REVIEW OF OS-9 CIS COBOL

Peter Dibble

Revision History
April 1985

OVERVIEW

COBOL is a big language, an old language, and an extremely popular language. Some languages were designed to be compiled and run on small computers: COBOL was not. COBOL is vehemently detested by many people involved with computers, but despite all the nasty publicity it gets, COBOL is probably the most used computer language in the world. If you need to hire an experienced programmer for a business application, you will find the hunting best if you snoot for a COBOL programmer. COBOL was one of the first compiled languages developed for computers (around 1960), and it has been being (arguably) improved since then. The fully "improved" version of COBOL is an enormous language whose compiler is fully capable of needing the best part of a megabyte of memory to run properly.

There are standards against which any version of COBOL should be measured. ANSI (American National Standards Institute) has defined a COBOL standard which constitutes the official definition of the language. CIS COBOL was written to conform to the ANSI standard definition of COBOL.

To quote the manual: "CIS COBOL is ANSI COBOL as given in 'American National Standard Programming Language COBOL' (ANSI X3.23 1974)." It includes level 1 of the ANSI definition of COBOL along with a few parts of level 2. This doesn't mean that CIS COBOL is the version of the language you may have used on a mainframe computer, but it does mean that if you don't use the enhancements that CIS COBOL includes, the programs you write using it will run essentially unmodified on any other computer that runs level 2 or higher of ANSI COBOL. Also, since CIS COBOL is compiled to intermediate code, programs written in it can be run on any computer that has the appropriate interpreter. If you read the adds in BYTE, you will see that CIS COBOL is implemented for many computers.

I didn't test CIS COBOL exhaustively for conformance to the standard, but I did write a few programs in it. I am used to IBM's VS-COBOL, and a version of UNIVAC COBOL; both are highly enhanced versions of higher levels of ANSI COBOL than CIS COBOL. It took me a while to learn which of my favorite programming tricks aren't possible under level 1 of ANSI COBOL, but after I learned the limitations I had to live with, I found that I could write programs with no more difficulty than I usually experience when writing in COBOL. I wish I had been able to transfer a program from the IBM to my micro and compile it, but I don't know of any real programs written to be compiled by ANSI level 1 COBOL. Transferring a program in the other direction is no problem.

There is far too much to CIS COBOL for me to say with certainty that it all works, but I understand that the language has actually been successfully tested against a set of standard test programs.

ENHANCEMENTS

Standard COBOL doesn't support the interactive microcomputer environment very well, but CIS COBOL includes enhancements to the ACCEPT and DISPLAY statements that make it relatively easy to display screens of data, and accept data from fields defined on the screen. Information can be accepted from, or displayed at, a particular cursor location. An input field can be defined as numeric only, in which case any inappropriate characters (like "A") won't be accepted. When a field is filled with data, the cursor automatically jumps to the beginning of the next field. There are special keys which jump the cursor forward and backward a field at a time. Special function keys can be defined. They act like a carriage return (terminate entry into a screen), but a program can determine whether a screen was terminated by a carriage return or a function key, and which function key was used.

The location of the cursor when carriage return was pressed is also available. Tne net effect of these enhancements is that it is fairly easy to write CIS COBOL programs that accept and display screens of data.

In addition to the usual COBOL file organizations (including ISAM), CIS COBOL allows an organization they call "line sequential." Line sequential files are variable length record files, in which the records are terminated by carriage returns. This makes it easy to read and write files that Pascal would call "files of text." The most generally important examples of files of this type are files created by text editors, and line by line output to a terminal or printer. The other access modes supported by CIS COBOL are: sequential, relative, and indexed. The names of files can be specified at run time using statements like:

```
SELECT FILE-15 ASSIGN TO FILE-15-NAME.
...
ACCEPT FILE-15-NAME.
OPEN INPUT FILE-15.
```

In addition to the standard ANSI debug features, CIS COBOL has a respectable interactive debugger. The commands available under this debugger are:

```
P - Display the current program counter
G - Set a breakpolnt
X - Single step
D - Display data at specified offset in data division
A - Change memory (ASCII)
S - Set block for display or change
/ - Display block
. - Change bytes in block
T - Trace paragraphs
L - Write CR,LF
M - Define a debug macro
$ - End a macro definition
C - Display a specified character
; - precedes a comment (for describing macros)
```

The interactive debugger can be used on any COBOL program by including +D on the command line that invokes the program, e.g., `RunC +D test.int`. This means that you can use the debugger on a program without having to do anything special when you compile it.

