved Word</th>
<th>Reserved Word</th>
<th>Reserved Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>ELSE</td>
<td>LINE</td>
</tr>
<tr>
<td>ACCESS</td>
<td>END</td>
<td>LINES</td>
</tr>
<tr>
<td>ADD</td>
<td>ENVIRONMENT</td>
<td>LINKAGE</td>
</tr>
<tr>
<td>ADVANCING</td>
<td>EQUAL</td>
<td>LOW-VALUE</td>
</tr>
<tr>
<td>AFTER</td>
<td>ERROR</td>
<td>LOW-VALUES</td>
</tr>
<tr>
<td>ALL</td>
<td>EXIT</td>
<td>MEMORY</td>
</tr>
<tr>
<td>ALPHABETIC</td>
<td>FD</td>
<td>MODE</td>
</tr>
<tr>
<td>ALTER</td>
<td>FILE</td>
<td>MODULES</td>
</tr>
<tr>
<td>AND</td>
<td>FILE-CONTROL</td>
<td>MOVE</td>
</tr>
<tr>
<td>AREA</td>
<td>FILLER</td>
<td>MULTIPLY</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>FIRST</td>
<td>NEXT</td>
</tr>
<tr>
<td>AT</td>
<td>FOR</td>
<td>NO</td>
</tr>
<tr>
<td>AUTHOR</td>
<td>FROM</td>
<td>NOT</td>
</tr>
<tr>
<td>BEFORE</td>
<td>GIVING</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>BLOCK</td>
<td>GO</td>
<td>OBJECT-COMPUTER</td>
</tr>
<tr>
<td>BY</td>
<td>GREATER</td>
<td>OCCURS</td>
</tr>
<tr>
<td>CALL</td>
<td>HIGH-VALUE</td>
<td>OF</td>
</tr>
<tr>
<td>CANCEL</td>
<td>HIGH-VALUES</td>
<td>OFF</td>
</tr>
<tr>
<td>CHARACTERS</td>
<td>I-O</td>
<td>OMITTED</td>
</tr>
<tr>
<td>CLOSE</td>
<td>I-O-CONTROL</td>
<td>ON</td>
</tr>
<tr>
<td>COLLATING</td>
<td>IDENTIFICATION</td>
<td>OPEN</td>
</tr>
<tr>
<td>COMMA</td>
<td>IF</td>
<td>OR</td>
</tr>
<tr>
<td>COMP</td>
<td>INITIAL</td>
<td>ORGANIZATION</td>
</tr>
<tr>
<td>COMPUTATIONAL</td>
<td>INPUT</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>CONFIGURATION</td>
<td>INPUT-OUTPUT</td>
<td>PAGE</td>
</tr>
<tr>
<td>CONTAINS</td>
<td>INSPECT</td>
<td>PERFORM</td>
</tr>
<tr>
<td>COPY</td>
<td>INSTALLATION</td>
<td>PIC</td>
</tr>
<tr>
<td>CURRENCY</td>
<td>INTO</td>
<td>PICTURE</td>
</tr>
<tr>
<td>DATA</td>
<td>INVALID</td>
<td>PROCEDURE</td>
</tr>
<tr>
<td>DATE-Compiled</td>
<td>IS</td>
<td>PROCEED</td>
</tr>
<tr>
<td>DATE-Written</td>
<td>JUST</td>
<td>PROGRAM</td>
</tr>
<tr>
<td>DEBUGGING</td>
<td>JUSTIFIED</td>
<td>PROGRAM-ID</td>
</tr>
<tr>
<td>DECIMAL-POINT</td>
<td>KEY</td>
<td>QUOTE</td>
</tr>
<tr>
<td>DELIMITER</td>
<td>LABEL</td>
<td>QUOTES</td>
</tr>
<tr>
<td>DEPENDING</td>
<td>LEADING</td>
<td>RANDOM</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>LEFT</td>
<td>READ</td>
</tr>
<tr>
<td>DIVIDE</td>
<td>LESS</td>
<td>RECORD</td>
</tr>
<tr>
<td>DIVISION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECORDS</td>
<td>SIGN</td>
<td>THRU</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>REDEFINES</td>
<td>SIZE</td>
<td>TIMES</td>
</tr>
<tr>
<td>RELATIVE</td>
<td>SOURCE-COMPUTER TO</td>
<td></td>
</tr>
<tr>
<td>REPLACING</td>
<td>SPACE</td>
<td>UNTIL</td>
</tr>
<tr>
<td>REWRITE</td>
<td>SPACES</td>
<td>USAGE</td>
</tr>
<tr>
<td>RIGHT</td>
<td>SPECIAL-NAMES</td>
<td>USING</td>
</tr>
<tr>
<td>ROUNDED</td>
<td>STANDARD</td>
<td>VALUE</td>
</tr>
<tr>
<td>RUN</td>
<td>STATUS</td>
<td></td>
</tr>
<tr>
<td>SAME</td>
<td>STOP</td>
<td></td>
</tr>
<tr>
<td>SECTION</td>
<td>SUBTRACT</td>
<td>WITH</td>
</tr>
<tr>
<td>SECURITY</td>
<td>SYNC</td>
<td>WORDS</td>
</tr>
<tr>
<td>SELECT</td>
<td>SYNCHRONIZED</td>
<td>WORKING-STORAGE</td>
</tr>
<tr>
<td>SENTENCE</td>
<td>TALLYING</td>
<td>WRITE</td>
</tr>
<tr>
<td>SEQUENCE</td>
<td>THAN</td>
<td>ZERO</td>
</tr>
<tr>
<td>SEQUENTIAL</td>
<td>THROUGH</td>
<td>ZEROES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ZEROS</td>
</tr>
</tbody>
</table>

The following words are NEVADA COBOL Reserved Words (Not ANSI-1974):

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

**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** is used in conjunction with the CP/M assembler for those of you wishing to write called programs in assembly language. It 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). If you do a lot of work in assembly language, you may want to get Nevada FORTRAN, as it comes with an assembler that is compatible with Nevada COBOL.

FILES THAT 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.
1 INTRODUCTION

COBOL (Common Business Oriented Language) is a programming language that has been used for business applications since the early 1960's. COBOL is based on English and uses certain words and syntax rules derived from English. NEVADA COBOL for the Commodore 64 is an updated subset of COBOL and is designed for small businesses with a Commodore 64 microcomputer.

As in English, the basic unit of COBOL is the word. A "word" may be a COBOL reserved word or a word that you define. Reserved words have specific meaning to the COBOL compiler; programmer-defined words can be assigned to data-names and procedure-names and must conform to the COBOL rules for the formation of names.

As the programmer, you combine Reserved Words and your programmer-defined words into clauses and statements. A clause or a statement specifies one action to be performed, one condition to be analyzed, or one description of data. These clauses and statements can then be combined into sentences.

Sentences may be simple (one statement or clause), or they may be compound (several statements or several clauses). Logically related sentences can be combined into paragraphs; related paragraphs can be combined into sections. These sections are then placed in one of the appropriate program divisions.

There are four divisions in a COBOL program:

INDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.

Each of the four divisions is briefly described in the chart at the end of this chapter. A more detailed description of the divisions is given in the subsequent chapters.
SETTING UP

The following is a list of the required Hardware:

- Your Commodore 64 computer
- The Commodore Z80 microprocessor (This is your CP/M Operating System cartridge.)
- A Commodore 1541 single disk drive or a Commodore IEEE interface and a CBM dual disk drive model 4040
- A video display monitor such as the Commodore Color Monitor Model 1701/1702

The following is a list of the required Software:

- Commodore’s CP/M Operating System disk
- A text editor ED. COM is found on your Commodore CP/M Operating System disk.

GETTING STARTED

Throughout our discussion we will be referring to the following disks:

NEVADA COBOL Data disk
   Included in your NEVADA COBOL software package, this disk should only be read. A listing of the files contained on this disk can be found at the front of this manual.

CP/M Operating System disk
   This is your Commodore CP/M Operating System disk that you use with your Z80 cartridge.

CP/M-NEVADA COBOL Operations disk
   This is a disk which you create.

Note that you should NEVER write on your NEVADA COBOL Data disk. To prevent any mistakes from occurring, be sure that your NEVADA COBOL Data disk is write protected. (Place a standard protection label over the “square cornered” notch on the disk.) Before continuing, consult your Commodore 64 CP/M Operating System User’s Guide if you are not familiar with the DIR, ERA, PIP, and STAT commands.

Follow these steps to get started using NEVADA COBOL:

1. Use one of your CP/M Operating System disk backup copies to create your CP/M-NEVADA COBOL Operations disk. If you don’t have a backup copy of the CP/M Operating System disk, see Section 4.2 The Copy Utility in your Commodore CP/M Operating System User’s Guide.
2. Use the CP/M ERA command to erase all of the files except the PIP.COM and ED.COM files from your newly created NEVADA COBOL Operations disk.

3. If you are using the IEEE interface and the Commodore dual disk drive, insert the newly created CP/M-NEVADA COBOL Operations disk into drive 0 (A). Insert the NEVADA COBOL Data disk into drive 1 (B). Now, boot CP/M.

   If you have a 1541 single disk drive, insert the newly created CP/M NEVADA COBOL Operations disk into the disk drive and boot CP/M.

   After CP/M is booted, the computer automatically displays an ‘A >’ prompt. This signifies disk A is ready to be accessed. Here is a sample of how each file should be copied on a single disk drive system. Remember, we will refer to the NEVADA COBOL Data disk as disk ‘B’ and the CP/M-NEVADA COBOL Operations disk as disk ‘A’.

