```
/*   ~$F  LARGE FREE STRING SPACE
   A TEXT COMPRESSION PROGRAM.

   AUTHOR -- CHARLES WETHERELL  18 AUGUST 1976.
   LAST DATE MODIFIED -- 25 AUGUST 1976.

   THIS PROGRAM COMPRESSES TEXT FILES USING THE MAYNE-JAMES DICTIONARY
   CONSTRUCTION ALGORITHM.  INPUT IS AN ARBITRARY TEXT FILE AND OUTPUT
   IS THE COMPRESSED FILE ALONG WITH AN ENCODING DICTIONARY.  SOME
   STATISTICS ON PROGRAM EFFICIENCY AND COMPRESSION RATE ARE KEPT.
   TWO PASSES BY THE SOURCE FILE ARE NECESSARY.  ON THE FIRST PASS THE
   DICTIONARY IS BUILT AND A RECORD OF ALL CHARACTERS SEEN IS KEPT.
   BETWEEN PASSES THE CHARACTER RECORD IS USED TO ADD ENCODINGS TO THE
   DICTIONARY ENTRIES.  DURING THE SECOND PASS, LONG HIGH FREQUENCY
   STRINGS ARE REPLACED BY SHORTER ENCODING STRINGS AS THE COMPRESSED
   FILE IS WRITTEN.  FURTHER INFORMATION ABOUT THE TECHNIQUE CAN BE
   FOUND IN CHAPTER 11 OF

      WETHERELL, C.S.  ETUDES FOR PROGRAMMERS.  PRENTICE-HALL,
         ENGLEWOOD CLIFFS, NJ.  1978.

   IN THIS VERSION OF THE ALGORITHM, ENDS OF INPUT LINES STOP STRING
   MATCHES DURING DICTIONARY CONSTRUCTION AND TEXT ENCODING; THE
   CARRIAGE RETURN IS NOT TREATED LIKE A CHARACTER.  ONLY CHARACTERS
   WHOSE INTERNAL REPRESENTATIONS LIE IN THE RANGE 1 TO 127 WILL BE
   CONSIDERED FOR ENCODING PAIRS SO THAT THE PAIRS WILL HAVE
   REASONABLE PRINT REPRESENTATIONS. IN A FULL WORKING IMPLEMENTATION
   ALL 256 AVAILABLE CHARACTERS WOULD BE USED FOR ENCODING.

   THE DICTIONARY SEARCH, ENTRY, AND CLEANUP ROUTINES ARE WRITTEN
   SO THAT THEY MAY BE CHANGED QUITE EASILY.
   THE ALGORITHMS CAN BE MODIFIED BY REPLACING THE BODIES OF PROCEDURES
   SEARCH.DICTIONARY, CLEAN.DICTIONARY, AND BUILD.ENTRY, ALONG WITH
   MAKING ANY NECESSARY CHANGES TO PREPARE.THE.PROGRAM.  THE VARIOUS
   THRESHOLDS PARAMETERS ARE ALL CALCULATED BY FUNCTIONS AND CAN BE
   BE MODIFED BY CHANGING THE FUNCTION DEFINITIONS.
   IF THERE IS A DATA STRUCTURE ADDED FOR SEARCHING, MAKE SURE THAT
   BUILD.ENCODING.TABLE LEAVES THE STRUCTURE IN GOOD SHAPE AFTER CODES
   ARE ADDED AND ENTRIES OF LENGTH TWO AND LESS ARE DELETED.

   THIS VERSION USES SIMPLE LINEAR SEARCH, A HYPERBOLIC THRESHOLD
   FOR COALESCENCE, A MEAN THRESHOLD FOR DELETION, AND AN INITIAL
   COUNT OF ONE FOR COALESCED ENTRIES.
