STRCOMP

AND

ISRCOMP

A USER'S MANUAL

p. 13-14 missing

BOLT BERANEK AND NEWMAN INC
50 MOULTON STREET CAMBRIDGE, MASS. 02138

STRCOMP AND ISRCOMP

A USER'S MANUAL

Virginia M. Dominick

Medical Information Technology Department
BOLT BERANEK AND NEWMAN INC
Cambridge, Massachusetts 02138

April 1968

The work described in this document received support through a contract, PH43-62-850, from the Institute of General Medical Sciences, National Institutes of Health, and through a grant from the American Hospital Association.

TABLE OF CONTENTS

LIST OF FIGURES

PREFACE

STRCOMP is a user-oriented, general-purpose, procedural programming language. With this language, a user may write programs to collect data, manipulate the data, retrieve all or part of the data, and perform statistical or textual analyses. Little or no previous experience in computer technology is necessary to learn and use the STRCOMP language. ISRCOMP is an extension of STRCOMP which operates on data files stored by the Information Storage and Retrieval (ISR) System.

The string-manipulation features in the STRCOMP language were originally designed by D. Bobrow, W. Feurzeig, and S. Papert; the design was modified by J. Barnaby and P. Wexelblat and implemented by J. Barnaby with assistance from P. La Follette and P. Wexelblat.

INTRODUCTION

This Manual is intended for new users of STRCOMP and ISRCOMP. The body of the Manual includes instructions for using each function and examples of programs in which these functions are used. For users with knowledge of JOSS, ABACUS, or TELCOMP, only a review of the Manual is necessary. The Appendices include detailed operating instructions (Teletype keyboard, etc.), a glossary of terms and commands, library programs, and a summary of differences between STRCOMP and ISRCOMP. These are for quick reference and may benefit both experienced and new users.

In all typescripts, messages printed by the computer are shown in black and user typing is shown in red for illustrative clarity.

I. THE STRCOMP LANGUAGE

A. General Aspects and Direct Commands

Calling in to the Computer

Terminals are connected to the Bolt Beranek and Newman Inc. (BBN) computer by telephone lines. Activation of the terminal is effected by first pressing the ORIG button, then dialing the BBN number (491-5220) where a computer operator will connect the telephone line to the computer after the user is identified. To initialize the computer system, depress the BREAK key (Appendix A includes a Teletype keyboard illustration for those unfamiliar with the Teletype keyboard).

A BRK RLS (break release) key will light to indicate that the keyboard is locked —— this must be depressed before typing. While the light is on, the computer will print the activation identification (consisting of date, time, and BBN line number) which means that the computer is ready to accept a call.

```
3:Ø3 PM 3/2Ø BBN12-12
```

No questions are asked; the user types in the name of the program, followed by a space, his initials, and depression of the ESC key. The ESC key is known as an end-of-message key. At this point, it is the only message terminator.

```
12:48 PM 12/18 BBN1Ø-1Ø
STRCOMP VMD
```

The STRCOMP program then advances to a clean page (on certain

Teletypes*) and prints an arrow (←). This is the sign in STRCOMP which means that the user is in control and may now use the system for direct calculations, for writing programs, or for data manipulation.

HALT

When finished with STRCOMP, the user types HALT, which causes the time to be printed and the Teletype to shut off (on certain Teletypes*). This command terminates any program within STRCOMP. If STRCOMP is restarted, the same series of operations (beginning with depression of the BREAK key) must be performed. The HALT function erases all other data and programs which were entered during a STRCOMP session. In order to preserve any work completed, the program or data must be filed before halting (see page 28).

The user need not call the BBN computer room to sign off. The HALT function shuts off the Teletype (on certain Teletypes*), then transmits a message to the computer room at BBN, indicating that the line is no longer in use.

Terminators

While using STRCOMP, several keys are available for end of message transmission. The ESC key, the RETURN key, the LINE FEED key, or the FORM FEED (CONTROL L) key terminates the line, meaning that the message typed in by the user is transmitted to the computer for processing. One of the terminating keys must be depressed after each entry.

---

*Optional equipment available from New England Telephone Company.

Erasing

Although back space or physical erasure is not possible on a Teletype, a means for deleting either a single character or an entire line has been established. To delete a single character or space, depress the SHIFT key and L simultaneously. A backslash (\) or bow tie (⌁) will print on the Teletype and the character directly preceding it will be disregarded. If SHIFT L key is depressed twice, the two preceding characters are disregarded, etc. In the following example, three characters are erased with the backslash key.

```
←TYPE2\\\PE 2+2
```

The line will be interpreted as TYPE 2+2.

If an entire line must be erased (which is expedient if the error occurs toward the beginning of the line), the RUBOUT key may be used. When this key is depressed, the carriage returns to the beginning of the same line, prints an X over the arrow (↑), then returns control to the user by typing another arrow on the next line.

Advancing a Page

To begin at the top of a new page, simultaneously depress the CONTROL button and the L, which will cause a form feed (page advance) to be executed* and control returned to the user (indicated by an arrow). Page advance is accurate only if the paper has been properly adjusted at the beginning of the STRCOMP session. LINE and PAGE are STRCOMP commands which are executed

---

*This is possible on certain Teletypes available from the New England Telephone Company.

as follows: LINE advances one line; PAGE advances to the top of a clean page.

Desk-Calculator Operations

STRCOMP can be used for computing the values of formulas. Operators are available and function as they do in arithmetic. Signs for operations are

- + addition
- - subtraction
- * multiplication
- / division
- ↑ exponentiation (raising to a power)

Operations are performed in standard order. Exponentiation is performed right to left and before any operation to its left; multiplication or division before addition or subtraction to its left. Other than that, order of operation is from left to right. Parentheses may be used to alter the order of execution. Expressions within parentheses are evaluated before those outside.

TYPE

The TYPE command is the only one necessary for desk-calculator operations.

```
←TYPE 2+2
        2+2=       4
←TYPE 22+3*(2+3)↑2
 22+3*(2+3)↑2=      97
←TYPE 2+2*4/6*3*(4-1)
 2+2*4/6*3*(4-1)=      14
```

In algebra the third statement might be 2+2X4/6X3(4-1). STRCOMP does not recognize 3(4-1) as 3 times (4-1). A multiplication sign must be part of that expression. The error comment NO GOOD, SOMETHING MISSING will print out if this or any sign is omitted.

The TYPE command may have several arguments (objects) separated by commas. Expressions are evaluated on separate lines as seen in the example. (An expression is any combination of numbers, variables, algebraic operators, and functions having a unique value.)

```
←TYPE 3+3, 6*7/4, 12-6/7
      3+3=       6
    6*7/4=      10.5
  12-6/7=      11.14286
```

In addition to numeric expressions, the TYPE command recognizes textual values. Textual values (called string values) may include any character and must be enclosed within quotation marks.

```
←TYPE "THIS IS AN EXAMPLE"
THIS IS AN EXAMPLE
```

In the following example, because the string value "2" is involved in an arithmetic operation, it is interpreted numerically.

```
←TYPE 2+"2"
    2+"2"=     4
```

PRINT

The PRINT command enables the user to format Teletype output.

Unlike the TYPE command, the PRINT command prints only the value of its argument with no spaces or carriage returns.

```
←PRINT 2+2
4
←PRINT 3+3-1*7
-1
←PRINT 3+3, 2+2
64
```

Like TYPE, the PRINT command accepts both string and numeric expressions.

```
←PRINT 3+3," IS LESS THAN ",12+3
6 IS LESS THAN 15
```

Carriage returns must be inserted into a PRINT statement when necessary. The symbol for carriage return is a number sign (#). Because the number sign is considered one of the arguments of the command, it must be separated from other arguments by commas.

```
←PRINT#, 3+3,#,"ANSWER"

6
ANSWER
```

A provision for spaces between arguments can be specified with spaces included within quotation marks.

```
←PRINT#, 3+3,"   ",2+2

6   4
←PRINT "ANSWER ",23+4
ANSWER 27
```

Line Limit

Any statement in STRCOMP is limited to the length of a Teletype line (71 characters). A statement cannot be continued on the next line.

Spaces within Expressions

Spaces may be entered *between* numeric units or words but not *within* numeric units. For example, ←TYPE 2 + 22 is acceptable but ←TYPE 2 +2 2 is not. In the latter case, STRCOMP interprets the space as the omission of an operator.

Errors

Many errors in STRCOMP are readily detected by the program. Generally, when an error is made, a statement beginning with NO GOOD or ERROR ABOVE prints out. A second line attempts to diagnose the problem.

```
←TYPE "THIS IS AN ERROR
NO GOOD
SECOND QUOTE MISSING
```

Since the comments lose their originality after the first 50 times, the user may type the command SHORT, which will cause STRCOMP to suppress all diagnostic phrases and merely print NO GOOD. If a particular error is unclear, the option to type WHY is available.

```
←SHORT
←TYPE 2 2+2
NO GOOD
←WHY
LOOKS LIKE COMMA OR OPERATOR MISSING
```

Variables

In writing programs and algebraic formulas in STRCOMP, it is necessary to establish variables to carry over intermediate results from one line to another. A variable may be established and given a value by using the SET command.

```
←SET A=1Ø
```

The variable A now has the value 1Ø.

```
←TYPE A
       A=1Ø
←TYPE 2Ø+A
        2Ø+A=     3Ø
```

The value of A will remain equal to 1Ø until A is reset or deleted. It is not necessary (or possible) to declare variable names in advance (as is done in Fortran). A variable cannot exist without a value.

The word SET is understood if it is omitted.

```
←B=5Ø
←TYPE B
    B = 5Ø
```

Variables in STRCOMP may be set equal to a string of characters.

```
←SET C = "THE ANSWER IS"
←PRINT C
THE ANSWER IS
```

A variable so defined is known as a string-valued variable. It

can be used as follows:

```
←PRINT C, " ", A+B
THE ANSWER IS 6Ø
```

(A and B having been set in previous examples).

Variables may also be truth-valued, i.e., true or false (see page 22 for a discussion of truth values).

```
←D=$T
←TYPE D
          D=TRUE
```

A variable name may be a combination of 1-6 alphabetic characters and digits provided that the first character is a letter.

```
←AB12=1ØØ
←TYPE AB12
        AB12=    1ØØ
```

Variable names may not be combinations of characters reserved for STRCOMP and ISRCOMP commands and functions. (Appendix B includes a complete list of STRCOMP and ISRCOMP commands.)

Variables and the values which have been accumulated in STRCOMP may be listed by typing the following:

```
←TYPE ALL VALUES
           A=     1Ø
        AB12=    1ØØ
           B=     5Ø
           C= THE ANSWER IS
```

A variable may be deleted by the DELETE command.

```
←B=5Ø
←DELETE B
←TYPE B
NO GOOD
THERE IS NO VARIABLE B
```

All variables are deleted by typing DELETE ALL VALUES.

Numbers

Numbers in STRCOMP range from 10^{-38} to 10^{38} or -10^{-38} to -10^{38} or 0. Numbers outside the ranges will initiate the error comment NO GOOD, NUMBER TOO LARGE. Numbers are printed in decimal notation from .001 to 99999.99. Numbers outside of this range are printed in terms of powers of 10. Numbers are rounded to seven significant digits.

```
←TYPE 1234567890
  1234567890=      1.234568*10↑9
←TYPE 11234.567890
 11234.567890=  11234.57
```

Concatenation

In addition to arithmetic operators, there is a text operator called concatenate which connects two strings into one. The symbol of concatenation is a period (.).

```
←X="GRAPE"
←Y="FRUIT"
←TYPE X.Y
         X.Y= GRAPEFRUIT
```

Numbers can also be treated as text strings for concatenation.

```
←A=2
←TYPE A.A
          A.A= 22
```

String-Valued Expressions

A string-valued expression may be a string-valued variable, quoted text, or a combination of these, including operators.

B. Programming in STRCOMP

Programs

When a problem cannot be solved with one algebraic formula, it may be necessary to use several statements (program) to obtain a solution. A program is a series of statements designed to accomplish a specific purpose. Programs are comprised of stored statements, each of which is preceded by a decimal number called a step number. Although statements need not be entered in numerical order into STRCOMP, STRCOMP executes the program in order by step number unless directed elsewhere.

Step numbers may range from 1.00000 to 99.99999. The integer part of each step number is called the part number. Generally, all steps with the same part number are designed to accomplish a specific phase in the problem.

Comments may be attached to stored statements by typing a semicolon followed by any number of characters, provided that the entire line does not exceed 71 characters.

Following is a short, one-part program written to solve a simple problem. The problem is: what percent of the population was under 20 years of age in 1958. Data from the 1958 census are used.

```
←1.1 SET AGE1=66697; NUMBER OF PEOPLE LESS THAN 2Ø
←1.2 SET AGE2=45524; 2Ø-39 YRS
←1.3 SET AGE3=39583; 4Ø-59 YRS
←1.4 SET AGE4=22263; OVER 6Ø
←1.5 TOTAL=AGE1+AGE2+AGE3+AGE4
←1.6 ANS=1ØØ*AGE1/TOTAL
←1.7 PRINT#,"PERCENT OF POPULATION UNDER 2Ø IN 1958 IS ",ANS,"%"
```

DO

Execution of a program is initiated with the DO command. When a part is specified to the DO command, STRCOMP executes all steps in that part of the program before stopping.

```
←DO PART 1
PERCENT OF POPULATION UNDER 2Ø IN 1958 IS 38.31685%
```

When a step is specified (DO STEP 1.5), only that step is executed. If DO is specified to begin in the middle of a part (DO PART 1.5), it begins there and continues to the end of the program part.

DONE

The DONE command causes STRCOMP to skip over statements in a part. When the DONE command is executed, the remaining steps in the part are skipped. Parts containing DONE commands are usually initiated

panding the previous program by inserting step 1.05 at the beginning allows the user to enter a title.

```
←1.Ø5 ACCEPT#,"TITLE ",*T
←DO STEP 1.Ø5

TITLE POPULATION STUDY
```

Now T has the value POPULATION STUDY. If no asterisk precedes the variable name, a textual answer may be entered within quotation marks. This quoted text becomes the value of the variable.

Errors made in response to DEMAND or ACCEPT may be erased with the RUBOUT key. In this case STRCOMP prints a number sign (#) and waits for a new answer. (Also, SHIFT L may be used to erase individual characters.)

During the construction of a program, a summary of the user's work is obtained by typing the direct command TYPE ALL PARTS.

```
←TYPE ALL PARTS
1.Ø5 ACCEPT#,"TITLE ",*T
1.1 TOTAL=Ø
1.2 DEMAND AGE1,AGE2,AGE3,AGE4
1.3 TOTAL=AGE1+AGE2+AGE3+AGE4
1.4 ANS=1ØØ*AGE1/TOTAL
1.7 PRINT#,"PERCENT OF POPULATION UNDER 2Ø IN 1958 IS ",ANS,"%"
```

The TYPE command may have individual parts or steps as an argument, e.g., TYPE STEP 1.4 or TYPE PART 1.

DELETE

The DELETE command enables the user to erase one step, several steps, a part, or all parts.

```
←DELETE STEP 1.1
```

The above example will delete that step only. The following statement will delete the steps and parts named.

```
←DELETE STEP 1.1, STEP 2.1, PART 3
```

The following statement will delete the step named and all following steps in that part.

```
←DELETE PART 1.5
```

An entire part can be deleted with the following command:

```
←DELETE PART 1
```

The command DELETE ALL PARTS will delete all parts. DELETE ALL will delete all parts, variables, and functions. DELETE ALL FCNS will delete any functions defined by the user (with DEFINE as described on page 50).

Subscripts

A variable name may be modified with one or more subscripts to denote successive values of that variable. Subscripts are indicated by brackets (never parentheses). In the following example, the program will determine the mean score from a group

of five test scores. The test scores are successive values of the variable S.

```
←1.1 DEMAND  S[1],S[2],S[3],S[4],S[5]
←1.2 SUM= S[1]+S[2]+S[3]+S[4]+S[5]
←1.3 AVE=SUM/5
←1.4 PRINT#, "AVERAGE IS ",AVE

←DO PART 1

       S[1]= 89.5
       S[2]= 91.5
       S[3]= 96
       S[4]= 88
       S[5]= 9Ø.5

AVERAGE IS 91.9
```

Each value may be referenced by its appropriate subscript.

```
←TYPE S[3]

       S[3]= 96
```

All values can be printed with the command TYPE ALL S.

Subscripts may consist of integers from 0 to 131070. Negative subscripts are not allowed. Fractional subscripts are rounded to the nearest integer.

