DAN ROSS

B5700 SNOBOL MANUAL

this listing is a copy of
tape CUBEA13
file SNOBOL/L2ØØØ11

when a copy of this file is on
the disk, it should be named
SNOBOL/MANUAL
type = SEQ

print this manual using program
(source) SNOBOL/LISTER
(object) ØSNOBOL/LISTER
(see the source in the
B57ØØ UTILITIES binder)

```
LABEL  000000000LPAPRNT00176177?USER=SPO    ; EXECUTE COPY   /SPO                                          COPY   /SPO
                                        FILE: SNOBOL/ MANUAL      FRI 6/25/76 06:14PM

        1000 %PL 78   FOR 8 LINES/INCH.
        2000 B5500 SNOBOL3 MANUAL.
        3000
        4000
        5000
        6000
        7000
        8000
        9000
       10000
       11000
       12000
       13000 JOHN M. CHAMBERS
       14000 COMPUTER SCIENCES DEPT.
       15000 1210 W. DAYTON STREET
       16000 UNIVERSITY OF WISCONSIN
       17000 MADISON, WISCONSIN  53706
       18000
       19000
       20000 %SECTION "CONTENTS."
       21000 %PAGE 1
       22000 CONTENTS:
       23000
       24000 1. INTRODUCTION
       25000 2. CONTROL CARDS
       26000    2.1. MCP CONTROL CARDS.
       27000    2.2. SNOBOL CONTROL CARDS.
       28000 3. PROGRAM ORGANIZATION
       29000    3.1. INSTRUCTIONS
       30000    3.2 PROGRAM-DEFINED FUNCTIONS
       31000       3.2.1. FUNCTION CALLS.
       32000       3.2.2. FUNCTION RETURNS.
       33000       3.2.3. FUNCTION CODE.
       34000       3.2.4. CALLING DEFINE().
       35000    3.3. PROGRAM ENTRY POINTS.
       36000 4. RULES.
       37000    4.1. THE STRING REFERENCE.
       38000    4.2. RULES WITH STRING REFERENCE ONLY.
       39000    4.3. RULES WITH PATTERNS.
       40000    4.4. RULES WITH REPLACEMENTS.
       41000    4.5. RULES WITH BOTH PATTERN AND REPLACEMENT.
       42000 5. EXPRESSIONS.
       43000    5.1. NAMES AND VALUES.
       44000       5.1.1. LITERALS.
       45000       5.1.2. NAMES
       46000       5.1.3. INDIRECTION.
       47000    5.2. CONCATENATION.
       48000    5.3. FUNCTION CALLS.
       49000    5.4. ARITHMETIC.
       50000       5.4.1. DIVISION MODES.
       51000       5.4.2. FAILURE OF ARITHMETIC.
```

```
        5.4.3. ARITHMETIC FUNCTIONS.
        5.4.4. ADDITIONAL COMMENTS ON ARITHMETIC.
6. PATTERNS.
    6.1. CONSTANT PATTERN ELEMENTS.
    6.2. STRING VARIABLES.
        6.2.1. ARBITRARY STRING VARIABLES.
        6.2.2. BALANCED STRING VARIABLES.
        6.2.3. FIXED-LENGTH STRING VARIABLES.
    6.3. BACK REFERENCES.
    6.4. PATTERN MATCH MODES.
    6.5. ASSIGNMENT OF STRING VARIABLES.
    6.6. PATTERN MATCH ALGORITHM.
7. GO-TO PARTS, LABELS.
    7.1 LABELS.
    7.2. SIMPLE GO-TO PARTS.
    7.3. RESERVED LABELS.
    7.4. COMPUTED GO-TOS.
    7.5. ERRORS IN GO-TO PARTS.
8. FAILURE.
    8.1. CAUSES OF FAILURE.
    8.2. ORDER OF EXECUTION.
9. INTRINSIC FUNCTIONS.
10. INPUT/OUTPUT.
    10.1. I/O STRINGS.
    10.2. OPEN AND CLOSED FILES.
    10.3. STANDARD I/O STRINGS.
    10.4. TELETYPES.
        10.4.1. TELETYPE INPUT AND OUTPUT.
        10.4.2. TELETYPE WAITING TIMES.
        10.4.3. TELETYPE ADDRESSES.
        10.4.4. SPECIAL CHARACTERS FOR THE TELETYPE.
    10.5. CAUSES OF I/O.
11. DEBUGGING AIDS.
    11.1. TRACING.
    11.2. INTERACTIVE DEBUGGING.
12. MISCELLANEOUS.
APPENDIX A. BNF DEFINITIONS FOR SNOBOL3.