Microware has included eight subroutines in the COBOL run time system which can be called from a COBOL program. MOVE-BLOCK is a procedure that can be used to do a high speed move of a block of data. ABORT terminates the program with an error code. CHAIN makes the standard OS-9 F$Chain system call available. The FUN-KEY subroutine can be used after a ACCEPT statement to find out if a function key was pressed instead of the carriage return key, and which one. DATE returns the date and, optionally, the time. SHELL invokes a shell, passing it a specified string. CHX and CHD change the execution and data directories for the program.

The subroutines in the run time system are called by number. CIS COBOL can also call subroutines which are either COBOL I-code, or object code. The CALL statement looks like:

```
CALL "/D0/SUBLIB/TEST.SUB.1"
USING ...
  ON OVERFLOW ....
```

The called program is loaded into memory if it is not already there, and, depending on whether the module header indicates that it is I-code or object code, interpreted or executed. If there is no room in memory for the new module, the ON OVERFLOW clause in the CALL statement gets control. The CANCEL verb unlinks a subroutine, freeing the memory it is using.

In addition to these methods of calling external subroutines, CIS COBOL supports program segmentation, which can be used to divide the program into sections that will remain on disk until they are needed. Segments use memory efficiently at the cost of extra disk I/O by sharing a common pool of overlay memory.

In addition to supporting ANSI COBOL level 1, including:

```
The Nucleus
Table Handling
Sequential Input and Output
Relative Input and Output
Indexed Input and Output
Segmentation
Library (Copy)
Inter-program communication debug
```

CIS COBOL supports parts of level 2 of ANSI COBOL including:

- Nested IF
- PERFORM UNTIL
- The START statement for Relative and Indexed I/O
- Full level 2 Inter-program communication

LIMITATIONS

I was disappointed with some of the restrictions of the low level of COBOL implemented for CIS COBOL, but not very surprised. I am more upset by some problems with terminal support, and the CONFIG utility that is used to customize the run time package for a particular type of terminal.

The features of advanced levels of COBOL that I missed most were AND and OR in IF statements. It is possible to do without boolean operations in IF statements, but I am not used to having to work around a limitation like that. Another very popular feature which is missing in CIS COBOL is the SORT statement. A surprising number of production COBOL program s include at least one sort, and it would be hard to eliminate a sort from a program without a major redesign.

The run time system which interprets the COBOL intermediate code also includes routines for terminal control. It is customized for a terminal by a utility program called CONFIG. I was not impressed with CONFIG. My favorite terminal uses the ANSI standard terminal control sequences CONFIG was clearly not written with my terminal in mind. I struggled for two evenings tryng to get RunC configured for my TeleVideo with no success. Finally, I gave up and turned to my H-19, which was much more like what CONFIG wanted ... I had COBOL running in ten minutes. There were three fundamental problems with CONFIG's handling of my TeleVideo's control sequences. CONFIG expected most terminal control strings to be no more than three characters long; several of the ANSI strings are longer than that. CONFIG simply can't deal with the ANSI direct cursor positioning sequence; I circumvented that problem by pretending that my terminal didn't have a direct cursor positioning command, and specifying relative positioning. CONFIG can only deal with commands that move the cursor one row or column at a time in relative positioning mode. Since the ANSI strings that cause the cursor to move one row or column are three characters long, this is a slow way to adjust the cursor position. The clear-screen sequence for my terminal is four characters long; so I couldn't use it. RunC tries to fake a clear-screen somehow, but it makes a real mess of it. The clear-screen sequence somehow came out as a string of thousands of <bell> characters. I understand that a more recent version of CONFIG than the one I have allows a four character string for the clear-screen sequence. I think that would have made it possible for me to get my TeleVideo working with COBOL.

CONFIG forms a trap for the unwary user. Once you start into it there is no turning back. If you change your mind about the response you just keyed in, you have to wait until you reach the end of the entire (long) string of questions, and ask to be allowed to change a large subset of your answers. When you

are going through CONFIG to fix a mistake or change an existing terminal description to fit a new terminal, you have to fill in the correct answer to each question. There is no way to select a default, or keep the old value. It is true that CONFIG is not likely to be a heavily used utility, but I found it so hard to use that I would much rather have written a few subroutines to support my terminals.