*/

/* SOME MACROS TO IMPROVE XPL AS A LANGUAGE.                              */

```

```
DECLARE  LOGICAL  LITERALLY 'BIT(1)',
         TRUE     LITERALLY '1',
         FALSE    LITERALLY '0',
         NOT      LITERALLY '~',         /* TO IMPROVE PRINTING */
         COMPUTE.TIME LITERALLY 'COREWORD("1E312")*2'; /* THE CLOCK */

/* DECLARATIONS FOR I/O UNITS                                             */

DECLARE  SOURCE.FILE LITERALLY '1',
         ECHO.FILE   LITERALLY '2',
         PRINT LITERALLY 'OUTPUT(ECHO.FILE) =';

/* DECLARATIONS FOR THE INPUT ROUTINE                                     */

DECLARE  INPUT.BUFFER CHARACTER,
         (PRINT.SOURCE, CHECK.CHARACTERS) LOGICAL,
         CHARACTER.USED("FF") LOGICAL;  /* I.E. 256 DIFFERENT ENTRIES*/

/* DECLARATIONS FOR THE DICTIONARY                                        */

DECLARE   DICTIONARY.SIZE LITERALLY '100',
          DICTIONARY.STRING(DICTIONARY.SIZE) CHARACTER,
          DICTIONARY.COUNT(DICTIONARY.SIZE) FIXED,
          DICTIONARY.CODE(DICTIONARY.SIZE) CHARACTER,
          DICTIONARY.USAGE(DICTIONARY.SIZE) FIXED,
          DICTIONARY.TOP FIXED;

/* CONTROL FOR ENCODING PRINT.                                            */

DECLARE  PRINT.ENCODING LOGICAL;

/* DECLARATIONS FOR ENCODING STATISTICS */

DECLARE  SEARCH.COMPARES FIXED,
         BUILD.COMPARES FIXED,
         COMPRESS.COMPARES FIXED;

DECLARE  TIME.CHECK(10) FIXED;

DECLARE  (INPUT.LENGTH, OUTPUT.LENGTH) FIXED;

I.FORMAT: PROCEDURE(NUMBER, WIDTH) CHARACTER;

   /* FUNCTION I.FORMAT CONVERTS ITS ARGUMENT NUMBER INTO A STRING
      AND THEN PADS THE STRING ON THE LEFT TO BRING THE LENGTH UP
      TO WIDTH CHARACTERS.  ALL OF THIS IS JUST THE FORTRAN
      INTEGER FORMAT.
   */
```

```

   DECLARE  NUMBER FIXED,
            WIDTH FIXED;

   DECLARE  STRING CHARACTER,
            BLANKS CHARACTER INITIAL ('                          ');

   STRING = NUMBER;
   IF LENGTH(STRING) < WIDTH
      THEN STRING = SUBSTR(BLANKS,0,WIDTH-LENGTH(STRING)) || STRING;
   RETURN STRING;

END I.FORMAT;

PREPARE.THE.PROGRAM: PROCEDURE;

   /* ONLY SIMPLE CLEARING OF THE DICTIONARY, THE CHARACTER RECORD,
      THE STATISTICS, AND A FEW SCALARS IS REQUIRED.
   */

   DECLARE I FIXED;

   DO I = 0 TO DICTIONARY.SIZE;
      DICTIONARY.STRING(I), DICTIONARY.CODE(I) = '';
      DICTIONARY.COUNT(I), DICTIONARY.USAGE(I) = 0;
   END;
   DICTIONARY.TOP = -1;

   DO I = 0 TO "FF"; CHARACTER.USED(I) = FALSE; END;
   INPUT.BUFFER = '';

   SEARCH.COMPARES = 0;
   INPUT.LENGTH, OUTPUT.LENGTH = 0;

END PREPARE.THE.PROGRAM;

FILL.INPUT.BUFFER: PROCEDURE;

   /* IF INPUT.BUFFER IS EMPTY, THEN THIS ROUTINE TRIES TO READ A
      LINE FROM THE SOURCE.FILE.  THE LINE READ GOES INTO
      INPUT.BUFFER WITH A NULL LINE THE SIGNAL FOR END OF FILE.  IF
      FLAG PRINT.SOURCE IS ON, THEN THE INPUT IS ECHOED.  IF FLAG
      CHECK.CHARACTERS IS ON, THE INPUT IS SCANNED FOR CHARACTER
      USAGE.