APPENDIX B. BNF NOTATION.
APPENDIX C. B5500 CHARACTER SET.
APPENDIX D. RUNNING JOBS FROM A TELETYPE.
INDEX.

%SECTION 1
%PAGE 1
1. INTRODUCTION

   THIS MANUAL DESCRIBES THE PROGRAMMING LANGUAGE SNOBOL3, AS
IMPLEMENTED ON THE BURROUGHS B5500 AT THE UNIVERSITY OF WISCONSIN
(MADISON) BY JOHN M. CHAMBERS, WITH MUCH HELP IN IN THE DESIGN AND
INITIAL CODING STAGES BY WILLIAM KRUEGER, AND SOME HELP IN WRITING
THE COMPILER BY DAVID WILSON.
   THE VERSION OF SNOBOL3 DESCRIBED HERE IS BASICALLY IDENTICAL
WITH THE LANGUAGE DEVELOPED AT BELL LABS, WITH THE OBVIOUS DIFFERENCES
OF CHARACTER SET AND I/O FACILITIES.  (THERE IS ALSO A SLIGHT PROBLEM
WITH DIVISION, CAUSED BY THE FACT THAT THE B5500 HARDWARE ROUNDS
QUOTIENTS, WHILE MOST OTHER MACHINES TRUNCATE.  SEE THE SECTION
ON ARITHMETIC.)  THERE ARE, AS USUAL, A NUMBER OF "EXTENSIONS",
MOSTLY IN THE INTRINSIC FUNCTIONS.  THE BASIC SNOBOL3 FUNCTIONS
ACT AS THEY DO ON OTHER SYSTEMS, BUT THOSE WHOSE ACTIONS ARE MACHINE
DEPENDENT ARE OFTEN DIFFERENT.  THERE ARE A NUMBER OF NEW FUNCTIONS
THAT REFLECT THE PECULIARITIES OF THE B5500 OPERATING SYSTEM.
   MOST SNOBOL3 PROGRAMS WRITTEN ON OTHER MACHINES CAN BE RUN ON
THE B5500 WITH ONLY MINOR CHANGES IN THE I/O, MOST OF WHICH CAN BE
MADE BY THE COMPILER ITSELF.
   AN EXTREMELY USEFUL FACILITY OF THE B5500 IS THE AVAILABILITY OF
TELETYPES AS I/O DEVICES.  NATURALLY, THIS IS AVAILABLE TO SNOBOL
IN AN EASILY-USED MANNER, AND A SIMPLE, VERY USEFUL INTERACTIVE
DEBUGGING AID HAS BEEN SUPPLIED FOR TELETYPE USERS.
%SECTION 2
%PAGE 1
2. CONTROL CARDS

   THERE ARE TWO TYPES OF CONTROL CARDS RELEVANT FOR SNOBOL JOBS--
THOSE THAT GIVE INFORMATION TO THE OPERATING SYSTEM (THE "MASTER
CONTROL PROGRAM", OR "MCP"), WHICH START WITH A QUESTION MARK; AND
THOSE THAT GIVE INFORMATION TO THE SNOBOL COMPILER, WHICH START WITH
A MINUS SIGN ("-").

2.1. MCP CONTROL CARDS.
   AT THE START OF A DECK OF CARDS FOR BATCH JOBS, OR TYPED TO
INITIATE A TELETYPE JOB, THE USER NEEDS A SET OF MCP CONTROL CARDS
FOR BATCH JOBS.  THESE ARE SEPARATE CARDS, EACH STARTING WITH A
QUESTION MARK ("ILLEGAL CHARACTER").  FOR TELETYPE JOBS, THE ENTIRE
SET OF CONTROL CARDS IS PRECEDED BY A PAIR OF QUESTION MARKS, AND
THE CARDS ARE SEPARATED BY SEMICOLONS (";"), WITH AN ARROW AFTER THE
LAST CONTROL CARD.