Arrays, Matrices

The successive values of one subscripted variable constitute an array. An array is an ordered set of data. A one-dimensional array is comparable to a vector and a two-(or more) dimensional array to a matrix. (Up to 6 subscripts, i.e., a 6-dimensional matrix, are allowed.) The test scores assimilated in the above example constitute a one-dimensional array.

A two-dimensional array (matrix) can be constructed by assigning two subscripts to one variable. In this example L[I,F] is the subscripted variable used. (Generally, arrays are printed by varying the righthand subscript first.)

```
←TYPE ALL L
 L[1,1]=      4
 L[1,2]=      7
 L[1,3]=      3
 L[2,1]=      3
 L[2,2]=      6
 L[2,3]=      4
 L[3,1]=      2
 L[3,2]=      1
 L[3,3]=      2
```

The FOR Clause

A clause beginning with FOR is used to modify commands such as step 1.1 in the example. A FOR clause causes a statement to be repeated, assigning successive numeric and/or string values to a variable each time.

```
←1.1 DEMAND "AGE ",A[I] FOR I=1,2,3
←DO STEP 1.1
AGE
        A[1]=44
AGE
        A[2]=34
AGE
        A[3]=23
```

In the example above, the successive values of A[I] specified by 1,2,3 are entered by the user of the program.

The same series can be assigned by using a FOR clause with intervening colons. A[I] FOR 1=1:1:4 means I begins at 1 and is incremented in steps of 1 until it reaches 4.

The increment is flexible; for example, A[I] FOR I=1:3:19 meaning I begins with a value of 1 is incremented by steps of 3 until it reaches 19. Decrementing is also possible—for example, A [I] FOR I=12:-1:4 which means begin with I equal to 12, then decrement in steps of one until I equals 4.

A FOR clause (such as TYPE I FOR I=Ø:1:3) causes I to equal 1, and the command is executed. Then the value of I is compared to the final value (3, in the example). If I plus the increment (1) is greater than the final value (3), the command is not executed. The step is completed and the program proceeds.

```
←TYPE I FOR I=Ø:3:7
          I=      Ø
          I=      3
          I=      6
```

Several ranges may be grouped in a FOR clause. In the following example, I is incremented in steps of 1 until I is equal to 4, then I is incremented in steps of 2 until the final increment is reached.

```
←TYPE I FOR I=1:1:4:2:1Ø
          I=      1
          I=      2
          I=      3
          I=      4
          I=      6
          I=      8
          I=     1Ø
```

In the following example, the two methods of specification are combined.

```
←TYPE I FOR I=1,5,6:1:9,3.3,-1
          I=      1
          I=      5
          I=      6
          I=      7
          I=      8
          I=      9
          I=      3.3
          I=     -1
```

Multiple FOR clauses may be used to modify a command. FOR clauses are interpreted from right to left. In the following example, the righthand FOR clause is executed first.

```
←TYPE L⌈I.F⌉ FOR F=1:1:2 FOR I=1:1:3
      L[1,1]=     12
      L[1,2]=     22
      L[2,1]=     22
      L[2,2]=     15
      L[3,1]=     14
      L[3,2]=     21
```

By reversing the order of the FOR clauses, the statement will be executed as follows.

```
←TYPE L[I,F] FOR I=1:1:3 FOR F=1:1:2
      L[1,1]=     12
      L[2,1]=     22
      L[3,1]=     14
      L[1,2]=     22
      L[2,2]=     15
      L[3,2]=     21
```

FOR can modify all commands except GO, REPEAT, FILE, and PUNCH.

WHILE and UNTIL

WHILE and UNTIL provide more-flexible increments and decrements in FOR clauses. These two modifiers may be used only in FOR clauses.

WHILE causes iterations to be executed in specified steps as long as the relation following WHILE is *true*. (See page 22 for a full discussion of relations.)

```
←DEMAND L[I] FOR I=1:1:5
        L[1]=1
        L[2]=3
        L[3]=5
        L[4]=6
        L[5]=7
←TYPE L[I] FOR I=5:-1 WHILE L[I]>3
        L[5]=      7
        L[4]=      6
        L[3]       5
```

In contrast, UNTIL causes iterations to be executed until the relation following UNTIL is *true*, that is, as long as the relation is *false*.

```
←TYPE L[I] FOR I=1:1 UNTIL L[I]=5
        L[1]=      1
        L[2]=      3
```

TO

Ordinarily, program statements are executed in order by step number. This sequential processing may be altered by directing the program to branch to a particular part or step. The branching is effected by the TO command.

When directed to a particular step by the TO command, that step, as well as any additional steps in that part, is executed. The program does not return to the original part — it remains in the part directed until given further commands.

```
←1.1 TYPE 3+2
←1.2 TO STEP 1.4
←1.3 TYPE 6*4
←1.4 TYPE "THE END"

←DO PART 1

3+2=5
THE END
```

IF

An IF clause (like a FOR clause) modifies a STRCOMP command. When a command is modified with an IF clause, it is executed only if the expression following IF is true. If the expression is not true, the command is not executed and the program proceeds to the next step.

```
←1.1 ACCEPT "AGE ", A
←1.2 TO PART 1.5 IF A>18
←1.3 PRINT#, "YOU ARE TOO YOUNG FOR THIS QUESTIONNAIRE"
←1.4 DONE
←1.5 ACCEPT#, "NAME ", *N

←DO PART 1

AGE 19
NAME
```

Arithmetic-Comparison Operators

An expression whose value is either true or false is called a truth-valued expression. In the example above, the TO command

(in step 1.2) was executed because the truth-valued expression following IF has the value true. The expression is a comparison. Several arithmetic comparison operators are available as follows:

=	equal to
<	less than
>	greater than
><,<>	not equal to (less or greater than)
=<,<=	less than or equal to

The following series of statements illustrates the use of comparison operators in IF clauses.

```
←TYPE "EQUAL" IF 1Ø=1Ø
EQUAL
←TYPE "LESS" IF 1Ø<3Ø
LESS
←TYPE "MORE" IF 1Ø>5
MORE
←TYPE "NOT EQUAL" IF 5<>9
NOT EQUAL
←TYPE "LESS OR EQUAL" IF 5<=8
LESS OR EQUAL
←TYPE "MORE OR EQUAL" IF 7>=7
MORE OR EQUAL
```

It is possible to replace the comparison with a truth-valued variable explicitly. STRCOMP has two special symbols for true and false.

$T	True
$F	False

In the following example, the text was printed because the value of A is true.

```
←A=$T
←TYPE "THE SUN IS SHINING" IF A
THE SUN IS SHINING
←TYPE A
        A=TRUE
```

It is also acceptable to give a truth value to a variable implicitly.

```
←V= 1=2
←TYPE V
        V=FALSE
←TYPE "ERATO" IF V
```

Since V is false, the TYPE command is not executed.

CTNS — A String-Comparison Operator

In addition to the arithmetic-comparison operators described above, STRCOMP includes a string-comparison operator called CTNS (an abbreviation of the word *contains*). The CTNS operator determines whether the value of one string-valued expression is found within the value of another string-valued expression. The value of the entire expression is either true or false.

```
←TYPE "APENECK SWEENEY" CTNS "ELIOT"
 "APENECK SWEENEY" CTNS "ELIOT"=FALSE
←TYPE "ANNA LIVIA PLURABELLE" CTNS "BELL"
 "ANNA LIVIA PLURABELLE" CTNS "BELL"=TRUE
```

Execution of a command can depend on an IF clause including CTNS.

```
←TYPE "BLAKE" IF "0 ROSE, THOU ART SICK!" CTNS "ROSE"
BLAKE
←A="QUOTH THE RAVEN,'NEVERMORE'"
←TYPE "E.A.POE" IF A CTNS "RAVEN"
E.A.POE
←B="NEVER"
←TYPE "THE RAVEN" IF A CTNS B
THE RAVEN
```

The command is not executed if the comparison is false.

```
←TYPE "SOUTHEY" IF "MAISIE" CTNS "PROUD MAISIE"
```

Looping

In many programs it is necessary to repeat certain operations until a particular condition is met. This procedure is known as looping. Following is an example of a one-step loop.

```
←1.1 ACCEPT#, "AGE GROUP POPULATION", AGE[I] FOR I=1,2,3,4
```

The ACCEPT command is repeated for each value of AGE as specified.

A loop encompassing more than one step follows:

```
←1.1 SUM=Ø
←1.2 ACCEPT#, "SCORE", S
←1.25 TO PART 2 IF SCORE=Ø
←1.3 SUM=SUM+S
←1.4 TO STEP 1.2
```

In this example, the program sums the values of S until a zero (Ø) is entered. Step 1.25 terminates the loop.

DO as a Stored Command

In previous examples, DO was used as a direct command to initiate a program. DO may also be part of a program step. When that step is executed, the program branches to the part or step specified, executes that part or step, then returns to the step immediately following the DO command.

```
←TYPE ALL PARTS
1.1 ; EXAMPLE OF THE STORED 'DO'
1.2 PRINT "1.2",#
1.3 DO PART 2
1.4 PRINT "NOW I AM BACK IN PART 1, AT STEP 1.4",#
1.5 DO PART 3
1.6 PRINT "HERE I AM AT STEP 1.6",#,#

2.1 PRINT "THIS IS PART 2",#
2.2 PRINT "THIS IS THE LAST STEP IN PART 2",#

3.1 PRINT"THIS IS THE ONLY STEP IN PART 3 WHICH WILL GET DONE",#
3.2 DONE
3.3 PRINT "THIS SECTION OF PART 3 WILL NOT GET DONE BECAUSE",#
3.4 PRINT " 'DO PART 3' WAS TERMINATED BY THE 'DONE' AT",#
3.5 PRINT "STEP 3.2"

←DO PART 1
1.2
THIS IS PART 2
THIS IS THE LAST STEP IN PART 2
NOW I AM BACK IN PART 1, AT STEP 1.4
THIS IS THE ONLY STEP IN PART 3 WHICH WILL GET DONE
HERE I AM AT STEP 1.6
```

DO may be modified by FOR or IF.

Break

If a program or printout must be interrupted, depress the BREAK

key for a full two seconds. Some unusual characters will print, then TERMINATED (in the case of a direct command) or INTERRUPTED AT STEP 1.1 (in the case of program execution) will print and control will be returned to the user.

STOP and GO

A break can be programmed. The STOP command allows the user to stop a program at any point without losing work that has already been completed. The command is useful for debugging purposes (program checkout). STOP may be used as a stored command only.

```
←TYPE ALL PARTS
1.1 SUM=Ø, C=Ø
1.2 ACCEPT#,"AGE ",A
1.4 SUM=SUM+A
1.5 STOP
1.6 TO PART 1.2
```

This program is incomplete. STOP is used to test the program as far as it has gone.

```
←DO PART 1

AGE 34
STOPPED AT STEP 1.5
```

Control returns to the user after the program has been stopped. After an interruption caused by a depression of the BREAK key or a STOP command, the program may be continued by typing GO.

REPEAT

Errors that exist in programs because of oversight cause a comment to print out indicating the nature of the error. If it involves only a single step and/or ones following it, the corrections may be made at that time if the user types REPEAT. The program will begin with the step that was in error and proceed. In the following example, an error was detected by STRCOMP. The user corrected it and resumed program execution.

```
←TYPE ALL PARTS
1.1 ACCEPT#,"SCORE ",A[I] FOR I=1:1:3
1.2 SUM=Ø, AVE=Ø
1.3 SUM=SUM+A[I] FOR I=1:1:3
1.4 AVE=SUM/B
1.5 TYPE AVE

←DO PART 1

SCORE 341
SCORE 45Ø
SCORE 333
ERROR AT STEP 1.4
THERE IS NO VARIABLE B
←1.4 AVE=SUM/3
←REPEAT
          AVE=     374.6667
```

Filing

Programs and/or variables may be stored in a STRCOMP file. All entries in a file must have a hyphenated name of not more than 12 alphabetic characters (digits are not allowed). The first six letters are the file name, and the last six letters are the name of an entry in that file. The file name may be repeated with separate

entry names. The file-entry name may be followed by a semicolon and a comment. This comment will be printed out at any time that the file entry is loaded.

Confidential codes may be assigned to STRCOMP files. The code can consist of letters and digits (with a letter first), not longer than six characters, separated from the file-entry name by a colon. This code does not prevent file access. The file is protected from revision or deletion; that is, the code must be given if the file is refiled with the same file and entry name or if the file or any entry in the file is deleted.

The confidential code must be assigned the first time that the file name is used. Then all new entries to this file are automatically assigned the same confidential code. Once a file has been established, subsequent code assignments are ignored.

In the following example, only part 1 of a program is to be filed. The file VMD has not yet been created, so it is possible to assign a confidential code.

```
←FILE PART 1 AS VMD-QUES:AA1234; QUESTIONNAIRE PROGRAM
```

All variables of a program may be filed by typing the following:

```
←FILE ALL VALUES AS VMD-TABLE; RESULTS
```

Both variables and a program may be filed by typing the following:

```
←FILE ALL AS VMD-TABQUE; QUES AND ANS
```

The FILE command does not erase data from STRCOMP; it merely copies them into storage.

Loading a File Entry

After variables or a program has been stored as a file entry, the user may return the information to STRCOMP with the LOAD command. LOAD must be followed by the entire file-entry name. The confidential code is not necessary for this procedure.

```
←LOAD VMD-TABLE
RESULTS
```

The comment is printed out as the file is being loaded.

A backarrow (←) indicates that the loading has been completed.

Starting a Program

The user may execute a program without first explicitly loading it, by using the START command.

```
←START VMD-QUES
```

The file entry is loaded and the program is executed. START begins execution of the program at the part with the lowest step number loaded (not necessarily part 1).

Deleting a File

The DELETE command may be used to expunge a file entry from storage. If the file has a confidential code, it must be entered along with the file name.

```
←DELETE FILE VMD-TABLE:AA1234
```

Deleting all entries from a file does not delete the file name itself. Another command must be given.

```
←DELETE FILE VMD: AA1234
```

However, if a file name is deleted while it has many entries, all of the entries as well as the file name are expunged.

Program Examples

The following program examples combine many of the STRCOMP capabilities described up to this point.

The first program calculates cost per ounce of coke when it is sold in various combinations. Part 1 prints instructions to the user. The answer given to the question in step 1.8 determines a program branch. If the user types Y or YES, the program branches to part 2. If the answer is N or NO, the program branches to part 4.