   Use the PIP command to copy the files from your NEVADA COBOL Data disk to the CP/M-NEVADA COBOL Operations disk. PIP will prompt you throughout the entire copy process. To invoke the PIP program, input PIP after the ‘A >’ prompt:

   
   A> PIP <CR>

   After RETURN is pressed, an asterisk (*) is displayed on the following line. Now, copy and verify the file CONFIG.CBL:

   *A:CONFIG.CBL = B:CONFIG.CBL[V]

   The following prompt will then be displayed:

   Insert disk B into drive 0, press return

   Insert the NEVADA COBOL Data disk and press RETURN. The PIP program will read the file CONFIG.CBL from the disk. After a short period of time, the following prompt will be displayed:

   Insert disk A into drive 0, press return

   Insert the CP/M-NEVADA COBOL Operations disk and press RETURN. The PIP program will now write onto the disk the CONFIG.CBL file. Upon completion, an asterisk will appear. You can now continue copying your files from the NEVADA COBOL Data disk to the CP/M-NEVADA COBOL Operations disk using the following format:

   *A:destination = B:source[V]
Continue this process until all files from the NEVADA COBOL Data disk are copied to the CP/M-NEVADA COBOL Operations disk. PIP can be terminated at any time by pressing RETURN after any asterisk (*) prompt.

We suggest now placing your NEVADA COBOL Data disk in a safe place. You will not need it unless something happens to your Operations disk. Depending on how much program development you do, it may be wise to backup your CP/M-NEVADA COBOL Operations disk at least once a day.

**COBOL PROGRAMMING CONCEPTS**

In English, vocabulary and punctuation are used to form sentences so that concepts can be clearly understood. In COBOL, similar techniques are used to form program statements. Here is a table to highlight some of these techniques.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Punctuation</td>
<td>A, ; must immediately follow the last character of a word and be followed by a space. The , and ; are interchangeable. The opening parenthesis, ‘(‘, cannot be followed by a space; the closing parenthesis, ‘)’ cannot be preceded by a space.</td>
<td>0001 MOVE MONEY (10) TO SAVING AND LOAN.</td>
</tr>
<tr>
<td>Verbs</td>
<td>Verbs are used in the PROCEDURE DIVISION. All verbs fall into the following categories:</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>Imperative</td>
<td>Gives a direct processing instruction</td>
<td>GO TO PERFORM</td>
</tr>
<tr>
<td>Conditional</td>
<td>Tests a condition (IF cannot appear in imperative statements)</td>
<td>IF A = B</td>
</tr>
<tr>
<td>Compiler</td>
<td>Instructs the compiler during compilation time</td>
<td>COPY</td>
</tr>
<tr>
<td>Directing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input-Output</td>
<td>Assists in the transfer of data between peripherals and memory</td>
<td>OPEN, CLOSE, READ, WRITE</td>
</tr>
<tr>
<td>MOVE Verb</td>
<td>Transfers data from one area of memory to another</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To send to more than one field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To transfer numbers, use a numeric MOVE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a) Align the decimal points</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b) Move the digits</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c) Fill in zeros</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE (old field) TO (new field)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE (old field) TO (new field), (new field)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>To add, subtract, multiply and divide values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADD (value) TO (field)</td>
</tr>
<tr>
<td></td>
<td>SUBTRACT (field1) FROM (field2) GIVING (field3)</td>
</tr>
<tr>
<td></td>
<td>MULTIPLY (number of times) BY (receiving field)</td>
</tr>
<tr>
<td></td>
<td>DIVIDE (divisor) INTO (dividend) GIVING (quotient)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence Control</th>
<th>To pass control to a statement that is not in sequential order</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>To permanently transfer control</td>
</tr>
<tr>
<td></td>
<td>To temporarily transfer control and return to the statement following the sequence interruption</td>
</tr>
<tr>
<td></td>
<td>The last portion of a sequence control procedure consists of either of these</td>
</tr>
<tr>
<td></td>
<td>GOTO (procedure name)</td>
</tr>
<tr>
<td></td>
<td>PERFORM</td>
</tr>
<tr>
<td></td>
<td>PERFORM (A) THRU (B)</td>
</tr>
<tr>
<td></td>
<td>EXIT</td>
</tr>
</tbody>
</table>
THE FOUR DIVISIONS OF A COBOL PROGRAM

The IDENTIFICATION DIVISION lets you specify:
  • Program Name
  • Programmer’s Name
  • System or application
  • Dates when written and compiled
  • Security restrictions

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003 T6WF.
0004* THIS PROGRAM CREATES A FILE OF FIXED LENGTH
0004* RECORDS IF THE RECORD SIZES ARE CHANGED TO
0004* YOUR NEEDS, CAN BE USED TO CREATE THE SPACE
0004* NEEDED (ALLOCATE) FOR A RANDOM FILE.
```

The ENVIRONMENT DIVISION lets you specify:
  • Source — Computer used to compile the program
  • Object — Computer used to execute the compiled Object program
  • The Input-Output section for identifying the File — Control; i.e., location of each file referenced and how each file will be used
  • Filenames may be up to 30 characters

```
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 DATA DIVISION lets you:
  • Give a detailed description of all the data to be used, i.e., format of each file and record within each file
  • Assign data-names to each of the items of data to be used
  • Describe in the Working Storage Section records and data items that are not part of the files, but are used during the processing of the object program
  • The Working Storage Section identifies intermediate storage areas along with constant values used.
DATA DIVISION.
FILE SECTION.
FD FILE1
LABEL RECORDS ARE STANDARD VALUE OF FILE-ID IS OUT-FILE-NAME BLOCK CONTAINS 1 RECORD DATA RECORDS ARE O-RECORD.
01 O-RECORD.
02 SEQ PIC 9999.
02 REC1 PIC X(156).
02 SEQ2 PIC 9999.
WORKING-STORAGE SECTION.
01 OUT-FILE-NAME PIC X(14) VALUE "A:TESTF.WRK".
01 X1 PIC 9999 VALUE 0001.

The PROCEDURE DIVISION lets you:
- Define the necessary instructions for solving the program.

PROCEDURE DIVISION
BEGIN.
DISPLAY "ENTER OUTPUT FILE NAME".
DISPLAY OUT-FILE-NAME WITH NO ADVANCING.
* TO ACCEPT AND USE THE FILE-NAME JUST DISPLAYED YOU CAN HIT THE <CR> KEY. SEE #2 UNDER ACCEPT.
ACCEPT OUT-FILE-NAME.
OPEN OUTPUT FILE1.
MOVE SPACES TO O-RECORD.
BEGIN2.
MOVE X1 TO SEQ.
MOVE X1 TO SEQ2.
ADD 1 TO X1.
DISPLAY O-RECORD.
WRITE O-RECORD.
IF X1 IS = TO 201
GO TO EOJ.
GO TO BEGIN2.
EOJ.
CLOSE FILE1.
STOP RUN.
END PROGRAM T6WF.
2 RUNNING NEVADA COBOL

Now, boot up the newly created NEVADA COBOL Operations disk. Notice that CP/M display's the amount of memory available. Compiling and executing of COBOL programs should be done with 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 disk because during compile time, data will be written onto it.

The next step is to compile the program called CONFIG. This is done by typing the following:

A> CC CONFIG

The disk drive(s) will work away and the COBOL compiler will finally display SUCCESSFUL COMPILE. If you have any problems compiling, read ahead about compiling a program, as on small disk drives you may have to assign files to other disk drives or make space available on the default drive. Normally, everything should go smoothly and work properly if the compiler has been copied correctly.

Next, type the following:

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
   enter 08
ENTER 2-DIGIT HEXADECIMAL CODE FOR BACKSPACE CURSOR
   enter 08
IS THE BACKSPACE PRECEDED WITH AN ESCAPE CHARACTER (Y/N)?
   enter N
ENTER # OF CHARACTERS ACROSS SCREEN
   enter 40
ENTER # OF LINES PER SCREEN PAGE
   enter 25
DOES YOUR BIOS ISSUE A CR/LF AT THE END OF EACH LINE (Y/N)?
   enter Y
DOES YOUR PRINTER REQUIRE A LINE FEED (Y/N)?
   enter Y
DO YOU WANT TO USE CPM FUNCTION 1 & 2 CONSOLE I-O (Y/N)?
   usually Y (user's option)
   If N, other information will be displayed
   Answer N if you will be sending escape characters to the CRT.

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

CC RENUMBER

Compiling RENUMBER.LBL creates RENUMBER.OBJ which automatically numbers or renumbers user written programs.

Once the CONFIG program has been run and you are satisfied with the ACCEPT/DISPLAY functions, the programs are no longer needed on the CP/M-NEVADA COBOL Operations disk and may be removed as follows:

A> ERA CONFIG.*

A> ERA RENUMBER.CBL

On some single density 5¼ disks, you may want to have a separate disk for compiling only. This disk needs only the following files:

   CC.COM  about 6K.
   W4.COM  about 30K.
   W5.COM  about 6K.

And at run time, you can also have a separate disk. It only needs one file:

   RUN.COM  about 12K.

With this disk file arrangement, the COBOL compiler will work on disks with very limited disk space.
BUILDING A PROGRAM

The first step is to create a COBOL source program file. This file will later be submitted to the COBOL compiler for compilation. Create the file by using a text editor. You can copy an existing COBOL source file such as RENUMBER.CBL that is on the NEVADA COBOL Data disk and create a new program file called, for example, MYPROG.CBL. Then, modify MYPROG.CBL as required. This saves keying time as well as avoiding the possibility of misspelling required keywords.

Each line of the COBOL source file must be terminated with a carriage return line feed. This is automatically done with text editor, ED.COM which should now be on your CP/M-NEVADA COBOL Operations disk.

