DISCLAIMER

THIS REPORT IS ISSUED FOR INFORMATION AND DOCUMENTATION PURPOSES ONLY. NO REPRESENTATION IS MADE BY THE GOVERNMENT WITH RESPECT TO THE INFORMATION IN THIS REPORT AND THE GOVERNMENT ASSUMES NO LIABILITY OR OBLIGATION WITH RESPECT THERETO; NOR IS THE MATERIAL PRESENTED TO BE CONSTRUED AS THE OFFICIAL POSITION OF THE DEPARTMENT OF THE ARMY, UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.
MISD User Manual No. 77-2

CHARACTER STRING PROCESSING LIBRARY ROUTINES
USER MANUAL

BRUCE D. CARREREY

APRIL 1977

MANAGEMENT INFORMATION SYSTEMS DIRECTORATE
ARRADCOM
DOVER, NEW JERSEY 07801

DISTRIBUTION STATEMENT A
Approved for public release; Distribution Unlimited
ABSTRACT

THIS REPORT DESCRIBES A SET OF GENERAL-PURPOSE, FORTRAN-CALLABLE FUNCTIONS AND SUBROUTINES FOR MANIPULATING CHARACTER-STRINGS. CHARACTER-STRINGS OF ARBITRARY LENGTH CAN BE TREATED AS SIMPLE (NON-DIMENSIONED) VARIABLES. FUNCTIONS ARE PROVIDED FOR STRING DEFINITION, STRING CONCATENATION, SUBSTRING EXTRACTION, STRING COMPARISON, PATTERN MATCHING AND SEARCHING, NUMERIC-STRING/STRING-NUMERIC CONVERSIONS, STRING INPUT/OUTPUT, AND OTHER TASKS. THE ROUTINES DESCRIBED ARE MODERATELY SYSTEM-DEPENDENT, AND ARE CURRENTLY AVAILABLE AS AN OBJECT PROGRAM LIBRARY ON THE CDC 6500/6600 COMPUTERS AT THE ARDCOM DOVER, NEW JERSEY SITE.
ASSUMPTIONS

THE READER IS EXPECTED TO BE AN EXPERIENCED FORTRAN PROGRAMMER WITH A SOLID UNDERSTANDING OF COMPUTER CONCEPTS AND A GENERAL KNOWLEDGE OF THE CDC SCOPE OPERATING SYSTEM AND CDC FORTRAN EXTENDED LANGUAGE.

WARNING

THE FUNCTIONS AND SUBROUTINES DESCRIBED IN THIS REPORT ARE MODERATELY SYSTEM-DEPENDENT. USERS ANTICIPATING A REQUIREMENT FOR TRANSPORTABILITY OF PROGRAMS TO A COMPUTER ENVIRONMENT OTHER THAN A CDC 6600 COMPUTER WITH SCOPE OPERATING SYSTEM SHOULD SEEK ALTERNATIVE METHODS OR CONSULT THE AUTHOR FOR IMPLEMENTATION GUIDANCE ON OTHER COMPUTERS.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Notation used in this manual</td>
<td>8</td>
</tr>
<tr>
<td>General rules for using the routines</td>
<td>9</td>
</tr>
<tr>
<td>Description of the routines</td>
<td>11</td>
</tr>
<tr>
<td>Appendices</td>
<td></td>
</tr>
<tr>
<td>A. Control cards / commands</td>
<td>43</td>
</tr>
<tr>
<td>B. Sample programs</td>
<td>44</td>
</tr>
<tr>
<td>C. Memory requirements for routines</td>
<td>64</td>
</tr>
<tr>
<td>References</td>
<td>65</td>
</tr>
<tr>
<td>Quick reference index to routines</td>
<td>66</td>
</tr>
</tbody>
</table>
INTRODUCTION

ONE OF THE MOST SERIOUS SHORTCOMINGS OF THE FORTRAN LANGUAGE IN COMPARISON TO MODERN HIGH-LEVEL LANGUAGES SUCH AS ALGOL, PL/1 OR SN23OL IS THE ABSENCE OF A CHARACTER DATA TYPE. IN MANY LANGUAGES CHARACTER STRINGS OF ANY LENGTH MAY BE DEFINED AND MANIPULATED AS A SIMPLE VARIABLE, JUST AS IS THE CASE FOR REAL OR INTEGER NUMBERS. FORTRAN, HOWEVER, HAS NO SUCH FACILITY, MAKING IT EXTREMELY DIFFICULT TO PROGRAM MANY APPLICATIONS IN WHICH CHARACTER STRINGS MUST BE MANIPULATED. (THE FUTURE ANSI FORTRAN PROPOSED BY COMMITTEE X3J3 INCLUDES A LIMITED IMPLEMENTATION OF THE CHARACTER TYPE VARIABLE. FOR BACKGROUND INFORMATION ON STRING HANDLING IN VARIOUS LANGUAGES SEE REFERENCES 6, 7, AND 8).

THE ROUTINES DESCRIBED IN THIS REPORT GO A LONG WAY TOWARD THE GOAL OF GIVING THE USER A CHARACTER-TYPE VARIABLE AND CHARACTER STRING OPERATORS AS IMPLEMENTED IN THE NEWER HIGH-LEVEL LANGUAGES. THE MOST SIGNIFICANT FEATURE OF THESE ROUTINES IS THAT THEY ENABLE THE USER TO DEFINE AND MANIPULATE CHARACTER STRINGS OF ANY LENGTH IN A MANNER SIMILAR TO OTHER VARIABLES. WITH THESE ROUTINES, ANY LENGTH STRING CAN BE REPRESENTED AS A SIMPLE INTEGER VARIABLE. VARIOUS FUNCTIONS AND SUBROUTINES ARE PROVIDED FOR DEFINING STRINGS, INPUT AND OUTPUT OF STRINGS, CONCATENATION (JOINING TOGETHER) OF STRINGS, SUBSTRING EXTRACTION, COMPARISON OF STRINGS, PATTERN MATCHING, CONVERSION OF STRINGS TO OTHER DATA TYPES, AND OTHER USEFUL TASKS. IN MANY APPLICATIONS, JUDICIOUS USE OF THESE ROUTINES CAN REDUCE THE NUMBER OF LINES OF CODE REQUIRED FOR A PROGRAM BY A FACTOR OF 10 OR BETTER, AND REDUCE DEVELOPMENT TIME BY EVEN LARGER FACTORS.

AS AN EXAMPLE OF THE USEFULNESS OF THE CHARACTER STRING ROUTINES PRESENTED HERE, CONSIDER THIS HYPOTHETICAL ANECDOTE...

PROGRAMMER ARA MODWUN HAS A PROBLEM TO CODE IN FORTRAN.
THE PROGRAM REQUIREMENTS INCLUDE THE NEED TO READ A PROJECT NAME FROM A CARD AND PRINT IT OUT IN THE MIDDLE OF A SENTENCE ON THE LINE PRINTER, SO THAT THE RESULTING LINE WILL READ:

THE <NAME> IS UNDER DEVELOPMENT.

WHERE <NAME> IS THE TEXT READ FROM THE CARD. FOR INSTANCE, IF THE CARD READ, "XM65OE3 ROCKET ASSISTED PROJECTILE", THEN THE PRINTOUT SHOULD READ:

THE XM65OE3 ROCKET ASSISTED PROJECTILE IS UNDER DEVELOPMENT.

ON THE OTHER HAND, IF THE CARD READ, "MI RIFLE", THEN THE LINE PRINTED SHOULD READ:

THE MI RIFLE IS UNDER DEVELOPMENT.
THIS PROBLEM REQUIRES APA TO PERFORM A SIMPLE CHARACTER STRING
MANIPULATION. IN FORTRAN, THOUGH, THE PROBLEM IS NOT ALL THAT
T-I-VIAL. ONE (IF THE THINGS THAT MAKES THE PROBLEM NON-TPI'IIAL
IS THE FACT THAT THE NAME READ IS OF ARBITRARY LENGTH, SO HE CANT
ALLOCATE A FIXED LENGTH OF SPACE FOR THE NAME IN THE FORMAT
STATEMENT WHEN PRINTING THE SENTENCE, SINCE APA MIGHT WIND UP
WITH HIS SENTENCE LOOKING LIKE:

THE M1 RIFLE IS UNDER DEVELOPMENT.

WHICH LOOKS SLOPPY AND IS DIFFICULT TO READ.

TO SOLVE THIS PROBLEM, APA CODED THE FOLLOWING PROGRAM SEGMENT.

**

```
DIMENSION NAME(45)
DIMENSION IPRED(2)
C
DATA IPRED / 1H, 1HI, 1HS, 1H ,
X 1HU, 1HN, 1HO, 1HE, 1HP, 1H ,
X 1HD, 1HE, 1MV, 1HE, 1ML, 1HO, 1HM, 1HE, 1HN, 1HT, 1H , /
C
READ NAME FROM CARD.
READ(5,19) (NAME(L),L=1,84)

105 FORMAT(8,4A1)
C
COUNT BACKWARD TO CUT OFF ALL TAILING BLANKS FROM NAME
DO 1 J=1,79
J=J-1
IF(NAME(J).NE.1H )GO TO 101
100 CONTINUE
101 CONTINUE
WRITE(6,106) (NAME(L),L=1,J), (IPRED(L),L=1,22)
106 FORMAT(* THE*, 1A1)
**

THE ABOVE PROGRAM WORKS, BUT ILLUSTRATES HOW A LOT OF CODE
MAY BE REWARDED FOR EVEN THE MOST TRIVIAL TEXT MANIPULATION.
IF APA HAD KNOWN HOW TO USE THE ROUTINES IN THIS
MANUAL, HE COULD HAVE REPLACED THE ENTIRE BLOCK OF CODE ABOVE
WITH A SINGLE STATEMENT (INCLUDING READING AND PRINTING), BY
SIMPLY CODING,