Part 2 is the main part of the program. It requests values for each variable, then calculates the cost per ounce in the formula in step 2.6. Step 2.7 reassigns a value to MN (minimum) if the current answer (ANS) is less than the previous minimum (MN). In part 3, the PRICE PER OUNCE is printed, then the program loops back to step 2.1. The program keeps looping until the value ∅ is entered in response to the TOTAL COST question in step 2.2. When this answer is entered, the program branches to part 3.5 where the minimum is printed. Then the program goes to part 4 for the amenities.

```
←TYPE ALL PARTS
1.1 PRINT#,"PROBLEM: WHAT IS THE BEST BUY IN COKE?"
1.2 PRINT#,"ENTER VARIOUS COMBINATIONS OF COKE CASES, E.G.,"
1.3 PRINT#,"8 BOTTLES OF COKE-16 OUNCES EACH-$.Ø2 DEPOSIT ON EACH"
1.4 PRINT#,"BOTTLE-COSTS $1.Ø9"
1.5 PRINT#,#,"THE PROGRAM WILL ASK FOR DIFFERENT VARIABLES UNTIL"
1.6 PRINT#,"YOU ANSWER WITH A ZERO (Ø)."
1.7 PRINT#,"THEN IT WILL PRINT THE MINIMUM AND STOP."
1.75 MN=1Ø
1.8 ACCEPT#,ARE YOU READY TO BEGIN? ",*T
1.9 TO PART 2 IF T CTNS "Y"
1.91 TO PART 4.1 IF T CTNS "N"

2.1 TC=Ø,NB=Ø,D=Ø,BB=Ø
2.2 ACCEPT#,#,#,"TOTAL COST ",TC
2.3 TO PART 3.5 IF TC=Ø
2.35 ACCEPT#,"NUMBER OF BOTTLES ",NB
2.4 ACCEPT#,"DEPOSIT PER BOTTLE ",D
2.5 ACCEPT#,"OUNCES PER BOTTLE ",BB
2.6 ANS=(TC-NB*D)/(NB*BB)
2.7 MN=ANS IF ANS<MN
2.8 TO PART 3

3.Ø1 PRINT#
3.1 PRINT#,"PRICE PER OUNCE ",ANS
3.2 TO PART 2.1
3.5 PRINT#,#,#,"THE MINIMUM IS ",MN
3.51 TO PART 4

4.1 PRINT#,"THANK YOU"
```

The program is executed as follows.

```
←DO PART 1

PROBLEM: WHAT IS THE BEST BUY IN COKE?
ENTER VARIOUS COMBINATIONS OF COKE CASES, E.G.,
8 BOTTLES OF COKE-16 OUNCES EACH-$.Ø2 DEPOSIT ON EACH
BOTTLE-COSTS $1.Ø9

THE PROGRAM WILL ASK FOR DIFFERENT VARIABLES UNTIL
YOU ANSWER WITH A ZERO (Ø).
THEN IT WILL PRINT THE MINIMUM AND STOP.
ARE YOU READY TO BEGIN? YES
```

```
TOTAL COST 1.99
NUMBER OF BOTTLES 12
DEPOSIT PER BOTTLE .Ø2
OUNCES PER BOTTLE 12

PRICE PER OUNCE .Ø1215278

TOTAL COST .99
NUMBER OF BOTTLES 8
DEPOSIT PER BOTTLE Ø
OUNCES PER BOTTLE 6.5

PRICE PER OUNCE .Ø19Ø3846

TOTAL COST .88
NUMBER OF BOTTLES 6
DEPOSIT PER BOTTLE .Ø2
OUNCES PER BOTTLE 6.5

PRICE PER OUNCE .Ø1948718

TOTAL COST Ø

THE MINIMUM IS .Ø1215278
THANK YOU
```

The following program may be used to balance a checkbook. Part 1 allows the user to enter his starting balance (step 1.2), then enter checks and deposits in any order (step 1.3). Step 1.35 causes the program to branch to part 2 if an entry is zero (Ø). The answer to the question in step 1.4 determines whether the program will skip to step 1.8. This will occur if the answer is DEPOSIT (a positive value). The program will continue on to the next step (which will make the entry a negative value) if the answer is CHECK. Step 1.6 adds a minus sign to the entry because it is not a deposit. Step 1.8 keeps a cumulative total. When the loop is terminated with an answer (zero) to the ENTRY question, the program goes to part 2 where the CURRENT BALANCE is printed.

```
←TYPE ALL PARTS
1.1 PRINT#,"CHECK BOOK BALANCING PROGRAM"
1.13 BAL=Ø,ENTRY=Ø
1.15 MINUS="-"
1.2 ACCEPT#,"STARTING BALANCE ",BAL
1.3 ACCEPT#,"ENTRY ",ENTRY
1.35 TO PART 2 IF ENTRY = Ø
1.4 ACCEPT#,"CHECK OR DEPOSIT? ",:ANS
1.5 TO PART 1.8 IF ANS CTNS "D" OR ANS CTNS "DEP"
1.6 ENTRY=MINUS.ENTRY
1.8 BAL=BAL+ENTRY
1.9 TO PART 1.3

2.1 PRINT#,#,#,"CURRENT BALANCE IS ","$",BAL
2.2 PRINT#,#,"THANK YOU."
```

The program is executed as follows.

```
←DO PART 1

CHECK BOOK BALANCING PROGRAM
STARTING BALANCE 239
ENTRY 13.98
CHECK OR DEPOSIT? CHECK
ENTRY 1ØØ
CHECK OR DEPOSIT? DEPOSIT
ENTRY 23.45
CHECK OR DEPOSIT? CHECK
ENTRY 19.98
CHECK OR DEPOSIT? CHECK
ENTRY 39.14
CHECK OR DEPOSIT? CHECK
ENTRY 5Ø
CHECK OR DEPOSIT? DEPOSIT
ENTRY Ø

CURRENT BALANCE IS $292.45

THANK YOU.
```

The final example is part of a medical history-taking program.

Part 1 gives the respondent instructions and asks for basic information. Steps 1.36 and 1.37 initiate separate part execution if certain conditions are true. Step 1.45 causes the program to jump to part 6, bypassing some parts. Part 2 is designed for a female respondent. Many of the questions are aimed at married women. Part 3 is designed for the male respondent. In both parts, the age of the respondent is requested (step 2.001) and part 4 is executed if the respondent is under 18 years. Part 6 is the beginning of the current-illness history. Branches are determined on the basis of the one-sentence response given when the question in step 6.2 is asked.

```
←TYPE ALL PARTS
1.1 PRINT#,"GOOD MORNING.  THE COMPUTER WILL ASK YOU SEVERAL"
1.2 PRINT#,"QUESTIONS. PLEASE ANSWER THEN AS CONCISELY AS"
1.3 PRINT#,"POSSIBLE."
1.31 ACCEPT#,"ARE YOU READY TO PROCEED? ",*A
1.32 TO PART 9 IF A CTNS "N"
1.321 ACCEPT#,"PLEASE TYPE M OR F INDICATING SEX ",*S
1.33 ACCEPT#,"PLEASE TYPE YOUR FIRST NAME ",*FN
1.34 ACCEPT#,"AND YOUR LAST NAME ",*LN
1.35 ACCEPT#,"AND YOUR TITLE (MR.,MRS.,MISS) ",*T
1.36 DO PART 2 IF S="F"
1.37 DO PART 3 IF S="M"
1.45 ACCEPT#,"DO YOU WANT TO BE IN THE THROAT CLINIC? ",*TC
1.46 TO PART 6 IF TC CTNS "Y"
1.47 TO PART 9 IF TC CTNS "N"

2.ØØ1 ACCEPT#,"HOW OLD ARE YOU, ",T." ".LN." ",*AGE
2.ØØ2 TO PART 4 IF AGE < 18
2.ØØ3 TO PART 6 IF T CTNS "MISS"
2.1 ACCEPT#,"DO YOU HAVE ANY CHILDREN? ",*C
2.2 TO PART 2.3 IF C CTNS "N"
2.21 ACCEPT#,"HOW MANY GIRLS? ",G
2.22 ACCEPT#,"HOW MANY BOYS? ",B
2.23 ACCEPT#,"HAVE YOU EVER HAD A MULTIPLE BIRTH (E.G.,TWINS) ",*TW
2.24 ACCEPT#,"IS YOUR HUSBAND LIVING? ",*HU
2.25 DONE IF HU CTNS "N"
2.3 ACCEPT#,"HOW LONG HAVE YOU BEEN MARRIED? ",*MA
```

```
3.1 DO STEP 2.ØØ1
3.2 TO PART 4 IF AGE < 18
3.3 ACCEPT#,"ARE YOU MARRIED? ",*MAR
3.4 DONE IF MAR CTNS "N"
3.5 ACCEPT#,"IS YOUR WIFE LIVING? ",*WI
3.6 DONE IF WI CTNS "N"
3.7 DO STEP 2.3

4.1 PRINT",,"THE PEDIATRIC CLINIC IS NEXT DOOR. ALL PATIENTS"
4.2 PRINT#,"UNDER 18 MUST GO THERE FOR TREATMENT."
4.3 DONE

6.1 PRINT#,#,#,"THIS IS THE THROAT CLINIC"
6.2 PRINT#,"PLEASE DESCRIBE YOUR ILLNESS IN ONE SENTENCE ".T." ".LN
6.3 ACCEPT#," ",*ILL
6.4 TO PART 7 IF ILL CTNS "IRR" OR ILL CTNS "RED"
6.41 TO PART 7 IF ILL CTNS "HUR" OR ILL CTNS "ACH"
6.5 TO PART 6.8 IF ILL CTNS "COU" OR ILL CTNS "SPI"
6.51 TO PART 6.8 IF ILL CTNS "SPU" OR ILL CTNS "CHO"
6.6 TO PART 6.9 IF ILL CTNS "LUM" OR ILL CTNS "SWA"
6.8 ACCEPT#,"DO YOU SMOKE? ",*SM
6.81 DONE; INCOMPLETE PROGRAM
6.9 ACCEPT#,"DOES THIS APPEAR ONLY AT MEALTIME? ",*ME
6.91 DONE; INCOMPLETE PROGRAM

7.1 ACCEPT#,"HOW LONG HAVE YOU HAD THIS SYMPTOM ",T." ".LN." ",*AA
7.2 DONE; INCOMPLETE PROGRAM

9.1 PRINT#,"THANK YOU"
```

The program is executed as follows.

```
←DO PART 1

GOOD MORNING.  THE COMPUTER WILL ASK YOU SEVERAL
QUESTIONS.  PLEASE ANSWER THEM AS CONCISELY AS
POSSIBLE.
ARE YOU READY TO PROCEED? YES
PLEASE TYPE M OR F INDICATING SEX M# F
PLEASE TYPE YOUR FIRST NAME HARRIET
AND YOUR LAST NAME CRONIN
AND YOUR TITLE (MR.,MRS.,MISS) MRS
HOW OLD ARE YOU, MRS CRONIN 35
```

```
DO YOU HAVE ANY CHILDREN? YES
HOW MANY GIRLS? 1
HOW MANY BOYS? 1
HAVE YOU EVER HAD A MULTIPLE BIRTH (E.G.,TWINS) NO
IS YOUR HUSBAND LIVING? YES
HOW LONG HAVE YOU BEEN MARRIED? 5 YEARS
DO YOU WANT TO BE IN THE THROAT CLINIC? YES

THIS IS THE THROAT CLINIC
PLEASE DESCRIBE YOUR ILLNESS IN ONE SENTENCE MRS CRONIN
 I HAVE TROUBLE SWALLOWING
DOES THIS APPEAR ONLY AT MEALTIME? NO
```

C. Boolean Operators, Mathematical Functions, and String-Manipulating Functions

1. Boolean Operators

The IF clause can be expanded by using Boolean operators between comparisons. The Boolean operators available are

AND

OR

NOT

The AND operator combines two truth-valued expressions to be evaluated. (Appendix C contains truth tables used in determining truth values.) The command illustrated in the following example was executed because both the first *and* the second expressions were true (the condition was met). If one or neither of the comparisons were true, the command would not have been executed.

```
←TYPE "ROMEO AND JULIET" IF 1Ø=1Ø AND 8=8
ROMEO AND JULIET
```

It is possible to test whether either the first *or* the second expression (or both expressions) is true by using the OR operator. In the following example, the second comparison is false but the first one is true, so the TYPE command is executed.

```
←TYPE "ONE OR THE OTHER" IF 1Ø=1Ø OR 3=2
ONE OR THE OTHER
```

NOT is the negative operator; i.e., the command is executed if the condition is found to be *not true* (false). The condition following NOT can be a single comparison.

```
←TYPE "THE ALBATROSS" IF NOT 1Ø<8
THE ALBATROSS
```

Two comparisons connected by a Boolean operator may be included after the NOT operator, but the comparisons must be placed within parentheses.

```
←TYPE "SARTOR RESARTUS" IF NOT (1Ø=1Ø AND 8<6)
SARTOR RESARTUS
```

Since the comparisons within parentheses (in the statement above) are not true, (only one of the comparisons is true), the command is executed. When the comparisons within the parentheses are true the command is not executed.

```
←TYPE "BEDDOES" IF NOT(1Ø=1Ø AND 8=8)
```

If OR is used within parentheses, then the command is executed if only one side of the operator is false.

```
←TYPE "MORTE D'ARTHUR" IF 1Ø>8 AND (3Ø=2Ø OR 15<3Ø)
MORTE D'ARTHUR
```

The command in the following example was executed because the first comparison (10>8) is true and the clause following AND is also true.

```
←TYPE "TENNYSON" IF 10>8 AND (30=20 OR 15<30)
TENNYSON
```

The numbers used in these examples would probably be expressed as variables in practice. Following is the first part of a questionnaire program illustrating the use of comparison and Boolean operators.

```
←TYPE ALL PARTS
1.1 M=18
1.2 ACCEPT#,"AGE ",AGE
1.3 TO PART 2 IF AGE <M
1.4 ACCEPT#,"HOURS OF TELEVISION ",T
1.5 DO PART 3 IF T>1 AND T<4

2.1 DONE

3.1 PRINT#,"THE PROGRAM JUMPED TO PART 3"

←DO PART 1

AGE 23
HOURS OF TELEVISION 3
THE PROGRAM JUMPED TO PART 3
```

2. Mathematical Functions

STRCOMP includes a group of standard mathematical functions which are predefined and internally stored. When a function

is applied to a numeric expression, it is itself considered a numeric expression with a new value. Finding the square root of a number is a familiar task. In STRCOMP, the square root is calculated by the following operation.

```
←TYPE SQRT(64)
   SQRT(64)=      8
```

The calculation is done by the computer.

To obtain the negative square root of a number, the command is as follows.

```
←TYPE -SQRT(64)
  -SQRT(64)=      -8
```

Several similar functions are defined in STRCOMP. In the following list, A, B, and C are used to mean any numeric expression [e.g., SQRT(A) means the square root of the value of A].

Basic mathematical functions include

SQRT(A)	positive square root of A
LOG(A)	base 10 logarithm of A
LN(A)	natural logarithm of A
EXP(A)	e to the power of A
SIN(A)	sine of A (A is assumed to be in radians)
COS(A)	cosine of A (A is expressed in radians)
ATN(A)	arc tangent of A (result in radians)
ATN(A,B)	arc tangent of A/B (result in radians)

Since trigonometric functions require that angles be expressed

in radians, the numerical constant pi (π) is frequently used in working with these functions. The value of pi is permanently stored in STRCOMP. It may be referenced by typing $PI.

```
←TYPE $PI
          $PI=       3.141593
```

The natural-logarithm base is also stored. It may be referenced by typing $E.

```
←TYPE $E
          $E=        2.718282
```

Number-dissection functions include

IP(A)	integer part of A
FP(A)	fractional part of A
XP(A)	exponent part of A
DP(A)	digital part of A
SGN(A)	algebraic sign (-1,Ø,+1)
'A'	absolute value of A

Following are examples of the above.

```
←A=12.345
←TYPE IP(A), FP(A), XP(A), DP(A), SGN(A)
       IP(A)=     12
       FP(A)=       .345
       XP(A)        1
       DP(A)=       1.2345
      SGN(A)=       1
```

Other functions include

RAN(A)	pseudorandom number between Ø and A; typing RAN(Ø) reinitializes the RAN function
MIN(A,B,C)	minimum of a list
MAX(A,B,C)	maximum of a list

Following are examples of the above.

```
←TYPE RAN(1Ø), MIN(2,4,1,5), MAX(2,4,1,5)
     RAN(1Ø)=       5.ØØØ229
MIN(2,4,1,5)=        1
MAX(2,4,1,5)=        5
```

3. String-Manipulating Functions

In addition to the mathematical functions previously described, several string-manipulating functions are available in STRCOMP. All of these functions operate on string-valued expressions.

String Length

To determine the number of characters in a string-valued expression, the string-length (STL) function is used. The value of STL(A) is always numeric.

```
←TYPE STL("I WANDERED LONELY AS A CLOUD")
 STL("I WANDERED LONELY AS A CLOUD")=     28
←TYPE STL(12.333)
 STL(12.333)=        6
←A="IT IS A BEAUTEOUS EVENING"
←TYPE A IF STL(A)>1Ø
           A= IT IS A BEAUTEOUS EVENING
←TYPE A IF STL(A)>5Ø
←TYPE STL(A)
      STL(A)=     25
```

The Number Function

The number (NUM) function determines whether a string-valued expression can be used as a number, i.e., if it is a combination of sign, digits, decimal points, exponent, etc.

The return from the NUM(A) function is either true or false. The return is not the value of the expression.

```
←TYPE NUM(3*7↑4*8+3)
 NUM(3*7↑4*8+3)=TRUE
←TYPE NUM("ABACUS")
 NUM("ABACUS")=FALSE
←TYPE NUM("4 SCORE AND 7 YEARS AGO")
 NUM("4 SCORE AND 7 YEARS AGO")=FALSE
←TYPE "ADONAIS" IF NUM(34.33)=$T
ADONAIS
```

String Verification

The string-verifying (OK) function determines whether the string-valued expression is a numeric expression, a string-valued variable, a simple truth-valued expression, or an invalid expression (none of the above). The OK function returns another value for Indeterminate Field, which applies only in ISRCOMP and will be discussed in that section.