COBOL CODING FORMAT

<table>
<thead>
<tr>
<th>Field</th>
<th>ANSI-1974</th>
<th>NEVADA COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td>1-6</td>
<td>1-4</td>
</tr>
<tr>
<td>Indicator area</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>A-field</td>
<td>8-11</td>
<td>6-9</td>
</tr>
<tr>
<td>B-field</td>
<td>12-72</td>
<td>10-70</td>
</tr>
</tbody>
</table>

1. You can use either format because 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 (i.e. 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 a line number.
4. Each line must be terminated by a carriage return line feed (ODOA hex).

EXAMPLE:
Columns
123456789012345678901234567890
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. This is because the computer 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 logged-in drive. The optional [u] would be the disk drive of where the OBJECT resides if other than drive (A).

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, the following prompt appears:

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 an 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. Warning, on some implementations of CP/M, it has been reported that these blank lines cause the file to be destroyed. If this should happen, you must not use blank lines or tabs!

EXAMPLE:

Columns
1234567890123456
0001*
*
9999/ head of form comment line is OK
LISTING A PROGRAM
To list a NEVADA COBOL source code program, use the CP/M TYPE command; and, if you have a printer, use CTRL-P.

A> TYPE RENUMBER.CBL[CTRL-P] <CR>
A> TYPE CONFIG.CBL <CR>
A> TYPE W5.CBL <CR>
3 IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION of a COBOL program is entirely 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.

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.]

EXAMPLE:

0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID. TEST1.
0003 AUTHOR. COMMODORE BUSINESS MACHINES.
0004 INSTALLATION. WEST CHESTER, PA
0005 DATE-WRITTEN. JULY 1, 1983.
0006 DATE-COMPILED. JULY 17, 1983.
0007 SECURITY. COPYRIGHT CBM, INC.
0008* comment lines with * in column 5 can be lower-case.

Another statement that can be placed in the IDENTIFICATION DIVISION is the COPY statement. The COPY statement inserts text into the source program at compile time.

FORMAT:
COPY u:filename.

RULES:

1. A COPY cannot occur within another COPY.
2. The disk unit (u:) is optional. The current logged-in disk drive is used as the default if the unit is not specified.
3. The COPY statement should be preceded 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. COMMODORE BUSINESS MACHINES.
0005 INSTALLATION. WEST CHESTER, PA
0006 DATE-WRITTEN. AUGUST 7, 1982.
0007 DATE-COMPILED. AUGUST 7, 1982.
4 ENVIRONMENT DIVISION

The ENVIRONMENT DIVISION identifies the computer to use for program compilation and execution. The ENVIRONMENT DIVISION may consist of a CONFIGURATION SECTION, INPUT-OUTPUT SECTION and COPY information.

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. [CURRENCYSIGN 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 CURRENCYSIGN 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.
0018  MEMORY BEGINNING 16384 ENDING 32767.

Here is a Memory Map to help you visualize where the various portions of your program may be placed in memory.

**MEMORY MAP**

0000    -----------------------------------------------------------0000
         |
100    -------------------beginning of runtime package -----------100
         |
       12K runtime package
         |
2E00    ---------------------------------------------------------2E00
         |
Your COBOL object code gets loaded starting here
and continues up...
         |
         |
         |
         and...
         |
         continues loading down from the top of
         memory or the memory ending statement.
         |
         So loading takes place in both directions!

         |
Your called program can go anywhere as long as
it’s +1 byte above the calling program’s
ending memory statement.

         |
         |
         A second or third called program could be here
to the bottom of CP/M which varies on each
machine. Be careful when porting to other
machines that you don’t overwrite CPM!!!

         |
---------Bottom of CP/M  ----------
         |
         CPM
         |
---------end of memory ----------
The **INPUT-OUTPUT SECTION** names each file used and specifies the associated external hardware devices.

**FORMAT:**

**INPUT-OUTPUT SECTION.**

**FILE-CONTROL.**

```plaintext
SELECT filename-1 ASSIGN TO
   {PRINTER}
   {DISK}
{SEQUENTIAL}
   [, ORGANIZATION is {RELATIVE}]
   {SEQUENTIAL}
   [, ACCESS MODE IS {RANDOM}]
[RELATIVE KEY IS dataname-1]
[RECORD DELIMITER IS STANDARD]
[, FILE STATUS IS dataname-2].

**I-O-CONTROL.**

**SAME [RECORD] AREA FOR filename-1, filename-2...**

**RULES:**

1. Each filename-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 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. Dataname-2 must be defined in the WORKING-STORAGE section as a two (2) character alphanumeric data item.

<table>
<thead>
<tr>
<th>Position 1 (STATUS KEY 1)</th>
<th>Position 2 (STATUS KEY 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 = Successful completion</td>
<td>0 = No information available</td>
</tr>
<tr>
<td>1 = AT END</td>
<td>X = SEE ERROR CODES</td>
</tr>
<tr>
<td>2 = INVALID KEY</td>
<td></td>
</tr>
<tr>
<td>3 = PERMANENT ERROR</td>
<td></td>
</tr>
<tr>
<td>9 = SEE STATUS KEY 2</td>
<td></td>
</tr>
</tbody>
</table>

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. This number 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). It's 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. Therefore, 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 your Commodore
64 CP/M Operating System User’s Guide for a description of the PIP (T) option.

18. If the DISK is SELECTed, then the information will be associated with the DISK. However, if the PRINTER is SELECTed, you have the option of sending information to the PRINTER or DISK. The choice can be made at compile time or at run time depending on which LABEL RECORDS clause is chosen in the (FD) File Description entry, described later.

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
0028   STATUS IS STA-1
0029   SELECT LISTING ASSIGN TO PRINTER.
0030   SELECT NEW-PAYROLL-MASTER-FILE
0031   ASSIGN TO DISK
0032   ACCESS MODE IS RANDOM
0033   RELATIVE KEY IS KEY3
0034   STATUS IS STA-2

NOTE: Also see Appendix I Sample Programs at the end of this manual.

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 preceded 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:

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

The DATA DIVISION specifies 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 only be used for SELECTed PRINTER files and sends output directly to the printer which cannot be redirected at run time. Also, if this clause is used, then the clause VALUE OF FILE-ID cannot be used.

However, if the clause LABEL RECORDS ARE STANDARD is used in conjunction with a SELECTed PRINTER file, then the clause VALUE OF FILE-ID must be used. This combination allows the user the choice of redirecting the printer output to the disk at compile time or run time.
At compile time, the user can specify the printer by using a literal which contains the key word PRINTER. Any other name will be treated as a disk file name and the information will be sent to it. If the file does not exist, it will be created.

If the user wishes to reassign the printer at run time, then a data-name is used in place of the literal. The keyword PRINTER is used as the value of the data-name if the information is to be sent to the printer. Any other name will send the information to the disk. If the disk file does not exist, it will be created.

EXAMPLE:

0041 DATA DIVISION.
0042 FILE SECTION.
0043 FD NEW-PAYROLL-MASTER-FILE
0044   LABEL RECORDS ARE STANDARD
0045   VALUE OF FILE-ID IS "A:MASTER.ACT"
0046   DATA RECORDS ARE HOURLY, SALARY.
0047* 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
0068* see cp/m STAT command for printer assignment
0069* using the LST: to serial or parallel port
0070   VALUE OF FILE-ID IS "PRINTER"
0071   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
0106* printing. see cpm TYPE command using control-p.
0107   VALUE OF FILE-ID IS "B:LIST.TXT"
0108   DATA RECORD IS PRT-LINE.
0109 FD LIST2
0110* note the next line sends data directly to printer
0111   LABEL RECORD IS OMITTED
0112   DATA RECORD IS PRT-LINE2.

RECORD DESCRIPTION — A description of each record is stated in the DATA DIVISION. Here, the particular characteristics of the data fields for each record are specified.
FORMAT:
\[
\begin{align*}
\text{data-name-1} & \quad \text{FILLER} \quad \text{REDEFINES data-name-2} \\
\text{level-number} & \quad [\text{OCCURS integer-1 TIMES}] \\
& \quad \text{PIC} \\
& \quad \{\text{SYNCHRONIZED} \quad \text{JUSTIFIED} \quad \text{RIGHT} \} \\
& \quad \{	ext{COMP} \quad \text{COMP-3} \quad \text{DISPLAY} \quad \text{COMPUTATIONAL-3} \} \\
& \quad \{	ext{JUST} \quad \text{BLANK WHEN ZERO} \} \\
& \quad \{	ext{USAGE IS} \quad \text{COMPUTATIONAL} \} \\
\end{align*}
\]

WORKING-STORAGE SECTION.

same as above and
\[
\begin{align*}
\{\text{ALL} \quad \text{litera}\} \\
\{\text{QUOTE} \quad \text{HIGH-VALUE} \} \\
\{\text{ZERO} \quad \text{LOW-VALUE} \} \\
\{\text{VALUE IS} \quad \text{SPACE} \} \\
\end{align*}
\]...

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 form 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 strings 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.


E. Numeric edited character strings contain the symbols “B, /, V, Z, O, 9”.

The following characters can also be contained “* . + _ $ CR DB”. (Note: CR and DB may cause a shift to the left in the placement of the decimal point.)

A description of each individual character 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 ($) represents a character position into which the ($) 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. CR and DB may cause a shift in the placement of the decimal point.

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 elementary 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.
0062 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 “SECOND”.
0090  02 T3 PIC X(5) VALUE “THIRD”.
0091 01   B-TABLE REDEFINES A-TABLE.
0092  02 ORDER OCCURS 3 TIMES PICTURE X(5).
*by using the ACCEPT verb the next file name can be
changed at object time.

THE-FILE PICTURE X(14) VALUE "A:FILENAME.WRK".

KEY3 PIC 9(7) COMP-3 VALUE 1.

KEY1 PIC X.

KEY2 PIC X.

maximum record or item size is 4095

BIG-ITEM PIC X(4095).

Also, within the DATA DIVISION is the COPY statement. 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 preceded 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:

DATA DIVISION.
COPY A:FILE6.
COPY A:FILE7.
COPY A:FILE8.
6 PROCEDURE DIVISION.

The PROCEDURE DIVISION of a COBOL program specifies the procedures that will be used to solve the 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 of 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: Keywords that can be used as part of the solution in the PROCEDURE DIVISION follow. They are arranged in alphabetical order for easy reference.

ACCEPT lets you input data from the keyboard and assigns that data to the specified data item (identifier).

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 automatically 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:FFFFFFF.FEE >".
0106          ACCEPT THE FILE-NAME.
0107*         clear the screen on a sol-20 next.
0108*         DISPLAY " "OB" ".
0110*         note screen-full can be 80*24 = 1920 size item.
0111         DISPLAY SCREEN-FULL.
0112*         set the cursor using a hexadecimal string.
0113         DISPLAY " "1B,01,3F" ".
0114         ACCEPT INPUT-ITEM.
```
**ADD** lets you add two numeric data items and 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
0151 ROUNDED ON SIZE ERROR GO TO ERROR-ROUTINE.