```
CALL KOUT(KON(" THE"), KIN(5), KON(" IS UNDER DEVELOPMENT."))
```

THIS ONE STATEMENT READS THE NAME FROM A CARD AND
PRINTS THE DESIGNED OUTPUT ON THE PRINTER. ALTHOUGH THE
STATEMENT MAY LOOK COMPLEX, IT IS ACTUALLY QUITE EASY TO
CONSTRUCT ONCE YOU HAVE MASTERED THE BASIC FUNCTIONS
PRESENTED IN THIS MANUAL.
FUNDAMENTAL CHARACTER STRING OPERATIONS

THIS SECTION INTRODUCES SOME OF THE BASIC OPERATIONS INVOLVED WITH CHARACTER STRINGS AS THEY APPLY TO THIS REPORT. LATER WE SHALL EXAMINE THE FUNCTIONS AND SUBROUTINES IN DETAIL.

1. STRING DEFINITION.

A CHARACTER STRING IS A GROUP OF FROM 0 TO 500 CHARACTERS TREATED AS A SINGLE ENTITY. A CHARACTER STRING HAVING 0 CHARACTERS IS CALLED THE NULL STRING.

TO DEFINE A CHARACTER STRING CONSTANT, WE CAN USE FUNCTION KON. FOR EXAMPLE,

ISTR = KON("ARRADCOM")
JSTR = KON("DOVER, NEW JERSEY 07801")

THE ABOVE TWO STATEMENTS DEFINE TWO STRINGS. THE FIRST STRING HAS 9 CHARACTERS. THE VALUE RETURNED BY FUNCTION KON IS CALLED THE STRING REPRESENTATION. IN THE FIRST CASE THE STRING REPRESENTATION OF "ARRADCOM" WAS ASSIGNED TO ISTR. STRING REPRESENTATIONS ARE ALWAYS ASSIGNED TO INTEGER VARIABLES. ANY STRING MAY BE REPRESENTED BY A SIMPLE INTEGER VARIABLE. THE SECOND STATEMENT GENERATES ANOTHER STRING CONSTANT REPRESENTATION ASSIGNED TO JSTR. FOR A STRING OF 23 CHARACTERS.

FOR THE PRESENT WE NEED NOT BE CONCERNED WITH WHAT THE INTERNAL FORM OF THE REPRESENTATION OF THE STRING IS; IT IS SUFFICIENT TO KNOW THAT ISTR AND JSTR UNIQUELY DEFINE THE STRINGS.

2. STRING CONCATENATION

CONCATENATION MEANS "JOINING TOGETHER". TWO OR MORE STRINGS MAY BE CONCATENATED BY USING FUNCTION KAT. FOR EXAMPLE, THE STRINGS PREVIOUSLY DEFINED CAN BE JOINED BY,

KSTR = KAT(ISTR, JSTR)

THE VARIABLE KSTR NOW CONTAINS THE REPRESENTATION OF "ARRADCOMDOVER, NEW JERSEY 07801".

3. SUBSTRING EXTRACTION.

SUBSTRING IS THE INVERSE FUNCTION OF CONCATENATION. SUBSTRING MEANS TO TAKE A STRING APART. IN ORDER TO EXTRACT A PORTION OF A LARGER CHARACTER STRING, FUNCTION KSUB MAY BE USED. FOR EXAMPLE,
NEWSTR=KSUB(ISTR+4,5)

NEWSTR IS DEFINED AS THE STRING REPRESENTATION OF THE STRING "ADCOM". BECAUSE KSUB EXTRACTED 5 CHARACTERS BEGINNING AT THE FOURTH CHARACTER OF ISTD, SIMILARLY, KSUB(ISTR+3) RESULTS IN THE STRING REPRESENTATION OF "NEW".

4. STRING INPUT/OUTPUT.

TWO ROUTINES ARE PROVIDED FOR GENERAL-PURPOSE INPUT AND OUTPUT OF STRINGS. FUNCTION KIN MAY BE USED LIKE A READ STATEMENT. FOR EXAMPLE,

LST =KIN(5)

THIS STATEMENT READS A RECORD (CARD) FROM UNIT 5 AND ASSIGNS THE STRING REPRESENTATION OF THE CARD IMAGE TO LSTR.

IF THE CARD READ FROM TAPES CONSISTED OF:

THIS IS AN EXAMPLE OF A CHARACTER STRING

THEN LSTR WOULD BE THE REPRESENTATION OF THE 40 CHARACTER STRING, "THIS IS AN EXAMPLE OF A CHARACTER STRING".

LATER A NUMBER OF OPTIONS WILL BE EXPLAINED FOR FUNCTION KIN.

TO OUTPUT A STRING, SUBROUTINE KOUT MAY BE USED. FOR EXAMPLE TO OUTPUT THE STRING PREVIOUSLY ASSIGNED TO ISTR, WE COULD SIMPLY CODE:

CALL KOUT(6,ISTR)

THIS WOULD OUTPUT THE DESIRED STRING TO UNIT 6. IF UNIT 6 WERE A PRINTER, THEN THE STRING "ADCOM" WOULD BE PRINTED (THE FIRST CHARACTER WOULD BE LOST AS CARRIAGE CONTROL, JUST LIKE ANY OTHER WRITE STATEMENT). SEVERAL CARRIAGE CONTROL OPTIONS MAY BE SPECIFIED IN THE CALL TO KOUT TO AVOID LOSS OF THE FIRST CHARACTER AS CARRIAGE CONTROL, IF DESIRED. THESE WILL BE INTRODUCED LATE.

ALREADY WE CAN SEE THAT IT WOULD BE VERY EASY TO WRITE A SECTION OF CODE TO COPY A CARD FROM THE READER TO THE PRINTER. WE COULD SIMPLY USE THE STATEMENT:

CALL KOUT(6,KIN(5))
5. STRING COMPARISON.

Using regular forTRAN statements, it is easy to compare
numbers by using an IF statement. For instance,

IF(xVAL .GE. YVAL) GO TO 23

compares XVAL and YVAL. Two strings may be compared
in a similar manner using function LEXIC. LEXIC compares
two strings for lexical sequence (that is, for dictionary
order). For example,

IF(LEXIC(ISTR, "GE", JSTR)) GO TO 23

compares string representations ISTR and JSTR. If the string
represented by ISTR would fall after JSTR in the dictionary,
or if both strings are identical, then control is transferred
to statement 23. Otherwise, control proceeds to the next
statement after the IF statement. Function LEXIC is
a logical function and always returns either .TRUE. or .FALSE..
The second argument passed to LEXIC is the keyword describing
the type of comparison desired, and may be any of the following,
"LT"  "LE"  "EQ"  "NE"  "GE"  "GT"

6. STRING LENGTH.

The length of a string is the number of characters it
contains. To find the length of any string, use function
LEN. For example,

NCHAR=LEN(ISTR)

will assign NCHAR = 8, based on our previous definition of ISTR.

7. STRING PATTERN MATCHING.

Pattern matching involves searching a string for any
substring which matches a given string. For example, if
a string consists of "THIS IS A STRING" and the pattern
is "S IS", then we can say the pattern matches the string
starting at the 4th character. The pattern matching function
is MATCH, and it returns the column of the string at
which the given pattern matches. For example,

IPAT = KON(" ")
IBLANK=MATCH(JSTR,"FIRST",IPAT)

This statement results in IBLANK=7 based on our prior
definitions, because MATCH searches for the first occurrence
of a blank (the pattern) in the string represented by JSTR.
The second argument is a keyword telling MATCH what kind of
A search to make. For instance, if we had used "FIRST NOT" instead of "FIRST", MATCH would have returned the position of the first non-blank character (which would be 1 in the example). Later we shall see how other keywords may be used and how more than one pattern may be searched for simultaneously. Pattern matching is often used in conjunction with substring to break up a string into its components. For example, if we wished to extract a substring all the information contained in parentheses in another string, we could use the following statements.

C ASSUME INSTR IS STRING TO BE SEARCHED...
C LOCATE COLUMN CONTAINING 1ST "("...
ICOL1=MATCH(INSTR,"FIRST",KON("("))
C ...AND COLUMN CONTAINING LAST "...
ICOL2=MATCH(INSTR,"LAST",KON(""))
C TRANSFER STRING BETWEEN PARENTHESES TO IPSTR...
IPSTR=KSUB(INSTR,ICOL1+1,ICOL2-ICOL1)

8. STRING CONVERSION.

Three functions are provided for conversion of strings to numbers and numbers to strings. Function INTVAL returns the integer value of the number represented by the string. For example:

NUMSTR=KON(" -237 ")
N=INTVAL(NUMSTR)
NSTRAR = NNN

In this case, the value of N will be -237. Similarly, function RLVAL can be used for string-to-real conversions. As a simple example of the utility of these functions, suppose we wanted to read a data card which contained a number in parentheses somewhere on the card. If we wished to use the numeric value inside those parentheses for a computation, we could do so by first reading the card using function KIN; second, extracting the substring between the parentheses, as already illustrated for function MATCH; finally, coding:

NUM=INTVAL(IPSTR)

At this point NUM is the integer value of the number we want, and can be directly used in computation.
FUNCTION KSR PROVIDES THE INVERSE FUNCTION OF BOTH INTVAL AND RLVAL. IT CONVERTS A REAL OR INTEGER NUMBER TO A STRING REPRESENTATION OF THE NUMBER. FOR EXAMPLE:

\[
\text{NUM}=23.14 \\
\text{NSTR}=\text{KSR(NUM)} \\
\text{CALL KOUT(6,ΚΝΥ(" NUM"),NSTR)}
\]

WILL RESULT IN THE PRINTING OF:

\[
\text{NUM}=37
\]

FUNCTION KSR HAS SEVERAL OPTIONAL FORMS WHICH WE SHALL EXAMINE LATER. IT IS IMPORTANT TO NOTE THOUGH THAT KSR IS MUCH MORE POWERFUL THAN FORTRAN WRITE/FORMAT STATEMENTS BECAUSE THE SIZE OF THE STRING CAN BE ADJUSTED AUTOMATICALLY TO THE MAGNITUDE OF THE NUMBER. FOR EXAMPLE, KSR (0) WILL GIVE A 1 CHARACTER STRING, "0", BUT KSR (-23.4588655) WILL PRODUCE THE STRING REPRESENTATION OF "-23.4588655", WHICH IS 11 CHARACTERS. THIS CAPABILITY IS CALLED DYNAMIC FORMATTING AND IS VERY USEFUL IN MANY APPLICATIONS.

9. STRING ROUTINE INITIALIZATION.

BEFORE ANY OF THE FUNCTIONS OR SUBROUTINES IN THIS MANUAL MAY BE USED, SUBROUTINE KINIT MUST BE CALLED. FOR THE PRESENT, WE SHALL LIMIT OUR DISCUSSION OF KINIT TO SAY THAT IT HAS THE FORM:

\[
\text{CALL KINIT(0)}
\]

LATER OTHER ARGUMENTS WILL BE INTRODUCED. KINIT DOES NOT PERFORM ANY FUNCTION THAT IS APPARENT TO THE USER BUT IS ESSENTIAL TO THE OPERATION OF THE ROUTINES. FAILURE TO CALL KINIT WILL RESULT IN THE FATAL DAYFILE DIAGNOSTIC.

STOP KINIT NEVER CALLED.

THESE ARE THE BASIC CONCEPTS AND FUNCTIONS INVOLVED WITH THE HANDLING OF STRINGS. A NUMBER OF OTHER FUNCTIONS AND SUBROUTINES WILL BE INTRODUCED LATER, AND A NUMBER OF OPTIONAL FORMS WILL BE PRESENTED FOR THE ROUTINES ALREADY INTRODUCED, SO AS TO IMPROVE THE UTILITY OF THE CHARACTER STRING HANDLING PACKAGE.
NOTATION USED IN THIS MANUAL

THE FOLLOWING NOTATION IS USED IN DESCRIBING THE SUBROUTINES AND FUNCTIONS IN THIS MANUAL:

WHEN DESCRIBING ARGUMENT LISTS FOR STRING ROUTINES, IF AN ITEM IS ENCLOSED IN ANGLE BRACKETS, FOR INSTANCE,

<INTEGER>

THEN IT IS A DESCRIPTION OF THE TYPE OF ENTRY REQUIRED. FOR EXAMPLE, IF WE WERE TO DESCRIBE THE SYNTAX OF AN ORDINARY FORTRAN WRITE STATEMENT, WE MIGHT WRITE:

WRITE (<UNIT NO.>,<FORMAT LABEL>) <VARIABLE LIST>

OFTEN ARGUMENTS ARE OPTIONAL, AND MAY BE OMITTED IF DESIRED. AN ITEM CONTAINED IN SQUARE BRACKETS IS OPTIONAL. TO EXPAND OUR EXAMPLE FOR THE FORTRAN WRITE STATEMENT, RECALL THAT THE <FORMAT LABEL> MAY BE OMITTED FOR UNFORMATTED WRITES, AND THAT WE NEED NOT HAVE ANY <VARIABLE LIST>. THEREFORE WE COULD IMPROVE OUR DEFINITION TO READ:

WRITE(<UNIT NO.> [<FORMAT LABEL>]) (<VARIABLE LIST>)

THREE DOTS (...) IS USED TO INDICATE AN ELEMENT WHICH MAY BE REPEATED AN ARBITRARY NUMBER OF TIMES. WE COULD REPLACE <VARIABLE LIST> IN OUR DEFINITION OF FORTRAN WRITE WITH <VARIABLE>, (<VARIABLE>...) TO SHOW ONE POSSIBLE FORM OF <VARIABLE LIST> AS AN ARBITRARY NUMBER OF VARIABLES SEPARATED BY COMMAS.
GENERAL RULES

It is extremely important that variables used with the string functions and subroutines be of the right type. All string representations must be assigned to an integer variable. Assigning a string representation to any other type of variable will result in the fatal dayfile diagnostic.

STOP ILLEGAL STRING REPRESENTATION

Being produced when the variable is subsequently referenced in a string function or subroutine. As a memory aid, remember that all functions which return a string representation as the value of the function begin with the letter K. The only function in this manual returning a real result is RLVAL. The only function returning a logical value is LEXIC. All other functions return integer values.

This rule is so important it bears repeating...

Any <string representation> can only be assigned to a variable that is type integer.

For all routines in this manual, unless otherwise specified, all arguments must be type integer.

ARRAYS OF STRINGS

We may use arrays of character strings just as we may use arrays of numeric variables. Each element of the array can represent a string of up to 50 characters. For example:

```
DIMENSION IST(5)
***
DO 100 I=1,5
100 IST(I)=KIN(5)
```

This reads five cards and assigns each card-image-string to a separate element of array IST. Of course, IST must be an integer array.

PASSING STRINGS TO SUBROUTINES

String representations may be passed to other routines just like any other variable. They may appear in common or as arguments in a subroutine or function call. Attempting to perform any operations on string representations other than those provided by routines in this manual will result in undefined results. For instance, trying to print a string representation using an ordinary print or write statement will
PRODUCE GARBAGE, NOT THE STRING DESIRED. THE ONLY LEGAL
ARITHMETIC OPERATIONS WHICH MAY BE PERFORMED ON STRING
REPRESENTATIONS ARE:

1. SIMPLE ASSIGNMENT STATEMENTS, FOR EXAMPLE,

   \texttt{KSTR=JSTR}

2. ASSIGNING THE VALUE 0 TO A STRING REPRESENTATION, WHICH
   DEFINES A NULL STRING, FOR EXAMPLE,

   \texttt{KSTR=0}

SETS KSTR TO THE REPRESENTATION OF A NULL STRING (0 CHARACTERS).

THERE ARE TWO KEYS TO SUCCESS IN USING THESE ROUTINES...

1. REMEMBER TO CALL KINIT BEFORE ANY OTHER STRING ROUTINE.
2. USE THE RIGHT TYPE OF VARIABLE.

KEEPING IN MIND THESE IMPORTANT GENERAL RULES, LET US
EXAMINE THE STRING ROUTINES IN MORE DETAIL.
INTEGER FUNCTION KON — DEFINE A STRING CONSTANT.

LEGAL FORMATS...

<STRING REP.> = KON(<LITERAL STRING>)          (1.1)
<STRING REP.> = KON(<LITERAL STRING>,<LENGTH>) (1.2)
<STRING REP.> = KON(<LITERAL STRING>,<PADDING>) (1.3)

ARGUMENTS...

<LITERAL STRING> IS EITHER A CHARACTER STRING ENCLOSED IN QUOTATION MARKS, OR A HOLLERITH CONSTANT OF THE FORM, <NO.CHAR.><TEXT>. UNDE<EMPH> NO CIRCUMSTANCES WHATSOEVER CAN THIS ARGUMENT BE A VARIABLE.

(NOTE... ON SOME PRINTERS AND KEYPUNCHES, THE QUOTE (") APPEARS AS A "NOT_EQUAL" SYMBOL. THE CORRECT KEYPUNCH FOR THE QUOTE IS AN 8-4 PUNCH ON AN 126 KEYPUNCH, AND AN 8-7 PUNCH ON AN 029 KEYPUNCH.)

<LENGTH> IS AN OPTIONAL PARAMETER SPECIFYING THE COMPLETE LENGTH DESIRED FOR THE STRING. IT MAY BE A VARIABLE. IF <LENGTH> IS GREATER THAN THE LENGTH OF THE <LITERAL STRING>, ADDITIONAL BLANKS WILL BE ADDED AS REQUIRED. IF <LENGTH> IS SHORTER THAN <ACTUAL STRING>, THE STRING WILL BE TRUNCATED.

<PADDING> IS AN OPTIONAL PARAMETER DISTINGUISHED FROM <LENGTH> BY THE FACT THAT IT IS PRECEDED BY A MINUS SIGN. <PADDING> IS A NUMBER SPECIFYING THE NUMBER OF BLANKS TO BE APPENDED TO THE END OF THE <LITERAL STRING>.

IMPORTANT NOTES...

DUE TO THE INTERNAL DATA REPRESENTATION OF BLANKS IN CDC FORTRAN EXTENDED, IT IS IMPOSSIBLE TO DISTINGUISH THE NUMBER OF TRAILING BLANKS ON A <LITERAL STRING>. FOR INSTANCE, "HI " AND "HI " BOTH HAVE PRECISELY THE SAME REPRESENTATION IN CDC FORTRAN. FOR THIS REASON, IT IS NECESSARY FOR FUNCTION KON TO IMPLEMENT THE FOLLOWING RULE TO AVOID AMBIGUITY...

WHEN USING FORM (1.1), ANY TRAILING BLANKS IN <LITERAL STRING> WILL BE AUTOMATICALLY REMOVED BY KON. IF <LITERAL STRING> IS COMPOSED SOLELY OF BLANKS, THE RESULT WILL BE A SINGLE BLANK. FROM THIS RULE, IT IS APPARENT THAT KON("HI") AND KON("HI ") BOTH DEFINE THE SAME STRING, NAMELY THE TWO CHARACTER STRING, "HI ". IF IT IS DESIRED TO HAVE ONE OR MORE TRAILING BLANKS INCLUDED IN THE STRING, THEN USE FORM (1.2) OR FORM (1.3) INSTEAD.

FOR EXAMPLE, KON(" HI ", -1) WILL PAD THE <LITERAL STRING> WITH ONE TRAILING BLANK, RESULTING IN " HI ". THE CHARACTERS IN THE <LITERAL STRING> MAY BE ANY CHARACTERS IN THE 64 CHARACTER ASCII SUBSET WITH THE EXCEPTIONS NOTED BELOW.
1. THE COLON (:) MAY NOT BE THE LAST CHARACTER OF A STRING.
2. TWO COLONS MAY NEVER APPEAR TOGETHER. AT LEAST ONE OTHER CHARACTER MUST INTERVENE.
3. THE QUOTE CHARACTER ( OR NOT-EQUALS SIGN ON SOME PRINTERS AND PUNCHES ) SHOULD NOT BE USED IN THE FORM OF <LITERAL STRING> WHICH IS DELIMITED BY THE QUOTE MARKS. IT MAY BE USED FREELY IN THE HolERITH-COUNT FORM.
   (EXCEPTION... THE CHARACTER " MAY BE SUCCESSFULLY REPRESENTED BY REPLACING IT WITH "" IN A STRING DELIMITED BY "" CHARACTERS).
   THESE RULES STEM FROM THE CDC IMPLEMENTATION OF RECORDS, NOT FROM ANY IDIOSYNCRASIES OF KON (SEE REFERENCES (3), (4), AND (5)).

EXAMPLES OF LEGAL STRING DEFINITIONS...
FUNCTION CALL... RESULT REPRESENTS...
---------------------------
IA=KON("THIS IS A STRING") "THIS IS A STRING"
IB=KON("THIS IS A STRING") "THIS IS A STRING"
IC=KON("SO IS THIS "") "SO IS THIS"
ID=KON("SO IS THIS") "SO IS THIS"
IE=KON("PANGED STRING",3) "PANGED STRING"
IG=KON("PANGED STRING ",-3) "PANGED STRING"
IH=KON("TRUNCATED STRING",0) "TRUNCATED"
II=KON(" ") " 
IJ=KON("SAY "HI") "SAY "HI"
IK=KON("SAY "HI","-1") "SAY "HI"
IL=KON("LENGTHENED STRING","20") "LENGTHENED STRING"

EXAMPLES OF ILLEGAL STRING DEFINITIONS...
FUNCTION CALL... REASON FOR ERROR
---------------------------
DATA LS /3HII /
IP=KON(LS) "<LITERAL STRING> MAY NOT BE VAR.
ST-ING=KON("HELLO") "<STRING REP> MAY NOT BE REAL VAR.
IT=KON("ANS:WER:" ) "COLON MAY NOT BE LAST CHARACTER.

12
SUBROUTINE KONLST - DEFINE ARRAY OF STRING CONSTANTS

LEGAL FORMAT...

CALL KONLST(<INT. ARRAY>,<LIT. STRING>[,<LIT. STRING>...]) [21]

ARGUMENTS...

<INT. ARRAY> IS AN INTEGER ARRAY TO RECEIVE THE STRING REPRESENTATIONS.
<LIT. STRING> IS A LITERAL STRING AS DESCRIBED FOR FUNCTION KON. IT MAY NOT BE A VARIABLE.

ANY NUMBER OF <LITERAL STRING>S MAY BE SPECIFIED, SEPARATED BY COMMAS, AND USING CONTINUATION CARDS IF NEEDED.

NOTE... IF THIS FUNCTION OR OTHER FUNCTIONS WITH VARIABLE-LENGTH ARGUMENT LISTS IS CALLED SEVERAL TIMES WITH DIFFERENT NUMBERS OF ARGUMENTS (IN THE SAME PROGRAM), THE FORTRAN COMPILER WILL PRODUCE THE INFORMATIVE DIAGNOSTIC, "ARGUMENT COUNT INCONSISTENT WITH PRIOR USAGE". THIS DIAGNOSTIC SHOULD BE IGNORED.

EXAMPLE 2....

DIMENSION ISTRG(*)
CALL KINIT( )
***
CALL KONLST(ISTRG,"MOVE","DRAW","DASH","TEXT","ERASE")

THE ABOVE EXAMPLE IS FUNCTIONALLY EQUIVALENT TO...

DIMENSION ISTRG(*)
CALL KINIT( )
***
ISTRG(1)=KON("MOVE")
ISTRG(2)=KON("DRAW")
ISTRG(3)=KON("DASH")
ISTRG(4)=KON("TEXT")
ISTRG(5)=KON("ERASE")

EXAMPLE 2...*

COMMON /STRINGS/ JS(50)
CALL KINIT( )
***
CALL KONLST(JS(10),"+","-","#","/"")

THIS EXAMPLE DEFINES ONLY JS(10) THROUGH JS(13) INCLUSIVE. THE REMAINING ELEMENTS OF THE ARRAY ARE NOT AFFECTED.
INTEGER FUNCTION KAT - CONCATENATE STRINGS

LEGAL FORMAT...

<STRREP>=KAT(<STRREP>,<STRREP>,(*<STRREP>)) (13.1)

ARGUMENTS...

<STRREP> IS A STRING REPRESENTATION.

ALL OF THE STRINGS REPRESENTED BY THE ARGUMENTS ARE CONCATENATED IN THE SEQUENCE GIVEN, AND THE RESULTING REPRESENTATION RETURNED AS THE VALUE OF THE FUNCTION. ANY NUMBER OF ARGUMENTS (GREATER THAN 1) MAY BE USED, USING CONTINUATION CARDS IF NEEDED.

EXAMPLE 3:

DIMENSION LS(4)
CALL KINIT()
***
CALL KONLST(LS," IS THE TIME"," AID OF"," PARTY"," NOW")
JS=KON(" FOR ALL GOOD MEN TO COME TO ")
NEWSTR=KAT(LS(4)*LS(1)*JS*KON(" TH=")*LS(2)*LS(3))

NEWSTR WILL THEN REPRESENT THE STRING,

" NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF PARTY"
INTEGER FUNCTION KSUB - SUBSTRING EXTRACTION

LEGAL FORMATS...

<STR.REP.> = KSUB(<STR.REP.>,<START.COL.>)  (4.1)
<STR.REP.> = KSUB(<STR.REP.>,<START.COL.>,<NO.CHAR.>)  (4.2)

ARGUMENTS...

<STR.REP.> IS A STRING REPRESENTATION.
<START.COL.> IS THE STARTING COLUMN FOR THE SUBSTRING EXTRACTION
WITHIN THE STRING.
<NO.CHAR.> IS THE NUMBER OF CHARACTERS TO BE EXTRACTED.

IF FORM (4.1) IS USED, THE ENTIRE REMAINDER OF THE STRING
FROM CHARACTER NUMBER <START.COL.> WILL BE EXTRACTED.
IF FORM (4.2) IS USED, THEN ONLY THE SPECIFIED NUMBER OF
CHARACTERS WILL BE EXTRACTED.

EXAMPLE 4...

CALL KINIT()
...
JST = KON("MISO/SEA0, BLDG 353")
IBLOG = KSUB(JST, 2)

THE VARIABLE IBLOG WILL REPRESENT "BLDG 353".

EXAMPLE 4...

DIMENSION IALPHA(26)
CALL KINIT()
...
ISTR = KON("ABCD:FGHIJKLMNOPQRSTUVWXYZ")
DO 10 I=1,26
  10 IALPHA(I) = KSUB(ISTR, I+1)

AFTER THE ABOVE CODE IS EXECUTED, IALPHA(1) WILL REPRESENT "A",
IALPHA(10) WILL REPRESENT "J", ETC.
INTEGER FUNCTION LEN - LENGTH OF STRING

LEGAL FORMAT...

<INTEGER RESULT>=LEN(<STRING REP.>)

ARGUMENTS...

<STRING REP.> IS A STRING REPRESENTATION.

FUNCTION LEN RETURNS THE LENGTH OF THE STRING REPRESENTED BY THE ARGUMENT.

EXAMPLE 5:....

CALL KINIT(""

***
KST=KON(" ALPHA")
LSTR=YAT(KSTR*KON("BETA"))
N=L N(LSTR)

N WILL BE 10 AFTER EXECUTION OF THIS CODE.
LOGICAL FUNCTION LEXIC - COMPARE STRINGS

LEGAL FORMAT...
<.TRUE./.FALSE.>=LEXIC(<STRING REP.>,<KEY>,<STRING REP.>) (6.1)

ARGUMENTS...

<STRING REP.> IS A STRING REPRESENTATION.
<KEY> IS A LITERAL KEYWORD INDICATING THE OPERATOR TO BE USED
FOR THE COMPARISON. <KEY> MAY ONLY BE ONE OF THE FOLLOWING:
"LT" "LE" "EQ" "NE" "GE" "GT"

THE KEYWORDS HAVE THE SAME MEANING AS THEIR REGULAR FORTRAN
COUNTERPARTS.

FUNCTION LEXIC RETURNS THE LOGICAL VALUE .TRUE. OR .FALSE.
THE STRINGS REPRESENTED BY THE FIRST AND THIRD ARGUMENTS ARE
COMPA RED LEXICALLY. IF THE CONDITION INDICATED BY <KEY> IS MET,
THEN LEXIC RETURNS .TRUE. OTHERWISE,.FALSE. IS RETURNED.
THE STRINGS NEED NOT BE OF THE SAME LENGTH TO BE COMPARED.
THE STRINGS ARE COMPARED FOR DICTIONARY-ORDER USING THE
DISPLAY CODE COLLATING-SEQUENCE OF THE CDC COMPUTER.
IN ORDER FOR TWO STRINGS TO BE CONSIDERED EQUAL, THEY MUST
HAVE THE SAME LENGTH AND COMPOSITION. THUS "HI" AND "HI" ARE
NOT EQUAL. IN FACT, "HI" WILL ALWAYS PRECEDE "HI" IN THE
COLLATING PROCESS, SO "HI" IS LESS THAN "HI".
IT IS IMPORTANT TO REALIZE THAT YOU MAY NOT COMPARE TWO
STINGS WITHOUT USING FUNCTION LEXIC. TWO STRINGS WHICH ARE
IDENTICAL IN EVERY RESPECT WILL NOT IN GENERAL HAVE THE SAME
STRING REPRESENTATION. AND IF YOU COMPARE THE REPRESENTATIONS
WITH AN ORDINARY IF STATEMENT WITHOUT LEXIC, THEY WILL
EVALUATE AS UNEQUAL. THE SOLE EXCEPTION IS THE NULL STRING,
WHICH IS ALWAYS REPRESENTED BY THE NUMERIC VALUE 0.

NOTE... WHEN USING FUNCTION LEXIC, IT IS IMPORTANT TO
REMEMBER TO DECLARE LEXIC AS TYPE LOGICAL IN THE CALLING
PROGRAM.

EXAMPLE 6.:....

LOGICAL LEXIC
CALL KINIT()

***
IST =KON("KOWALSKI")
JSTR=KON("KOWAL")
IF(LEXIC(JSTR,"LT",ISTR))GO TO 10
***

CONTROL WILL BE TRANSFERRED TO STATEMENT 10 AT THE IF STATEMENT.
EXAMPLE 6.

PROGRAM SORT(INPUT,OUTPUT,TAPE INPUT,TAPE OUTPUT)
C PROGRAM TO READ 100 CARDS, SORT THEM ALPHABETICALLY, C
C AND PRINT THE SORTED LIST.
LOGICAL LEXIC
DIMENSION NSTR(100)
CALL KINIT(.)
DO 100 I=1,100
100 NST(I)=KIN(5)
DO 300 I=1,99
   IIP1-I+1
   DO 200 J=IIP1,100
      IF(LEXIC(NSTR(J),"GE",NSTR(I)))GO TO 200
      ITEP=NSTR(I)
      NST(I)=NSTR(J)
      NSTR(J)=ITEP
200 CONTINUE
300 CALL KOUT(NSTR(I),1)
   CALL KOUT(NSTR(100),1)
STOP
END

NOTE... THE ABOVE EXAMPLE USES A VERY INEFFICIENT SORTING