   */

   DECLARE  I FIXED;

```

```
      IF LENGTH(INPUT.BUFFER) > 0 THEN RETURN;
      INPUT.BUFFER = INPUT(SOURCE.FILE);
      IF PRINT.SOURCE THEN PRINT INPUT.BUFFER;
      IF CHECK.CHARACTERS THEN
         DO I = 0 TO LENGTH(INPUT.BUFFER)-1;
            CHARACTER.USED(BYTE(INPUT.BUFFER,I)) = TRUE;
         END;

END FILL.INPUT.BUFFER;

COALESCENCE.THRESHOLD: PROCEDURE FIXED;

   /* THIS PROCEDURE CALCULATES THE THRESHOLD FOR COALESCING
      TWO DICTIONARY ENTRIES INTO ONE.  HERE, THE REQUIREMENT IS
      THAT THE ENTRIES HAVE FREQUENCIES GREATER THAN THE RECIPROCAL
      OF THE RATIO OF SPACE REMAINING IN THE DICTIONARY.
   */

   RETURN DICTIONARY.SIZE/(DICTIONARY.SIZE-DICTIONARY.TOP+1) + 1;

END COALESCENCE.THRESHOLD;

DELETION.THRESHOLD: PROCEDURE FIXED;

   /* THIS FUNCTION RETURNS THE THRESHOLD NECESSARY FOR AN ENTRY
      TO BE RETAINED IN THE DICTIONARY AT CLEANUP TIME.
      IN THIS VERSION, THE FREQUENCY MUST BE GREATER THAN THE
      ROUNDED UP MEAN FREQUENCY.
   */

   DECLARE  SUM FIXED,
            I FIXED;

   SUM = 0;
   DO I = 0 TO DICTIONARY.TOP;
      SUM = SUM + DICTIONARY.COUNT(I);
   END;
   RETURN SUM/(DICTIONARY.TOP+1) + 1;

END DELETION.THRESHOLD;

FIRST.COUNT: PROCEDURE FIXED;

   /* THIS FUNCTION RETURNS THE COUNT GIVEN A COALESCED ENTRY WHEN
      IT IS FIRST ENTERED IN THE DICTIONARY.
   */

   RETURN 1; /* CURRENTLY GIVE A COUNT OF 1. */
```

```

END FIRST.COUNT;

SEARCH.DICTIONARY: PROCEDURE(TEST.STRING) FIXED;

    /* THIS FUNCTION SEARCHES THE DICTIONARY FOR THE LONGEST MATCH
       WITH THE HEAD OF THE ARGUMENT TEST.STRING.  IF NO MATCH IS
       FOUND, THE ROUTINE RETURNS -1 AS VALUE; IF A MATCH IS FOUND,
       THE INDEX OF THE MATCH IS RETURNED AS VALUE.

       THIS ROUTINE PERFORMS A SIMPLE LINEAR SEARCH OF THE DICTIONARY
       FROM THE ZEROTH ENTRY TO THE ENTRY DICTIONARY.TOP.  IF AN
       ENTRY'S LENGTH IS LONGER THAN THE LONGEST CURRENT MATCH AND
       STILL NO LONGER THAN THE ARGUMENT, THEN THE ENTRY IS MATCHED
       AGAINST THE ARGUMENT.  EQUALITY WILL CAUSE THE MATCH TO BE
       UPDATED. NOTICE THAT BY STARTING THE INDEX AT -1, THE RETURN
       VALUE WILL BE PROPER EVEN IF NO MATCH IS FOUND.