**ALTER** modifies 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-6.
0204 END-OF-JOB.

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-name 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 five 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 because 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 (assembled with proper origin). 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:

<table>
<thead>
<tr>
<th>#BYTES</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Number of code and data bytes in segment</td>
</tr>
<tr>
<td>2</td>
<td>Load address of code and data belonging to the segment.</td>
</tr>
</tbody>
</table>

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.

The CANCEL statement releases the memory areas occupied by the referenced 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.

Note: See the Sample Programs in Appendix I.

**CLOSE** terminates 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.

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 not present, the default drive will be used.
3. The COPY statement should be preceded 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** lets you display data on the video monitor.

**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 video monitor.
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 and line feed are executed before data transfer begins unless WITH NO ADVANCING is specified.
4. Data is 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:
   - "OX" 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 you are 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 to 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 " "OB" ".
0390* the next line clears the screen on Hazeltine-1520
0392* DISPLAY " "7E,1C" ".
0393* each CRT is different but if you know the commands you
0394* can also set the cursor and display in reverse.
0395* the next line sets the I-O driver for BIOS any
0396* incoming character will be passed to user.
0397* DISPLAY UNIT "OX".
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** lets you 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 results 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 specifies 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.

2. This is a Compiler Directing statement that tells the Compiler it is the last statement in the COBOL source program file to be processed.

EXAMPLE:

*   all program statements must be above
9999   END PROGRAM TEST1.

EXIT furnishes 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** lets you leave the normal sequence of procedures and continue at another area of the program.

**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 integers 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.
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:

IF {condition}  NEXT SENTENCE  ELSE NEXT SENTENCE

{condition}:

{ = \(<\) \(\rangle\) \(\text{EQUAL TO}\) \(\text{LESS THAN}\) \(\text{GREATER THAN}\) \(\text{IDENTIFIER-3 IS [NOT]}\) \(\text{NUMERIC}\) \(\text{IDENTIFIER-3 IS [NOT]}\) \(\text{ALPHABETIC}\)

FORMAT-2:

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 algebraic 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.
The **INSPECT** statement provides the ability to tally, replace, or tally and replace occurrences 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 alphanumericedited, numericedited 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 any 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, until either a match is found or there is no successive literal-1, literal-3. When there is no 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 5b above, except 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 determines the occurrences of literal-1 to be tallied and/or occurrences of literal-3 to be replaced and is affected by the BEFORE and AFTER phrase as follows:
Using the BEFORE Phrase

a. If the BEFORE phrase is specified, then the associated literal-1, literal-3 or implied operand of the CHARACTERS phrase participate only in comparison cycles involving 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.

The position of this first occurrence of literal-2 is determined before the first cycle of the comparison operation is begun.

b. If there is no occurrence of literal-2, literal-5 within the contents of data item referenced by identifier-1, then 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, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase does not match the contents of the data item referenced by identifier-1, then they are not eligible to participate in the comparison operation.

Using the AFTER Phrase

a. If the AFTER phrase is specified, then the associated literal-1, literal-3, or implied operand of the CHARACTERS phrase participates only in comparison cycles involving contents of data items referenced by identifier-1 from its character position immediately to the right of the rightmost character position of the first occurrence of literal-2, literal-5, 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 is begun.

b. If there is no occurrence of literal-1, literal-5 within the contents of the data item referenced by identifier-1, then its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase is not eligible to participate in the comparison operation.

c. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase does not match the contents of the data item referenced by identifier-1, then they are not eligible to participate in the comparison operation.
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.

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, providing that the leftmost occurrence is at the point where comparison began in the first comparison cycle that 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 through two successive INSPECT statements specifying the same identifier but 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.

Here 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 = HGNDEX.

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

Where word = ADJECTIVE, count = 6, word = BJECTIVE.
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: BBBBBBABCD
**MOVE** transfers data from one data area to another.

**FORMAT:**

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

**RULES:**

1. Identifier-1 and literal-1 represent the sending area and identifier-2 indentifier-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. 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 alphanumerically edited or alphanumerically 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 replaced 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 alphanumerically 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 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 alphanumerically to alphanumerically 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 lets you 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 an elementary numeric item, except identifier-3 which may be an 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
0402                 P-ERR.
0402   MULTIPLY WAGE-RATE BY OVERTIME-HOURS.
**OPEN** lets you 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** lets you 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 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 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** makes 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., whatever 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 SPACE 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** replaces 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 preceded 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** causes 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** lets you 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 an elementary numeric item, except identifier-3 which may refer to an 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** releases a record to an output file and allows for vertical positioning if the output device is a printer.

**FORMAT:**

```plaintext
{PAGE
{LINE

WRITE record-name [BEFORE ADVANCING {integer 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 (OCH) 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.
```
7 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 > ERROR <CR>.

Using the CP/M feature CTRL-P, the error messages can also be sent to the printer as they are displayed. Also, CTRL-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.

Note: The Level codes are F for Fatal (no object code generated) and W for Warning Possible Error. Also, (not shown below) each line is preceded by the source program’s actual line number.

<table>
<thead>
<tr>
<th>SEQ. NO.</th>
<th>COL.</th>
<th>ERROR NO.</th>
<th>LEVEL</th>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999</td>
<td>70</td>
<td>001</td>
<td>F</td>
<td>SYNTAX ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>002</td>
<td>F</td>
<td>NOT A COBOL WORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>003</td>
<td>F</td>
<td>SYNTAX ERROR OR PERIOD MISSING FROM PRIOR LINE</td>
</tr>
<tr>
<td></td>
<td>dataname</td>
<td>004</td>
<td>F</td>
<td>FILE NOT SELECTED IN THE I-O SECTION</td>
</tr>
<tr>
<td></td>
<td>dataname</td>
<td>005</td>
<td>F</td>
<td>OCCURS LIMITED TO ONE LEVEL</td>
</tr>
<tr>
<td></td>
<td>dataname</td>
<td>006</td>
<td>F</td>
<td>SUBSCRIPTED ITEMS CANNOT BE REDEFINED</td>
</tr>
<tr>
<td></td>
<td>dataname</td>
<td>007</td>
<td>F</td>
<td>PICTURE ITEMS MUST BE ELEMENTARY</td>
</tr>
<tr>
<td></td>
<td>dataname</td>
<td>008</td>
<td>F</td>
<td>EDITED PICTURE CONTAINS ILLEGAL COMBINATIONS</td>
</tr>
<tr>
<td></td>
<td>dataname</td>
<td>009</td>
<td>F</td>
<td>MAX RECORD LENGTH OF 4095 EXCEEDED</td>
</tr>
<tr>
<td></td>
<td>dataname</td>
<td>010</td>
<td>F</td>
<td>ELEMENTARY ITEM DOES NOT HAVE PICTURE CLAUSE</td>
</tr>
<tr>
<td></td>
<td>dataname</td>
<td>011</td>
<td>F</td>
<td>ILLEGAL REDEFINES DUE TO INCORRECT REFERENCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>012</td>
<td>F</td>
<td>SUBSCRIPT ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>013</td>
<td>F</td>
<td>ILLEGAL COMBINATION OF CHARACTERS IN PICTURE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>014</td>
<td>F</td>
<td>DUPLICATION OF PREVIOUS NAME IS ILLEGAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>015</td>
<td>F</td>
<td>ENVIRONMENT DIVISION MISSING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>016</td>
<td>F</td>
<td>FD MUST CONTAIN A LABEL RECORD CLAUSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>017</td>
<td>F</td>
<td>VALUE OF FILE-ID MISSING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>018</td>
<td>F</td>
<td>SUBSCRIPT LITERAL CONTAINS ILLEGAL VALUE</td>
</tr>
</tbody>
</table>
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
024 dataname F MUST HAVE RELATIVE KEY
025 dataname F MUST BE IN WORKING-STORAGE
026 F KEY NOT ELEMENTARY
027 dataname F RELATIVE KEY MUST BE PIC 9(7)
028 dataname F PARAGRAPH NAME IS NOT DEFINED
029 dataname F PARAGRAPH NAME IS NOT ALTERABLE
030 dataname F TOO MANY FILES SELECTED
031 dataname F NEED MORE MEMORY OR REDUCE SIZE OF LABELS
032 dataname F CORRECT ALL ERRORS AND RECOMPILE
033 dataname F MISSING DIVISION STATEMENT
034 dataname F TOO MANY PARAGRAPH NAMES
035 dataname F TOO MANY FORWARD REFERENCES
036 dataname F 01-10 AND 77 LEVELS ONLY
037 dataname F IS NOT DEFINED
038 dataname F AREA B MUST START WITH " ON CONTINUED LITERAL
039 dataname F ILLEGAL HEXADECIMAL CHARACTER
040 dataname F ILLEGAL FILE-ID "U:FILE"
041 dataname F ASCII (DISPLAY) DATA TYPE REQUIRED
042 dataname F RANDOM FILES MUST USE INVALID KEY CLAUSE
043 dataname F RESERVED WORD NOT YET IMPLEMENTED
044 dataname F VALUE/PICTURE SIGN ERROR
045 dataname F COPY CANNOT ALSO COPY
046 dataname F COPY FILE NAME TOO LONG
047 dataname F COULD NOT FIND REDEFINED ITEM NAME
048 dataname F LITERAL OVER 120 CHARACTERS LONG
049 dataname F LITERAL TRUNCATED RIGHT END
050 dataname W MORE THAN 30 CHARACTERS IN A WORD
051 dataname W LITERAL LONGER THAN PICTURE
052 dataname W REDEFINED AREA ADJUSTED
053 dataname W EDITED PICTURE MODIFIED
054 dataname W TWO RECORDS IN A FILE HAVE DIFFERENT SIZES
055 dataname W COLUMN 5 OR 7 TREATED AS COMMENTS
056 dataname W LINE NUMBER OUT OF SEQUENCE
057 dataname W RANDOM FILE CANNOT BE DELIMITED
058 dataname W PERIOD IS MISSING AFTER PREVIOUS WORD
059 dataname W DECIMAL POINT SIZES DIFFERENT
060 dataname W PRINTER CANNOT BE DELIMITED
061 dataname W VALUE EXCEEDS 5 DIGITS FOR COMP
062 dataname W ILLEGAL VALUE FOR COMP
063 dataname W ILLEGAL CURRENCY SIGN
064 dataname W COPY FILE-NAME MISSING
065 dataname W ALL LITERAL LIMITED TO 1 BYTE
066 dataname W ZERO MISSING IN BLANK WHEN ZERO
067 dataname W BLANK WHEN ZERO NOT ALLOWED AT GROUP LEVEL
068 dataname W