TECHNIQUE FOR THE SAKE OF SIMPLICITY. SEE REFERENCE [1] FOR A
DESCRIPTION OF BETTER METHODS.
INTEGER FUNCTION MATCH — FIND SUBSTRING MATCHING PATTERN

L'GAL FORMATS...

<COL.NO.> = MATCH(<S.R.>,<KEY>,<PAT.S.R.>)   \[7.1\]  
<COL.NO.> = MATCH(<S.R.>,<KEY>,<PAT.LIST S.R.>,<NO.PAT.>,<I MATCH>)  \[7.2\]

ARGUMENTS...

<S.R.> IS A STRING REPRESENTATION FOR THE STRING TO BE SEARCHED.
<KEY> IS A KEYWORD INDICATING THE TYPE OF SEARCH TO BE MADE,
SELECTED FROM THE FOLLOWING CHOICES.

"FIRST"    "FIRSTNOT" "LAST"    "LASTNOT"

<PAT.S.R> IS THE STRING REPRESENTATION OF THE PATTERN TO BE SEARCHED FOR.
<PAT.LIST S.R.> IS AN ARRAY OF REPRESENTATIONS OF PATTERNS TO BE SEARCHED FOR.
<NO.PAT> IS THE NUMBER OF ELEMENTS IN THE PATTERN ARRAY.
<I MATCH> IS RETURNED TO THE CALLING PROGRAM AS THE SUBSCRIPT
OF <PAT.LIST S.R> WHICH RESULTED IN A SUCCESSFUL MATCH.

FORM [7.1] IS USED WHEN A SINGLE PATTERN IS TO BE SEARCHED FOR IN THE STRING. FORM [7.2] IS USED WHEN SEVERAL PATTERNS ARE TO BE TRIED IN SUCCESSION. THE KEYWORDS INDICATE THE DIRECTION OF THE SEARCH, AND WHETHER THE NORMAL CONDITION OR INVERSE CONDITION CONSTITUTES A "SUCCESSFUL" MATCH.

"FIRST" SEARCHES LEFT TO RIGHT FOR THE FIRST OCCURRENCE OF ANY OF THE PATTERNS. "FIRSTNOT" SEARCHES FOR THE FIRST CHARACTER WHICH DOES NOT MATCH ANY OF THE PATTERNS.

"LAST" SEARCHES FOR THE LAST OCCURRENCE OF ANY OF THE PATTERNS IN THE STRING. "LASTNOT" SEARCHES FOR THE LAST CHARACTER WHICH DOES NOT MATCH ANY OF THE PATTERNS GIVEN. THESE DISTINCTIONS ARE BEST CLARIFIED BY EXAMPLES.

EXAMPLE 7...

IST = KON("THE VALUE OF X IS 3 IF Y IS 7 AND IF Z IS ",")  
ICOL = MATCH(IST,"FIRST",KON("IF"))  
LCOL = MATCH(IST,"LAST",KON("IF"))

THIS EXAMPLE RESULTS IN ICOL=21 AND LCOL=75.

EXAMPLE 7...

DIMENSION IPAT(4)  
CALL KINIT(4)  
***  
CALL KONLST(IPAT, "+", "-", ",", "/")  
IST = KIN(5)  
ICOLOP = MATCH(ISTR,"FIRST",IPAT,4,I MATCH)
This example reads a card from tapes and searches for the first arithmetic operator. Assuming that the card contained:

"378.45*23.11+3.0"

Then the results would be that ICOLOP=7 and IMATCH=3, because the first column that matched any of the listed patterns was column 7, and the pattern it matched was PAT(3).

Example 7:

\[ \text{ILN} = \text{MATCH}([\text{IST: }] \text{"LASTNOT": \text{KON}([" "])})\]

This example searches a string to find the last non-blank character in the string.

Notes on pattern matching...

If the match fails (that is, the specified pattern matching condition is not met by any of the patterns), the value returned by function MATCH is . When using form 17.21, <1 MATCH> is meaningless when "NOT" keywords are used. Searches using "LASTNOT" are not recommended for patterns longer than one character.
INTEGER FUNCTION KIN - INPUT A STRING

LEGAL FORMATS...

<STRING REP> = KIN(<LFN>)                   [R.1]
<STRING REP> = KIN(<LFN>,<LENGTH>)        [R.2]
<STRING REP> = KIN(<LFN>,<PADDING>)       [R.3]

ARGUMENTS...

<LFN> IS THE LOGICAL FILE NAME. IT MAY BE EITHER A UNIT
NUMBER OR LEFT-JUSTIFIED-WITH-ZERO-FILL FILE NAME.
<LENGTH> IS THE NUMBER OF CHARACTERS TO BE DEFINED FOR THE
RESULT STRING.
<PADDING> IS THE NUMBER OF BLANKS TO BE APPENDED TO THE END OF
THE STRING (DISTINGUISHED FROM <LENGTH> BY THE MINUS SIGN).

IF FORM [R.1] IS USED THEN THE STRING RESULTING WILL
BE THE LENGTH OF THE RECORD READ WITH ALL TRAILING BLANKS
REMOVED. FOR EXAMPLE, IF THE INPUT DEVICE IS AN 80 COLUMN
CARD READER AND THE 10 NON-BLANK COLUMN ON THE CARD READ IS
COLUMN 63, THEN THE STRING REPRESENTED BY <STRING REP> WILL
BE 60 CHARACTERS LONG. IF FORM [R.2] IS USED, THEN THE
STRING WILL BE DEFINED AS <LENGTH> CHARACTERS LONG REGARDLESS
OF THE NUMBER OF CHARACTERS IN THE ACTUAL RECORD. FOR
EITHER FORM, THE MAXIMUM RECORD LENGTH IS LIMITED BY THE
CDC OPERATING SYSTEM TO 150 CHARACTERS.

END-OF-FILE ENCOUNTERED USING EITHER FORM WILL CAUSE KIN
TO RETURN THE NULL STRING. FOR FORM [R.1], AN ENTIRELY BLANK
CARD WILL RETURN A STRING CONSISTING OF A SINGLE BLANK (" ").

EXAMPLE A.  .  .  .

PROGRAM TEXT(INPUT,TAPE=INPUT,OUTPUT)
CALL KINIT(')
***
IST =KIN(5)
***

THIS EXAMPLE READS A CARD FROM THE INPUT FILE. THE LENGTH OF
THE STRING DEPENDS ON THE CONTENTS OF THE RECORD READ.

EXAMPLE B.  .  .  .

PROGRAM TEXT(INPUT,OUTPUT)
CALL KINIT(')
INP=5LINPUT
***
IST:=KIN(INP)
IF(IST.EQ.3)STOP "END-OF-FILE ENCOUNTERED ON INPUT"
***
THIS EXAMPLE ILLUSTRATES HOW A LEFT-JUSTIFIED-WITH-ZERO-FILL FILE NAME MAY BE USED INSTEAD OF A UNIT NUMBER. IT ALSO ILLUSTRATES A METHOD FOR TESTING FOR END-OF-FILE. ALTERNATIVELY, THE USER COULD TEST FOR LEN(ISTR) EQUAL TO 0, WHICH ALSO WOULD BE TRUE WHEN END-OF-FILE WAS ENCOUNTERED.

EXAMPLE:......

PROGRAM TEST3(INPUT,TAPES=INPUT,OUTPUT,TAPE6=OUTPUT,TAPE1)
CALL KINIT
FILE=6,TAPE1
***
ISTR=KIN(FILE,100)
***

REGARDLESS OF THE CONTENT OF THE RECORD READ, LEN(ISTR) WILL BE 100 CHARACTERS. IF THE RECORD READ WAS LONGER THAN 100 CHARACTERS, IT WILL BE TRUNCATED. IF IT WAS SHORTER THAN 100 CHARACTERS, IT WILL BE PADDED WITH BLANKS. THE ONLY EXCEPTION IS THAT THE NULL STRING WILL BE ASSIGNED TO ISTR IF END-OF-FILE IS ENCOUNTERED ON TAPE1.
SUBROUTINE KOUT - OUTPUT A STRING

LEGAL FORMATS...

CALL KOUT(<LFN>,<STRING REP.>)
CALL KOUT(<LFN>,<STRING REP.>,<CARRIAGE CON.>)

ARGUMENTS...

<LFN> IS THE LOGICAL FILE NAME EXPRESSED EITHER AS AN INTEGER NUMBER, OR AS A LEFT-JUSTIFIED-WITH-ZERO-FILL FILE NAME. 
STRING REP. IS THE REPRESENTATION OF THE STRING TO BE OUTPUT.
CARRIAGE CON. IS A CARRIAGE CONTROL INDICATOR, CHOSEN FROM THE LIST BELOW.

<table>
<thead>
<tr>
<th>&lt;CARRIAGE CON.&gt;</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SINGLE SPACE  (NORMAL PRINTING)</td>
</tr>
<tr>
<td>2</td>
<td>DOUBLE SPACE  (SKIP 1 LINE)</td>
</tr>
<tr>
<td>3</td>
<td>OVERPRINT PREVIOUS LINE</td>
</tr>
<tr>
<td>-1</td>
<td>SKIP TO TOP OF NEXT PAGE (FORM-FEED)</td>
</tr>
<tr>
<td>(OMITTED)</td>
<td>NO CARRIAGE CONTROL (1ST CHAR. USED FOR CARRIAGE CONTROL, IF APPLICABLE TO PHYSICAL DEVICE).</td>
</tr>
</tbody>
</table>

EXAMPLE ... 

PROGRAM ONE (OUTPUT,TAPE6=OUTPUT)
CALL KINIT()
***
IST =KON("THIS IS A STRING")
CALL KOUT(6,IST,1)
***

THIS EXAMPLE WILL PRINT "THIS IS A STRING" ON THE NEXT LINE OF THE OUTPUT DEVICE.
INTEGER FUNCTION KINTEK - INPUT FROM TEKTRONIX TERMINAL

LEGAL FORMAT...

<STRING> = KINTEK(<LENGTH>)

(10.1)

ARGUMENTS...

<LENGTH> IS THE DESIRED LENGTH OF STRING TO BE INPUT. IF SPECIFIED AS ', THEN THE RESULTING STRING LENGTH WILL BE THE LENGTH OF THE INPUT RECORD AFTER ALL TRAILING BLANKS ARE REMOVED. THE MAXIMUM ALLOWABLE <LENGTH> IS 133, LIMITED BY THE TEKTRONIX SOFTWARE (TCS PLOT-10).

THIS FUNCTION IS SIMILAR TO KTN, EXCEPT IT IS COMPATIBLE WITH THE TCS SOFTWARE AND MAY BE USED ANYWHERE TEKTRONIX USERS WOULD ORDINARILY CALL SUBROUTINE AINST. NOTE THAT <LENGTH> IS A REQUIRED PARAMETER.

EXAMPLE 10.1...

PROGRAM T001(TAPE61,TAPE62)
CALL INITT(300)
CALL KINIT(')

... CALL MOVABS(0,200)
ISTR = KINTEK(':')
...

THIS EXAMPLE MOVES THE CURSOR TO COORDINATES (0,200) AND WAITS FOR THE TERMINAL USER TO ENTER A LINE FROM THE KEYBOARD. THE PROGRAM CONVERTS THE INPUT LINE TO A STRING REPRESENTATION, ISTR.

NOTE THAT THIS ROUTINE REQUIRES THE USE OF THE TEKTRONIX TCS SOFTWARE, SUCH AS THE TEK30 OR TEK48 LIBRARIES. TCS IS A REGISTERED TRADEMARK OF TEKTRONIX, INC.
SUBROUTINE KOUTEK - OUTPUT TO TEKTRONIX

LEGAL FORMATS...

CALL KOUTEK(<STRING REP.>)
CALL KOUTEK(<STRING REP.>,<CARRIAGE CON.>)

ARGUMENTS...

<STRING REP.> IS THE REPRESENTATION OF THE STRING TO BE OUTPUT.
<CARRIAGE CON.> IS CARRIAGE CONTROL CODE CHOOSEN FROM THE
LIST BELOW:

<CARRIAGE CON.> MEANING...

-----------
1           NEW LINE, THEN OUTPUT
2           NEW LINE, NEW LINE, OUTPUT
OMITTED    OUTPUT AT PRESENT CURSOR POSITION

THIS SUBROUTINE IS PROVIDED TO GIVE COMPATIBILITY WITH
TEKTRONIX TERMINAL USERS USING TCS (PLOT-10) SOFTWARE.
IT IS FUNCTIONALLY EQUIVALENT TO STRING FUNCTION KOUT, BUT MAY BE
USED ANYWHERE TCS SUBROUTINE KOUSTW WOULD NORMALLY BE USED.

NOTE THAT THIS ROUTINE REQUIRES THE TCS SOFTWARE SUCH
AS IS CURRENTLY AVAILABLE ON THE TEK30 OR TEK48 LIBRARY.
INTEGER FUNCTION INTVAL - STRING TO INTEGER CONVERSION

LEGAL FORMATS...

\[ \langle \text{INTEGER VAL.} \rangle = \text{INTVAL}(<\text{STRING REP.}>)) \] \hspace{1cm} (12.1)

\[ \langle \text{INTEGER VAL.} \rangle = \text{INTVAL}(<\text{STRING REP.}>;<\text{IF ERROR}>)) \] \hspace{1cm} (12.2)

ARGUMENTS...

\(<\text{STRING REP.}>\) IS A REPRESENTATION OF THE STRING TO BE CONVERTED.
\(<\text{IF ERROR}>\) IS AN INTEGER VARIABLE PREVIOUSLY ASSIGNED TO A VALID STATEMENT NUMBER IN THE CALLING PROGRAM BY A FORTRAN ASSIGN STATEMENT.

INTVAL PERFORMS STRING TO INTEGER CONVERSION. IN NORMAL FORTRAN APPLICATION PROGRAMS, USERS ARE OFTEN UNAWARE THAT SUCH CONVERSIONS TAKE PLACE, BECAUSE THE READ AND FORMAT STATEMENTS PERFORM THE CONVERSION AUTOMATICALLY. FOR EXAMPLE,

\[
\text{READ}(5,10) \text{IVAL}
\]

100 \text{ FORMAT}(15)

CAUSES THE COMPUTER TO READ A CHARACTER STRING FROM UNIT 5, AND CONVERT THE FIRST 5 CHARACTERS TO THE NUMERIC VALUE THEY REPRESENT. TO ACCOMPLISH THE SAME OPERATION USING THE CHARACTER STRING ROUTINES, WE MIGHT WRITE,

\[
\text{IVAL = INTVAL(KIN(5)))}
\]

THE PRIMARY DIFFERENCE IN THESE TWO EXAMPLES IS THAT THE NUMBER COULD BE PUNCHED ANYWHERE ON THE CARD FOR THE SECOND CASE, AND COULD BE WRITTEN WITH OR WITHOUT A DECIMAL POINT, AS WE SHALL PRESENTLY SEE.

CONVERSION RULES...

1. THE STRING TO BE CONVERTED MAY CONTAIN ANY LEGAL REPRESENTATION OF A NUMBER. IT MAY BE WRITTEN WITH OR WITHOUT A DECIMAL POINT, SIGNED OR UNSIGNED, AND MAY EVEN HAVE AN EXPONENT OF 10 SPECIFIED IN THE USUAL FORTRAN REPRESENTATION (E.G., "1.23E4").
2. THE STRING MAY HAVE ANY NUMBER OF LEADING BLANKS, WHICH ARE IGNORED.
3. CONVERSION IS CONSIDERED FINISHED WHEN A TRAILING BLANK IS ENCOUNTERED OR THE STRING IS EXHAUSTED, WHICHEVER COMES FIRST.
4. IF AN ILLEGAL CHARACTER IS ENCOUNTERED IN THE STRING, CONVERSION IS STOPPED AND ONE OF THE FOLLOWING OCCURS...

A. IF \(<\text{IF ERROR}>\) WAS NOT SPECIFIED ON THE CALL, THEN A FATAL DIAGNOSTIC ERROR MESSAGE IS ISSUED AND THE PROGRAM STOPS.
B. IF \(<\text{IF ERROR}>\) WAS SPECIFIED, THEN CONTROL IS PASSED DIRECTLY TO THE DESIGNATED STATEMENT NUMBER IN THE CALLING PROGRAM. THIS FEATURE ALLOWS THE USER COMPLETE CONTROL OVER
RECOVERY FROM CONVERSION ERRORS.
5. AN ENTIRELY BLANK STRING RETURNS 0 WITHOUT ERROR.

EXAMPLES OF STRINGS TO BE CONVERTED AND THE RESULT...

IF THE STRING IS... THEN INTVAL RETURNS...