The value of OK(A) is numeric ranging from 0 to 4. If the argument (A) is a valid numeric expression, the value of OK(A) is 1.

```
←TYPE OK(23+33)
   OK(23+33)=       1
←A=23.44*45
←TYPE OK(A)
        OK(A)=      1
```

If the argument of OK is a string-valued variable, the value of OK(A) is 2. The variable itself must be within quotation marks. If the variable is used without quotation marks, the result will be zero (invalid expression). A textual string within quotation marks is also an invalid expression.

```
←B="CHILDE HAROLD'S PILGRIMAGE"
←TYPE OK("B")
     OK("B")=      2
←TYPE OK(B)
       OK(B)=      Ø
←TYPE OK("APRIL")
 OK("APRIL")=      Ø
```

A valid truth-valued expression gives OK(A) the value 3. The expression to be evaluated must be within quotation marks.

```
←A=$T, B=1, C=3
←TYPE OK("A")
     OK("A")=      3
←TYPE OK("B=1")
   OK("B=1")=      3
←TYPE OK("$T")
     OK("$T")=     3
```

The value 4 is returned when the argument is an Indeterminate Field. (To be discussed in the ISRCOMP section.)

String Evaluator

The string-evaluator (EVAL) function evaluates its argument and prints the value of A. This function is used for indirect addressing. The argument of EVAL may be any string-valued ex-

pression. For example, if the value of the variable A is the string MARS and the value of the variable MARS is the string VENUS, the value of EVAL(A) is VENUS.

```
←A="MARS"
←MARS="VENUS"
←TYPE EVAL(A)
     EVAL(A)= VENUS
```

The argument may also be a numeric expression. The expression is evaluated and its value is printed.

```
←X=4,Y=46,Z=34
←TYPE EVAL(X*Y*Z)
 EVAL(X*Y*Z)=   6256
←TYPE EVAL(X)
     EVAL(X)=      4
←TYPE "FRA LIPPO LIPPI" IF EVAL(X+Y)<=50
FRA LIPPO LIPPI
```

Alphabetical Verifier

The alphabetical-verifying (ALPH) function determines whether two string-valued expressions are in alphabetical order. The value of ALPH(A,B) is either true or false.

```
←A="ARNOLD", B="RUSKIN", C="WILDE"
←TYPE ALPH(A,C)
   ALPH(A,C)=TRUE
←TYPE ALPH(C,B)
   ALPH(C,B)=FALSE
←TYPE "MATTHEW" IF ALPH(A,B)
MATTHEW
```

Single quotation marks precede the letter A in alphabetizing.

```
←D="'QUOTATION'"
←TYPE ALPH(D,A)
   ALPH(D,A)=TRUE
←TYPE ALPH(A,D)
   ALPH(A,D)=FALSE
```

If the expressions are equal, the value of ALPH(A,B) is true.

```
←TYPE ALPH(A,A)
   ALPH(A,A)=TRUE
```

String-Locating Function

The locating (LOC) function indicates the location of one character string within another character string. The LOC function requires two arguments: the first is the string to be searched; the second is the string for which the function is searching. The LOC function returns a number as its value. This number corresponds to the first character position *after* the last character position of the requested string.

```
←S="LINES WRITTEN IN KENSINGTON GARDENS"
←TYPE LOC(S,"W")
  LOC(S,"W")=       8
←TYPE LOC(S,"IN")
 LOC(S,"IN")=       4
←TYPE LOC(S,"N IN")
 LOC(S,"N IN")=     17
```

If the second argument (character string to be found) is not found within the first argument (character string to be searched), the value of LOC(A,B) is Ø.

```
←TYPE LOC(S,"WATER")
 LOC(S,"WATER")=       Ø
```

The value of LOC(A,B) is always first character position after the last character position of the located segment where it has *first occurred*.

```
←D="ONE TWO THREE ONE TWO THREE"
←TYPE LOC(D,"ONE")
 LOC(D,"ONE")=       4
←TYPE LOC(D,"E ONE")
 LOC(D,"E ONE")=     18
```

Extracting Segments

While the LOC function returns the location of a specified string-valued expression as a number, the EXT function returns the character(s) as a string value. The EXT function may have two arguments: the first argument is a string-valued expression; the second is a numeric expression specifying a character location in the first expression.

```
←A="IN A SOLITUDE OF THE SEA"
←TYPE EXT(A,4)
    EXT(A,4)= A
←TYPE EXT(A,6)
    EXT(A,6)= S
```

The value of EXT(A) is null (a string of 0 characters) if the argument of EXT is larger than the number of characters in the string to be searched.

```
←TYPE EXT(A,55)
   EXT(A,55)=
```

Three arguments may be used with EXT to specify a segment to be extracted from a string-valued expression. The first argument is still a string-valued expression, the second is the

first character position of the segment to be extracted, and the third argument is the last character position of the segment to be extracted. The third argument may be larger than the total number of characters.

```
←Q="FIEND, I DEFY THEE!"
←PRINT EXT(Q,8,72)
I DEFY THEE!
←PRINT EXT(Q,8,13)
I DEFY
```

LOC and EXT can be combined to extract a section of text by specifying the first character and indicating the end of the extraction with another LOC operation.

```
←B="JANUARY 23, 1968 IS A SUNDAY, NOT A MONDAY"
←PRINT EXT(B,LOC(B,","),LOC(B,",")+4)
 1968
```

Decode-Date Function

The decode-date (DDT) function converts a date (string-valued expression) into the number of days since January 1, 1849.

```
←TYPE DDT("3/3/68")
 DDT("3/3/68")=  43525
←TYPE DDT("3/4/1868")
 DDT("3/4/1868")=   7ØØ2
```

It is necessary to decode a date whenever arithmetic operations on that date are planned; e.g., to find the age of a student, his date of birth may be subtracted from today's date and that answer divided by the number of days in the year to result in age in years.

```
←AGE=(DDT("3/5/68")-DDT("2/4/1945"))/365.25
←TYPE AGE
          AGE=     23.ØØØØ8
```

The COND Function

A series of expressions, separated by commas, comprises the argument of the COND function. The odd-numbered expressions (first, third, etc.) must be truth-valued expressions. The COND function evaluates each expression in the argument. When the first true, odd-numbered expression is encountered, the COND function returns the value of the expression following that one.

In the following example, the first odd-numbered expression is true, so the value of COND(A,B,C,D) is the value of W.

```
←W="O WILD WEST WIND"
←T="HAIL TO THEE, BLITHE SPIRIT!"
←TYPE COND(1=1,W,2=2,T)
 COND(1=1,W,2=2,T)= O WILD WEST WIND
```

The first odd-numbered expression is false in the next example but the second odd-numbered expression is true, so the value of COND(A,B,C,D) is the value of T.

```
←TYPE COND(2=3,W,3=3,T)
 COND(2=3,W,3=3,T)= HAIL TO THEE, BLITHE SPIRIT!
```

An error will result if one of the arguments is not a true truth-valued expression. To avoid this in programming, it may be use-

ful to add $T (true) or a variable with that value onto the series of expressions, followed by a string-valued expression.

```
←TYPE COND(2=3,W,3=4,T)
NO GOOD
COND FUNCTION WITH NO TRUE LOGICAL EXPRESSION
←TYPE COND(2=3,W,3=4,T,A,"NO TRUE EXPRESSION") FOR A=$T
COND(2=3,W,3=4,T,A,"NO TRUE EXPRESSION")= NO TRUE EXPRESSION
```

4. Defining Functions

A new function may be defined in STRCOMP. The procedure is similar to that performed by SET, but DEFINE is *not* optional. The new function name directly follows DEFINE and the components of the function follow.

```
←DEFINE TAN(A)=SIN(A)/COS(A)
←TYPE TAN(60)
     TAN(60)=        .320040
←DEFINE EXTNEW(A,B,C)=EXT(A,LOC(A,B),LOC(A,B)+C)
←X="JANUARY 23, 1968"
←Y=","
←Z=4
←TYPE EXTNEW(X,Y,Z)
 EXTNEW(X,Y,Z)=  1968
```

In the above examples A, B, and C are used as dummy variables; i.e., these characters do not interact with any variables previously set in the program.

A function (other than standard STRCOMP functions) may be deleted by the DELETE command.

```
←DELETE FCN TAN
```

All new functions may be deleted by typing ←DELETE ALL FCNS.

The TYPE command may be used to print the string value of a new function (e.g., ←TYPE FCN TAN) or the values of all new functions (e.g., ←TYPE ALL FCNS).

D. Advanced Techniques

The following section describes techniques recommended for the advanced user.

Editing Functions

The STP and FN functions enable the user to access the text of a specific step or function and to set variables to that text.

```
←T=STP(1.1)
←TYPE T
            T= 1.1 TO STEP 1.4 IF B=17
←P=FN("EXTNEW")
←TYPE P
            P= DEFINE EXTNEW(A,B,C)=EXT(A,LOC(A,B),LOC(A,B)+C
```

The user may now apply string-manipulating functions to the variables in order to change their values. The altered value (edited step or function) may be reset by using the DO command as described on page 52.

The Backarrow Operator

The backarrow (←) operator enables the user to establish variable names or expressions (within arguments to commands) within

a program. A backarrow causes the string value of the argument following it to be used in place of it in a statement.

```
←X="+2"
←TYPE 2←X
        2←X=      4
←PRINT 3←OP 4, #FOR OP="+","*","/"
7
12
.75
```

Text strings may be addressed indirectly with the backarrow. In the following example, A represents the value of ANS where ANS has the value of the text string I HATE RAIN.

```
←ANS="I HATE RAIN"
←A="ANS"
←PRINT#,"WHAT DO YOU MEAN BY "."'".←A."'?"

WHAT DO YOU MEAN BY 'I HATE RAIN'?
```

The EVAL function is similar to the backarrow operator (see page 44 for description of the EVAL function).

DO Revisited

The DO command contains some sophisticated features not previously mentioned. In addition to initiating the execution of a step or part, DO will execute a string-valued argument (provided that the value of the argument is an acceptable STRCOMP command).

```
←X="PRINT A*37,# FOR A=1:1:3"
←DO X
37
74
111
```

If the string value of the argument begins with a step number, it becomes part of the program. A step number may be attached to the argument by using the concatenation operator.

```
←DO "2.72 ".X
```

In this example, the value of X becomes step 2.72 in the program and is executed at the appropriate time. The new step remains in the program until deleted.

DO may be modified by conditional phrases within a program.

```
←1.3 DO "1.41 TO STEP 1.1 IF A>3Ø" IF ANS="YES"
←DO STEP 1.3 FOR ANS="YES"
←TYPE STEP 1.41
1.41 TO STEP 1.1 IF A>3Ø
```

In this example, step 1.41 will not be part of the program until the condition (ANS="YES") is true.

Another example of this type of execution follows (X has been defined above).

```
←DO STAR FOR STAR="2.72 ".X
←TYPE STEP 2.72
2.72 PRINT A*37,# FOR A=1:1:3
```

INDEX Command

The INDEX command enables the user to determine the subscripts of each existing element in an array in sequence. This is quite useful in sparse arrays. Before the INDEX command is executed, the indexing variables are set to reference any element. The INDEX

command references the next existing element in the array and increments the indexing variables to the values of the subscripts of this element. Then the program executes any following steps which may act on that element in the array.

INDEX is modified by IN, whose argument denotes the array name. The INDEX command must contain the same number of subscripts as are assigned to the variable name. INDEX must be part of a loop to be effective.

The values in the following example are as follows.

```
←DEMAND L[I] FOR I=1:1:3
        L[1]=12
        L[2]=23
        L[3]=1Ø
```

The following example is actually an infinite loop. When the last element in the array is reached, the INDEX command cannot go any further. Another command must be added to the statement to provide for loop termination.

```
←TYPE PART 1
1.1 I=Ø
1.2 INDEX I IN L
1.3 TYPE L[I]
1.4 TO PART 1.2

←DO PART 1
        L[1]=     12
        L[2]=     23
        L[3]=     1Ø
        L[3]=     1Ø
        L[3]=     1Ø
INTERRUPTED AT STEP 1.3
```

ELSE is a command which is executed only if the INDEX command can no longer access new variables; i.e., the end of the array has been reached. In the following example, an array (made from the subscripted variable L[I,F]) exists as follows.

```
L[1,1]=       6
L[1,2]=       7
L[2,1]=       5
L[2,2]=       8
L[3,2]=       4
L[3,3]=       6
```

The following program will print the value of L[I,F] only if it is greater than 5.

```
*TYPE PART 1
1.Ø5 I=Ø,F=Ø
1.1 INDEX I,F IN L ELSE TO PART 2
1.2 TYPE L[I,F] IF L[I,F]>5
1.3 TO STEP 1.1

2.1 PRINT #,"END OF PROGRAM"

*DO PART 1
      L[1,1]=      6
      L[1,2]=      7
      L[2,2]=      8
      L[3,3]=      6

END OF PROGRAM
```

The IS Operator

The IS operator is a truth-valued operator which matches a string expression with a syntactic pattern. (In certain cases, evaluation of an IS operation also assigns new values to variables as directed by a pattern.)

The components of a pattern are text-describing symbols, concatenation symbols (.), logical "or" symbols (!), and subordinating parentheses. A pattern may consist of a single text-describing symbol or the concatenation of two or more such symbols. Further, a pattern may consist of the alternation of two or more such patterns. Any pattern may be enclosed in parentheses and used as a text-describing symbol in forming patterns. There are two kinds of text-describing symbols: string-valued expressions and number sign (#) with optional modifiers.

The evaluation process consists of attempting to match the subject of IS to the pattern, testing each alternative, left to right, until a match is found or all alternatives have failed to match. Any string expression or string-valued variable may be matched with an identical expression or a variable of the same value.

String-valued expressions in patterns must be matched character for character.

```
←TYPE "EXPRESSION" IS "EXPRESSION"
 "EXPRESSION" IS "EXPRESSION"=TRUE
←A="VERNAL EQUINOX"
←TYPE A IS "VERNAL EQUINOX"
 A IS "VERNAL EQUINOX"=TRUE
←TYPE "VERNAL EQUINOX" IS A
 "VERNAL EQUINOX" IS A =TRUE
```

The number sign is used to indicate any text string of zero or more characters when used after the IS operator. In the following example, A has the value of HE IS 25 YEARS OLD.

```
←TYPE "SATISFACTORY" IF A IS #
SATISFACTORY
```

A number sign may be used in conjunction with a string expression to match a key word in the middle of a string.

```
←B="HE IS 27 YEARS OLD."
←TYPE "CORRECT" IF B IS #."YEARS".#
CORRECT
```

A number sign will match any of a class of expressions, depending on the modifiers. There are three types of modifiers, any or all of which may be used in any given instance: category, length, and assignment modifiers.

Whereas the number sign without category modifier matches any text characters, it may be followed by A (alphabetic), D (numeric digits), B (blanks and carriage returns), or O (all other characters). The number sign followed by any combination of these category modifiers matches a string with a mixture of the respective categories.

Alphabetic characters may be matched as follows.

```
←TYPE "LETTERS" IF "ABC" IS #A
LETTERS
```

Digits are matched by appending a D to the number sign.

```
←TYPE "DIGITS" IF 123 IS #D
DIGITS
```

Blanks are matched with a B.

```
←TYPE "BLANK" IF " " IS #B
BLANK
```

Other signs are matched with the modifier O.

```
←TYPE "OTHERS" IF "&%$@" IS #O
OTHERS
```

Without length modifiers, the number sign matches text of any length (including zero length). The number sign may be followed by a numeric-valued expression within parentheses, such as (5) or (A), in which case it will match only text of the specified length.

```
←BX="LITTLE TROTTY WAGTAIL"
←C=21
←TYPE "CLARE" IF BX IS #(C)
CLARE
```

The number sign may be followed by a variable name within square brackets, such as [X], in which case the text which matches the number sign will be assigned as the value of the variable. The assignment is made only if that element of the pattern is interpreted and found to match one of the alternatives.

```
←B="MATCH"
←1.1 TO STEP 1.4 IF B IS #[X]
←1.2 DEMAND NET1
←1.3 TO PART 2
←1.4 TYPE X
←DO PART 1
          X= MATCH
```

If more than one of the above types of modifiers is used on a given sign, all category modifiers must appear first, followed

by a length modifier (if any), followed by the assignment modifier (if any).

```
←TYPE PART 4
4.2 PRINT#,"WHAT DO YOU THINK OF THE WORLD WIDE"
4.3 PRINT#,"WICKET COMPANY'S ADVERTISING?"
4.4 ACCEPT#,*ANS
4.5 TO PART 4.9 IF ANS IS #.("ANNOY"!"BAD"!"IRRIT").#
4.6 TYPE "SORRY"
4.9 PRINT#,"THIS IS PART 4.9"

←DO PART 4

WHAT DO YOU THINK OF THE WORLD WIDE
WICKET COMPANY'S ADVERTISING?
I FIND IT VERY ANNOYING
THIS IS PART 4.9
```

In the above, if the respondent types in an answer such as I FIND IT VERY ANNOYING, the program checks the entire text string to find that I FIND IT VERY meets the any text string requirement, ANNOY matches one of the literal text strings, and ING also meets the any text string requirement.