    */

    DECLARE  TEST.STRING CHARACTER;

    DECLARE  INDEX FIXED,
             (MATCH.LENGTH, ARG.LENGTH, ENTRY.LENGTH) FIXED,
             I FIXED;

    INDEX = -1;
    MATCH.LENGTH = 0;
    ARG.LENGTH = LENGTH(TEST.STRING);

    DO I = 0 TO DICTIONARY.TOP;
       ENTRY.LENGTH = LENGTH(DICTIONARY.STRING(I));
       IF ENTRY.LENGTH > MATCH.LENGTH
             & ENTRY.LENGTH <= ARG.LENGTH THEN
          IF DICTIONARY.STRING(I)
                = SUBSTR(TEST.STRING,0,ENTRY.LENGTH) THEN
             DO;
                INDEX = I;
                MATCH.LENGTH = ENTRY.LENGTH;
             END;
    END;
    SEARCH.COMPARES = SEARCH.COMPARES + DICTIONARY.TOP + 1;
    RETURN INDEX;

END SEARCH.DICTIONARY;

CLEAN.DICTIONARY: PROCEDURE;

```

```
    /* CLEAN.DICTIONARY ELIMINATES AT LEAST ONE LOW FREQUENCY ENTRY
       FROM THE DICTIONARY AND RESTORES THE SMALLER DICTIONARY TO
       THE FORMAT IT HAD BEFORE CLEANING.
       THE WHILE LOOP SURROUNDING THE BODY OF THE PROCEDURE GUARANTEES
       THAT AT LEAST ONE ENTRY IS DELETED FROM THE DICTIONARY BEFORE
       RETURN.  IF THE INITIAL THRESHOLD IS NOT HIGH ENOUGH TO DELETE
       AN ENTRY, THE THRESHOLD IS INCREMENTED UNTIL SOMETHING IS
       DELETED.

       THE DICTIONARY IS JUST A LINEAR TABLE WITH NO STRUCTURE SO
       ENTRIES CAN BE DELETED BY PUSHING THE RETAINED ENTRIES TOWARD
       THE ZERO END OF THE TABLE OVERWRITING THE REMOVED ENTRIES.
    */

    DECLARE  I FIXED,
             THRESHOLD FIXED,
             OLD.TOP FIXED,
             NEW.TOP FIXED;

    OLD.TOP = DICTIONARY.TOP;
    THRESHOLD = DELETION.THRESHOLD;
    DO WHILE OLD.TOP = DICTIONARY.TOP;
       NEW.TOP = -1;
       DO I = 0 TO DICTIONARY.TOP;
          IF DICTIONARY.COUNT(I) >= THRESHOLD THEN
             DO;
                NEW.TOP = NEW.TOP + 1;
                DICTIONARY.STRING(NEW.TOP) = DICTIONARY.STRING(I);
                DICTIONARY.COUNT(NEW.TOP) = DICTIONARY.COUNT(I);
             END;
       END;
       DICTIONARY.TOP = NEW.TOP;
       THRESHOLD = THRESHOLD + 1;
    END;

END CLEAN.DICTIONARY;

BUILD.ENTRY: PROCEDURE(ENTRY.STRING) FIXED;

    /* BUILD.ENTRY ADDS ENTRY.STRING TO THE DICTIONARY WITH A COUNT
       OF ZERO AND RETURNS AS VALUE THE INDEX OF THE NEW ENTRY.

       BECAUSE THE DICTIONARY IS SEARCHED LINEARLY, THE NEW ENTRY
       CAN SIMPLY BE ADDED AT THE END.  THE ONLY REQUIREMENT IS THAT
       THE DICTIONARY MAY NEED TO BE CLEANED BEFORE THE NEW ENTRY
       CAN BE ADDED.
    */

```

```
    DECLARE  ENTRY.STRING CHARACTER;

    IF DICTIONARY.TOP+2 >= DICTIONARY.SIZE
       THEN CALL CLEAN.DICTIONARY;
    DICTIONARY.TOP = DICTIONARY.TOP + 1;
    DICTIONARY.STRING(DICTIONARY.TOP) = ENTRY.STRING;
    DICTIONARY.COUNT(DICTIONARY.TOP) = 0;
    RETURN DICTIONARY.TOP;

END BUILD.ENTRY;

BUILD.DICTIONARY: PROCEDURE;

    /* DICTIONARY CONSTRUCTION CONTINUES UNTIL THE INPUT ROUTINE
       FAILS TO RETURN ANY DATA.  THE TEST FOR A NULL STRING IS
       SIMPLE IF WE CHECK THE LENGTH AGAINST ZERO.  THE
       DICTIONARY.SEARCH ROUTINE RETURNS -1 IF NO MATCH IS FOUND AND
       THEN THE FIRST CHARACTER OF THE INPUT IS FORCED AS THE MATCH
       AND INTO THE DICTIONARY.  NOTICE THAT THE ACTUAL STRING
       MATCHED IS PICKED UP FROM THE DICTIONARY ENTRY.  COALESCENCE
       TAKES PLACE AS NECESSARY, THE MATCH IS REMEMBERED, AND THE
       INPUT PREPARED FOR ANOTHER CYCLE.