```
"23"  23
"-23" -23
"0 ORANGE S" 0
"-200.0" (ERROR)
"85*63" 854
"-233e+3" 233
"*75" 0
"1"  1
"23*251" (ERROR)
```

EXAMPLE 12.1...

```
PROGRAM TEST(INPUT,OUTPUT,TAPES=INPUT,TAPES=OUTPUT)
CALL KINIT()
*** READ THE VALUE OF NRUN FROM ANYWHERE ON THE CARD... NRUN=INTVAL(KIN())
IF(NRUN.EQ.0) STOP
***

THIS EXAMPLE SHOWS HOW AN INTEGER VALUE MAY BE READ FROM
ANYWHERE ON A CARD. FIRST, KIN() INPUTS A STRING. THE
STRING IS THEN CONVERTED INTO ITS NUMERIC VALUE BY INTVAL.
SHOULD THERE BE INFORMATION ON THE CARD OTHER THAN A NUMBER,
THE PROGRAM WILL STOP AND A FATAL ERROR MESSAGE ISSUED.
```

EXAMPLE 12.2...

```
PROGRAM TEST2(INPUT,OUTPUT,TAPES=INPUT,TAPES=OUTPUT)
CALL CONNEC(S)
CALL CONNEC(S)
CALL KINIT()
***
ASSIGN 10 TO IERRED
***
101 CONTINUE
C READ USERS TYPED VALUE OF NRUN...
NRUN=INTVAL(KIN(),IERRED)
IF(NRUN.EQ.0) STOP
***
10 CONTINUE
C COME HERE ONLY ON CONVERSION ERROR.
CALL KOUT6(KOH("BAD DATA, TRY AGAIN"),1)
GO TO 101
***
```
THIS PROGRAM SEGMENT IS SIMILAR TO EXAMPLE 12.1 EXCEPT THAT THE PROGRAM IS INTERACTIVE, AND THEREFORE THE PROGRAM SHOULD GIVE THE USER AN OPPORTUNITY TO MEND HIS WAYS IF HE MAKES A TYPO ERROR WHILE ENTERING THE VALUE FOR NRUN AT THE KEYBOARD. SHOULD INTVAL DETECT ILLEGAL INPUT, IT WILL NOT RETURN VIA THE NORMAL PATH, BUT WILL INSTEAD BRANCH DIRECTLY TO STATEMENT 10. THUS IF THE USER TYPED "12", INTVAL WOULD RETURN NORMALLY, BUT IF HE TYPED "1K", INTVAL WOULD RETURN TO STATEMENT 10.
REAL FUNCTION RLVAL - STRING TO REAL CONVERSION

LEGAL FORMATS...

<REAL VAL.> = RLVAL(<STRING REP.>)  \hspace{1cm} (13.1)
<REAL VAL.> = RLVAL(<STRING REP.>,<IF ERROR>)  \hspace{1cm} (13.2)

ARGUMENTS...

<STRING REP.> IS THE REPRESENTATION OF THE STRING TO BE CONVERTED.
<IF ERROR> IS AN INTEGER VARIABLE PREVIOUSLY ASSIGNED TO A
VALID STATEMENT NUMBER IN THE CALLING PROGRAM BY
A FORTRAN ASSIGN STATEMENT.

THE RULES OF CONVERSION GIVEN FOR INTVAL APPLY ALSO TO
RLVAL, WITH THE EXCEPTION THAT THE FRACTIONAL PART OF NUMBERS
IS RETAINED SINCE RLVAL RETURNS A REAL VALUE. THE
STRING TO BE CONVERTED MAY STILL BE WRITTEN WITH OR WITHOUT A
DECIMAL POINT.

EXAMPLE 13.1...

IST =KON("1.3")
JST =KON("1.361")
X=RLVAL(IST)
***
ASSIGN 20 TO I-ERP
Y=RLVAL(KAT(IST,JST),I-ERP)
***
20 CONTINUE
PHI T I O
106 FORMAT(* ILLEGAL VALUE FOR Y*)
***

IN THIS EXAMPLE, X WILL BE 13.0 AND Y WILL BE 13.361.
INTEGER FUNCTION KSR - NUMERIC TO STRING CONVERSION

LEGAL FORMATS...

<STRING REPR> = KSR(<VALUE>)  \[14.1\]
<STRING REPR> = KSR(<VALUE>,<FTN.FMT.>)  \[14.2\]

ARGUMENTS...

<VALUE> IS A REAL OR AN INTEGER VALUE TO BE CONVERTED.
<FTN.FMT.> IS A LITERAL STRING CONTAINING A LEGAL FORTRAN FORMAT TO BE USED TO GOVERN THE CONVERSION.

KSR IS THE INVERSE FUNCTION OF FUNCTIONS INTVAL AND RLVAL.
IT ACCEPTS AS INPUT THE NUMERIC VALUE OF A REAL OR INTEGER NUMBER AND RETURNS THE STRING REPRESENTATION OF THE NUMBER.
IN AN ORDINARY FORTRAN WRITE STATEMENT, THIS CONVERSION IS PERFORMED AUTOMATICALLY BY THE FORMAT STATEMENT. FOR EXAMPLE,

```
WRITE(6,200) X
200 FORMAT(F10.4)
```

THIS CODE CAUSES THE COMPUTER TO TAKE THE NUMERIC VALUE OF X AND CONVERT IT TO A STRING WITH 10 CHARACTERS REPRESENTING THE VALUE OF X AND THEN OUTPUT IT TO THE DEVICE. TO ACCOMPLISH THE SAME THING USING THE STRING ROUTINES, WE COULD WRITE,