Once I got the screen support working, I found that I wasn't pleased with the way it worked. I believe that when the cursor leaves a numeric field, the field should be right justified and zero filled. The screen handling package in CIS COBOL seems to agree with me to some extent. If you enter a "." in an integer field it will right justify and zero fill, but if you exit the field with a carriage return (ending the entire screen) or down arrow (moving to the next field), a test for numeric in the program will indicate that the field is not numeric. If the field has editing characters in it the field is inclined to end up left justified and zero filled.

I am used to getting useful, english error messages from COBOL; CIS COBOL gives error messages with numeric codes in them indicating what the error is. Even after I looked up the error, it wasn't clear what the problem was. For instance, when I hadn't declared a variable it told me that there was a type mismatch in the statement using the undeclared variable. When I tried to use AND and OR, it gave me the same error. I ended up treating the error message as "something's wrong around here."

BENCHMARKS

I ran two benchmarks against this COBOL: one for speed at numeric processing (the sieve), the other for speed in handling ISAM files. I adjusted the prime number program from the January 1983 BYTE slightly to fit ANSI level one, and ran it. This version of COBOL would have fallen nearly at the bottom of the chart given in that BYTE, between Microsoft COBOL and RMCOBOL. It took 541 seconds to find the first 1899 primes. I could have made the program run somewhat faster by using indexing instead of subscripting, but that would have spoiled the benchmark. I have to admit that I felt silly writing a Eratosthenes Sieve program in COBOL. Testing COBOL for its ability to find prime numbers is like testing programmers for their ability to read Latin; they may be able to do it but it is hardly relevant. I ran that benchmark because it is the most used benchmark for microcomputer languages, but I also ran another non-standard, but, I think, more relevant, benchmark.

I constructed a benchmark program which gives a good measure of the speed with which the language handles indexed I/O. Indexed I/O is very important to the group of users who might use COBOL. Interpreting the results of a benchmark that involves I/O is a little tricky. Certainly the file structure the language uses is very important, especially with a large indexed file; but the access time for the disk is an important factor, and the time the operating system takes for a context switch is somewhat important.

I built a file 10,000 records long of 55 byte records with five byte keys and then read it randomly reading two records alternately from each end. It took 2615 seconds to build the file and 3233 seconds to read the file (it would, of course, have been possible to read it faster if I had read sequentially). I ran these benchmarks on a GIMIX system with a CM 5000 Winchester (a file that size would not have fit on my 8" floppies). I used OS-9 Level Two on a 2 mhz 6809. The performance would have been much worse if I had used a floppy instead of a Winchester, and somewhat better if I had used GMX - III.

I compiled three COBOL programs on the same machine I ran the benchmarks on. A simple merge program which I haven't included with this review took 45 seconds to compile, the sieve compiled in 35 seconds, and the ISAM test program took 43 seconds.

SUMMARY

It is possible to get past the problems with CONFIG, to learn to live with the primitive error messages, and to feel comfortable with the screen handling conventions. What is left is a substantial implementation of an old, but useful language. I don't think everyone should run out and buy this package, but, for a few people, it could be uniquely useful. If you want to use a group of COBOL programs on microcomputer, it would certainly be easier to convert them from one level of COBOL to another than to translate them into an entirely different language. CIS COBOL would be a good teaching tool for schools unable to afford time on a machine with a full-blown COBOL compiler.

It should be relatively easy to find programmers who can work in COBOL. With CIS COBOL, a microcomputer could be used as a development environment for COBOL programs, though the low level of CIS COBOL would prevent this in most cases. Perhaps the most significant advantage of CIS COBOL over other languages is that programs written in CIS COBOL can be moved in I-Code form to a variety of other machines and operating systems, and run without source code. UCSD Pascal has shown that this is an asset even though it can't generally run under a normal operating system.

CIS COBOL was written by Micro Focus Limited. Microware wrote a run time package for it that allows any program written in CIS COBOL, including CIS COBOL itself to be run under OS-9. By writing a run time package for CIS COBOL, and arranging to license it for OS-9. Microware made a large collection of business software available to OS-9 users. If you are looking for a nice accounting system, payroll, MRP system, or whatever, check with Microware. They have a long list of vendors offering programs which run under the CIS COBOL run time system.