64
RUN TIME AND COMPILEx TIME ERROR MESSAGES

The RUN time 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
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.
APPENDIX I  SAMPLE PROGRAMS  
Listing No. 1 — Sequentially Read a Fixed Length File

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 COMMODORE-64.
0009 OBJECT-COMPUTER.
0010 COMMODORE-64.
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.
DISPLAY I-RECORD
DISPLAY STATUS-KEY.
GO TO BEGIN2.
EOJ.
DISPLAY STATUS-KEY
CLOSE FILE1.
DISPLAY STATUS-KEY.
STOP RUN.
END PROGRAM T6RF.

Listing No. 2 — Read and Rewrite Fixed Length Records

IDENTIFICATION DIVISION.
PROGRAM-ID.
T6IOF.
THIS PROGRAM READS THEN REWRITES FIXED
LENGTH RECORDS.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
COMMODORE-64.
OBJECT-COMPUTER.
COMMODORE-64.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT FILE1 ASSIGN TO DISK
ORGANIZATION IS SEQUENTIAL
ACCESS MODE IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD FILE1
LABEL RECORDS ARE STANDARD
VALUE OF FILE-ID IS IN-OUT-FILE
DATA RECORDS ARE I-O-RECORD.
01 I-O-RECORD.
02 SEQ PIC 9999.
02 REC1 PIC X(160).
WORKING-STORAGE SECTION.
01 IN-OUT-FILE PIC X(14)
VALUE "A:TESTF.WRK".
01 X1 PIC 9999
VALUE 1001.
PROCEDURE DIVISION.
BEGIN.
DISPLAY "ENTER FILE NAME".
DISPLAY IN-OUT-FILE WITH NO ADVANCING.
ACCEPT IN-OUT-FILE.
OPEN I-O FILE1.
MOVE SPACE TO I-O-RECORD.
BEGIN2.
READ FILE1 AT END
GO TO EOJ.
DISPLAY SEQ.
DISPLAY "IN" WITH NO ADVANCING.
MOVE X1 TO SEQ.
ADD 1 TO X1.
DISPLAY SEQ.
REWRITE I-O-RECORD.
DISPLAY "OUT" WITH NO ADVANCING.
GO TO BEGIN2.
EOJ.
CLOSE FILE1.
STOP RUN.
END PROGRAM T6IOF.
Listing No. 3 — Create a File of Variable Length

0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003 T6WD.
0004* THIS PROGRAM CREATES A FILE OF VARIABLE
0005* LENGTH (DELIMITED) RECORDS. MOST TEXT EDITORS
0006* CREATE THIS TYPE OF FILE. EACH RECORD ENDS
0007* WITH A CARRIAGE RETURN AND LINE FEED.
0008 ENVIRONMENT DIVISION.
0009 CONFIGURATION SECTION.
0010 SOURCE-COMPUTER.
0011 COMMODORE-64.
0012 OBJECT-COMPUTER.
0013 COMMODORE-64.
0014 INPUT-OUTPUT SECTION.
0015 FILE-CONTROL.
0016 SELECT FILE1 ASSIGN TO DISK
0017 ORGANIZATION IS SEQUENTIAL
0018 ACCESS MODE IS SEQUENTIAL.
0019* the next statement tells the compiler each record is to be
0020* delimited (separated) by or ended with a carriage return
0021* and line feed.
0022 RECORD DELIMITER IS STANDARD.
0023 DATA DIVISION.
0024 FILE SECTION.
0025 FD FILE1
0026 LABEL RECORDS ARE STANDARD
0027 VALUE OF FILE-ID IS OUT-FILE
0028 DATA RECORDS ARE O-RECORD.
0029 01 O-RECORD.
0030 02 SEQ PIC 9999.
0031 02 REC1 PIC X(156).
0032 02 SEQ2 PIC 9999.
0033 WORKING- STORAGE SECTION.
0034 01 OUT-FILE PIC X(14)
0035 VALUE IS "A:TESTB.WRK".
0036 01 X1 PIC 9999
0037 VALUE 0001.
PROCEDURE DIVISION.
BEGIN.
DISPLAY "ENTER OUTPUT FILE NAME".
DISPLAY OUT-FILE WITH NO ADVANCING.
ACCEPT OUT-FILE.
MOVE SPACE TO O-RECORD.
OPEN OUTPUT FILE1.
DISPLAY "OPEN".
MOVE PAD TO REC1.
BEGIN2.
MOVE X1 TO SEQ.
MOVE X1 TO SEQ2.
ADD 1 TO X1.
DISPLAY O-RECORD.
WRITE O-RECORD
IF X1 = 011
GO TO EOJ.
GO TO EOJ.
EOJ.
CLOSE FILE1.
STOP RUN.
END PROGRAM T6WD.
Listing No. 4 — Read a Variable Length File

0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003 T6RD.
0004* THIS PROGRAM READS A VARIABLE LENGTH (DELIMITED) FILE.
    * this kind of file is 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 COMMODORE-64.
0009 OBJECT-COMPUTER.
0010 COMMODORE-64.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013 SELECT FILE1 ASSIGN TO DISK ORGANIZATION IS SEQUENTIAL 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 VALUE OF FILE-ID IS IN-FILE DATA RECORDS ARE I-RECORD.
0022 01 I-RECORD.
0023 02 SEQ PIC 9999.
0024 02 REC1 PIC X(160).
0025 WORKING- STORAGE SECTION.
0026 01 IN-FILE PIC X(14) VALUE "A:TESTB.WRK".
0028 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.
BEGIN2.
   the next statement is necessary because the delimited
   read only transfers data into the record area and if short,
   the data from prior reads will be in the record area on the
   right end.

   MOVE SPACE TO I-RECORD.
   READ FILE1
   AT END
   GO TO EOJ.
   DISPLAY I-RECORD.
   GO TO BEGIN2.
   EOJ.
   CLOSE FILE1.
   STOP RUN.
END PROGRAM T6RD.

Listing No. 5 — Read and Rewrite Variable Length Records

IDENTIFICATION DIVISION.
PROGRAM-ID.
   T6IOD.
   THIS PROGRAM READS THEN REWRITES VARIABLE LENGTH RECORDS.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
   COMMODORE-64.
OBJECT-COMPUTER.
   COMMODORE-64.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT FILE1 ASSIGN TO DISK
   ORGANIZATION IS SEQUENTIAL
   ACCESS MODE IS SEQUENTIAL
   RECORD DELIMITER IS STANDARD.
DATA DIVISION.
FILE SECTION.
   FD FILE1
   LABEL RECORDS ARE STANDARD
   VALUE OF FILE-ID IS I-O-FILE-NAME
   DATA RECORDS IS A-RECORD.
0023 01 A-RECORD.
0024 02 SEQ PIC 9999.
0025 02 REC1 PIC X(160).
0026 WORKING-STORAGE SECTION.
0027 01 X1 PIC 9999
0029 01 I-O-FILE 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.
Listing No. 6 — Read a Variable Length File, Output to the Printer

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. COMMODORE-64.
0006 OBJECT-COMPUTER. COMMODORE-64.
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 line 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 SPACE 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.
Listing No. 7 — Write Random Fixed Length Records to a File Previously Created Using a Sequential Fixed Length Write Program

0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003 T8WR.
0004* THIS PROGRAM WRITES RANDOM FIXED LENGTH
0004* RECORDS TO A FILE THAT HAS BEEN CREATED USING
0004* A SEQUENTIAL FIXED LENGTH WRITE PROGRAM TO
0004* ALLOCATE THE REQUIRED FILE SPACE.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008 COMMODORE-64.
0009 OBJECT-COMPUTER.
0010 COMMODORE-64.
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 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.
BEGIN2.
MOVE SPACE TO O-RECORD.
MOVE 0001 TO XX-KEY.
DISPLAY "ENTER RECORD NUMBER 0001".
ACCEPT XX-KEY.
IF XX-KEY IS NOT NUMERIC
    GO TO BEGIN2.
IF XX-KEY = 9999
    GO TO EOJ.
MOVE XX-KEY TO KEY-1.
MOVE XX-KEY TO SEQ.
DISPLAY "ENTER DATA FOR RECORD".
ACCEPT REC1.
WRITE O-RECORD
INVALID KEY
    DISPLAY "INVALID KEY" GO TO BEGIN2.
DISPLAY O-RECORD.
GO TO BEGIN2.
EOJ.
CLOSE FILE1.
DISPLAY "EOJ".
STOP RUN.
END PROGRAM T8WR.