    */

    DECLARE  (MATCH, LAST.MATCH) CHARACTER,
             (COUNT, LAST.COUNT) FIXED,
             INDEX FIXED,               /* THE ENTRY LOCATION    */
             THRESHOLD FIXED;           /* COALESCENCE THRESHOLD */

    LAST.MATCH = '';
    LAST.COUNT = 0;

    DO WHILE TRUE;
       CALL FILL.INPUT.BUFFER;
       IF LENGTH(INPUT.BUFFER) = 0 THEN RETURN;
       INDEX = SEARCH.DICTIONARY(INPUT.BUFFER);
       IF INDEX = -1
          THEN INDEX = BUILD.ENTRY(SUBSTR(INPUT.BUFFER,0,1));
       MATCH = DICTIONARY.STRING(INDEX);
       COUNT, DICTIONARY.COUNT(INDEX) = DICTIONARY.COUNT(INDEX) + 1;
       THRESHOLD = COALESCENCE.THRESHOLD;
       IF COUNT >= THRESHOLD & LAST.COUNT >= THRESHOLD THEN
          DICTIONARY.COUNT(BUILD.ENTRY(LAST.MATCH||MATCH))=FIRST.COUNT;
       LAST.MATCH = MATCH;
       LAST.COUNT = COUNT;
       INPUT.BUFFER = SUBSTR(INPUT.BUFFER, LENGTH(MATCH));
    END;
```

```
END BUILD.DICTIONARY;

BUILD.ENCODING.TABLE: PROCEDURE;

   /* CODE CONSTRUCTION HAS TWO STEPS.  IN THE FIRST, EVERY
      DICTIONARY ENTRY OF LENGTH TWO OR ONE IS THROWN OUT BECAUSE
      THERE IS NO POINT IN REPLACING SUCH STRINGS WITH A TWO
      CHARACTER CODE.  SECOND, CODES ARE ASSIGNED USING CHARACTERS
      UNSEEN IN THE TEXT AS STARTERS.  WHEN SUCH PAIRS RUN OUT,
      NO MORE CODES ARE ASSIGNED EVEN IF THERE ARE MORE ENTRIES IN
      THE DICTIONARY.

      NOTICE THE LINES BELOW WHICH CONSTRUCT THE DICTIONARY CODE.
      THE APPARENTLY SENSELESS CATENATION OF TWO BLANKS BUILDS A
      COMPLETELY NEW STRING INTO WHICH THE CODE CHARACTERS CAN BE
      INSERTED.  THIS IS A BAD GLITCH IN XPL AND YOU PROBABLY WON'T
      UNDERSTAND IT UNLESS YOU PROGRAM IN XPL FOR SOME TIME.
   */

   DECLARE  (I, J) FIXED,
            TOP FIXED;

   TOP = -1;
   DO I = 0 TO DICTIONARY.TOP;
      IF LENGTH(DICTIONARY.STRING(I)) > 2 THEN
         DO;
            TOP = TOP + 1;
            DICTIONARY.STRING(TOP) = DICTIONARY.STRING(I);
            DICTIONARY.COUNT(TOP) = DICTIONARY.COUNT(I);
         END;
   END;
   DICTIONARY.TOP = TOP;

   TOP = -1;
   DO I = 1 TO "7F";     /* LOOP OVER ELIGIBLE START CHARACTERS */
      IF NOT CHARACTER.USED(I) THEN
         DO J = 1 TO "7F";     /* LOOP OVER SECOND CHARACTERS   */
            IF TOP = DICTIONARY.TOP THEN RETURN;
            TOP = TOP + 1;
            DICTIONARY.CODE(TOP) = ' ' || ' ';
            BYTE(DICTIONARY.CODE(TOP),0) = I;
            BYTE(DICTIONARY.CODE(TOP),1) = J;
         END;
   END;
   DICTIONARY.TOP = TOP;

END BUILD.ENCODING.TABLE;
```

```
COMPRESS.TEXT: PROCEDURE;

    /* ENCODING WORKS ALMOST THE SAME WAY AS DICTIONARY CONSTRUCTION.