Some small number of people will find Microware's version of CIS COBOL just what they need. If you think you are one of those people, I recommend that you get the manual before you commit to the language. The manuals won't be any help to you if you don't know COBOL, but, if you do, they will leave you with an accurate impression of the language, and either leave you impatient to get the software, or disappointed about some important missing feature (most likely sort).

COBOL TEST PROGRAM

```
** CIS COBOL V4.4                    Test.CBL              PAGE: 0001
**
        IDENTIFICATION DIVISION.
        PROGRAM-ID. FIRST-TEST.PROGRAM.
        AUTHOR. PETER DIBBLE.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER. GIMIX.
        OBJECT-COMPUTER. GIMIX.
        INPUT-OUTPUT SECTION.
             FILE-CONTROL.
             SELECT INPUT-1 ASSIGN ":CI:"
                  ORGANIZATION IS LINE SEQUENTIAL.
             SELECT MERGE-FILE ASSIGN MERGE-NAME.
             SELECT TEMP-FILE ASSIGN "MERGE.TEMP".
        DATA DIVISION.
        FILE SECTION.
        FD  INPUT-1;
             RECORD 40;
             BLOCK 5;
             LABEL RECORDS ARE STANDARD.
        01  INPUT-1-LINE                  PIC X(40).
        FD  MERGE-FILE;
             RECORD 20;
             BLOCK 10;
             LABEL RECORDS ARE STANDARD.
        01  MERGE-LINE                    PIC X(20).
        FD  TEMP-FILE
             RECORD 20;
             BLOCK 10;
             LABEL RECORDS ARE S TANDARD.
        01  TEMP-LINE                     PIC X(20).
        WORKING-STORAGE SECTION.
        01  IN-THIS                       PIC X(40) VALUE SPACES.
        01  LINE-FMT REDEFINES IN-THIS.
```

```
           02  KEEP-THIS.
                04  FILLER               PIC X(19).
                04  CARRIAGE-RETURN       PIC X.
           02  FILLER                    PIC X(20).
       01  MERGE-THIS                    PIC X(20) VALUE SPACES.
       01  FILE-STAT                     PIC X VALUE "0".
       PROCEDURE DIVISION.
       START-UP.
      * PARAMETERS ARE GIVEN IN THE FIRST RECORD OF STD. INPUT
           OPEN INPUT INPUT-1.
           READ INPUT-1 INTO MERGE-NAME.
           OPEN INPUT MERGE-FILE.
           OPEN OUTPUT TEMP-FILE.
           DISPLAY IF MERGING STANDARD INPUT WITH ", MERGE-NAME.

** CIS COBOL V4.4                   Test.CBL              PAGE: 0002
           READ INPUT-1 INTO IN-THIS;
                AT END MOVE HIGH-VALUES TO IN-THIS.
           PERFORM FIX-IN.
           READ MERGE-FILE INTO MERGE-TH IS ;
                AT END MOVE HIGH-VALUES TO MERGE-THIS.
       MAIN-SECTION.
           PERFORM MERGE-LOOP UNTIL FILE-STAT EQUAL TO "1".
           MOVE "0" TO FILE-STAT.
           CLOSE MERGE-FILE.
           OPEN OUTPUT MERGE-FILE.
           CLOSE TEMP-FILE.
           OPEN INPUT TEMP-FILE.
           PERFORM READ-TEMP.
           PERFORM COPY-TEMP-TO-MERGE UNTIL FILE-STAT EQUAL TO "1".
           STOP RUN.
       MERGE-LOOP.
           PERFORM PICK-NEXT.
           WRITE TEMP-LINE.
       END-MERGE-LOOP.
           EXIT.
       PICK-NEXT.
           IF KEEP-THIS < MERGE-THIS
                THEN
                     PERFORM FIX-IN
                     MOVE KEEP-THIS TO TEMP-LINE
                     READ INPUT-1 INTO IN-THIS;
                          AT END PERFORM END-IN
                ELSE
                     MOVE MERGE-THIS TO TEMP-LINE
                     READ MERGE-FILE INTO MERGE-THIS;
                          AT END PERFORM END-MERGE.
       PICK-NEXT-END.
           EXIT.
       END-IN.
           MOVE HIGH-VALUES TO IN-THIS.
           IF MERGE-THIS = HIGH-VALUES
                THEN
                     MOVE "1" TO FILE-STAT.
       END-MERGE.
           MOVE HIGH-VALUES TO MERGE- THIS.
           IF IN-THIS = HIGH-VALUES
                THEN
```

```
                    MOVE "1" TO FILE-STAT.
           FIX-IN.
                MOVE X"OD" TO CARRIAGE-RETURN.
           COPY-TEMP-TO-MERGE.
                WRITE MERGE-LINE.
                PERFORM READ-TEMP.
           END-COPY-TEMP-TO-HERGE.
                EXIT.
           READ-TEMP.
                READ TEMP-FILE; AT END PERFORM END-TEMP.
                MOVE TEMP-LINE TO MERGE-LINE.
           END-READ-TEMP.
                EXIT.
           END-TEMP.
                MOVE "1" TO FILE-STAT.
           END-INPUT.
                MOVE HIGH-VALUES TO IN-THIS.
           END-MERGE-IN.
                MOVE HIGH-VALUES TO MERGE-THIS.
           END-PROGRAM.
                EXIT.
      ** CIS COBOL V4.4 REVISION 0                                    URN rp/
      ** COMPILER COPYRIGHT (C) 1978 , 1981 MICRO FOCUS LTD
      ** ERRORS=OOOOO DATA=00791 CODE=00489 DICT=00654:01229/01883 GSA FLAG
```