Listing No. 8 — Read Random Fixed Length Records
IDENTIFICATION DIVISION.
PROGRAM-ID.
T8RR.
* THIS PROGRAM WRITES RANDOM FIXED LENGTH RECORDS
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
COMMODORE-64.
OBJECT-COMPUTER.
COMMODORE-64.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT FILE1 ASSIGN TO DISK
ORGANIZATION IS
RELATIVE
ACCESS MODE IS RANDOM
RELATIVE KEY IS KEY-1.
DATA DIVISION.
FILE SECTION.
FD FILE1
LABEL RECORDS ARE STANDARD VALUE OF FILE-ID IS IN-FILE DATA RECORDS ARE I-RECORD.
01 I-RECORD.
02 PART-NUMBER PIC 9999.
02 ITEM-DESCRIPTION PIC X(160).
WORKING-STORAGE SECTION.
01 IN-FILE PIC X(14) VALUE "A:TESTF.WRK".
01 KEY-1 PIC 9(7) COMP-3.
01 XX-KEY PIC 9(4).
PROCEDURE DIVISION.
BEGIN.
DISPLAY "ENTER INPUT FILE NAME".
DISPLAY IN-FILE WITH NO ADVANCING.
ACCEPT IN-FILE.
OPEN INPUT FILE1.
DISPLAY "OPEN".
BEGIN2.
MOVE SPACE TO I-RECORD.
MOVE 0001 TO XX-KEY.
DISPLAY "ENTER RECORD NUMBER 0001" ACCEPT XX-KEY.
IF XX-KEY IS NOT NUMERIC GO TO BEGIN2.
IF XX-KEY = 9999 GO TO EOJ.
MOVE XX-KEY TO KEY-1.
READ FILE1 INVALID KEY DISPLAY "INVALID KEY" GO TO BEGIN2.
don’t display on invalid key as data is unspecified.
DISPLAY I-RECORD.
GO TO BEGIN2.
EOJ.
CLOSE FILE1.
DISPLAY "EOJ".
STOP RUN.
END PROGRAM T8RR.
Listing No. 9 — Read and Rewrite Fixed Length Records in Random Mode

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 COMMODORE-64.
0010 OBJECT-COMPUTER.
0011 COMMODORE-64.
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 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.
BEGIN.
DISPLAY "ENTER I-O FILE NAME"
DISPLAY I-O-FILE WITH NO ADVANCING.
ACCEPT I-O-FILE.
OPEN I-O FILE1.
BEGIN2.
MOVE SPACE TO A-RECORD.
MOVE 1 TO XX-KEY.
DISPLAY "ENTER RECORD NUMBER 0001".
ACCEPT XX-KEY.
IF XX-KEY IS NOT NUMERIC
   GO TO BEGIN2.
IF XX-KEY = 9999
   GO TO EOJ.
MOVE XX-KEY TO KEY-1.
READ FILE1
   INVALID KEY
   DISPLAY "READ INVALID KEY" GO TO BEGIN2.
DISPLAY A-RECORD.
DISPLAY "ENTER NEW DATA".
ACCEPT REC1.
REWRITE A-RECORD
   INVALID KEY
   DISPLAY "REWRITE INVALID KEY".
DISPLAY A-RECORD.
GO TO BEGIN2.
EOJ.
CLOSE FILE1.
DISPLAY "EOJ".
STOP RUN.
END PROGRAM T8IOR.
IDENTIFICATION DIVISION.

PROGRAM-ID.

THIS PROGRAM CALLS PROGRAM T20A WHICH IN
TURN CALLS PROGRAM T20B.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.

COMMODORE-64.

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.

8080-CPU MEMORY SIZE 16383 CHARACTERS.

DATA DIVISION.

WORKING-STOREAGE SECTION.

01 M1.

02 M1-2.

03 M1-3 PIC XXX.

02 M1-4 PIC 99.

02 M1-5 PIC 99V99 COMP VALUE 11.11.

02 M1-6 PIC 999999V99 COMP-3 VALUE 012345.78.

02 M1-7 PIC $99,999.99.

01 M2 PIC S9V9999 VALUE 0.6143.

01 M3 PIC X(10) VALUE "A:T20A".

01 M4 PIC X(120).

01 M5 PIC X(20) JUSTIFIED.

PROCEDURE DIVISION.

BEGIN.

DISPLAY "START T20".

MOVE ALL "A" TO M4.

CALL "T20A" USING M1, M2, M3, M4, M5.

DISPLAY "EOJ-T20".

STOP RUN.

END PROGRAM T20.
IDENTIFICATION DIVISION.
PROGRAM-ID.
T20A.
* THIS PROGRAM IS CALLED BY T20 AND IN
* TURN CALLS PROGRAM T20B.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
COMMODORE-64.
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.
8080-CPU MEMORY BEGINNING 16384 ENDING 20000.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 L3 PIC X(10) VALUE "A:T20A".
LINKAGE SECTION.
01 M1.
02 M1-2.
03 M1-3 PIC XXX.
02 M1-4 PIC 99.
02 M1-5 PIC 99V99 COMP.
02 M1-6 PIC 999999V99 COMP-3.
02 M1-7 PIC $99,999.99
01 M2 PIC S9V9999.
77 M3 PIC X(10).
77 M4 PIC X(120).
77 M5 PIC X(20) JUSTIFIED.
PROCEDURE DIVISION.
* no period after the word division when using
USING M1, M2, M3, M4, M5.
BEGIN.
DISPLAY "THIS IS T20A".
DISPLAY M3.
DISPLAY M4.
CALL "T20B" USING L3.
CANCEL "T20B".
EOJ1.
EXIT PROGRAM.
EOJ.
STOP RUN.
END PROGRAM T20A.
IDENTIFICATION DIVISION.

PROGRAM-ID.

T20B.

THIS PROGRAM IS CALLED BY T20A AND EXITS BACK TO IT. NOTE HOW THE MEMORY IS ALLOCATED.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.

COMMODORE-64.

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.

8080-CPU MEMORY BEGINNING 20001 ENDING 24000.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 L1 PIC X(10) VALUE SPACE.

LINKAGE SECTION.

01 L3 PIC X(10).

PROCEDURE DIVISION

USING L3.

BEGIN.

DISPLAY “THIS IS T20-B”.

DISPLAY L3.

EOJ1.

EXIT PROGRAM.

EOJ.

STOP RUN.

END PROGRAM T20B.
Listing No. 11 — Chain to Execute the Next Program Using CP/M’s Submit

0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003 TSUBMIT.
0004* THIS PROGRAM CHAINS TO EXECUTE THE NEXT
0005* PROGRAM USING CP/M’s SUBMIT WHEN THE NEXT
0006 PROGRAM IS NOT TYPE (.OBJ)
0007 ENVIRONMENT DIVISION.
0008 CONFIGURATION SECTION.
0009 SOURCE-COMPUTER.
0010 COMMODORE-64.
0011 OBJECT-COMPUTER.
0012 COMMODORE-64.
0013 INPUT-OUTPUT SECTION.
0014 FILE-CONTROL.
0015 SELECT FILE1 ASSIGN TO DISK
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 “A:$$$.SUB”
0022 DATA RECORDS ARE NEXT-PROGRAM.
0023 01 NEXT-PROGRAM PIC X(16).
0024 WORKING-STORAGE SECTION.
0025 01 W-NEXT-PROGRAM.
0026 02 NAME-SIZE PIC X VALUE “07”.
0027 02 NAME PIC X (7) VALUE “ED TEXT”.
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.
Listing No. 12 — Call an Assembly Language Program used to Transfer Files from CP/M to PTDOS.

0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID. TRANSFER.
   * This program calls an assembly language program call
   * "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. COMMODORE-64.
0006 OBJECT-COMPUTER. COMMODORE-64.
   * 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 (½)?".
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.
THIS PROGRAM IS "TRANS"

IT IS AN ASSEMBLY LANGUAGE PROGRAM THAT IS CALLED BY THE PRIOR COBOL PROGRAM NAMED TRANSFER. IT TRANSFERS CP/M FILES TO PT DOS A UNIX LIKE OPERATING SYSTEM.

IT IS AN EXAMPLE OF AN ASSEMBLY LANGUAGE CALLED PROGRAM

after this program is assembled, the .HEX file must be converted to an .OBJ file. use the program called CONVHEX to do the conversion.