```
CALL KOUT(6,KS"(X,"(F10.4))")
```

THIS WILL CAUSE EXACTLY THE SAME RESULT AS THE WRITE STATEMENT.
THE MORE INTERESTING CASE THOUGH, IS THE CASE WHERE KSR IS USED WITHOUT FORMAT SPECIFICATION, AS SHOWN IN FORM [14.1].
FOR THIS CASE, KSR WILL AUTOMATICALLY SELECT A FORMAT WHICH WILL EXACTLY FIT THE MAGNITUDE OF THE NUMBER, REGARDLESS OF ITS VALUE.

CONVERSION RULES FOR KSR...

A. IF <FTN.FMT> IS NOT SPECIFIED...

1. IF <VALUE> IS AN INTEGER NUMBER, THEN KSR WILL SELECT AN I-TYPE FORMAT WITH THE SMALLEST NUMBER OF CHARACTERS POSSIBLE TO REPRESENT THE NUMBER.

2. IF <VALUE> IS A REAL NUMBER, KSR WILL SELECT A REAL FORMAT REPRESENTATION AS FOLLOWS: IF <VALUE> IS BETWEEN -1.0E-8 AND +1.0E9, AN F-TYPE SPECIFICATION WILL BE USED WITH ALL LEADING BLANKS REMOVED AND ALL TRAILING ZEROS SUPPRESSED FROM THE NUMBER. NUMBERS OUTSIDE THIS RANGE WILL BE CONVERTED USING IPE11.S FOR POSITIVE VALUES AND IPE12.S FOR NEGATIVE VALUES. THE ONLY EXCEPTION IS THE EXACT VALUE 0.0, WHICH WILL BE CONVERTED TO "0", SINCE THERE IS NO DISTINCTION BETWEEN A REAL 0.0 AND AN INTEGER 0 IN THE CDC REPRESENTATION OF VALUES.
3. IF <FMT> IS SPECIFIED...

1. Any legal format may be used which does not cause the system maximum record length of 150 characters to be exceeded.

2. Any trailing blanks generated by the format will be ignored, for reasons explained in the discussion of function KUN.

3. The format need not agree in type with <VALUE>.
For example if a <VALUE> of 13.5 is specified with a format of "(1"), KSP will automatically convert the value to integer before conversion, so the string result would be "13".

Example 1-1...

PROGRAM TEST OUTPUT TAPED = OUTPUT
CALL KINIT()
***
N=1
***
N=10000
***
WRITE (6,20)n
20 FOR AT(*) THERE WERE *17 * OCCURRENCES.*
***
IST=KAT(KON("THERE WERE",-1),KSR(N)*KON(" OCCURRENCES."))
CALL KOUT(3,ISTF,1)
***

The above example illustrates the differences between using an ordinary Fortran WRITE and using a free-format conversion with function KSR. Assuming N can range in value from 0 to 1000000 depending on the situation, the programmer specified an I7 conversion for N, which is the minimum size which will fit any contingency. Now suppose N was 3. The FORTRAN WRITE/FORMAT combination would produce:

THERE WERE >3 OCCURRENCES.

The KSR conversion would produce:

THERE WERE 3 OCCURRENCES.

Which is much more readable.

The dynamic formatting capability illustrated above is very useful in preparing attractive output, especially in interactive applications. Floating-point numbers may also be used.
EXAMPLE 14.2

PROGRAM TEST(OUTPUT,TAPE=OUTPUT)
CALL KINIT()
***
XVAL =49.4
***
ISLASH=KUN("//")
IST=KAT(ISLAS..,KSP(XVAL),ISLASH)
CALL KOUT(6,IST*1)
***

THE PRINTED OUTPUT WOULD BE:

/ / 12.25/ / 

THIS EXAMPLE ILLUSTRATES THE ABILITY OF KSP TO SELECT SUITABLE FORMATS FOR REAL NUMBERS.

EXAMPLE 14.3

PROGRAM TEST(OUTPUT,TAPE=OUTPUT)
CALL KINIT()
***
X=7.0/0
***
CALL KOUT(6,KG,*'(X"(F1.5)")'))

THIS EXAMPLE ILLUSTRATES THE USE OF FORM (14.2) AND THE RESULTING PRINTOUT WOULD READ:

37.500000
SUBROUTINE KREPL - REPLACE STRING WITH STRING

LEGAL FORMAT...

CALL KREPL(<STRING REP.>,<STRING REP.>)

ARGUMENTS...

<STRING REP.> IS A STRING REPRESENTATION

KREPL REPLACES THE FIRST NAMED STRING WITH THE SECOND STRING.

FOR EXAMPLE:

CALL KREPL(IST,KON(" NEW"))

CAUSES THE VARIABLE IST TO BE ASSIGNED THE REPRESENTATION
OF " NEW". THIS SEEMS EQUIVALENT TO:

IST=KON(" NEW")

SO IT SEEMS REASONABLE TO QUESTION THE UTILITY OF SUBROUTINE KREPL. LATER WE SHALL SEE THAT KREPL IS MORE EFFICIENT IN MEMORY USAGE THAN USING THE "=" ASSIGNMENT FORM. IT IS IMPORTANT TO USE KREPL INSTEAD OF SIMPLE ASSIGNMENT STATEMENT ANY TIME THE ASSIGNMENT IS IN A LOOP WHERE THE VARIABLE IS TO BE "RE-USED" OVER AND OVER. OTHERWISE, THE SIMPLE ASSIGNMENT STATEMENT IS ACCEPTABLE.

NOTE THAT THE VARIABLE TO BE REPLACED MUST BE A LEGAL STRING REPRESENTATION. IT MAY, OF COURSE, BE THE NULL STRING.

EXAMPLE 15.1...
SUBROUTINE KINIT - INITIALIZE CHARACTER STRING ROUTINES

LEGAL FORMAT...

CALL KINIT(<BUF_SIZE>)

ARGUMENTS...

<BUF_SIZE> IS THE DIMENSION OF THE MEMORY BUFFER TO BE USED FOR STORAGE OF ACTUAL-STRINGS. IF SPECIFIED AS 0, THE DEFAULT SIZE OF 512 WILL BE USED.

KINIT INITIALIZES THE COMMON VARIABLES SHARED AMONG THE VARIOUS FUNCTIONS AND SUBROUTINES IN THE CHARACTER STRING PACKAGE. NORMALLY, THE FORM OF THE CALL IS:

CALL KINIT()

OPTIONAL VALUES FOR <BUF_SIZE> WILL BE INTRODUCED IN THE NEXT SECTION.
DEFINITION OF <STRING REPRESENTATION>

UNTIL NOW, WE HAVE BEEN USING THE NOTION OF <STRING REP.> WITHOUT KNOWING WHAT IT IS. A <STRING REPRESENTATION> IS A SINGLE VARIABLE WHICH UNIQUELY DESCRIBES A STRING OF CHARACTERS. THE READER WHO IS FAMILIAR WITH THE ARCHITECTURE OF THE CDC 6500 SERIES COMPUTER WILL KNOW THAT ONLY 16 CHARACTERS MAY BE STORED IN A SINGLE COMPUTER WORD. SINCE THE STRING ROUTINES ARE CAPABLE OF REPRESENTING STRINGS OF UP TO 500 CHARACTERS USING A SINGLE WORD, IT SHOULD BE APPARENT THAT IT IS IMPOSSIBLE TO STORE THE ACTUAL STRING IN THE VARIABLE REPRESENTING THE STRING.

ACTUALLY THE <STRING REP.> IS JUST A POINTER TO A LOCATION IN MEMORY CONTAINING THE ACTUAL STRING. THE ACTUAL STRING IS A CONTINUOUS BLOCK OF CHARACTERS OCCUPYING AS MANY WORDS AS NEEDED. ACTUAL STRINGS ARE NORMALLY ALLOCATED AUTOMATICALLY TO A RESERVED BLOCK OF MEMORY CALLED THE BUFFER. WHEN A <STRING REP.> IS GENERATED AUTOMATICALLY BY A STRING FUNCTION OR SUBROUTINE, IT ALWAYS CONTAINS THE FOLLOWING SUBFIELDS WHICH ARE PACKED INTO THE SINGLE WORD <STRING REP.>

<table>
<thead>
<tr>
<th>BITS</th>
<th>SUBFIELD NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-8</td>
<td>&lt;LENGTH&gt;</td>
<td>NUMBER OF CHARACTERS IN ACTUAL-STRING.</td>
</tr>
<tr>
<td>9-20</td>
<td>&lt;SUB&gt;</td>
<td>SUBSCRIPT OF THE BUFFER TO ADDRESS 1ST CHARACTER OF ACTUAL-STRING.</td>
</tr>
<tr>
<td>21-47</td>
<td>&lt;EXTENT&gt;</td>
<td>THE DISK EXTENT (BLOCK) ON WHICH THE ACTUAL-STRING IS STORED.</td>
</tr>
</tbody>
</table>
| 48-53| <INTERLOCK>   | ALWAYS SET TO 768, A SOFTWARE INTERLOCK DESIGNATING THIS WORD AS A <STRING REP.>.
| 54-59| <CHAR.>       | DISPLAY CODE OF ACTUAL-STRING FOR STRINGS OF LENGTH 1, 0 FOR ALL OTHER STRINGS, NOT USED AND SET TO 000. |

LET US SEE HOW THE STRING ROUTINES WOULD FORM A REPRESENTATION FOR INSTR WHERE INSTR IS DEFINED BY:

INSTR=KIN(INSTR)

FUNCTION KIN READS A STRING INTO MEMORY, Removes all trailing blanks, and copies the remaining string to some location in the buffer. THIS BUFFER IS CALLED USA FOR "DYNAMIC STORAGE AREA". THE STRING REPRESENTATION ASSIGNED TO INSTR WILL CONSIST OF THE LENGTH OF THE STRING IN THE LOW ORDER 8 BITS AND THE SUBSCRIPT OF USA NEEDED TO ADDRESS THE ACTUAL STRING IN THE NEXT 16 BITS. EACH TIME THIS STATEMENT IS EXECUTED KIN WILL PUT THE NEW ACTUAL-STRING IN THE BUFFER AND UPDATE THE STRING REPRESENTATION FOR INSTR. IT WILL NOT, HOWEVER, DESTROY OR REPLACE THE OLD ACTUAL-STRING PREVIOUSLY ASSIGNED TO INSTR. THIS IS BECAUSE KIN IS A FUNCTION AND IT DOESN'T "KNOW" WHERE ITS RETURNED VALUE WILL BE ASSIGNED. THERE IS NO WAY KIN CAN
Determine that the string representation will be replacing instr. on the other hand, if we had coded:

CALL KREPL (INSTR, KIN(-))

Then KREPL would know that INSTR was to be replaced, and so it would release the storage space used by the old actual-string for INSTR. This explains why it is important to use KREPL instead of using simple assignment statements when a variable is re-used for string representations.

What happens when the buffer area is completely filled? Previously this report claimed that any number of strings could be stored. Now it seems that we might run out of space in the buffer for actual-strings. In fact, if the buffer is small, we shall run out of space rather quickly. When this happens, the character string routines automatically copy the current buffer onto disk and clear the entire OSA for new actual-strings. Each time the buffer is completely filled, another block of disk space is automatically allocated for its storage. Each of these blocks is called an <extent>, and is identified by a unique number. The extent becomes an integral part of the <string rep.>, so that when an actual-string is needed by one of the string routines, it looks to see if the string was defined in the same <extent> as the <extent> which is currently in the buffer. If not, the correct <extent> is recalled from disk into the OSA. This procedure is done automatically with a random access disk file called KFILE. KFILE is generated when KINIT is first called, and should not be declared on the program card. Although the disk operations performed are much more efficient than would be possible using ordinary FORTRAN READ and WRITE statements, a disk access still takes about 1000 times longer than a memory access. Therefore it should be apparent that a big buffer (USA) is desirable if a lot of character string operations are to be performed. Since the larger array USA is, the fewer disk accesses will be required. Of course, if memory is at a premium in a certain application, the routines will still work with a USA as small as 64 words. Regardless of the size of the OSA buffer, the total storage capacity for actual-strings will be several million characters. In all cases the user will run out of memory for storage of <string representations> before storage for actual-strings is exhausted, supporting the claim to virtually unlimited storage.

In order to conserve space in the buffer, several special cases are defined for forming string representations:

1. If the actual-string is only one character long, then the actual string will not be allocated in the buffer, but will be stored directly in the upper 2 bits of the <string rep.>.
SINGLE CHARACTER STRINGS ARE FREQUENTLY ENCOUNTERED AND THIS SAVES BOTH MEMORY SPACE AND ACCESS TIME.

2. STRING CONSTANTS (DEFINED BY FUNCTION KON AND SUBROUTINE KONLST) ARE NOT COPIED INTO THE DSA BUFFER BECAUSE THE CONTENTS OF A STRING CONSTANT SHOULD NEVER CHANGE. THE ACTUAL-STRING USED IS THE ACTUAL-STRING PASSED AS AN ARGUMENT, AND THE VALUE OF \(<\text{SUM}>\) IS THE SUBSCRIPT OF DSA NECESSARY TO ADDRESS THE ACTUAL-STRING IN THE USER'S PROGRAM. IT SHOULD BE NOTED THAT \(<\text{SUM}>\) MAY BE A NEGATIVE NUMBER IN THIS CASE IF THE LOCATION OF THE ACTUAL-STRING IS A MEMORY ADDRESS LOWER THAN THE ADDRESS OF DSA(1). A CONSEQUENCE OF THIS SYSTEM IS THAT PROGRAMS USING OVERLAYS MUST MAINTAIN THE OVERLAY DEFINING A GIVEN STRING CONSTANT IN CORE WHENEVER THAT STRING CONSTANT IS REFERENCED BY ITS \(<\text{STRING REP}.>\).

THE SIX BITS COMPRISING \(<\text{INTERLOCK}>\) ARE A PATTERN OF BITS WHICH WOULD NOT NORMALLY BE FOUND IN THAT POSITION IN ANY REAL OR INTEGER NUMBER. ALL STRING Routines WHICH DEFINE A STRING RESULT SET THIS PATTERN, AND ALL ROUTINES EXPECTING A \(<\text{STRING REP}.>\) FOR AN ARGUMENT CHECK FOR IT. IF THE INTERLOCK PATTERN DOES NOT EXIST, THE DIAGNOSTIC "ILLEGAL STRING REPRESENTATION IS ISSUED. THIS HELPS TRAP INCORRECT STRING REPRESENTATIONS WHICH WOULD OTHERWISE YIELD UNPREDICTABLE AND USUALLY DISASTEROUS ERRORS.
SUBROUTINE KINIT REVISITED

Now that we understand how actual-strings are allocated to memory and disk, we are ready to revisit subroutine KINIT to see how we may alter the size of the buffer if we wish. Subroutine KINIT has a default buffer in labelled common as follows...

COMMON / KSPACE / DSA(512)

In order to change the buffer size, all we have to do is re-define the common block in the main program, and use the new dimension in the call to KINIT. For example, to increase the buffer size from 512 to 2048 words,

PROGRAM BIG(INPUT,OUTPUT)
COMMON / KSPACE / DSA(2048)
CALL KINIT(2048)

***

The dimension chosen for DSA should be an exact multiple of 64. When this method is used, a number of non-fatal loader errors are generated:

COMMON BLOCK REDEFINITION - KSPACE
LAST PROGRAM READ - XXXXX
LAST FILE ACCESSED - XXXXX

These diagnostics may be ignored.
SUBROUTINE KRLS - RELEASE STRING SPACE NO LONGER NEEDED

LOCAL FORMAT...

CALL KRLS(STRING REP.*, ..., STRING REP.*...))

ARGUMENTS...

STRING REP.* IS THE STRING REPRESENTATION OF THE STRING WHICH IS NO LONGER NEEDED.

THIS ROUTINE MAY BE CALLED WITH ANY NUMBER OF ARGUMENTS.
THE STRING SPACE IN THE BUFFER ALLOCATED FOR THE STRINGS whose REPRESENTATIONS ARE PASSED AS ARGUMENTS WILL BE FREED-UP FOR RE-USE.

EXAMPLE 17.1...

CALL KRLS(I STR, J STR, K STR)

THIS CALL WILL FREE SPACE OCCUPIED BY I STR, J STR, AND K STR.
IF THE ACTUAL-STRING$ FOR ANY OF THESE STRINGS ARE STORED IN A LOCATION OTHER THAN ARRAY DSA, THAT PARAMETER WILL BE IGNORED.
SUBROUTINE KRLSL - RELEASE SPACE FOR ARRAY OF STRINGS

LEGAL FORMAT...

CALL KRLSL(<STR.REP.ARRAY>,<NO.RLS.>) [IP.1]

ARGUMENTS...

<STR.REP.ARRAY> IS THE FIRST ELEMENT OF THE ARRAY TO BE RELEASED.
<NO.RLS.> IS THE NUMBER OF ELEMENTS TO BE RELEASED.

KRLSL MAY BE USED TO RELEASE THE SPACE USED FOR STORAGE
OF ACTUAL STRINGS FOR ALL OR PART OF AN ARRAY OF STRINGS.

EXAMPLE 18.1...

PROGRAM TEST(INPUT,OUTPUT)
DIMENSION ISTR(1:O)
CALL KINIT('')
***
DO 100 I=I,100
100 ISTR(I)=KIN(5)
***
LAST 40 STRINGS NO LONGER NEEDED, FREE-UP SPACE.
CALL KRLSL(ISTR(61),40)