COBOL SIEVE

```
      ** CIS COBOL V4.4                  siev.cbl             PAGE: 0001
      **
           IDENTIFICATION DIVISION.
           PROGRAM-ID. SIEVE.
           AUTHOR. PETER DIBBLE.
           ENVIRONMENT DIVISION.
           WORKING-STORAGE SECTION.
           77  PRIME                          PIC 9(5) COMP.
           77  PRIME-COUNT                    PIC 9(5) COMP.
           77  I                              PIC 9(4) COMP.
           77  K                              PIC 9(5) COMP.
           01  BIT-ARRAY.
                03 BIT OCCURS 8191 TIMES       PIC 9 COMP.
           PROCEDURE DIVISION.
           START-UP.
                DISPLAY "TEN ITERATIONS".
                PERFORM SIEVE THROUGH SIEVE-END.
                DISPLAY "PRIMES FOUND: ", PRIME-COUNT.
                STOP RUN.
           SIEVE.
                MOVE ZERO TO PRIME-COUNT.
                MOVE 1 TO I.
                PERFORM INIT-BITS 8191 TIMES.
                MOVE 1 TO I.
                PERFORM SCAN-FOR-PRIMES THROUGH END-SCAN-FOR-PRIMES
                     8191 TIMES.
           SIEVE-END.
                EXIT
           INIT-BITS.
                MOVE 1 TO BIT (I).
```

```
               ADD 1 TO I.
          END-INIT-BITS.
               EXIT.
          SCAN-FOR-PRIMES.
               IF BIT (I) = 0
                    THEN
                         GO TO NOT-PRIME.
               ADD I I 1 GIVING PRIME.
         *     DISPLAY PRIME.
               ADD I PRIME GIVING K.
               PERFORM STRIKOUT UNTIL K > 8191.
               ADD 1 TO PRIME-COUNT.
          NOT-PRIME.
               ADD 1 TO I.
          END-SCAN-FOR-PRIMES.
               EXIT
          STRIKOUT.
               MOVE 0 TO BIT (K).
               ADD PRIME TO K.
          END-PROGRAM.
               EXIT.
     ** CIS COBOL V4.4 REVISION 0                                    URN rp/
```