RELOC EQU 0 ;4200H FOR TRS-80
SET UP AS FOLLOWS
BO LOAD PTDOS
*S GO TO SOLOS
BO LOAD CP/M FROM LIFEBOAT 32K
COPY PTDEFS ;THIS FILE CONTAINS THE PTDOS DEFINITIONS
ORG 16384 + RELOC
XEQ START ;necessary for ptdos assembler
START EQU $ ;ENTRY FROM COBOL PROGRAM
SHLD SAV1 ;OUT-FILE-NAME
LXI H,O
DAD SP
SHLD SAVSP
LXI SP,STACK ;SET UP THE STACK
CALL GETP
LHLD SAV3 ;TRANSFER-FUNCTION
MOV A,M ;GET CODE
CPI '1' ;OPEN?
JZ OPEN
CPI '2' ;CLOSE?
JZ CLOSE
CPI '3' ;WRITE?
JZ WRITE
ERROR TRANSFER FUNCTION NOT 1, 2, 3
ERRT LXI D,3232H ;22
EXIT EQU $
LHLD SAV4 ;TRANSFER-ERROR
MOV M,D
DCX H
MOV M,E
LHLD SAVSP
SPHL
RET
GETP EQU $
XCHG
SHLD SAV2 ;TRANSFER-TYPE
PUSH B
POP H ;POINTS TO TABLE TO ADDRESS LEFT END
MOV E,M
INX H
MOVE D,M
XCHG
SHLD SAV3 ;TRANSFER-FUNCTION
XCHG
INX H
MOV E,M
INX H
MOV D,M
XCHG
SHLD SAV4 ;TRANSFER-ERROR
XCHG
INX H
MOV E,M
INX H
MOV D,M
LXI H,255
MOV A,E
SUBL
MOV A,D
SBBH
SHLD SAV5 ;LEFT END OF RECORD TO BE OUTPUT
RET
LXI ONAME+9
MVI C,1D
MVIA.40H;OPENCREATEIFNECESSARY
LXI OBUF
LXI ONAME
;LEFTENDOFRECORDTOBEOUTPUT
OPEN EQU$
LHLD SAV1 ;OUT-FILE-NAME RIGHT END
LXI D,ONAME+9
MVI C,10
OP1 EQU$
MOV A,M
STAX D
DCX H
DCX D
DCR C
JNZ OP1
;the next 9 lines is a ptdos open function
MVI A,40H ;OPEN CREATE IF NECESSARY
LXI D,OBUFF
LXI H,ONAME
0088       CALL PSCAN
0089       JC ERROR
0090       JZ ERROR
0091       MOV A,E ;FILE NUMBER
0092       CPI 255 ; -1 for cp
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
0102       LDA OFILENUMBER
0103       CALL SYS
0104       DB EOFOP ; END FILE
0105       JMP ERROR
0106     LDA OFILENUMBER
0107       CALL SYS
0108       DB CLOOP
0109       JMP $ ; NO ERRORS RETURNED ON CLOSE
0110     LXI D,3030H ; good close message for the cobol program
0111       JMP EXIT
0112       WRITE EQU $
0113       LHLD SAV2 ; TRANSFER-TYPE
0114       MOV A,M
0115       CPI '2' ; DROP THE LF'S
0116       JZ WT2
0117       CPI '1'
0118       JNZ ERRT ; ERROR TRANSFER-TYPE CODE
0119       LHLD SAV5 ; LEFT-END
0120       XCHG
0121     LXI B,256
0122     WT1 EQU$ ; ptdos writer function follows
0123     LDA OFILENUMBER
0124       CALL SYS
0125       DB WBLOP ; WRITE BLOCK
0126       JMP EXIT
0127     LXI D,3030H ; GOOD WRITE for cobol program
0128     JMP ERROR
0129     WT2 EQU $ ; DROP THE LF'S
0130     LHLD SAV5
0131     LXI D,BUFF2
0132     LXI B,256
0133     WT2A EQU $
0140  MOV A,M
0141  CPI OAH  ;LF
0142  JZ WT2B
0143  CPI 1AH  ;CP/M's EOF FOR ASCII FILES
0144  JZ WT2C
0145  STAX D
0146  INX D
0147  WT2B EQU $
0148  INX H
0149  DCX B
0150  MOV A,C
0151  ORA B
0152  JNZ WT2A
0153  WT2C EQU $
0154  LXI H, BUFF2
0155  MOV A,E
0156  SUB L
0157  MOV E,A
0158  MOV A,D
0159  SUBB H
0160  MOV D,A
0161  PUSH D
0162  POP B  ;SIZE OF THIS WRITE FOR PTDOS
0163  XCHG
0164  JMP WT1
0165  SAV1 DW 0  ;OUT-FILE-NAME
0166  SAV2 DW 0  ;TRANSFER-TYPE
0167  SAV3 DW 0  ;TRANSFER-FUNCTION
0168  SAV4 DW 0  ;TRANSFER-ERROR
0169  SAV5 DW 0  ;OUTPUT RECORD
0170  SAVSP DW 0  ;STACK POINTER
0171  ; all that follows is for ptdos
0172  DB ";"+80H
0173  DW 04COH
0174  DB 0
0175  OBUFF DS 20
0176  DS 20
0177  STACK DW 0
0178  ONAME DS 10
0179  DB 0
0180  OFILENUMBER DB 0
0181  BUFF2 DS 256
0182  LAST DB 0
0183  END START  ;necessary for cpm assembler
APPENDIX II GLOSSARY

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

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 arithmetic expression can be an identifier, a numeric elementary item, or a numeric literal. Such identifiers and literals are separated by arithmetic operators or two arithmetic expressions are separated by an arithmetic operator, or an arithmetic expression is enclosed in parentheses.

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

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>–</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>**</td>
<td>exponeniation</td>
</tr>
</tbody>
</table>

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**
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:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1,...9</td>
<td>digit</td>
</tr>
<tr>
<td>A,B,...Z</td>
<td>letter</td>
</tr>
<tr>
<td>+</td>
<td>plus sign</td>
</tr>
<tr>
<td>-</td>
<td>minus sign (hypen)</td>
</tr>
<tr>
<td>*</td>
<td>asterisk</td>
</tr>
<tr>
<td>/</td>
<td>slash</td>
</tr>
<tr>
<td>=</td>
<td>equal sign</td>
</tr>
<tr>
<td>$</td>
<td>currency sign</td>
</tr>
<tr>
<td>;</td>
<td>comma</td>
</tr>
<tr>
<td>;</td>
<td>semicolon</td>
</tr>
<tr>
<td>.</td>
<td>period (decimal point)</td>
</tr>
<tr>
<td>&quot;</td>
<td>quotation mark</td>
</tr>
<tr>
<td>(</td>
<td>left parenthesis</td>
</tr>
<tr>
<td>)</td>
<td>right parenthesis</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than symbol</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than symbol</td>
</tr>
</tbody>
</table>

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 slash (/) 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 general 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 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 zeros, and 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 of values upon 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:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>space</td>
</tr>
<tr>
<td>0</td>
<td>zero</td>
</tr>
<tr>
<td>+</td>
<td>plus</td>
</tr>
<tr>
<td>-</td>
<td>minus</td>
</tr>
<tr>
<td>CR</td>
<td>credit</td>
</tr>
<tr>
<td>DB</td>
<td>debit</td>
</tr>
<tr>
<td>Z</td>
<td>zero suppress</td>
</tr>
<tr>
<td>*</td>
<td>check protect</td>
</tr>
<tr>
<td>$</td>
<td>currency sign</td>
</tr>
<tr>
<td>,</td>
<td>comma</td>
</tr>
<tr>
<td>.</td>
<td>period (decimal point)</td>
</tr>
<tr>
<td>/</td>
<td>slash</td>
</tr>
</tbody>
</table>

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:

- ZERO, ZEROS, or ZEROES represent one or more occurrences of the character zero (0).
- SPACE or SPACES represent one or more occurrences of the character space (blank).
- QUOTE or QUOTES represent one or more occurrences of the character quote (").
- HIGH-VALUE or HIGH-VALUES represent one or more occurrences of the character FF Hexadecimal.
- LOW-VALUE or LOW-VALUES represent one or more occurrences of the character 00 Hexadecimal.
- ALL "literals" represent 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-VALUE TO OUT-RECORD.
```

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 register, 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 which also provides information required for transmission and handling of data during execution of the object program.

Integer
A numeric literal or a 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 or 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 an 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.

LineNumber
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 from 1 to 120 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. A second set of quotation marks (‘”’) can be used to bound hexadecimal values. Each hexadecimal 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 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 containing from 1 to 18 numeric characters that can also have either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character, nor can it be to the immediate left of a minus sign. The algebraic sign, if present, must be the leftmost character. A numeric literal cannot be bounded by quotation marks.

EXAMPLE:

* numeric literals
  0060  MATH.  ADD 1 TO TOTAL-ITEMS.
  0061  ADD 3.75 TO AMT-SAVED.

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.

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 occurrence of this entry). The (39) is the subscript.

**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 Environment 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:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>comma</td>
</tr>
<tr>
<td>;</td>
<td>semicolon</td>
</tr>
<tr>
<td>.</td>
<td>period (decimal point)</td>
</tr>
<tr>
<td>&quot;</td>
<td>quotation mark</td>
</tr>
<tr>
<td>(</td>
<td>left parenthesis</td>
</tr>
<tr>
<td>)</td>
<td>right parenthesis</td>
</tr>
<tr>
<td>=</td>
<td>equal sign</td>
</tr>
</tbody>
</table>

**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 alternative 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.

**REDEFINES Clause**
This clause allows you to give a name to a field which crosses from one elementary item into the next. In the FILE SECTION, REDEFINES may not be used on the 01 level. To redefine an entire record, you only need to name the new record in the DATA RECORDS clause to implicitly redefine it. For correct format, nothing should come between data-name-1 and the REDEFINES clause.

**Reference Format**
A format that provides a standard method for describing COBOL source programs.
Relation Character
A character that belongs to the following set:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>greater than symbol</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than symbol</td>
</tr>
<tr>
<td>=</td>
<td>equal to</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Relational operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS [NOT] GREATER THAN</td>
<td>Greater than or not greater</td>
</tr>
<tr>
<td>IS [NOT] &gt;</td>
<td></td>
</tr>
<tr>
<td>IS [NOT] LESS THAN</td>
<td>Less than or not less than</td>
</tr>
<tr>
<td>IS [NOT] &lt;</td>
<td></td>
</tr>
<tr>
<td>IS [NOT] EQUAL TO</td>
<td>Equal to or not equal to</td>
</tr>
<tr>
<td>IS [NOT]=</td>
<td></td>
</tr>
</tbody>
</table>

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, the section header is composed of a section-name, the reserved word SECTION, 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 as either a 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:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>plus sign</td>
</tr>
<tr>
<td>-</td>
<td>minus sign</td>
</tr>
<tr>
<td>*</td>
<td>asterisk</td>
</tr>
<tr>
<td>/</td>
<td>slash</td>
</tr>
<tr>
<td>=</td>
<td>equal sign</td>
</tr>
<tr>
<td>$</td>
<td>currency sign</td>
</tr>
<tr>
<td>;</td>
<td>comma</td>
</tr>
<tr>
<td>;</td>
<td>semicolon</td>
</tr>
<tr>
<td>.</td>
<td>period (decimal point)</td>
</tr>
<tr>
<td>&quot;</td>
<td>quotation mark</td>
</tr>
<tr>
<td>(</td>
<td>left parenthesis</td>
</tr>
<tr>
<td>)</td>
<td>right parenthesis</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than symbol</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than symbol</td>
</tr>
</tbody>
</table>

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. 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.

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:

```cobol
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 TIME PIC IS X(9).
0017 PROCEDURE DIVISION.
0018 DATA-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.