***

THE CALL TO KRLSL WILL FREE BUFFER SPACE IN ARRAY DSA PREVIOUSLY
OCCUPIED BY THE ACTUAL-STRINGS FOR ISTR(61) THROUGH ISTR(100).
THE SPACE RECLAIMED WILL BE USED BY THE STRING ROUTINES FOR
STORAGE OF NEW ACTUAL-STRINGS AS NEEDED.
INTEGER FUNCTION KDC - DECODE ACTUAL-STRING TO STRING REPRESENTATION
LEGAL FORMAT...

<STRING REPRESENTATION> = KDC(<ACTUAL-STRING>,<COL>,<LENGTH>) \[19.1]\n
ARGUMENTS...

<ACTUAL-STRING> IS A VARIABLE, ARRAY, OR LITERAL STRING CONTAINING
THE ACTUAL STRING TO BE DECODED.
<COL> IS THE NUMBER OF THE FIRST CHARACTER IN THE <ACTUAL-STRING>
TO BE DECODED.
<LENGTH> IS THE NUMBER OF CHARACTERS TO BE DECODED.

KDC CAN BE USED IN A MANNER SIMILAR TO THE FORTRAN EXTENDED
DECODE STATEMENT. KDC WILL RETURN THE STRING REPRESENTATION
OF A STRING OF DISPLAY-CODE CHARACTERS PACKED INTO THE WORD,
STARTING IN CHARACTER POSITION <COL>, <LENGTH> CHARACTERS
LONG. THIS SUBSTRING IS COPIED INTO THE CHARACTER ROUTINES BUFFER
PRIOR TO GENERATION OF THE STRING REPRESENTATION.

EXAMPLE 19.1

PROGRAM TEST(INPUT,TAPE5=INPUT,OUTPUT,TAPE6=OUTPUT)
DIMENSION IBCD/1/)
CALL INIT( )

*** READ(S*20)(IBC(L),L=1,8)
2; FORMAT(2*ALP,A2)
***
ISTR = KDC(IBC,'A2')
***

THIS EXAMPLE READS A CARD IMAGE INTO ARRAY IBCD IN DISPLAY CODE
FORMAT, AND THEN EXTRACTS THE CONTENTS OF COLUMN 21 THROUGH 40
AS STRING REPRESENTATION ISTR.
SUBROUTINE KEC - ENCODE STRING REPRESENTATION TO ACTUAL-STRING
LEGAL FORMAT...
CALL KEC<string rep.>,<dest.>,<col.1>)

ARGUMENT LIST...
<string rep.> IS THE REPRESENTATION FOR THE STRING TO BE ENCODED.
<dest.> IS A VARIABLE OR ARRAY TO RECEIVE THE ACTUAL STRING.
<col.1> IS THE CHARACTER POSITION IN THE <dest.> WHERE THE
ACTUAL-STRING IS TO START.

THIS SUBROUTINE PROVIDES THE INVERSE FUNCTION TO FUNCTION KDC,
AND IS SIMILAR TO THE FORTRAN EXTENDED ENCODE STATEMENT.
THE DISPLAY CODE ACTUAL-STRING REPRESENTED BY <STRING REP.> IS
COPIED INTO THE DESTINATION ARRAY OR VARIABLE STARTING AT <COL.1>.
THE NUMBER OF CHARACTERS ENCODED WILL BE LEN(<STRING REP.>).
CHARACTERS IN <DEST.> BEFORE <CHAR.1> AND AFTER THE END OF THE
ACTUAL-STRING WILL NOT BE AFFECTED.

EXAMPLE 20.1...
PROGRAM TEST(input,output,tape5=input,tape6=output)
DIMENSION I*100(1)
CALL KINIT()
***
N=27
ENCOD(40*105:BOD)N
105 FORMAT(* OF THE *i29 * ENTRIES *)
***
J=11
IST=KAT(KSR(J),KON(" WERE VALID."))
CALL KEC(ISTR,1*BCD,20)
PRINT 200*(1*BCD(L),L=I,4)
200 FORMAT(3A10,A5)
***

THIS EXAMPLE WILL RESULT IN THE PRINTING OF,
OF THE 27 ENTRIES 11 WERE VALID.

42
APPENDIX A

CONTROL CARDS TO GAIN ACCESS TO KSPLIB Routines

The following sequence of control cards should be executed for batch programs prior to loading the FORTRAN object program:

\[
\text{ATTACH\*KSPLIB\*IO=CARBREY,} \\
\text{LIBRARY\*KS-LIB.}
\]

Example Program Deck for Batch Job...

MYJOB.
COMMENT\*(XXX=YYY=JDD000=01)\*MYNAME BILLING INFO CARD
\text{FTN\*(FP\*AT)}
\text{ATTACH\*KSPLIB\*IO=CARBREY,} \\
\text{LIBRARY\*KS-LIB.}
LGO.
7/8/9
\text{PROGRAM CHARA\*(INPUT,OUTPUT,TAPE\#=OUTPUT))}
\text{CALL KINIT(\$)}
\text{*** END}
7/8/9
\text{*** DATA CARDS USED BY PROGRAM CHARA}
7/8/9

Interactive Programs compiled using Intercom may use the following sequence of commands:

\[
\text{ATTACH\*KSPLIB\*ID=CARBREY,} \\
\text{RUN\*F\*N.} \\
\text{XEQ\*LDSET\*LIB=KSPLIB\*LOAD=LGO.}
\]

The source deck for the routines described in this manual has not been included due to its length. The routines are written in CDC FORTRAN extended and in CDC COMPASS ASSEMBLY LANGUAGE.

Requests for copies of the source deck should be made in writing to Chief, Scientific and Engineering Applications Division, Management Information Systems Directorate, ARACOM, Dover, N.J. After distribution is approved, the routines will be provided on a user-supplied 7 or 9 track magnetic tape.

43
APPENDIX B

SAMPLE PROGRAMS USING KSPLIB Routines

The following pages give two sample programs. A brief explanation of each program is provided, along with a listing of the program and the output from a sample run.
EXAMPLE PROGRAM I - FOUR-FUNCTION CALCULATOR

This program interprets arithmetic expressions in a form as would be used on an ordinary four-function calculator. For example, if the user types:

3 * 1.20 + 17 =

then the program will print:

20.60

This program analyzes the expression left to right, without any hierarchy of operators, until an "=" sign is encountered. The result is then printed. Parenttheses are not permitted. The program may be run either in batch mode or interactively provided input and output files are connected. Like all the example programs in this manual, little or no error-checking is performed in the interest of simplicity.

The operation of the program is simple. The five operators are defined by a konst call. The program searches the input line character-by-character for an operator. When it is found, 'opn' is set equal to the subscript of the operator array corresponding to the new operator. All the columns up to but not including the operator are assumed to contain the operand, which is converted from string to numeric by function rlnval. The portion of the string processed is discarded by extraction of the remainder of the string, and the process repeated. Each time both operands and the operator have been determined, the result is computed. When "=" is reached, the running total is printed using the default conversion of function ksr.

Although this entire program requires less than 30 statements to implement using the character string package, it would take more than a thousand lines of code to implement the equivalent program without the string routines.
**** FORTRAN LISTING OF EXAMPLE PROGRAM 1 ****

PROGRAM FUNK(IN,TAPE5=INPUT,OUTPUT=TAPE6=OUTPUT)

**** FOUR-FUNCTION CALCULATOR EMULATION PROGRAM ****

THIS PROGRAM SOLVES ARITHMETIC EXPRESSIONS ENTERED AS YOU
WOULD ENTER THEM ON AN ORDINARY 4-FUNCTION CALCULATOR. E.G.,
"3.74 - 2.5 = 1.24" AND PRINTS THE VALUE OF THE RESULT. E.G.,
"2.84^2". EVALUATION PROCEEDS LEFT-TO-RIGHT WITHOUT HIERARCHY
OF OPERATORS, UNTIL "=" IS EncOUNTERED. PARENTHESES ARE NOT
REMITTED. PROGRAM HALTS WHEN "STOP" ENCOUNTERED.

LOGICAL LE·XIC
DIMENSION IOPFR(5)

INITIALIZE PROGRAM. SET STRINGS TO NULL. DEFINE OPERATORS.
CALL INIT(0)
ISTR=JSTR=
CALL KONST(IOPE,"+"","-","*","/","=")
ISTOP=KON("STOP")

CONTINUE
READ NEXT EXPRESSION. HALT IF "STOP" ENTERED.
ANS=0.
IOP=0
CALL KREPL(ISTR,ISTR(5))
IF(LEXIC(ISTR,"EO",ISTOP))STOP
CALL KREPL(JSTR,ISTR)

CONTINUE
LOCATE NEXT OPERATOR IN STRING. IOPN IS SUBSCRIPT OF IOPFR.
ICOLOP=MATCH(ISTR,"FIRST",IOPE(IOPN),IOPN)
IF(IOPN.EQ.0)STOP"ILLEGAL OR NULL EXPRESSION."
EVALUATE NUMBER PRECEDING OPERATOR.
VAL=RLVAL(KSUB(ISTR,ICOLOP-1))
UPDATE ANSWER.
IF(I0P.EQ.0)ANS=VAL
IF(I0P.EQ.1)ANS=ANS*VAL
IF(I0P.EQ.2)ANS=ANS/VAL
IF(I0P.EQ.3)ANS=ANS-VAL
IF(I0P.EQ.4)ANS=ANS+VAL
DISCARD PORTION OF EXPRESSION ALREADY EVALUATED.
CALL KREPL(ISTR,KSUB(ISTR,ICOLOP+1))
IOP=IOPN
IF(I0P.NE.5)GO TO 11
COME HERE WHEN "=" ENCOUNTERED. PRINT RESULT.
CALL KREPL(JSTR,KAT(JSTR,KSR(ANS)))
CALL KOUT(6,JSTR,1)
GO TO 1
END

46
SAMPLE OUTPUT FROM PROGRAM FUNK4 RUN

2 + 2 = 4.
2*3 + 1.28318 = 7.28318
100/3 + 56 - .6666 = 82.6667333333
26 + 23 + 21+20+17+10+1-117 = 91.
2*222.1 + 100.2 = 344.
SAMPLE PROGRAM 2 - ALGEBRAIC ASSIGNMENT STATEMENT INTERPRETER

THIS PROGRAM IS A MUCH-EXPANDED VERSION OF PROGRAM 1.

THIS PROGRAM INTERPRETS COMPLETE ALGEBRAIC ASSIGNMENT STATEMENTS SIMILAR TO FORTRAN ASSIGNMENT STATEMENTS, INCLUDING PARENTHESES AND VARIABLES. FOR EXAMPLE, IF THE INPUT LINE READ,

\[ X_{\text{DISPERSION}} = \frac{1}{(2 + 1.577)} \]

THEN THE PROGRAM WOULD PRINT,

\[ X_{\text{DISPERSION}} = 0.27956383947 \]

IF THE NEXT INPUT LINE READ,

\[ Y_{\text{DISPERSION}} = 1 + 2 \times \left( 1 - X_{\text{DISPERSION}} \right) / (X_{\text{DISPERSION}}^{0.707}) \]

THEN THE RESULT PRINTED WOULD READ,

\[ Y_{\text{DISPERSION}} = 4.54783459936 \]

THE PROGRAM WILL AUTOMATICALLY REMEMBER THE VALUE OF THE VARIABLE \( X_{\text{DISPERSION}} \) FROM THE PREVIOUS EQUATION.

THE OPERATORS ALLOWED INCLUDE \( +, - \) INCLUDING UNARY NEGATION, \( *, /, ** \), AND PARENTHESES. THE HIERARCHY OF EVALUATION IS THE SAME AS FORTRAN (E.G., MULTIPLIES DONE BEFORE ADDS IN THE ABSENCE OF PARENTHESES). VARIABLES MAY BE ANY LENGTH, UNLIKE FORTRAN. HOWEVER, IMBEDDED BLANKS ARE NOT PERMITTED IN NUMERIC CONSTANTS OR VARIABLES. NO DISTINCTION IS MADE BETWEEN REAL AND INTEGER VARIABLES OR CONSTANTS; ALL ARITHMETIC IS PERFORMED IN FLOATING POINT MODE. AS IN THE FIRST EXAMPLE, ALMOST NO ERROR CHECKING IS DONE IN ORDER TO KEEP THE PROGRAM REASONABLY SIMPLE.

IN ADDITION TO ILLUSTRATING SOME TECHNIQUES FOR USAGE OF THE STRING ROUTINES, THIS PROGRAM CAN ALSO SERVE AS A PEDAGOGICAL TOOL FOR TECHNIQUES USED IN HIGH-LEVEL LANGUAGE COMPILERS. THE METHOD USED FOR INTERPRETING THE ASSIGNMENT STATEMENTS IN THIS PROGRAM IS FUNDAMENTALLY THE SAME AS THE TECHNIQUE USED IN MOST MODERN HIGH-LEVEL LANGUAGE COMPILERS.

REFERENCE [2] GIVES FURTHER INFORMATION.

THERE ARE TWO BASIC TOOLS USED IN THIS PROGRAM THAT MAKE POSSIBLE THE TRANSLATION PROCESS: POLISH NOTATION AND THE PUSHDOWN STACK. BECAUSE OF THE PROLIFERATION OF HEWLETT-PACKARD CALCULATORS, IT IS ASSUMED THAT THE PRINCIPLES OF OF POLISH NOTATION ARE WELL-KNOWN TO THE READER. TO REVIEW BRIEFLY, POLISH NOTATION IS A SYSTEM ALLOWING US TO WRITE AN EXPRESSION OF ARBITRARY COMPLEXITY WITHOUT PARENTHESES AND WITHOUT AMBIGUITY. FOR EXAMPLE, AN ALGEBRAIC (OR "INFIX") EXPRESSION SUCH AS,

\[ x = 2 \times (3 + 4) \]

COULD BE WRITTEN IN POLISH NOTATION AS:
34 + 2 * x =

This form is usually called reverse Polish notation (RPN) since the operators follow the operands. The advantage of Polish notation is that the expression is simply scanned from left to right and evaluated as each operator is encountered. Users of pocket calculators based on RPN are already well acquainted with the principles of Polish notation, although they may not be aware of it. These users may also be aware of the tool that makes the execution of the Polish expression possible: the push-down stack. The operation of a push-down stack is illustrated in Figure B-1. The stack may be visualized as a stack of trays in a cafeteria. The last tray placed on the stack is always the first one taken off. Such a stack is sometimes called a last-in-first-out stack, or LIFO. To execute a Polish expression, proceed as follows:

1. Moving left to right, if an operand is encountered, push it on the stack.
2. When an operator is encountered, pop the last two operands off the stack and perform the indicated operation on them.
3. Push the result back on the stack, forming a new operand.

When using an RPN calculator, the "Enter" key is used to push an operand on the stack. The other operations are performed automatically by the calculator.

The push-down stack, which is the key to executing Polish notation, is also the key to converting infix notation to Polish. Figure B-3 illustrates a flowchart for converting algebraic expressions to Polish expressions using a stack. Figure B-4 shows how we may use the same stack to execute the Polish expression. Thus we have a two-step conversion process which can handle expressions of arbitrary complexity. Figure B-1 also illustrates how a push-down stack is implemented in software. The stack is just an array with a pointer maintained elsewhere which always points to the top of the stack. When the stack is pushed, the pointer is incremented. When the stack is popped, the array element pointed to is returned and the pointer decremented. In the program listing, three stack routines are provided: IPUSH, which pushes the argument on the stack; IPOP, which returns the item popped off the stack, and ISEE, which allows us to examine the content of the top of the stack without popping it. The flow chart in Figure B-2 illustrates the overall program flow.

The variables used in the sample program are stored in an array as encountered. In practice, this method is too inefficient for real compilers, and a hash-table should be used for any real applications (see reference [11]).
AFTER THE READER UNDERSTANDS THE PROGRAM OPERATION, AN INTERESTING EXERCISE CAN BE TO EXPAND THE PROGRAM TO INCLUDE FUNCTIONS, SUBSCRIPTED VARIABLES, AND EVEN CONDITIONAL STATEMENTS. ANOTHER INTERESTING EXERCISE IS TO WRITE A HIGH-LEVEL LANGUAGE FOR PROGRAMMABLE CALCULATORS. THIS HIGH-LEVEL LANGUAGE COULD BE CONVERTED INTO A SERIES OF KEYSTROKES USING RPN LOGIC BY A ROUTINE SIMILAR TO SUBROUTINE POLISH.

THIS PROBLEM WAS SELECTED BECAUSE IT ILLUSTRATES THE EASE WITH WHICH EVEN A RELATIVELY COMPLICATED STRING-MANIPULATION PROBLEM CAN BE HANDLED USING THE CHARACTER STRING PROCESSING LIBRARY. WITHOUT THE LIBRARY, THIS PROGRAM OF LESS THAN 220 STATEMENTS WOULD PROBABLY REQUIRE IN EXCESS OF 2000 STATEMENTS.
FIGURE B-1

PUSH-DOWN STACK

**OPERATION - LAST IN, FIRST OUT (LIFO)**

1. (EMPTY)
2. PUSH "HELLO"
3. POP.
4. POP.

**IMPLEMENTATION - ARRAY WITH POINTER**

1. STACK POINTER
2. STACK POINTER
3. STACK POINTER
4. STACK POINTER
FIGURE B-2
PROGRAM XPRESN
SOLVE ALGEBRAIC ASSIGNMENT EXPRESSION

START

INITIALIZE VARIABLES

INPUT INFIX ASSIGNMENT STATEMENT

IS IT "STOP"?

YES → STOP

NO → CONVERT INFIX TO POLISH NOTATION