In blocks of textual data, leading spaces may cause problems in manipulation. The IS operator enables the user to delete all leading spaces with a step similar to the following.

```
←X="  ROSEBUD"
←1.1 TO STEP 1.1 IF X IS " ".#[X]
```

For example, the value of X may be a string which has two leading spaces. The step will be executed thrice because the test specified by IS " ".#[X] will remove one leading space from X until the value of X begins with a nonspace character. When this occurs, the condition is not met; the step is not completed, but goes on

to the next step. The result of the execution of step 1.1 is X=ROSEBUD.

FORM

The FORM function refers to the format of a single value, rather than several values in table form (as it does in TELCOMP). Two functions FORM(A), and FORM(B,FORMAT) are available.

FORM(A) converts the value of a number to a string in standard format 20 characters long. FORM(A) is useful for specific types of tables.

```
←PRINT FORM(I) FOR I=1:1:3
      1                    2                    3
```

The object of FORM must be a variable if it is to be useful. The command may be modified with a FOR clause, as above. The maximum number of columns is four. A table may be requested as follows.

```
←PRINT FORM(I), FORM(I↑10), FORM(I↑2Ø),#FOR I=1Ø,15,2Ø
     1Ø                   1*1Ø↑1Ø              1*1Ø↑2Ø
     15                   5.766504*1Ø↑11       3.325257*1Ø↑23
     2Ø                   1.Ø24*1Ø 13          1.Ø48576*1Ø 26
```

FORM(B,FORMAT) converts a number to a string value in a format specified in that function by the user. Number signs are used to specify format.

```
←PRINT FORM(99.673,##.#)
99.7
```

When indicated, the number is rounded, not truncated.

```
←PRINT FORM(99.679,##)
100
```

Formats may include minus, plus, and exponentiation signs.

```
←PRINT FORM(58.123,+##)
+58
←PRINT FORM(-123.456,-###)
-123
```

Exponentiation symbols must occur exactly three or seven times together — no other combination is valid. The exponentiation signs indicate that the value is a decimal number multiplied by 10 raised to a power (indicated by righthand digits and sign).

```
←PRINT FORM(123.456,#####↑↑↑↑↑↑↑)
12346*10↑-02
←PRINT FORM(123.4567,#↑↑↑)
1+02
```

Tabular Printout

STRCOMP includes a flexible means of tabular printout. The following are required:

(1) a step to set the value of the row variable,

(2) a branch statement that will be executed when all rows have been printed,

(3) the actual printing statement including a FORM function,

(4) a carriage return after each row,

(5) another step to increment the value of the row variable,

(6) an initiating statement.

The initiating statement and the value of the row-variable increment may be combined. All other parts of the tabulating segment must be separate statements.

For example, assume that the values of L[I,F] as shown below are the result of an opinion poll. (A program has obtained the values listed there.) The following segment may be appended to obtain a tabular printout *provided that the answers are single digits*.

```
←TYPE PART 1
1.85 F=1
1.9 TO PART 2 IF F=4
1.91 PRINT "   ",L[I,F] FOR I=1:1:3
1.92 PRINT #
1.93 DO PART 1.9 FOR F=F+1
```

The PRINT statement (step 1.91) provides for spaces between columns.

The values obtained in the opinion poll are as follows.

```
L[1,1]=        1
L[1,2]=        2
L[1,3]=        3
L[2,1]=        4
L[2,2]=        5
L[2,3]=        6
L[3,1]=        7
L[3,2]=        8
L[3,3]=        9
```

The tabular printing segment as shown above will cause these results to be printed as follows.

```
←DO PART 1
   1     4     7
   2     5     8
   3     6     9
```

If the values are unknown, it is useful to use the FORM function to standardize output. Step 1.91 may be revised to accomplish this.

```
←1.91 PRINT "    ", FORM(L[I,F],###) FOR I=1:1:3
```

The values of L[I,F] are now as follows.

```
L[1,1]=      12.4
L[1,2]=      13.66
L[1,3]=     123
L[2,1]=     133.23
L[2,2]=      12.3
L[2,3]=       1
L[3,1]=      23.1
L[3,2]=      1Ø.99
L[3,3]=      23.2
```

To obtain a tabular printout using the new values, the same segment with the revised step 1.91 is executed. The results are as follows.

```
 12    133    23
 14     12    11
123      1    23
```

The TAB Function

The TAB function provides for more flexibility in formatting printouts. TAB advances the carriage to the space number specified in its argument. For example, TAB 2Ø advances the carriage to the 20th character position from the lefthand margin. (TAB 2Ø does *not* advance the carriage 20 character positions from its

current location.) In the following example, TAB has replaced the literal spaces, as seen in step 1.91 of the previous example. A title is added and the table is centered on the page by using the new program.

```
←TYPE PART 1
1.1 F=1, T=9, B=2Ø
1.15 PRINT TAB 22, "TABLE OF RESULTS",#,#
1.2 TO PART 2 IF F=4
1.3 PRINT TAB B
1.4 PRINT TAB T, FORM(L[I,F],###) FOR T=T+9 FOR I=1:1:3
1.5 PRINT #
1.6 DO PART 1.2 FOR F=F+1 FOR T=9

2.1 STOP
```

The values of L[I,F] are taken from the previous example. The result is shown below.

```
←DO PART 1

                    TABLE OF RESULTS

                   12    133      23
                   14     12      11
                  123      1      23
STOPPED AT STEP 2.1
```

Plotting Graphs

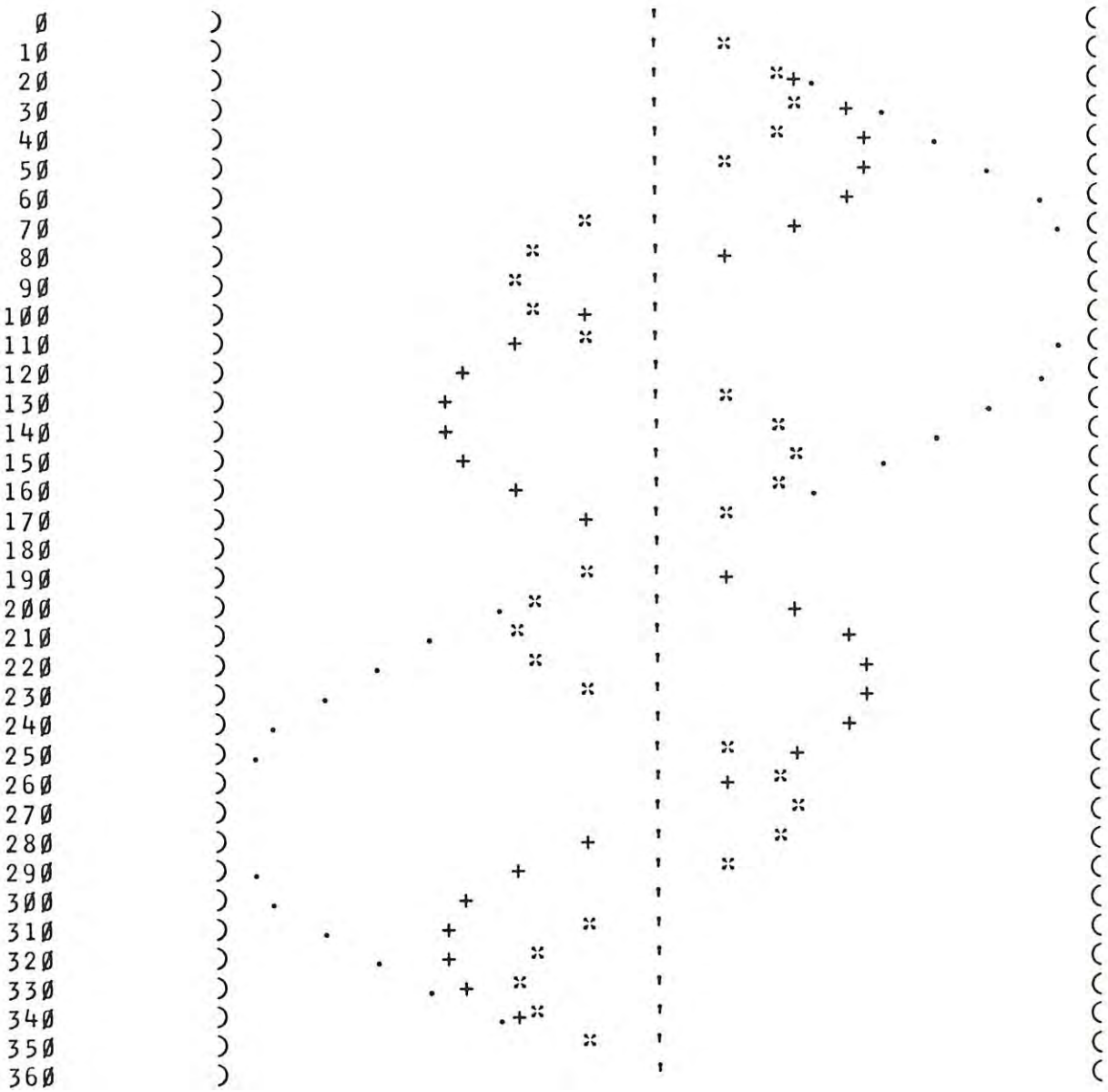
The PLOT command provides for graphing results of computations. The user can plot any number of variables, numbers, or expressions (provided that they fit on one line).

```
←PLOT SIN(X*$PI/18Ø) FOR X=Ø:15:36Ø
```

In the curve above, what is normally regarded as the *Y* axis runs horizontally, while the normal *X* axis is vertical. To provide a scale for the *X* axis (vertical), the PLOT command can be modified with an ON clause.

The ON clause specifies the scale increment of the *X* (vertical) axis. The scale is stated in three parts with intervening colons: the minimum, the increment, and the maximum. In the following example, DEG begins at Ø, is incremented by 1Ø for each vertical point on the graph, and ends at 36Ø.

```
←1.1 SET R=DEG*$PI/180
←1.2 PLOT SIN(R), SIN(2*R)/2,SIN(3*R)/3, -1,1,0 ON DEG
←2.1 DO PART 1 FOR DEG=0:10:360
←DO PART 2
```



Values are plotted with Teletype characters whose precedence is from left to right as follows.

. + *) (' ^ % $ # " !] [ZYXWVUTSRQ

It is possible to choose symbols by inserting a number sign (#) to omit certain symbols.

```
←PLOT -1,#,-.5,#,#,Ø,#,#,#,.5,#,#,#,#,1
.                 *           '           #           Z
```

STRCOMP plots horizontally on the range -1 to +1. Values must be plotted within this range. This restriction is not as limiting as it sounds. Values are scaled to conform to this format. If the values of P are known to be between zero and some maximum (MX), a graph of the values may be obtained by plotting the value of PA, where PA=P/MX.

Since Ø is in the middle of the graph, scaling the horizontal axis of a graph on Ø to a maximum results in values plotted only on one-half of the page. To distribute the values over the entire page, PA must be redefined as follows.

```
←PA=2*(P/MX)-1
```

The scaling definitions can be made even more general by determining the values to be between a minimum and a maximum.

The following definition of PA ensures that this range fills the

page for any values of MX and MN (provided that MX>MN).

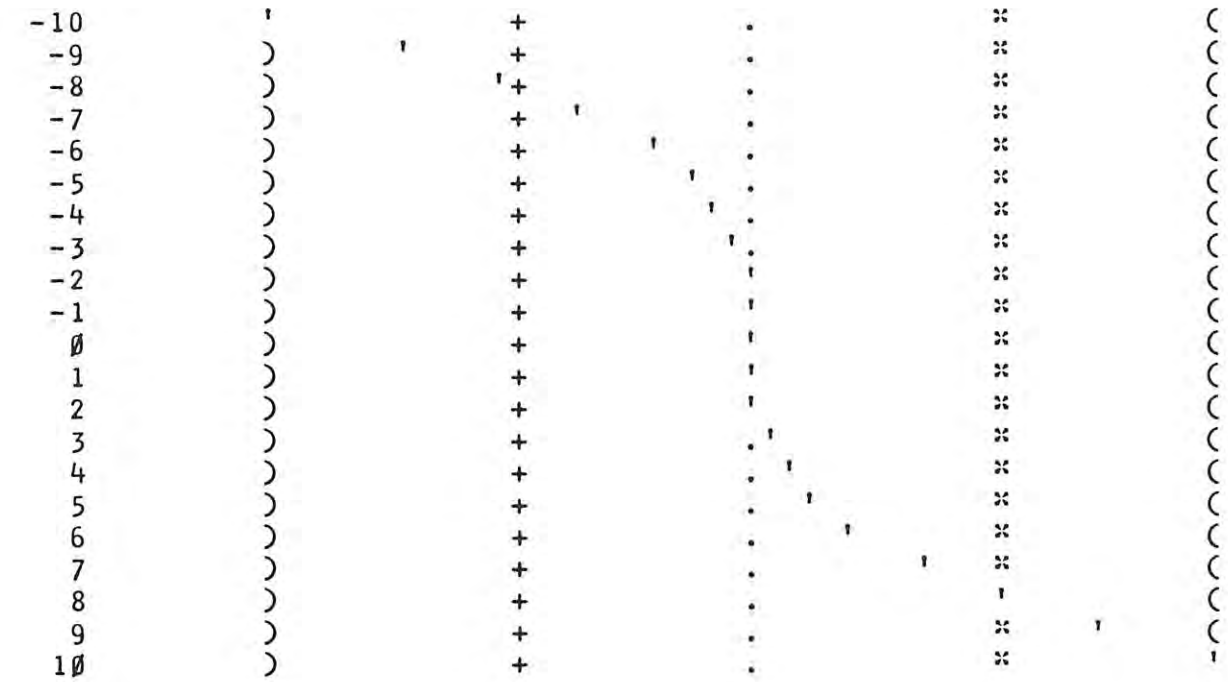
```
←PA=((P-MN)/(MX-MN))*2-1
```

The following program plots PA over the range -10 to 10.

```
←TYPE PART 1
1.1 MX=1ØØØ,  MN=-1ØØØ
1.2 DO PART 2 FOR X=-1Ø:1:1Ø

2.1 PA=((P-MN)/(MX-MN))*2-1 FOR P=X↑3
2.2 PLOT Ø,-.5,.5,-1,1,PA ON X

←DO PART 1
```



If two or more variables have the same value when the PLOT command

is executed, the character assigned to the righthand-most variable will be plotted.

If the value to be plotted is outside of the range of -1 to +1, an exclamation point (!) is placed in the margin corresponding to the exceeded limit.

```
←PLOT 1.5*SIN(X*$PI/180) FOR X=0:15:360
```

Resolution of a plotted graph is one Teletype character or about 0.1 inches. The plotting range -1 to +1 comprises 51 characters separated by 50 spaces. Consequently, the plot increment along

the *Y* axis is .04 or 2% of the full scale. When a value fails to match one of the available plot positions, it is plotted at the nearest available location; thus, .0199 is plotted at 0 and .03 is plotted at .04.

The *X*-axis (vertical) plot increment is equal to a line space of the Teletype (.167 inches). The value of each *X*-axis increment is determined by the use of a FOR clause. More-distinct numerical resolution can be obtained by manipulating the PLOT command to plot on a smaller increment on the *X* (vertical) axis. For example, if a PLOT with an increment of 1 on the *X* (vertical) axis is not sufficiently clear, the graph may be replotted with an increment of 0.1 or 0.05.

PUNCH and READ

Programs in STRCOMP may be punched onto paper tape. The PUNCH command is similar to FILE in that one may PUNCH a part, functions, variables, or all of these. A comment may be added to the command after a semicolon. The comment will be punched onto the tape and will be printed out when the tape is entered into the computer.

```
←PUNCH PART 1;  STANDARD DEVIATION PROGRAM
```

Tapes are punched in an internal format usable by STRCOMP only. Before executing the PUNCH command, the user must notify the BBN computer operator. When the PUNCH command is executed, the tape is punched in the BBN computer room, not on the user's Teletype. A backarrow is printed when the tape has been punched.