COBOL BENCHMARK PROGRAM

```
     ** CIS COBOL V4.4                    Bnch.CBL              PAGE: 0001
     **
            IDENTIFICATION DIVISION.
            PROGRAM-ID. ISAM-BENCHMARK
            AUTHOR. PETER DIBBLE.
            ENVIRONMENT DIVISION,
            CONFIGURATION SECTION.
            SOURCE-COMPUTER. GIMIX.
            OBJECT-COMPUTER. GIMIX.
            INPUT-OUTPUT SECTION.
            FILE-CONTROL.
                 SELECT ISAM-FILE-1 ASSIGN "ISAM.FILE";
                      ORGANIZATION IS INDEXED ;
                      ACCESS MODE IS SEQUENTIAL ;
                      RECORD KEY IS ISAM-KEY-1.
                 SELECT ISAM-FILE-2 ASSIGN "ISAM.FILE";
                      ORGANIZATION IS INDEXED ;
                      ACCESS MODE IS RANDOM ;
                      RECORD KEY IS ISAM-KEY-2.
            DATA DIVISION.
            FILE SECTION.
            FD  ISAM-FILE-1;
                 DATA RECORD ISAM-RECORD-1.
            01  ISAM-RECORD-1.
                 03  ISAM-KEY-1               PIC 9(9) COMP-3.
                 03  FILLER                   PIC X(50).
            FD  ISAM-FILE-2;
                 DATA RECORD ISAM-RECORD-2.
            01  ISAM-RECORD-2.
                 03 ISAM-KEY-2                PIC 9(9) COMP-3.
                 03 FILLER                    PIC X(50).
            WORKING-STORAGE SECTION.
```

```
          77  KEY-NO                         PIC 9(9) COMP-3 VALUE 0.
          77  HI-NUMBER                      PIC 9(9) COMP-3.
          77  LO-NUMBER                      PIC 9(9) COMP-3.
          77  DATE                           PIC XXX VALUE "004"
          01  WORK-DATA.
               03 WORK-KEY                    PIC 9(9) COMP-3.
               03 I-DATA                      PIC X(50).
          01 SYSTEM-DATE.
               03 YEAR                        PIC 99.
               03 MONTH                       PIC 99.
               03 DAY                         PIC 99.
          01 SYSTEM-TIME.
               03 HOUR                        PIC 99.
               03 MINUTE                      PIC 99.
               03 SECOND                      PIC 99.
          PROCEDURE DIVISION.
          START-UP.
               OPEN OUTPUT ISAM-FILE-1.
               MOVE "ASSORTED DATA: NAME, ADDRESS, ETC, OR WHATEVER" TO
                    I-DATA.
               ADD 1 KEY-NO GIVING LO-NUMBER.
               MOVE KEY-NO TO WORK-KEY.
               DISPLAY "START BUILD".
               CALL DATE USING SYSTEM-DATE, SYSTEM-TIME.
               DISPLAY "TIME " HOUR, ":", MINUTE, ":", SECOND
** CIS COBOL V4.4                    Bnch.CBL              PAGE: 0002
**
               PERFORM ADD-RECORD 10000 TIMES.
               CLOSE ISAM-FILE-1
               DISPLAY "BUILD DONE".
               CALL DATE USING SYSTEM-DATE, SYSTEM-TIME.
               DISPLAY "TIME " HOUR, ":", MINUTE, ":", SECOND
               MOVE WORK-KEY TO HI-NUMBER.
               DISPLAY "READ STARTING".
               OPEN INPUT ISAM-FILE-2.
               PERFORM TEST-READS 2500 TIMES.
               CLOSE ISAM-FILE-2.
               CALL DATE USING SYSTEM-DATE, SYSTEM-TIME.
               DISPLAY "TIME " HOUR, ":", MINUTE, ":", SECOND
               DISPLAY "READ DONE".
               STOP RUN.
          ADD-RECORD.
               ADD 1 TO WORK-KEY.
               WRITE ISAM-RECORD-1 FROM WORK-DATA;
                    INVALID KEY PERFORM ERROR-1.
          ERROR-1.
               DISPLAY "INVALID KEY: ", ISAM-KEY-1.
          TEST-READS.
               PERFORM READ-HIGH.
               PERFORM READ-HIGH.
               PERFORM READ-LOW.
               PERFORM READ-LOW.
          READ-HIGH.
               MOVE HI-NUMBER TO ISAM-KEY-2, WORK-KEY.
               READ ISAM-FILE-2; INVALID KEY PERFORM ERROR-2.
               SUBTRACT 1 FROM WORK-KEY GIVING HI-NUMBER.
          ERROR-2.
               DISPLAY "INVALID KEY: ", WORK-KEY.
```

```
          READ-LOW.
               MOVE LO-NUMBER TO WORK-KEY, ISAM-KEY-2.
               READ ISAM-FILE-2, INVALID KEY PERFORM ERROR-2.
               ADD 1 WORK-KEY GIVING LO-NUMBER.
          END-PROGRAM.
               EXIT.
     ** CIS COBOL V4.4 REVISION 0                                    URN rp/
     ** COMPILER COPYRIGHT (C) 1978,1981 MICRO FOCUS LTD
     ** ERRORS-00000 DATA-00705 CODE=00703 DICT=00612:01271/01883 GSA FLAG
```