COMPUTE USING POLISH NOTATION
FIGURE B-3

SUBROUTINE POLISH

CONVERT INFIX TO POLISH NOTATION

OPERATOR HIERARCHY

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>0</td>
</tr>
<tr>
<td>)</td>
<td>1</td>
</tr>
<tr>
<td>=</td>
<td>2</td>
</tr>
<tr>
<td>+ -</td>
<td>3</td>
</tr>
<tr>
<td>* / ~</td>
<td>4</td>
</tr>
<tr>
<td>**</td>
<td>5</td>
</tr>
</tbody>
</table>

START

IT ← SCAN FOR NEXT ATOM

IS IT NULL?

YES → FLUSH STACK TO POLISH

NO → RETURN

IS IT OPERATOR?

NO → MOVE IT TO POLISH

YES → IT ← SCAN FOR NEXT ATOM

RANK OF TOP OF STACK ≥ IT?

NO → IS IT "(~)?

YES → POP

NO → IT ← SCAN FOR NEXT ATOM

MOVE IT TEMP TO POLISH

NOTE: ~ IS UNARY NEGATION OPERATOR.

AN ATOM IS AN OPERATOR, CONSTANT, OR VARIABLE.

53
FIGURE B-4
SUBROUTINE COMPUT
CALCULATE ANSWER FROM POLISH NOTATION

START

ANY 
ATOMS 
LEFT?

RETURN

IT = NEXT 
POLISH 
ATOM

IS IT 
OPERATOR?

PUSH IT

I = POP 
OP1 = VALUE (I)

PRINT 
ANSWER

IS IT 
"+"?

VARNAM = POP 
ASSIGN OP2 
TO VARNAM

IS IT 
"-"?

PUSH (-OP2)

IS IT 
"*"?

PUSH ((OP1 + OP2)

IS IT 
"/"?

PUSH ((OP1 - OP2)

PUSH ((OP1 • OP2)

PUSH ((OP1 / OP2)

A
PROGRAM XPRESN( INPUT, TAPE=INPUT, OUTPUT, TAPE=OUTPUT)

INTERACTIVE ARITHMETIC EXPRESSION INTERPRETER WITH VARIABLES

LOGICAL LEXIC, LASTOP
COMMON / XPRSNN / IPS, IP, LSTK(50), LPOL(200), LOLIM(10),
X LOPE(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, THLK, INSTR
X NVAR

CALL INIT(I)
CALL KONLST(LOPE, "(" " ", "=" " ", "-" " ", "*" " ", "/" " ",
CASE  " " " ", "*" " ", "/" " "," " " " ", "=" " ", "-" "")
CALL KONLST(LOLIM, "(" " ", "=" " ", "-" " ", "*" " ", "/" " ",
CASE  " " " ", "=" " ", "-" "")
IHLK=KON(" ")
INSTR=6
DO 10 I=1,100
LVAR(I)=
VARVAL(I)=0.0
10 CONTINUE

DO 11 CONTINUE

RESET COUNTERS IN PREPARATION FOR NEW EXPRESSION.
IP=6
LASTOP=.TRUE.
READ EXPRESSION.
CALL KREPL(INSTR, KIN(" "));
CALL KOUT(KIN(" "));
IF(LEXIC(INSTR,"FO",KON("STOP",".")))STOP

CONVERT INFIX TO POLISH NOTATION.
CALL POLISH

USE POLISH LIST TO COMPUTE AND PRINT RESULT.
CALL COMPUT
GO TO 11
END
SUBROUTINE POLISH

CONVERT EXPRESSION FROM INFIX NOTATION TO POLISH NOTATION

COMMON / XPRS/N / IPS, 1PS, PSTK(50), LPOL(200), LULIM(10),
X LOPER(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, INSTR,
X NVAR

51 CONTINUE
CALL SCAN REMAINDER OF STRING FIND NEXT ATOM IT.

IF (IT .EQ. 0) GO TO 251
IF (IT .EQ. 0) GO TO 261
IF (IT .EQ. 0) GO TO 301

51 CONTINUE
IF (IT .EQ. 0) GO TO 251
IF (IT .EQ. 0) GO TO 261
IF (IT .EQ. 0) GO TO 301
SUBROUTINE SCAN(IT, LOP)

SCAN REMAINING EXPRESSION FOR NEXT ATOM.
AN ATOM MAY BE AN OPERATOR, A VARIABLE, OR A NUMBER.

LOGICAL LEXIC, LASTOP, LOP
COMMON /XPRSN / I5, IP, LSTK(56), LPOL(200), L0LIM(10),
X LOPER(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IHLK, INSTR,
X NVAR

IF(INSTR.EQ.0)GO TO 151
LOCATE COLUMN WITH 1ST NON-BLANK CHARACTER.
I1NR=INDEX(INSTR,*FIRST*,IHLK)
IF(I1NR.EQ.0)GO TO 151
REMOVE ANY LEADING BLANKS.
IF(I1NR.GT.1)CALL KREPL(INSTR,KSUB(INSTR,I1NR))
FIND COLUMN WITH DELIMITER TERMINATING ATOM.
ICOLIM=INDEX(INSTR,*FIRST*,L0LIM,EM,IMIT)
IF(ICOLIM.EQ.1)GO TO 201
CO-E HERE IF IT IS AN OPERAND.
IT=KSUB(INSTR,ICOLIM-1)
DISCARD PORTION OF EXPRESSION WERE DONE WITH.
CALL KREPL(INSTR,KSUB(INSTR,ICOLIM))
GO TO 501
151 CONTINUE
CO-E HERE ON END-OF-STRING.
IT=
LOP=.TRUE.
GO TO 501
201 CONTINUE
CO-E HERE IF IT IS AN OPERATOR.
IT=KSUB(INSTR,1)
IF(LEXIC(IT,*EQ*,LOPER(6)))GO TO 251
IF(LEXIC(IT,*NE*,LOPER(9)))GO TO 401
CO-E HERE FOR - SIGN. SEE IF IT IS UNARY OPERATOR -.
IF(LASTOP)IT=LOPER(8)
GO TO 401
251 CONTINUE
CHECK FOR POSSIBLE ** OPERATOR INSTEAD OF JUST **.
IF(LEXIC(KSUB(INSTR,2),IT,*EQ*,LOPER(6)))IT=LOPER(9)
401 CONTINUE
LASTOP=.TRUE.
J=2
IF(LEXIC(IT,*EQ*,LOPER(9)))J=3
DISCARD PORTION OF EXPRESSION ALREADY DONE.
CALL KREPL(INSTR,KSUB(INSTR,J))
LOP=.TRUE.
GO TO 601
501 CONTINUE
LOP=.FALSE.
LASTOP=.FALSE.
601 CONTINUE
RETURN
END
SUBROUTINE IPUSH(ITEM)

PUSH ITEM ONTO PUSH-DOWN (LIFO) STACK.

LOGICAL LASTOP
COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LUIM(10),
X LOPED(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IBLK, INSTR,
X NVAR
IPS=IPS+1
LSTK(IPS)=ITEM
RETURN
END

INTEGER FUNCTION IPOP(NS)

RETURN ITEM POPPED OF TOP OF PUSH-DOWN STACK.

LOGICAL LASTOP
COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LUIM(10),
X LOPED(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, IBLK, INSTR,
X NVAR
IPOP=
NS=IPS
IF(NS.LT.1)RETURN
IPOP=LSTK(IP)
IPS=IPS-1
NS=IPS
RETURN
END

58
INTEGER FUNCTION ISEE(NS)

RETURN CONTENTS OF TOP OF PUSH-DOWN STACK WITHOUT AFFECTING
THE CONTENT OF THE STACK.
NS IS A VARIABLE, RETURNED AS PRESENT STACK DEPTH.

LOGICAL LASTOP
COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LOLIM(10),
X LOPEP(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, TBLK, INSTR,
X NVAR
ISEE=
NS=IPS
IF(IPS.LT.1)RETURN
ISEE=LSTK(IPS)
RETURN
END

INTEGER FUNCTION IRANK(ISTR)

COMPUTE RELATIVE RANK OF OPERATOR, ISTR, FROM GIVEN HIERARCHY.

LOGICAL LEXIC, LASTOP
COMMON / XPRSN / IPS, IP, LSTK(50), LPOL(200), LOLIM(10),
X LOPEP(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, TBLK, INSTR,
X NVAR
IRANK=-1
IF(ISTR.EQ.0)RETURN
DO 10 I=1,9
IF(LEXIC(LOPER(I),"EQ",ISTR))GO TO 11
10 CONTINUE
STOP "RANK ERROR"
11 CONTINUE
IRANK=LRANK(I)
RETURN
END
SUBROUTINE COMPUT

EXECUTE OPERATIONS INDICATED BY POLISH EXPRESSION.

LOGICAL LEXIC
COMMON / XPRSN / IPS*, IP*, LSTK(50), LPOL(200), LULIM(10),
 X LOPE(J), LRANK(J), LVAR(100), VARVAL(100), LASTOP, TBLK, INSTR,
 X NVAR
 IPS=0
 IMSG=
 CALL <REP’L(IMSG,’0’,””,””=””,””))
 ANS=0.
 I=6

1=1 CONTINUE
 C NEXT ITEM FROM POLISH LIST.
 I=I+1
 IF(I.GT.IP)GO TO 51
 C IS IT AN OPERAND...
 IT=LPOL(I)
 DO 15 J=1,9
 IF(LEXIC(IMSG,LOPER(J)).EQ.0)GO TO 21
 15 CONTINUE
 C NO. IT IS AN OPERAND IF THIS PATH. PUSH ON STACK.
 CALL IPUSH(IT)
 GO TO 101

21 CONTINUE
 C YES. IT IS OPERATOR IF THIS PATH. GET LAST 2 OPERANDS FROM
 C TOP OF STACK, UNLESS UNARY OPER. OR = SIGN SPECIAL CASE.
 OP2=VALUE(1P0P(NS))
 IF(,3)GO TO 31
 IF(J.EQ.6)OP1=VALUE(1P0P(NS))

 30 CONTINUE
 C PERFORM THE OPERATION.
 JM3=J-3
 GO TO (301,311,321,331,341,351)JM3

301 CONTINUE
 C ADD.
 ANS=OP1 + OP2
 GO TO 401

311 CONTINUE
 C SUBTRACT
 ANS=OP1 - OP2
 GO TO 401

321 CONTINUE
 C MULTIPLY
 ANS=OP1 * OP2
 GO TO 401

331 CONTINUE
 C DIVIDE
 ANS=OP1 / OP2
 GO TO 401

60
341 CONTINUE
C  UNARY NEGATION.
   ANS = -OP2
   GO TO 41
351 CONTINUE
C  EXPONENTIATION.
   ANS = OP1 ** OP2
   GO TO 41
391 CONTINUE
C  ASSIGNMENT (=).
   NAMVAR = I20P(NS)
   DO 39 K = 1, NVAR
      IF (LEXIC(NAMVAR "=" LVAR(K))) GO TO 391
   CONTINUE
C  VARIABLE NOT FOUND, PUT IT IN TABLE.
   NVAR = NVAR + 1
   LVAR(NVAR) = NAMVAR
   K = NVAR
391 CONTINUE
C  VARVAL(K) = OP2
   ANS = OP2
   CALL KREPL(IMSG, KAT(NAMVAR, IMSG))
   GO TO 301
41 CONTINUE
C  PLACE RESULT OF OPERATION ON STACK.
   CALL T_PUSH(KSR(ANS))
   GO TO 141
51 CONTINUE
C  CALL HERE TO PRINT ANSWER.
   CALL KOUT(6, KAT(IMSG, KSR(ANS)), 1)
   RETURN
END
REAL FUNCTION VALUE(ITEM)

RETURN NUMERICAL VALUE OF OPERAND ITEM.

COMMON / XPRSN / IP, LSTK(50), LPOL(200), LULIM(10),
X LOPE(9), LRANK(9), LVAR(100), VARVAL(100), LASTOP, TBLK, INSTR,
X NVAR
VALUE=0.
IF(ITEM.EQ.0)RETURN

1ST CHARACTER DETERMINES IF ITEM IS VARIABLE OR NUMBER.
IC1=KSUB(ITEM+1*1)
IF(LEXIC(IC1,"LT",KON("0")))GO TO 201

ITEM IS A LITERAL NUMBER IF THIS PATH.
VALUE=RLVAL(ITEM)
RETURN

201 CONTINUE
COME HERE IF ITEM = VARIABLE. FIND IT IN TABLE.
DO 25 K=1,NVAR
IF(LEXIC(ITEM,"E0",LVAR(K)))GO TO 26.
25 CONTINUE

VARIABLE UNDEFINED IF THIS PATH,_ASSUME = 0.
CALL KOUT(6,KAT(KON(" UNDEFINED",-1)),ITEM,KON(" SET TO 0."))
RETURN

261 CONTINUE
RETURN VALUE ASSOCIATED WITH VARIABLE.
VALUE = VARVAL(K)
RETURN
END
SAMPLE OUTPUT FROM PROGRAM XPRESN PUN

\begin{verbatim}
Z = 11
Z = 11.
ZFUNCTON = 1*(1/(2*(1/(6*7+21))))
ZFUNCTON = 1.49714285714
A = 1
A = 1.
B = 10.000
B = 10.
C = 1+5.5+3.5 -1
C = 21.
ROOT = (-B + (B*B - 4*A*C) / (2*A))
ROOT = 3.
T = 175
T = 175.
GR = 7.5E-3
GR = 7.500E+13
PR = .2256
PR = .2256
NUSUBL = .9246 * ((PR**1.17) / (1.049*PR**0.67)*GR)**0.4
NUSUBL = 4340.2204221
FINAL = NUSUBL/C + ZFUNCTION
FINAL = 2171.50735391
JVF = RALPH + 2.44
UNDEFINED RALPH SET TO 0.
JVF = 2.44
STOP
\end{verbatim}
### APPENDIX C

**ALPHABETICAL LISTING OF KSPLIB ROUTINES AND LENGTHS**

(Since the character string processing routines are available on a library, only programs actually needed are loaded)

<table>
<thead>
<tr>
<th>NAME</th>
<th>OCTAL LEN.</th>
<th>REFERENCES THESE ROUTINES...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>43</td>
<td>CHXFR</td>
</tr>
<tr>
<td>CENSUS</td>
<td>11</td>
<td>BIT CENSUS IFETS VFETP VIO</td>
</tr>
<tr>
<td>CHFILL</td>
<td>116</td>
<td>KAT KON KOUT KSAS NFLTFR</td>
</tr>
<tr>
<td>CHXFR</td>
<td>115</td>
<td>KAT KON KOUT KSAS KSMAKE LOCFP</td>
</tr>
<tr>
<td>DSGET</td>
<td>147</td>
<td>KAT KON KOUT KSAS KSMAKE</td>
</tr>
<tr>
<td>IFETS</td>
<td>75</td>
<td>KAT KON KOUT KSAS KSMAKE LOCFP</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>151</td>
<td>CHXFR DSGET KSAS KSMAKE</td>
</tr>
<tr>
<td>KAT</td>
<td>150</td>
<td>CHXFR DSGET KSAS KSMAKE</td>
</tr>
<tr>
<td>KOC</td>
<td>110</td>
<td>CHXFR DSGET KSAS</td>
</tr>
<tr>
<td>KEC</td>
<td>66</td>
<td>CHXFR KSAS</td>
</tr>
<tr>
<td>KIM</td>
<td>252</td>
<td>CHXFR DSGET KRACTR KSMAKE</td>
</tr>
<tr>
<td>KINIT</td>
<td>200</td>
<td>VFETC8 VFETP VIO</td>
</tr>
<tr>
<td>KINTEK</td>
<td>137</td>
<td>AINST CHXFR DSGET KRACTR KSMAKE</td>
</tr>
<tr>
<td>KON</td>
<td>26</td>
<td>CHFILL CHXFR DSGET KRACTR KSMAKE</td>
</tr>
<tr>
<td>KONLST</td>
<td>1</td>
<td>LOCFP</td>
</tr>
<tr>
<td>KOUT</td>
<td>215</td>
<td>KSAS</td>
</tr>
<tr>
<td>KOUTEK</td>
<td>103</td>
<td>KSAS</td>
</tr>
<tr>
<td>KRACTR</td>
<td>1</td>
<td>KSAS</td>
</tr>
<tr>
<td>KRAFT</td>
<td>37</td>
<td>KRLS</td>
</tr>
<tr>
<td>KRLS</td>
<td>120</td>
<td>KSAS LOCFP</td>
</tr>
<tr>
<td>KRLSL</td>
<td>70</td>
<td>KSAS</td>
</tr>
<tr>
<td>KSAS</td>
<td>222</td>
<td>IFETS VFETP VIO</td>
</tr>
<tr>
<td>KMAKE</td>
<td>114</td>
<td>KRLS</td>
</tr>
<tr>
<td>KSH</td>
<td>413</td>
<td>CHXFR DSGET KRACTR KSMAKE</td>
</tr>
<tr>
<td>KSUB</td>
<td>135</td>
<td>CHXFR DSGET KSAS KSMAKE</td>
</tr>
<tr>
<td>LEN</td>
<td>46</td>
<td>KSAS</td>
</tr>
<tr>
<td>LEXIC</td>
<td>353</td>
<td>CHXFR KRACTR KSAS</td>
</tr>
<tr>
<td>LOCFP</td>
<td>26</td>
<td>KON</td>
</tr>
<tr>
<td>MATCH</td>
<td>26</td>
<td>CHXFR KRACTR KSAS</td>
</tr>
<tr>
<td>NFLTFR</td>
<td>16</td>
<td>NFLTFR</td>
</tr>
<tr>
<td>NFLTER</td>
<td>27</td>
<td>NFLTFR</td>
</tr>
<tr>
<td>RLVAL</td>
<td>46</td>
<td>KAT KON KOUT KSAS NFLTTER</td>
</tr>
<tr>
<td>VFETC8</td>
<td>65</td>
<td>KAT KON KOUT KSAS NFLTTER</td>
</tr>
<tr>
<td>VFETP</td>
<td>67</td>
<td>KAT KON KOUT KSAS NFLTTER</td>
</tr>
<tr>
<td>VIO</td>
<td>60</td>
<td>KAT KON KOUT KSAS NFLTTER</td>
</tr>
</tbody>
</table>

*NOTE: ROUTINES ACCESSIBLE BY USER.
Length of all routines is subject to change as modifications are made to library or operating system.*

64
REFERENCES


<table>
<thead>
<tr>
<th>Call</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>KINIT(&lt;STPF.SIZE&gt;)</td>
<td>715</td>
</tr>
<tr>
<td>CALL</td>
<td>KINTE(&lt;LIT.6TM&gt;)</td>
<td>24</td>
</tr>
<tr>
<td>CALL</td>
<td>KONLST(&lt;INT.ARRAY&gt;,&lt;LIT.ST&gt;,&lt;LIT.ST&gt;)</td>
<td>13</td>
</tr>
<tr>
<td>CALL</td>
<td>KOUT(&lt;LFN&gt;,&lt;STR.REP.&gt;)</td>
<td>403</td>
</tr>
<tr>
<td>CALL</td>
<td>KOUT(&lt;LEN&gt;,&lt;STR.REP.&gt;,&lt;CARR.CNT.H&gt;)</td>
<td>25</td>
</tr>
<tr>
<td>CALL</td>
<td>KOUTEK(&lt;STR.REP.&gt;)</td>
<td>37</td>
</tr>
<tr>
<td>CALL</td>
<td>KREPL(&lt;STR.REP.&gt;,&lt;STR.REP.&gt;)</td>
<td>39</td>
</tr>
<tr>
<td>CALL</td>
<td>KRLS(&lt;STR.REP.&gt;,&lt;STR.REP.&gt;,&lt;STR.REP.&gt;)</td>
<td>40</td>
</tr>
<tr>
<td>CALL</td>
<td>KRLSL(&lt;STR.REP.ARRY&gt;,&lt;NUM.SLS&gt;)</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>KSR(&lt;VALUE&gt;)</td>
<td>31</td>
</tr>
<tr>
<td>CALL</td>
<td>KSR(&lt;VALUE&gt;,&lt;FTN.FMT&gt;)</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>KSUM(&lt;STR.REP.&gt;,&lt;START.COL.&gt;)</td>
<td>315</td>
</tr>
<tr>
<td>CALL</td>
<td>KSUM(&lt;STR.REP.&gt;,&lt;START.COL.&gt;,&lt;O.STRING&gt;)</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>LEN(&lt;STR.REP.&gt;)</td>
<td>516</td>
</tr>
<tr>
<td>CALL</td>
<td>LEXIC(&lt;STR.REP.&gt;,&lt;KEY&gt;,&lt;STR.REP.&gt;)</td>
<td>517</td>
</tr>
<tr>
<td>CALL</td>
<td>MATCH(&lt;STR.REP.&gt;,&lt;KEY&gt;,&lt;LIST.ST&gt;)</td>
<td>519</td>
</tr>
<tr>
<td>CALL</td>
<td>MATCH(&lt;STR.REP.&gt;,&lt;KEY&gt;,&lt;LIST.ST&gt;,&lt;IN.PAT&gt;,&lt;MATCH&gt;)</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>RLVAL(&lt;STR.REP.&gt;)</td>
<td>29</td>
</tr>
</tbody>
</table>
This report describes a set of general-purpose, FORTRAN-callable functions and subroutines for manipulating character-strings. Character-strings of arbitrary length can be treated as simple (non-dimensioned) variables. Functions are provided for string definition, string concatenation, substring extraction, string comparison, pattern matching and searching, numeric-string/string-numeric conversions, string input/output, and other tasks. The routines described are moderately system-dependent, and are currently available.
as an object program library on the CDC 6500/6600 computers at the ARRADCOM Dover, New Jersey site.