To reload a program or file from paper tape, the READ command is used. The computer operator at BBN must have the tape. Then the user must notify the operator of his intent (so that he can set up the tape drive). The command READ is all that is allowed.

```
←READ
```

When the entry is completed, a backarrow appears, returning control to the user. The file is now ready to be used.

Special Symbols

Certain values in STRCOMP are available through the following symbols.

$L	current line on this page
$P	page in use from beginning of current STRCOMP session
$SPC	Teletype carriage position in spaces from left margin
$Y	current year
$MON	month from beginning of year (number)
$D	days from beginning of month
$H	hours since midnight of current day
$MIN	minutes from beginning of current hour
$DAT	current date in string form (e.g., 2/14/68)
$TIM	current time in string form (e.g., 10:15 AM)
$T	true
$F	false
$IND and $END	are recognized by STRCOMP but are included only because they appear in ISR files (see page for a complete discussion of these values)

II. THE ISRCOMP LANGUAGE

A. Background

ISRCOMP is a version of STRCOMP which lacks a few of the more specialized facilities of STRCOMP but includes the capability to access data files built by the ISR System.* Specifically, ISRCOMP is oriented toward data retrieval and analysis of information stored in ISR system files. (ISRCOMP cannot create, alter, or append ISR system files.) Before discussing ISRCOMP, it may be useful to review the general points of the ISR System.

General Aspects of ISR System

The ISR System is comprised of several user-oriented programs designed to facilitate information storage and retrieval. There are three basic file-handling functions. First, through a question/answer dialogue with the computer, the user defines a file structure by describing fields in terms of names, position in the record, data type, and syntax definition of acceptable values, and by specifying which fields are to be used for ordering records in the file. Second, data are entered (in the format previously specified) via either Teletype or magnetic tape (card images). The former method enables the user to enter or update records as the data become available, whereas the latter method is useful when large blocks of data have already been collected. Finally, information may be retrieved (found and examined) from

*For a detailed description of the ISR System, see S.T. Castleman, *Information Storage and Retrieval System* (April 1968).

the data file, in summary form or in part. Using the dialogue-oriented search program, data manipulations may be specified by using basic arithmetic. Populations (subsets of the file) are selected through the use of Boolean operators and comparison operators. Arithmetic summaries may be printed as matrices or straight tabulations. However, there is no way of arriving at many results without a procedure-oriented program which is capable of branching, iterating, computing standard functions, etc. Thus, the need for ISRCOMP is established.

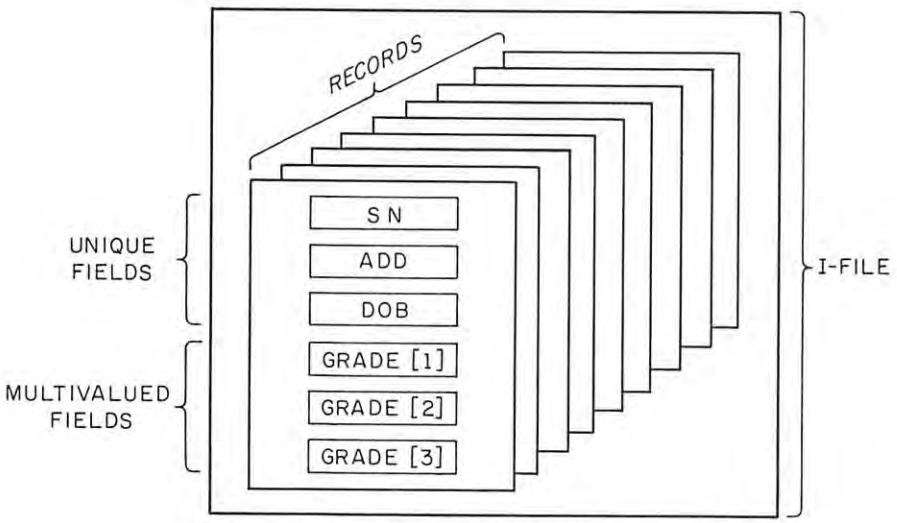
Some ISR Definitions

All the structure information and data records defined and entered via the ISR System are known as a FILE. In order to avoid confusion with program and variable files (stored by the STRCOMP or ISRCOMP FILE command), these ISR data files will be hereafter known as I-FILES. An I-FILE is comprised of records. Each record is comprised of units of information called *fields*.

The fields may have one unique value in each record or may be multivalued. For example, a student-information I-FILE includes many student records, each of which includes the unique fields, student name, address, and date of birth and the multivalued fields, grade, and date of grade.

Records are arranged in the I-FILE by the first identifying field (ID). The ID field must have a unique value. ID fields are arranged alphabetically when their values are *text* and in numeric order when their values are *integer* or *decimal number*.

FIGURE 1.
I-FILE
structure.



If more than one field is designated as an ID field, then *records* are ordered primarily on the first ID field and secondarily on the second ID field. For example, if three names are filed as SMITH, JOHN, then student number (SN) would be necessary to distinguish records (SMITH,JOHN 10115, SMITH,JOHN 11021, SMITH,JOHN 12112).

The terms ID field, field, record, and I-FILE will be used in reference to the ISR System specifically. See Fig. 1 for a diagram of I-FILE structure.

B. Using ISRCOMP

Calling ISRCOMP

It is quite possible to use the ISRCOMP language on ISR files

without intimate knowledge of the ISR System. To call the ISRCOMP program, follow the same directions on page 1, except type ISR-COMP instead of STRCOMP. A backarrow (←) is the sign of user control, as in STRCOMP.

Basic commands enable the user to reference the I-FILE, read each record, and close the I-FILE. Once an I-FILE has been accessed, the user may obtain a *dictionary* of field types in that I-FILE. Witn this information, he may write programs to manipulate the field values.

Opening and Closing an I-FILE

The OPEN command enables the user to access an I-FILE. (A list of I-FILES, including their numbers, can be obtained by typing TYPE ALL TITLES.) The OPEN command must be followed by the I-FILE number. If the I-FILE has a confidential code, the file number must be followed by a colon and a string-valued expression containing the correct code.

```
←OPEN FILE 28:"VMD"
```

There is no way to find a confidential code except from the originator of the file. The following example illustrates how a program might be written to protect a file from unauthorized access, even though the program may be started up by any user.

```
←1.1 ACCEPT#,"CODE  ",*CODE
←1.2 OPEN FILE 28:CODE
```

A file may be opened by several users at the same time. However,

if a file is being changed or updated by an ISR program, any attempt to OPEN it will cause an error message.

An I-FILE is automatically closed when ISRCOMP is halted or when another I-FILE is opened. It can also be specifically closed by the CLOSE command, which requires no argument.

Types of Information in an I-FILE

Once an I-FILE is open, the user may obtain a dictionary which acquaints him with the different types of information stored in the I-FILE.

```
←OPEN FILE 28:"VMD"
←TYPE DICT

28 SOCRATES (VMD)     1:04 PM 4/9/1968

N       ID    TEXT
SN      ID    FIXED 5
ADD     UNI   TEXT
DOB     UNI   DATE
H       UNI   DECIMAL
W       UNI   DECIMAL
GH      UNI   INTEGER
GRAREC
  DG      MUL   DATE
  GRADE   MUL   TEXT
```

Each field (in the above example) has listed

(1) its ISRCOMP name (a variable name);

(2) its position in the record (unique, multivalued, identifying, or group);

(3) its data type (fixed *n*, text, date, decimal number, or integer).

This file will be used for most of the examples in this Manual. The long names of each field are as follows.

SN	student number
N	name
ADD	address
DOB	date of birth
H	height
W	weight
GH	general health
GRAREC	grade record
GRADE	grade
DG	date of grade

Fixed *n*, text, and date fields are all string-valued. Fixed-*n* field values are always *n* arbitrary characters: text fields have values of 0-71 arbitrary characters; and date field values are of the form month/day/year (e.g., 12/1/1968).

Integer and decimal-number fields are numeric-valued. Decimal-number fields may have any value attainable by any normal STRCOMP variable (namely, 2.9×10^{-38} to $1.7 \times 10^{+39}$ in magnitude or zero), whereas integer field values are positive whole numbers or zero.

U, IND, and END

In addition, fields of all five data types may have the values U (unknown) or IND (indeterminate). U is interpreted as a string value in fields normally having string values. The literal expression U must be used in testing numeric fields. The U must have been entered at the time of data entry. For example,

```
←TO PART 2 IF N="U"
```

The value IND is automatically assigned to any field when a null entry (depression of the ESC key) is entered at the time of data entry. The symbol $IND is used to check for indeterminate fields.

```
←TO PART 2 IF GH=$IND
```

Multivalued fields may have more than one value, but the one after the last value entered in the series is automatically stored as END (symbolized by $END). In the following example, the values of GRADE[I] are typed until the value $END is reached.

```
←TYPE GRADE[I] FOR I=1:1 UNTIL GRADE[I]=$END
    GRADE[I]= C-
    GRADE[I]= C
    GRADE[I]= C
```

Accessing Records

Opening an I-FILE does not access the first record within that

I-FILE. The NEXT command accomplishes record access. Records are accessed in order (where file order is determined by ID fields). For example, in the Student Health Record I-FILE, the records are alphabetically ordered by name (not by order of entry). The following example accesses the first record.

```
←OPEN FILE 28:"VMD"
←NEXT
```

The only indication of command execution in the above example is a backarrow returning control to the user. Repeated use of the NEXT command accesses successive records.

As each record is accessed, the ISRCOMP variables having the same name as the field names are set to the values of the corresponding fields in that record. After a record is accessed, the TYPE command may be used to establish the contents of the record.

```
←OPEN FILE 28:"VMD"
←NEXT
←TYPE DOB
        DOB= 12/Ø1/1956
←TYPE ALL FIELDS
          N= ABRAMS, SUSAN
        ADD= 541 MASS AVE, CAMBRIDGE
         SN= 23221
        DOB= 12/Ø1/1956
          H=     57
          W=     9Ø
         GH=      1
  GRADE[1]= C-
  GRADE[2]= C
  GRADE[3]= C
     DG[1]=  9/18/1967
     DG[2]= 1Ø/15/1967
     DG[3]= 11/3Ø/1967
```

Because these field names are actually ISRCOMP variable names, they can be used in any expression.

```
*PRINT#,"HEIGHT IN FEET ",H/12
HEIGHT IN FEET 4.75
```

FIND

The FIND command provides a means of skipping through the file to a particular record without accessing any others. FIND is followed by an expression or a series of expressions, separated by commas. The first expression has the value for the first ID field, the second expression has the value for the second ID field, etc., not exceeding the number of ID fields specified in the I-FILE. Executing the FIND command accesses the first record in the file that has ID fields matching the arguments of FIND. For example, assume that the following records (identified by student name and number) are in an I-FILE in the following order.

```
ABRAMS, SUSAN    23221
FIORE, BEVERLY    22331
HARRIS, JAMES    33311
SMITH, JOHN    12121
SMITH, JOHN    12221
SMITH, JOHN    12323
SWEET, CAROLYN    23113
WATERS, MARYANN    23231
```

The FIND command in the following example causes the program to skip over records until the first record with SMITH, JOHN as the first ID field appears.

```
←OPEN FILE 28:"VMD"
←FIND "SMITH, JOHN"
←PRINT#,N," ",SN

SMITH, JOHN  12121
```

To find a SMITH, JOHN other than the first one, it is necessary to use the student number, also.

```
←FIND "SMITH, JOHN","12323"
←PRINT#,N,"  ",SN

SMITH, JOHN  12323
```

If no match is found, the first record after the place where an exact match might be expected is accessed. In the following example, GUNTHER is not part of the I-FILE, so the next record is accessed.

```
←OPEN FILE 28:"VMD"
←FIND "GUNTHER"
←PRINT#,N,"  ",SN

HARRIS, JAMES   33311
```

After a FIND has been executed, the user is positioned at that point in the I-FILE. (The NEXT command accesses subsequent records.) If the program is already positioned past the point to be found, it is possible to back up with the FIND command.

The FIND command is useful for finding blocks of data in files which have been reorganized on a field name other than the

original ordering field (using the ISR shuffle program described in the *Information Storage and Retrieval System User's Manual*). For example, the student record I-FILE is originally ordered on student name and student number. It is possible to reorganize the file so that it is ordered by date of birth. When this is done, the FIND command may be used to access the first of a specific date of birth; then all other identical dates of birth will follow. (See page 85 for a program example using the FIND command in this way.)

ELSE

NEXT and FIND can be followed by an ELSE clause, which gives the program an opportunity to branch on certain special cases. Used after the NEXT command, the ELSE clause will be executed if the end of the I-FILE has been reached.

```
←1.3 NEXT ELSE TO PART 2
```

Used after the FIND command, the ELSE clause is executed only if the exact record specified is not found.

Dictionary References

The ISRDIC function enables the user to decode field values by

referencing a dictionary which has been set up by an ISR program. The ISRDIC function requires two arguments: the field name and the dictionary number. For example, GH (general health) values are coded as follows.

1 excellent
2 good
3 fair
4 poor
5 needs care

Only the codes for GH are stored in the I-FILE records. An ISR dictionary (number 132) has been set up to include these values. In a printout, the user may specify the dictionary value to be printed rather than code number stored in the I-FILE.

```
←PRINT N," ",ISRDIC(GH,132)
ABRAMS, SUSAN  EXCELLENT
```

Multivalued Fields

Multivalued fields require a single subscript and are referenced like all other subscripted variables. (See page 16 for a detailed description of subscripts.)

```
←TYPE ALL GRADE
    GRADE[1]= C-
    GRADE[2]= C
    GRADE[3]= C
```

STRCOMP Features *not* in ISRCOMP

Some functions and operators of STRCOMP have been omitted from ISRCOMP. The functions STP, NST, FN, NUM, OK, and COND are not in ISRCOMP. The backarrow (←) and the IS operators and the INDEX command are not in ISRCOMP.

The EVAL function does much of what the backarrow operator would do. (See p. 44.) The function of the IS operator can be programmed in ISRCOMP by using EXT, STL, LOC, etc. The CTNS operator also replaces some IS operator functions.

Programming in ISRCOMP

The procedure for writing programs in ISRCOMP is similar to that in STRCOMP. Since I-FILE field names become variables in STRCOMP, the user must avoid using those names except as I-FILE variable names.

STRCOMP and ISRCOMP programs are interchangeable if the functions unique to either of the languages are not used. For example, a STRCOMP editing program can be used with a ISRCOMP program (in STRCOMP).

Program Examples

Following are two programs written in the ISRCOMP language.

The first program accesses a shuffled version of the student record file (now arranged by date of birth). It then skips over the first part of the file to the first record where the date of birth is January 1, 1956, or the first date after that. It then prints out date of birth, height, and weight for all students whose date of birth is in 1956.

```
←TYPE ALL PARTS
1.1 OPEN FILE 56:"VMD"
1.12 PRINT#,TAB 14,"HEIGHT AND WEIGHT OF CHILDREN BORN IN 1956"
1.13 PRINT#,#
1.14 PRINT#,TAB 2, "DATE OF BIRTH",TAB 18,"HEIGHT",TAB 33,"WEIGHT"
1.15 FIND "1/1/1956"
1.2 TO PART 1.3
1.25 NEXT ELSE TO PART 2
1.3 TO PART 2 IF NOT DOB CTNS "1956"
1.4 PRINT#,#,TAB 4,DOB,TAB 2Ø,H,TAB 35,W
1.5 TO PART 1.25

2.1 PRINT#,#,"END OF TABLE"

←DO PART 1
```

HEIGHT AND WEIGHT OF CHILDREN BORN IN 1956

DATE OF BIRTH	HEIGHT	WEIGHT
4/Ø5/1956	59	99
5/3Ø/1956	6Ø	9Ø
6/18/1956	58	88
12/Ø1/1956	57	9Ø

END OF TABLE

The second example computes the length of stay in the hospital of patients staying under 10 days. The I-FILE which is accessed contains the following fields.

```
←OPEN FILE 2:"DIS"
←TYPE DICT

2 EURIPIDES (VMD)    6:Ø9 PM 3/21/1968

DOD    ID     DATE
UN     ID     FIXED 9
N      UNI    TEXT
ADD    UNI    TEXT
VMD    UNI    TEXT
ADX    UNI    TEXT
AGE    UNI    TEXT
INS    UNI    TEXT
SER    UNI    FIXED 6
DOA    UNI    DATE
DIV    UNI    FIXED 3
DOB    UNI    DATE
SEX    UNI    FIXED 3
INIT   UNI    FIXED 3
LOX    MUL    TEXT
```

The program accesses the file which is arranged by date of discharge. As each record is opened, the essential fields are checked for indeterminate values or the literal expression "999" (this indicates a fictitious patient). If the fields have values, the length of stay is calculated by subtracting the admission date from the discharge date.