       HERE, THOUGH, THE INPUT STREAM IS CONVERTED TO OUTPUT LINES
       AS THE ENCODINGS ARE FOUND.  THE LOOP RUNS FOR AS LONG AS
       THERE IS INPUT.
    */

    DECLARE  OUTPUT.BUFFER CHARACTER,
             INDEX FIXED;

    INPUT.BUFFER = '';
    PRINT '';
    PRINT '*** THE COMPRESSED TEXT ***';
    PRINT '';
    CALL FILL.INPUT.BUFFER;
    DO WHILE LENGTH(INPUT.BUFFER) > 0;
       INPUT.LENGTH = INPUT.LENGTH + LENGTH(INPUT.BUFFER);
       OUTPUT.BUFFER = '';
       DO WHILE LENGTH(INPUT.BUFFER) > 0;
          INDEX = SEARCH.DICTIONARY(INPUT.BUFFER);
          IF INDEX > -1 THEN
             DO;
                OUTPUT.BUFFER = OUTPUT.BUFFER
                              || DICTIONARY.CODE(INDEX);
                DICTIONARY.USAGE(INDEX) = DICTIONARY.USAGE(INDEX) + 1;
                INPUT.BUFFER = SUBSTR(INPUT.BUFFER,
                                  LENGTH(DICTIONARY.STRING(INDEX)));
             END;
          ELSE
             DO;
                OUTPUT.BUFFER = OUTPUT.BUFFER
                              || SUBSTR(INPUT.BUFFER,0,1);
                INPUT.BUFFER = SUBSTR(INPUT.BUFFER,1);
             END;
       END;
       OUTPUT.LENGTH = OUTPUT.LENGTH + LENGTH(OUTPUT.BUFFER);
       IF PRINT.ENCODING THEN PRINT OUTPUT.BUFFER;
       CALL FILL.INPUT.BUFFER;
    END;

END COMPRESS.TEXT;

PRINT.SUMMARY.STATISTICS: PROCEDURE;

```

```
    /* SUMMARY STATISTICS INCLUDE A SECOND PRINTING OF THE SEARCH
       STATISTICS, THE DICTIONARY ITSELF, AND THE COMPRESSION RATE.
       IN A WORKING VERSION, BOTH THE COMPRESSED TEXT AND THE
       DICTIONARY WOULD HAVE ALSO BEEN PRINTED ON SEPARATE FILES FOR
       RE-EXPANSION LATER.  NOTICE THE COMPLICATION TO PRINT A RATIO
       AS A DECIMAL IN A LANGUAGE WITHOUT FLOATING POINT NUMBERS.