*************** important ***********************
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.
```

Subscripted Data-Name

An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses. Here are the rules for parentheses:

- An opening parenthesis must be preceded by a space and a closing parenthesis must be followed by a space.
- No spaces are allowed within a set of 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
True or False represents the result of an evaluated condition.

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 Words
A COBOL word that must be supplied by the user to satisfy the format of a clause or statement. 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.

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.
REFERENCES


COBOL for Students, Edward Arnold, 1975.


CP/M Bible, Howard E. Sams & Co., Product #22015.

CP/M PRIMER, Howard E. Sams & Co., Product #21791.


NEVADA COBOL Application Packages Book1, Ellis Computing, 1980.


INDEX

A
ABBREVIATED combined relation condition 91
ACCEPT 2, 32
ACCESS MODE 91
ACTUAL Decimal Point 91
ADD verb 34
ADVANCING 2
AFTER 2, 45, 47, 49
ALL "literal" 2, 27, 39, 45, 50
ALPHABETIC CHARACTER 20, 28, 43, 91
ALPHANUMERIC CHARACTER 91
ALPHANUMERIC edited character 28, 47, 53
ALTERNATE Record Key 91
ALTER 2, 34, 42
AND 2, 43-44, 55
ARITHMETIC Expression 91
ARITHMETIC Operator 91
ARITHMETIC VERBS 9
ASCENDING Key 91
ASSEMBLY LANGUAGE 85-90
ASSIGN TO 2, 22
ASSUMED Decimal Point 28, 92
ASTERISK (*) 14, 29
AT END Condition 23, 59, 92
AUTHOR 2

B
BEFORE 2, 45, 47, 49, 62
BINARY data types (COMP) 30
BLANK WHEN ZERO 27
BLOCK CONTAINS 25, 92
BUILDING A PROGRAM 14, 69

C
CALL 2, 20, 35, 81-83, 92
CANCEL 2, 35, 37
CC.COM 4, 13
CHARACTER-STRING 25, 45-46, 48-49, 92
CLASS condition 92
CLAUSE 5, 92
CLOSE 2, 38
COBOL Character Set 28, 93
COBOL Coding format 5, 8-9, 14-15
COLLATING Sequence 93
COLUMN 9, 14-15, 93
COMBINED Condition 93

114
COMMENT Line 93-94
COMPILE time 94
COMPILE TIME ERROR MESSAGES 63-65
COMPILER Directing Statement 8, 94
COMPILING A PROGRAM 15
COMPUTATIONAL (COMP) 2-3, 27, 29-30
COMPUTATIONAL-3 (COMP-3) 3, 23, 27, 29, 30, 59
COMPUTER-Name 94
CONDITION-Name 43, 94
CONDITIONAL Expression 94
CONDITIONAL Statement 8, 94
CONDITIONAL Variable 95
CONFIG.CBL 4, 7, 12, 15-17
CONFIGURATION Section 20-21, 95
CONNECTIVE 95
CONTIGUOUS Memory 20, 95
CONVHEX.COM 4, 36
COPY 2, 7, 18, 20, 24, 31, 38
COUNTER 95
CP/M 6, 21, 33, 84-90
CREDIT and debit symbols (CR) (DB) 28, 29
CURRENT Record 95
CURRENT Record Pointer 95
CURRENCY Symbol ($) 20, 29, 95

D
DATA Disk 4, 6
DATA DIVISION 5, 10-11, 25-26, 31, 60, 62
DATA Description Entry 25, 96
DATA RECORDS ARE 25
DATE-COMPILED 2, 10
DATE-WRITTEN 2, 10
DEBUGGING MODE 14, 20, 96
DECIMAL-POINT IS COMMA 21
DECLARATIVES 96
DELIMITED files 30
DEPENDING ON 42
DESCENDING Key 96
DIGIT Position 96
DISK 3, 23-24, 55
DISPLAY 27, 29, 33, 39, 43, 46
DIVIDE 2, 40, 97
DIVISION Header 5, 97
DYNAMIC Access 97
E
ED.COM 6-7, 14
EDITING 97
ELEMENTARY Item 54, 61, 97
ELSE 2, 43
END PROGRAM 32, 41, 97
ENTRY 97
ENVIRONMENT DIVISION 5, 10, 20, 24, 98
EQUAL TO 43
ERROR CODES AND MESSAGES 23, 63-65
ERRORS.COM 4
EXECUTING A PROGRAM 16
EXIT 2, 41, 56
EXIT PROGRAM 37, 41
EXTENDED Mode 98

F
FD, 2, 24-26
FIGURATIVE Constant 39, 46, 54, 98
FILE CONTROL 2, 10, 22, 98
FILE Description (FD) 24, 98
FILE Name 10, 98
FILE Organization 98-99
FILE SECTION 25-26, 60, 62, 99
FILLER 2, 77
FIRST 45, 50
FIXED-LENGTH RECORDS 66-68, 77-80
FORMAT 99
FROM 2, 61

G
GETTING STARTED 6
GIVING 34, 40, 54, 61
GLOSSARY 91-112
GO TO 34, 42, 56
GREATER THAN 43
GROUP Item 30, 99

H
HARDWARE REQUIRED 6
HEXADECIMAL literal 12
HIGH ORDER End 99
HIGH-VALUE 2, 27
M
MASS Storage File 101
MEMORY MAP 21
MEMORY SIZE 21, 36
MINUS SIGN (−) 29
MNEMONIC-Name 101
MOVE VERB 9, 52
MULTIPLY 2, 54

N
NATIVE Character Set 102
NEGATED Simple Condition 102
NEXT Executable Statement 102
NEXT SENTENCE 43-44, 102
NEXT Record 102
NONCONTIGUOUS Items 102
NONNUMERIC Literal 35, 43, 46, 52, 54, 102
NOT 2, 43
NUMERIC CHARACTER 28, 103
NUMERIC edited character 28, 47
NUMERIC Literal 43, 54, 103
NUMERIC MOVE 52-53

O
OBJ FILES 4
OBJECT-COMPUTER 2, 10, 103
OBJECT of Entry 103
OBJECT Program 103
OBJECT Time 103
OCCURS clause 27, 53, 103
ON SIZE ERROR 40-41, 54, 61
OPEN 8, 38, 55, 104
OPERATIONAL Sign 40, 53, 104
OPTIONAL Word 104
OR 2, 43-44, 55
ORGANIZATION IS 23
OUTPUT 2, 55, 74-75, 104
SAME RECORD AREA 23
SAMPLE PROGRAMS 66-90
SCREEN Information 12
SECTION Header 108
SECTION-Name 32, 108
SECURITY 3
SEGMENT-Number 108
SELECT 3, 22, 25
SENTENCE 3, 108
SEPARATOR 108
SEQUENCE CONTROL VERBS 9
SEQUENTIAL Access 14, 22, 55, 59-60, 66, 108
SIGN Condition 109
SIMPLE Condition 109
SOFTWARE REQUIREMENTS 6
SOURCE-COMPUTER 3, 10, 109
SOURCE Program 109
SPACE 3, 27, 52, 54
SPECIAL Character 109
SPECIAL-NAMES 3, 110
STANDARD Data Format 53, 110
STATEMENT 5, 110
STATUS KEY 23
STOP RUN 60
SUBJECT of Entry 110
SUBPROGRAM 110
SUBSCRIPTING 110
SUBTRACT 3, 61
SYMBOLS AND CONVENTIONS 1
SYNCHRONIZED (SYNC) 3, 27
SYSTEM-Name 10, 111

TAB characters 16, 59
TABLE 112
TABLE Element 112
TABLE OF CONTENTS
TALLYING 3, 45, 47, 51
TEXT-Word 112
THROUGH 3, 55
THRU 3, 55
TIMES 3, 27, 55
TRUTH Value 112
U
UNARY Operator 112
UNIT 18, 24-25, 31, 38-39
UNTIL 3, 55
USAGE IS clause 27, 29
USER-DEFINED WORDS 112
USING 3, 32, 35

V
VALUE clause 27
VALUE OF FILE-ID 25
VARIABLE 112
VARIABLE LENGTH RECORDS 22, 59, 69-75
VERBS 8, 112

W
W1.WRK 4
W3.WRK 4, 63
W4.COM 4, 13
W5.CBL 4, 17, 63
WITH NO ADVANCING 39
WORDS 3, 5, 112
WORKING-STORAGE SECTION 10, 23, 27, 112
WRITE 3, 8, 62, 76-77

Z
ZERO 3, 27, 52, 54