The hospital in the example had three divisions within it, so the calculations are done by by division. Step 1.4 indicates the division in which each patient resided. The results are printed by listing days of stay on the left and number of patients who stayed that long on the right.

```
←TYPE ALL PARTS
1.1 OPEN FILE 2:"DIS" FOR A=Ø FOR B=Ø FOR C=Ø FOR L[Ø,Ø]=Ø
1.2 NEXT ELSE TO PART 2
1.3 TO STEP 1.2 IF DOA=$IND OR DOD=$IND OR DIV=$IND OR UN CTNS "999"
1.4 F=LOC("GH BM PH ",DIV)
1.5 TO STEP 1.2 IF F=Ø
1.6 L[I,F]=L[I,F]+1 IF I>Ø FOR I=DDT(DOD)-DDT(DOA)
1.7 TO STEP 1.2

2.1 PRINT#,#,#,#,"FREQUENCY OF LENGTH OF STAY BY DIVISION "
2.2 PRINT#,#,
2.3 PRINT#,#,TAB 1Ø,"DIVISION I"
2.4 PRINT#,I,"          ",L[I,F] FOR I=1:1:1Ø FOR F=4
2.5 PRINT#,#,TAB 1Ø, "DIVISION II"
2.6 PRINT#,I,"          ",L[I,F] FOR I=1:1:1Ø FOR F=7
2.7 PRINT#,#,TAB 1Ø,"DIVISION III"
2.8 PRINT#,I,"          ",L[I,F] FOR I=1:1:1Ø FOR F=1Ø
2.9 CLOSE

←DO PART 1

FREQUENCY OF LENGTH OF STAY BY DIVISION

          DIVISION I
1          4Ø8
2          788
3          9Ø4
4          492
5          456
6          364
7          382
8          334
9          28Ø
1Ø          3Ø4

          DIVISION II
1          124
2          5Ø8
3          436
4          456
5          446
6          47Ø
7          33Ø
8          4Ø8
9          342
1Ø          33Ø
```

	DIVISION III
1	9Ø
2	294
3	286
4	254
5	2ØØ
6	226
7	18Ø
8	192
9	192
1Ø	14Ø

APPENDICES

APPENDIX A. TELETYPE OPERATING INSTRUCTIONS

To establish connection with the BBN computer ——

(1) Depress the ORIG button.

(2) Dial 491-5220 (BBN computer room).

(3) When the computer operator answers, give your name and request a line.

(4) When connection is established, replace the receiver, then depress the BREAK key.

(5) The BRK RLS button will light up and an identification code (including current time and date) will print out.

(6) Depress the BRK RLS button.

(7) Type the program name (STRCOMP or ISRCOMP) followed by a space, then your initials.

(8) Depress the ESC button.

A backarrow now prints, indication that the user is in control of the program; i.e., he may issue commands.

The Teletype Keyboard

The Model 33 ASR Teletype (Fig. A-1) is the one most commonly issued and is discussed below. If you have a Model 35 ASR Teletype, please refer to the *Teletypewriter Fundamentals Handbook* (Teletype Corporation, Skokie, Ill., 1965).

Terminating keys ——

ESC	always
RETURN	except in special cases
LINE FEED	except in special cases
CONTROL KEY and L	always

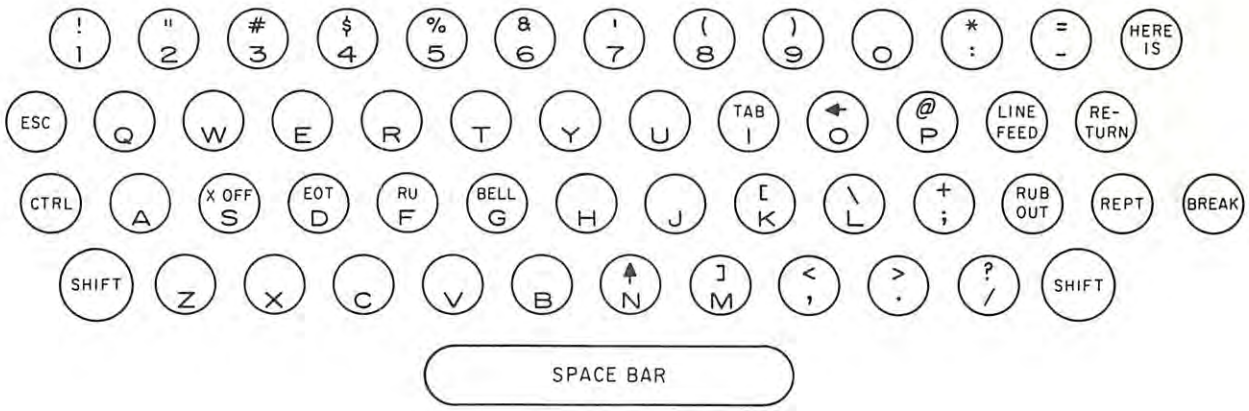


FIGURE A-1. Model 33 ASR Teletype terminal.

The BREAK key interrupts the program at any time. Each time that this key is depressed, the BRK RLS key will light up, indicating that the keyboard is locked. The BRK RLS key must be depressed before continuing. All entries by the user must be followed by depression of a terminating key.

Throughout the Manual, two combinations of keys are discussed: the CONTROL key plus one other and the SHIFT key plus one other. The SHIFT and CONTROL keys are not interchangeable. The interpretation of the combinations (e.g., CONTROL and L) are not always clear from markings on the keys.

SHIFT combines with all top-row characters (except HERE IS). The result (the same as on a standard typewriter) is the upper case indicated on those keys. The zero key has no upper case.

In the second row from the top, the following occur.

(1) SHIFT combines with I to result in a TAB of 15 character

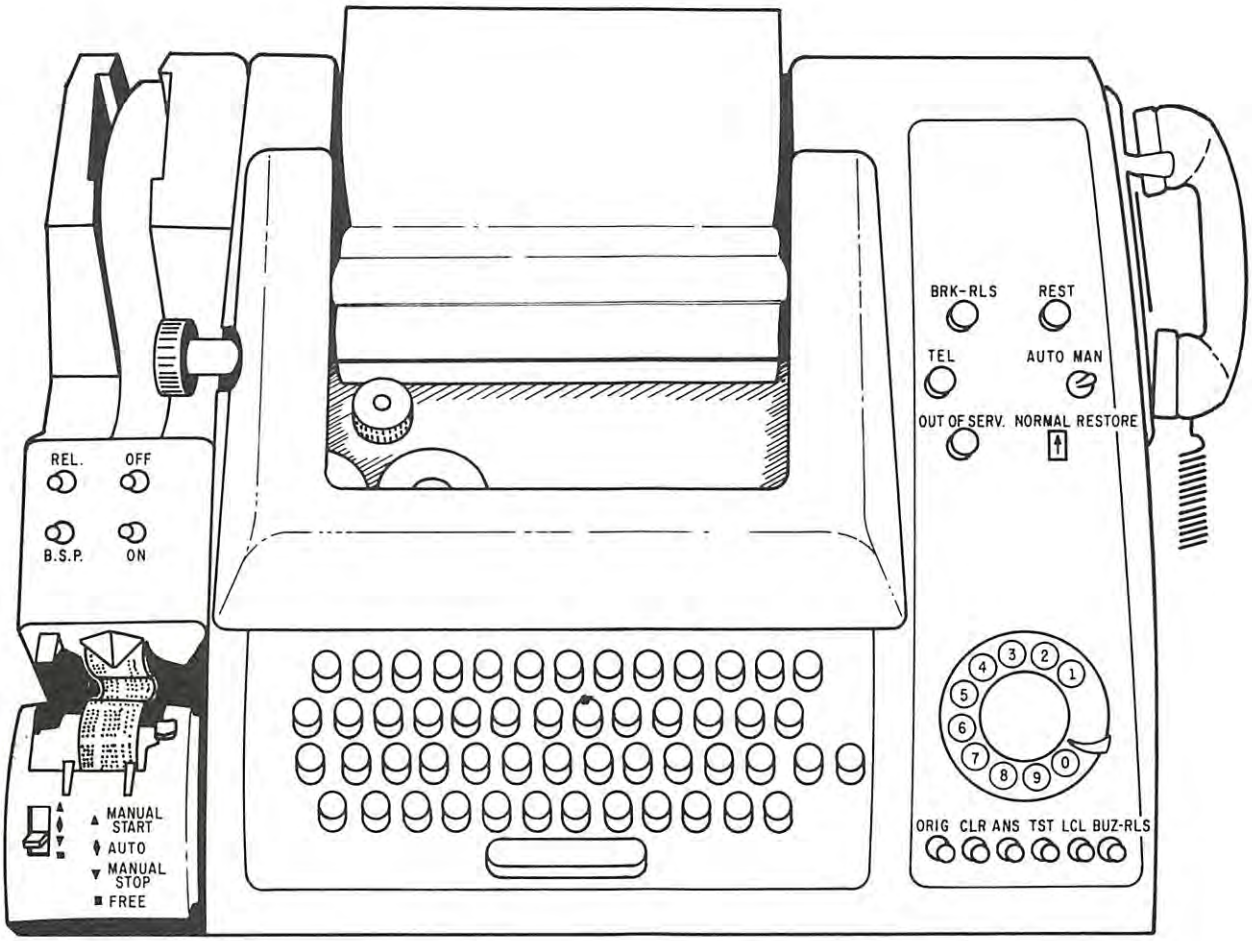


FIGURE A-2. Teletype keyboard.

spaces from the current location of the carriage.

(2) SHIFT plus 0 produces a backarrow (←).

(3) SHIFT plus P produces an at (@) sign.

The third row has three keys that may combine with SHIFT.

(1) SHIFT plus K produces a leftsided bracket ([).

(2) SHIFT plus L causes a backslash (\) or a bow tie (⁓) to print and causes the previous character (may be a space) to be deleted (disregarded) by the computer.

(3) SHIFT plus semicolon (;) produces a plus sign (+).

The bottom row includes the following combinations.

(1) SHIFT plus N produces an uparrow (↑).
(2) SHIFT plus M produces the rightsided bracket (]).
(3) SHIFT plus comma (,) produces a less-than sign (<).
(4) SHIFT plus period (.) produces a greater-than sign (>).
(5) SHIFT plus slash (/) produces a question mark (?).

Control characters are executed by depressing simultaneously the CONTROL key and the designated key.

CONTROL plus L causes the page to advance to the top of a clean one (if the paging is set accurately at first). CONTROL plus D will shut off the Teletype motor on certain machines. Other control characters exist but are illegal in STRCOMP and ISRCOMP.

When the user wants to end the STRCOMP program, the HALT command is used (e.g., ←HALT). This command stops the program, erasing all accumulated work, and shuts the Teletype off (on certain Teletypes).

APPENDIX B. GLOSSARY OF TERMS

	page	
'A'	41	absolute value of the numeric expression A
ACCEPT	14	direct or stored command —— provides for data entry directly into program
ALPH	45	a truth-valued function which determines whether two string-valued expressions are in alphabetical order
AND	37	Boolean operator —— combines two comparisons (may be symbolized by ∧ or &)
argument	5	the object of a function or command
array	17	an arranged set of subscripted data
ATN	40	stored mathematical function; arctangent of a variable(s) —— argument and result expressed in radians
backarrow (←)	51	a STRCOMP (only) function which causes the value of a string-valued variable following it to be executed in the statement
CLOSE	75	in ISRCOMP only —— terminates access to I-FILE
COND	49	a conditional function which returns the value of the first expression after the first true, odd-numbered argument —— in STRCOMP only
COS	40	a mathematical function; cosine of a variable —— argument and result expressed in radians
CTNS	24	a truth-valued operator which verifies the existence of one string-valued expression within another
DDT	48	decodes a date from a string value to number of days after 1/1/1849
DEFINE	50	a command which permits the user to define a function

DELETE	16	a command which erases the specified data, used with step(s), part(s), file(s), ALL, or FCN(s)
DEMAND	13	direct or stored command —— provides for data entry directly into program
DO	12,26,52	command used to initiate the execution of a program, step, or string-valued argument
DONE	12	stored command only —— terminates a program part
DP	41	mathematical function —— result is the digit part of its argument
DTM		decodes time from a string value to the number of minutes since midnight
ELSE	82	in ISRCOMP —— used with NEXT or FIND commands —— executed only if first part of statement cannot be executed;
	55	in STRCOMP —— after index executed when no more elements of array are explicitly defined
entry	28	a program or set of results, or both, permanently stored in the computer in a file
EXP	40	mathematical function meaning *e* to the power of its argument
expression	5	any combination of numbers, variables, algebraic operators, and functions having a unique value
EXT	47	a function which extracts one character or a segment of characters from a string-valued expression —— returns the characters which have been specified by number
EVAL	44	a function used for indirect addressing —— returns the value of the value of its argument
FCN	50	argument meaning *function*, used with TYPE, PRINT, FILE, or PUNCH
FIELD	73	in ISRCOMP only —— term used in ISR system to denote units of information; these become vari-

		able names in ISRCOMP
FILE	28	permits permanent storage of programs, variables, or functions
FIND	80	in ISRCOMP only —— used to locate a record within an I-FILE
FN	51	function which enables the user to set a variable to the text of a function
FORM	60	function used to print numeric values in specified format
FP	41	a mathematical function —— returns fraction part of its argument
GO	27	command (direct only) used to resume execution of a program after it has been stopped by a break or the STOP command
HALT	2	command used to terminate use of STRCOMP or ISRCOMP
ID field	73	see ordering field
I-FILE	73	an organized set of data compiled by means of the ISR system —— referenced in ISRCOMP only
IF	22	conditional modifier causing the statement to be executed only when the condition following is true
IN	54	modifies the INDEX command by denoting the name of subscripted variable to be accessed
INDEX	53	a command which accesses existing elements in an array —— in STRCOMP only
IP	41	mathematical function —— returns the integer part of its argument
IS	55	a STRCOMP (only) operator which matches character patterns with a string-valued expression
ISRDIC	83	a function in ISRCOMP only —— used to decode

		field values by referring to a previously constructed dictionary
iteration		repeated execution of one (or more) command to effect a certain operation
LINE	3	a command which causes the Teletype to advance one line
LOAD	30	command initiating reentry of a file entry which has been permanently stored in the computer
LOC	46	function which finds the value of its argument in a string-valued expression and returns one plus the numeric location of the last character
LOG	40	mathematical function which determines the base 10 logarithm of its argument
LONG	7	cancels previous SHORT if any
LN	40	mathematical function which determines the natural logarithm of its argument (base *e*)
MAX	42	a function which returns the value of the largest number in a series (arguments of MAX).
MIN	42	function which returns the value of the smallest number in a series (arguments of MIN)
multi-valued field	83	in ISRCOMP only —— refers to a unit of information having more than one value; similar to a subscripted variable
NEXT		ISRCOMP only —— command used to access a record in an I-FILE; opens records in sequence according to ordering field (ID field)
NOT	37	Boolean operator —— negates expression following it
NST	51	in STRCOMP only —— has the value of the step number following the one stated in its argument
NUM	43	a truth-valued function in STRCOMP (only) which determines whether its argument can be used as

		program continues after REPEAT is executed
SGN	41	mathematical function which returns the algebraic sign of its argument (-1,0,+1)
SHORT	7	suppresses diagnostic comments after error
SIN	40	mathematical function which calculates the sine of its argument —— argument and results are in radians
SQRT	40	mathematical function which calculates the square root of its argument
START	30	an initializing command which loads then begins program in one step
STOP	27	a command which allows a user to stop a program at any step of the program
STL	42	function which determines the number of characters in the value of its argument
STP	51	a function which permits a user to set a variable to the text of the step number which is the argument —— in STRCOMP only
string-valued variable	8	a variable whose value is any combination of characters defined by enclosing the combination within quotation marks
subscript	16	a modifier (within brackets) which allows the allocation of successive values to one variable name (usually called a subscripted variable)
TAB	63	a spacing function which advances the carriage to the character number (spaces from beginning of the left margin) rather than a specified number of spaces
TO	21	a command which causes the program to branch to another step (frequently modified by conditional phrases)
TYPE	4	command which prints out the name of the argument and the value of that argument, each on a separate line

APPENDIX C. TRUTH TABLES

AND

	T	F	I
T	T	F	I
F	F	F	F
I	I	F	I

OR

	T	F	I
T	T	T	T
F	T	F	I
I	T	I	I

NOT

T	F
F	T
I	I

APPENDIX D. SUMMARY OF NONINTERCHANGEABLE FEATURES

STRCOMP Features Not Found in ISRCOMP

← (backarrow operator)	Indirect address operator
COND(A,B,C)	Conditional evaluator
FN(A)	String value of a function
INDEX	Command to access elements in an array
IS operator	Pattern-matching operator
NST(A)	Next-step-number print function
NUM(A)	Number evaluator
OK(A)	String evaluator
STP(A)	Value of step number function

ISRCOMP Features Not Found in STRCOMP

CLOSE	Command to terminate access to I-FILE
FIND	Command to skip to a specified record in an I-FILE
NEXT	Command to access records in an I-FILE
OPEN	Command to initiate access to an I-FILE

APPENDIX A. TELETYPE OPERATING INSTRUCTIONS

To establish connection with the BBN computer ——

(1) Depress the ORIG button.