    */

    DECLARE  I FIXED,
             RATIO CHARACTER;

    PRINT '';
    PRINT '*** COMPRESSION STATISTICS ***';
    PRINT '';
    PRINT 'CODE FREQUENCY   USAGE   STRING';
    DO I = 0 TO DICTIONARY.TOP;
       PRINT ' ' || DICTIONARY.CODE(I) || '  '
                          || I.FORMAT(DICTIONARY.COUNT(I), 9) || ' '
                          || I.FORMAT(DICTIONARY.USAGE(I), 9)
                          || ' |' || DICTIONARY.STRING(I) || '|';
    END;
    PRINT '';
    PRINT '   CHARACTERS IN INPUT = ' || INPUT.LENGTH;
    PRINT '   CHARACTERS IN OUTPUT = ' || OUTPUT.LENGTH;
    RATIO = (1000*OUTPUT.LENGTH)/INPUT.LENGTH + 1000;
    PRINT '   COMPRESSION RATE = .' || SUBSTR(RATIO,1);
    PRINT '   COMPARES DURING DICTIONARY CONSTRUCTION = '
                   || BUILD.COMPARES;
    PRINT '   COMPARES DURING COMPRESSION = '
                   || COMPRESS.COMPARES;
    PRINT '*** TIME CHECKS IN MILLESECONDS ***';
    PRINT '   TIME TO PREPARE = '
                   || TIME.CHECK(0) - TIME.CHECK(1);
    PRINT '   TIME TO BUILD DICTIONARY = '
                   || TIME.CHECK(1) - TIME.CHECK(2);
    PRINT '   ENCODING TABLE TIME = '
                   || TIME.CHECK(2) - TIME.CHECK(3);
    PRINT '   COMPRESSION TIME = '
                   || TIME.CHECK(3) - TIME.CHECK(4);

END PRINT.SUMMARY.STATISTICS;

/* THE MAIN ROUTINE MUST ASSIGN THE I/O UNITS TO FILES, INITIALIZE
   NEEDED VARIABLES, CALL THE DICTIONARY CONSTRUCTION ALGORITHM, BUILD
   THE ENCODING TABLE, AND THEN ENCODE THE OUTPUT.  MOST OF THIS WORK
   IS DONE IN CALLED PROCEDURES.
*/
```

```
TIME.CHECK(0) = COMPUTE.TIME;
CALL ASSIGN('TEXT', SOURCE.FILE, "(1) 1000 0110");
CALL ASSIGN('COMPRESS', ECHO.FILE, "(1) 0010 1010");
PRINT '*** BEGIN THE TEXT COMPRESSION. ***';
PRINT '';

CALL PREPARE.THE.PROGRAM;
PRINT.SOURCE = TRUE; CHECK.CHARACTERS = TRUE;
TIME.CHECK(1) = COMPUTE.TIME;
CALL BUILD.DICTIONARY;
BUILD.COMPARES = SEARCH.COMPARES;
TIME.CHECK(2) = COMPUTE.TIME;
CALL BUILD.ENCODING.TABLE;

CALL ASSIGN('TEXT', SOURCE.FILE, "(1) 1000 0110"); /* A REWIND */
SEARCH.COMPARES = 0;
PRINT.SOURCE, CHECK.CHARACTERS = FALSE;
PRINT.ENCODING = TRUE;
TIME.CHECK(3) = COMPUTE.TIME;
CALL COMPRESS.TEXT;
COMPRESS.COMPARES = SEARCH.COMPARES;

TIME.CHECK(4) = COMPUTE.TIME;
CALL PRINT.SUMMARY.STATISTICS;

EOF EOF EOF EOF EOF EOF
```

RESULTS

We ran the program on a short prefix of itself used as data; the results are printed below with line numbers for reference. Lines 67-71 show the encoding dictionary; with such a short text file it is no wonder the dictionary is short. Compression is only 0.973, partly because text lines in the host system are not padded out with those blanks so beloved by most compression routines. Still some interesting compression takes place, as witness line 62. Notice that compression "D F" was not used because another compression ate up the "D" first. Here are the results.

```
 1 *** BEGIN THE TEXT COMPRESSION. ***
 2
 3 /* ~$F  LARGE FREE STRING SPACE
 4     A TEXT COMPRESSION PROGRAM.
 5
 6     AUTHOR -- CHARLES WETHERELL  18 AUGUST 1976.
 7     LAST DATE MODIFIED -- 25 AUGUST 1976.
 8
 9     THIS PROGRAM COMPRESSES TEXT FILES USING THE MAYNE-JAMES DICTIONARY
10     CONSTRUCTION ALGORITHM.  INPUT IS AN ARBITRARY TEXT FILE AND OUTPUT
11     IS THE COMPRESSED FILE ALONG WITH AN ENCODING DICTIONARY.  SOME
12     STATISTICS ON PROGRAM EFFICIENCY AND COMPRESSION RATE ARE KEPT.
13     TWO PASSES BY THE SOURCE FILE ARE NECESSARY.  ON THE FIRST PASS THE
14     DICTIONARY IS BUILT AND A RECORD OF ALL CHARACTERS SEEN IS KEPT.
```