(2) Dial 491-5220 (BBN computer room).

(3) When the computer operator answers, give your name and request a line.

(4) When connection is established, replace the receiver, then depress the BREAK key.

(5) The BRK RLS button will light up and an identification code (including current time and date) will print out.

(6) Depress the BRK RLS button.

(7) Type the program name (STRCOMP or ISRCOMP) followed by a space, then your initials.

(8) Depress the ESC button.

A backarrow now prints, indication that the user is in control of the program; i.e., he may issue commands.

The Teletype Keyboard

The Model 33 ASR Teletype (Fig. A-1) is the one most commonly issued and is discussed below. If you have a Model 35 ASR Teletype, please refer to the *Teletypewriter Fundamentals Handbook* (Teletype Corporation, Skokie, Ill., 1965).

Terminating keys ——

ESC	always
RETURN	except in special cases
LINE FEED	except in special cases
CONTROL KEY and L	always

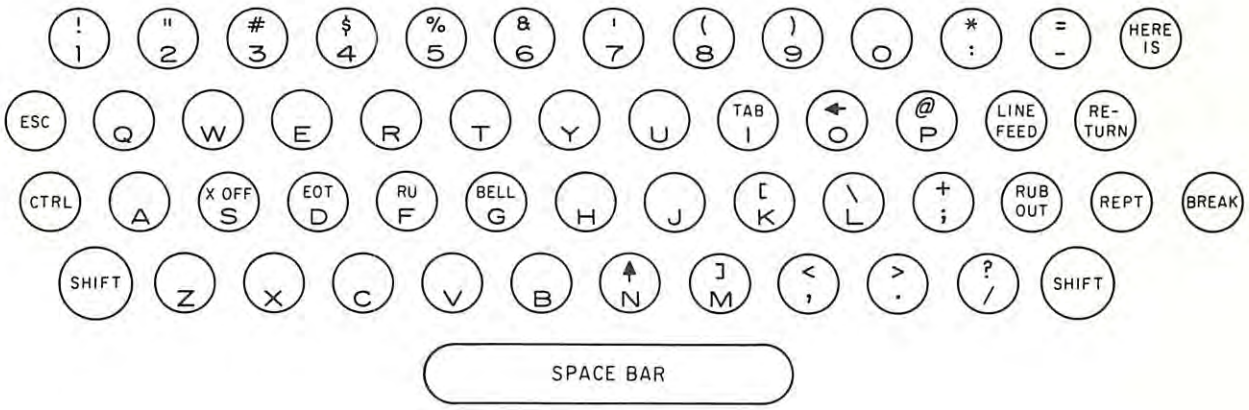


FIGURE A-1. Model 33 ASR Teletype terminal.

The BREAK key interrupts the program at any time. Each time that this key is depressed, the BRK RLS key will light up, indicating that the keyboard is locked. The BRK RLS key must be depressed before continuing. All entries by the user must be followed by depression of a terminating key.

Throughout the Manual, two combinations of keys are discussed: the CONTROL key plus one other and the SHIFT key plus one other. The SHIFT and CONTROL keys are not interchangeable. The interpretation of the combinations (e.g., CONTROL and L) are not always clear from markings on the keys.

SHIFT combines with all top-row characters (except HERE IS). The result (the same as on a standard typewriter) is the upper case indicated on those keys. The zero key has no upper case.

In the second row from the top, the following occur.

(1) SHIFT combines with I to result in a TAB of 15 character

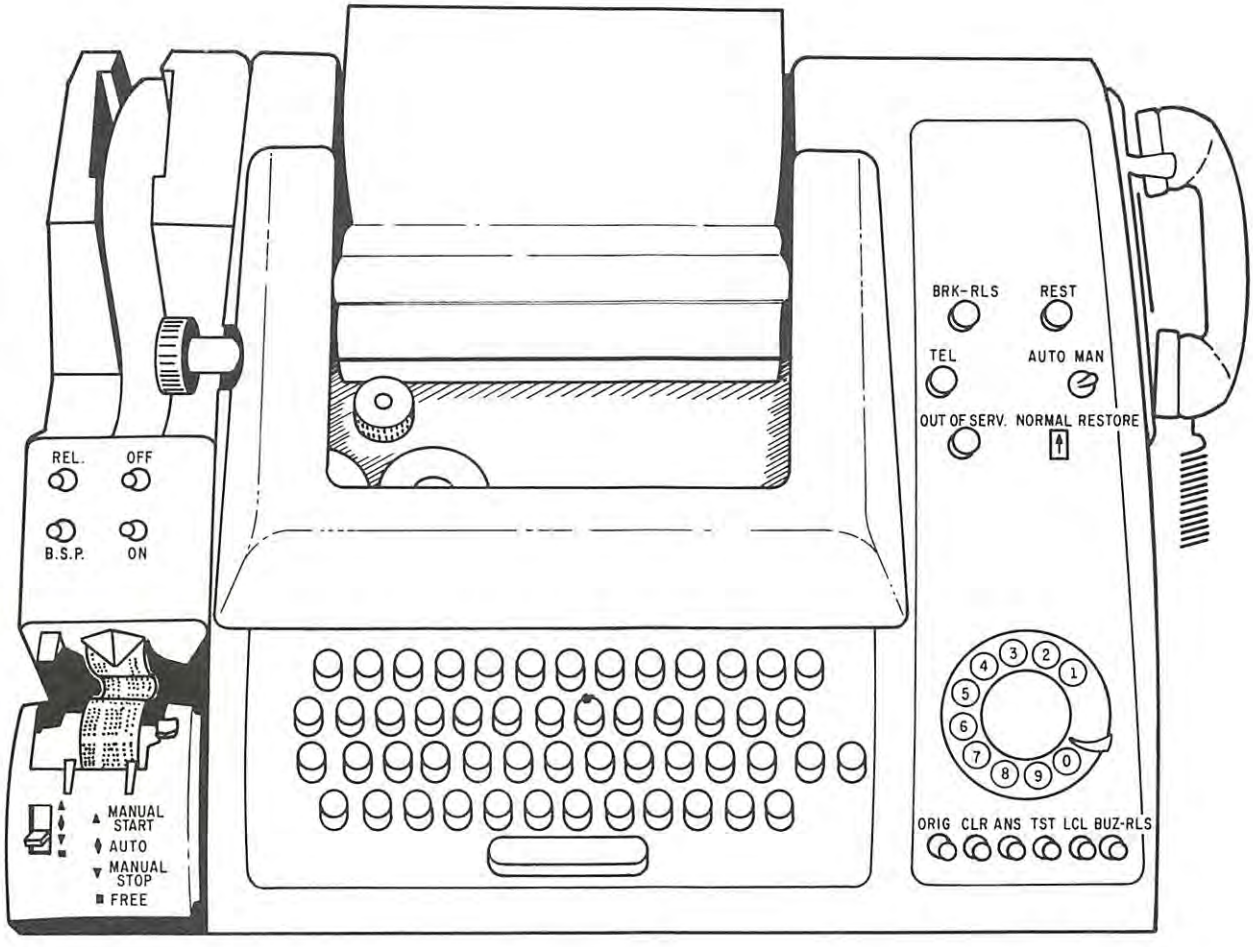


FIGURE A-2. Teletype keyboard.

spaces from the current location of the carriage.

(2) SHIFT plus 0 produces a backarrow (←).

(3) SHIFT plus P produces an at (@) sign.

The third row has three keys that may combine with SHIFT.

(1) SHIFT plus K produces a leftsided bracket ([).

(2) SHIFT plus L causes a backslash (\) or a bow tie (⋈) to print and causes the previous character (may be a space) to be deleted (disregarded) by the computer.

(3) SHIFT plus semicolon (;) produces a plus sign (+).

The bottom row includes the following combinations.

(1) SHIFT plus N produces an uparrow (↑).
(2) SHIFT plus M produces the rightsided bracket (]).
(3) SHIFT plus comma (,) produces a less-than sign (<).
(4) SHIFT plus period (.) produces a greater-than sign (>).
(5) SHIFT plus slash (/) produces a question mark (?).

Control characters are executed by depressing simultaneously the CONTROL key and the designated key.

CONTROL plus L causes the page to advance to the top of a clean one (if the paging is set accurately at first). CONTROL plus D will shut off the Teletype motor on certain machines. Other control characters exist but are illegal in STRCOMP and ISRCOMP.

When the user wants to end the STRCOMP program, the HALT command is used (e.g., ←HALT). This command stops the program, erasing all accumulated work, and shuts the Teletype off (on certain Teletypes).

APPENDIX B. GLOSSARY OF TERMS

	page	
'A'	41	absolute value of the numeric expression A
ACCEPT	14	direct or stored command —— provides for data entry directly into program
ALPH	45	a truth-valued function which determines whether two string-valued expressions are in alphabetical order
AND	37	Boolean operator —— combines two comparisons (may be symbolized by ∧ or &)
argument	5	the object of a function or command
array	17	an arranged set of subscripted data
ATN	40	stored mathematical function; arctangent of a variable(s) —— argument and result expressed in radians
backarrow (←)	51	a STRCOMP (only) function which causes the value of a string-valued variable following it to be executed in the statement
CLOSE	75	in ISRCOMP only —— terminates access to I-FILE
COND	49	a conditional function which returns the value of the first expression after the first true, odd-numbered argument —— in STRCOMP only
COS	40	a mathematical function; cosine of a variable —— argument and result expressed in radians
CTNS	24	a truth-valued operator which verifies the existence of one string-valued expression within another
DDT	48	decodes a date from a string value to number of days after 1/1/1849
DEFINE	50	a command which permits the user to define a function

DELETE	16	a command which erases the specified data, used with step(s), part(s), file(s), ALL, or FCN(s)
DEMAND	13	direct or stored command —— provides for data entry directly into program
DO	12,26,52	command used to initiate the execution of a program, step, or string-valued argument
DONE	12	stored command only —— terminates a program part
DP	41	mathematical function —— result is the digit part of its argument
DTM		decodes time from a string value to the number of minutes since midnight
ELSE	82	in ISRCOMP —— used with NEXT or FIND commands —— executed only if first part of statement cannot be executed;
	55	in STRCOMP —— after index executed when no more elements of array are explicitly defined
entry	28	a program or set of results, or both, permanently stored in the computer in a file
EXP	40	mathematical function meaning *e* to the power of its argument
expression	5	any combination of numbers, variables, algebraic operators, and functions having a unique value
EXT	47	a function which extracts one character or a segment of characters from a string-valued expression —— returns the characters which have been specified by number
EVAL	44	a function used for indirect addressing —— returns the value of the value of its argument
FCN	50	argument meaning *function*, used with TYPE, PRINT, FILE, or PUNCH
FIELD	73	in ISRCOMP only —— term used in ISR system to denote units of information; these become vari-

		field values by referring to a previously constructed dictionary
iteration		repeated execution of one (or more) command to effect a certain operation
LINE	3	a command which causes the Teletype to advance one line
LOAD	30	command initiating reentry of a file entry which has been permanently stored in the computer
LOC	46	function which finds the value of its argument in a string-valued expression and returns one plus the numeric location of the last character
LOG	40	mathematical function which determines the base 10 logarithm of its argument
LONG	7	cancels previous SHORT if any
LN	40	mathematical function which determines the natural logarithm of its argument (base *e*)
MAX	42	a function which returns the value of the largest number in a series (arguments of MAX).
MIN	42	function which returns the value of the smallest number in a series (arguments of MIN)
multi-valued field	83	in ISRCOMP only —— refers to a unit of information having more than one value; similar to a subscripted variable
NEXT		ISRCOMP only —— command used to access a record in an I-FILE; opens records in sequence according to ordering field (ID field)
NOT	37	Boolean operator —— negates expression following it
NST	51	in STRCOMP only —— has the value of the step number following the one stated in its argument
NUM	43	a truth-valued function in STRCOMP (only) which determines whether its argument can be used as

		a number
OK	43	a function in STRCOMP only which determines the type of argument it has and responds with a code indicating a numeric, string, truth-valued, indeterminate, or invalid argument
ON	65	modifying word used with the PLOT command to provide scale in the left margin on a graph
OPEN	7	in ISRCOMP only —— command used to initiate reference to an I-FILE
OR	37	Boolean operator —— used in comparative clauses (may be symbolized by ∇ or @)
ordering field	73	in ISRCOMP only —— the unit of information within the record used as identification of that record and used to index that file (ID field)
PAGE	3	a command causing the program to advance to a clean sheet (pages must be aligned at the beginning of a STRCOMP run)
PLOT	64	a command used to graph results of computation or a series of variables by using Teletype characters as symbols
PRINT	5	a command used to type out only the value of an expression or specific text
PUNCH	70	command used to store file entries on paper tape
RAN	42	a function which generates a pseudorandom number between 0 and the value of the argument
READ	70	command which loads (reenters) data from a paper tape which has been punched through STRCOMP
RECORD	73	in ISRCOMP only —— the unit of information which is comprised of fields; an I-FILE is comprised of records
REPEAT	28	a direct command only —— used to reexecute a step after it has been corrected or altered;

		program continues after REPEAT is executed
SGN	41	mathematical function which returns the algebraic sign of its argument (-1,0,+1)
SHORT	7	suppresses diagnostic comments after error
SIN	40	mathematical function which calculates the sine of its argument —— argument and results are in radians
SQRT	40	mathematical function which calculates the square root of its argument
START	30	an initializing command which loads then begins program in one step
STOP	27	a command which allows a user to stop a program at any step of the program
STL	42	function which determines the number of characters in the value of its argument
STP	51	a function which permits a user to set a variable to the text of the step number which is the argument —— in STRCOMP only
string-valued variable	8	a variable whose value is any combination of characters defined by enclosing the combination within quotation marks
subscript	16	a modifier (within brackets) which allows the allocation of successive values to one variable name (usually called a subscripted variable)
TAB	63	a spacing function which advances the carriage to the character number (spaces from beginning of the left margin) rather than a specified number of spaces
TO	21	a command which causes the program to branch to another step (frequently modified by conditional phrases)
TYPE	4	command which prints out the name of the argument and the value of that argument, each on a separate line

unique field	73	in ISRCOMP only —— a unit of information within a record having only one value
UNTIL	21	a possible modifier in FOR clauses
variable	8	a short code name assigned to values to facilitate manipulation and referencing
WHILE	21	a possible modifier in FOR clauses
WHY	7	causes printout of diagnostic of most recent error
XP	41	prints the exponent part of its argument
$[special symbols]	71	special system values

APPENDIX C. TRUTH TABLES

AND

	T	F	I
T	T	F	I
F	F	F	F
I	I	F	I

OR

	T	F	I
T	T	T	T
F	T	F	I
I	T	I	I

NOT

T	F
F	T
I	I

APPENDIX D. SUMMARY OF NONINTERCHANGEABLE FEATURES

STRCOMP Features Not Found in ISRCOMP

← (backarrow operator)	Indirect address operator
COND(A,B,C)	Conditional evaluator
FN(A)	String value of a function
INDEX	Command to access elements in an array
IS operator	Pattern-matching operator
NST(A)	Next-step-number print function
NUM(A)	Number evaluator
OK(A)	String evaluator
STP(A)	Value of step number function

ISRCOMP Features Not Found in STRCOMP

CLOSE	Command to terminate access to I-FILE
FIND	Command to skip to a specified record in an I-FILE
NEXT	Command to access records in an I-FILE
OPEN	Command to initiate access to an I-FILE