   THE FIRST TWO CARDS IN A BATCH JOB SHOULD BE:

      ? USER <PROJ#> / <USER#>
      ? EXECUTE SNOBOL/SNOBOL

THE FIRST CARD IDENTIFIES THE USER TO THE MCP; THE SECOND TELLS IT
TO INITIATE SNOBOL.  THE PROGRAM AND DATA DECK MUST BE PRECEDED BY

      ? DATA <NAME>

WHICH NAMES THE INPUT DECK.  BETWEEN THE EXECUTE CARD AND THE DATA
CARD CAN COME ANY NUMBER OF OTHER CONTROL CARDS; SOME OF THE CARDS
THAT ARE USEFUL TO SNOBOL USERS ARE DESCRIBED BELOW.  ONE OF THE
OTHER CONTROL CARDS MUST BE THE FOLLOWING:

      ? FILE PROGRAM = <NAME>

WHERE <NAME> IS THE SAME AS THE <NAME> ON THE DATA CARD.  THIS
ASSIGNS THE CARD DECK AS THE INPUT FILE TO THE COMPILER.  SOME OTHER
MCP CONTROL CARDS THAT CAN BE USED ARE:

      ? PROCESS = <INTEGER>
         THIS PUTS A CPU TIME LIMIT TO THE ENTIRE RUN OF <INTEGER>
         MINUTES.  IF THIS LIMIT IS EXCEEDED, THE JOB WILL BE TERMINATED
         BY THE MCP.  THE DEFAULT CPU TIME LIMIT IS 2 MINUTES.
      ? IO = <INTEGER>
         THIS PUTS A TIME LIMIT OF <INTEGER> MINUTES TO THE I/O TIME
         THAT CAN BE USED BY THE RUN.  THE DEFAULT I/O LIMIT IS
         5 MINUTES.
      ? STACK = <INTEGER>
         THIS ASSIGNS A STACK OF <INTEGER> WORDS TO THE RUN.  THIS
         IS ONLY NECESSARY IF THE PROGRAM REACHES GREAT DEPTHS OF
         FUNCTION CALLS, CAUSING STACK OVERFLOWS.  THE DEFAULT STACK
         SIZE IS 512.
      ? CORE = <INTEGER>
         THIS ASSIGNS A CORE ESTIMATE OF <INTEGER> WORDS TO THE
         RUN.  THE JOB WILL BE STARTED WHEN THIS MANY WORDS BECOME
         AVAILABLE.  THE DEFAULT CORE ESTIMATE IS 10000.

A BATCH JOB CAN EXECUTE A SNOBOL PROGRAM FROM A DISK FILE IF THE
DATA CARD IS OMITTED AND THE PROGRAM CARD IS CHANGED TO:

      ? FILE PROGRAM = <NAME1>/<NAME2> SERIAL

WHERE <NAME1>/<NAME2> IS THE NAME OF THE DISK FILE.  IN GENERAL,
THE INPUT FILE NAMED "PROGRAM" CAN BE ASSIGNED TO ANY FILE WHICH
CAN BE READ WITH 10,30 BLOCKING.
   ANOTHER CONTROL CARD WHICH IS OCCASIONALLY USEFUL, THOUGH THE
SAME RESULTS CAN BE ACHEIVED WITH THE SNOBOL FUNCTION FILL(), IS
THE FILE CARD, WHICH HAS THE GENERAL FORM:

      ? FILE <NAME> = <NAME1>/<NAME2> <UNIT>

WHERE <NAME> IS THE "INTERNAL" NAME OF THE FILE, AND <NAME1>/<NAME2>
IS THE "EXTERNAL" NAME OF THE FILE ACTUALLY BEING USED.  FOR SNOBOL
JOBS, <NAME> SHOULD BE THE I/O STRING NAME, WITH THE EXCEPTION
OF THE STRING READ, WHICH THE MCP THINKS IS CALLED "PROGRAM".
<UNIT> INDICATES WHAT SORT OF I/O DEVICE IS TO BE USED.  SOME
POSSIBLE VALUES FOR <UNIT> ARE:
   SERIAL  MEANS SERIAL DISK FILE
   PRINT   MEANS LINE PRINTER FILE--NO BACK-UP
   PRINT OR BACK UP  MEANS LINE PRINTER OR PRINTER BACK-UP
   TAPE   MEANS MAGNETIC TAPE
FOR INPUT FILES, IT IS ONLY NECESSARY TO DISTINGUISH DISK FILES
FROM NON-DISK FILES.  OF THE STANDARD SNOBOL FILES, NEWDISK IS
ASSUMED TO BE A DISK FILE UNLESS DECLARED OTHERWISE; ALL OTHERS
ARE NON-DISK UNLESS THE USER SPECIFICALLY ASKS FOR DISK.
SOME EXAMPLES OF FILE CARDS ARE:

   ? FILE CARD = RALPH/GRK SERIAL           (DISK FILE)
   ? FILE PROGRAM = MLX/PHLP                (NON-DISK; ASSIGNS READ)
   ? FILE PRINT = P PRINT OR BACK UP        (GOES TO PRINTER IF IT CAN)
   ? FILE PRINT = PRNT PRINT                (LP, NO BACK-UP)
   ? FILE NEWDISK = ART/BEVL                (GOES TO DISK)

NOTE THAT THE FILE CARD ONLY APPLIES TO THE FIRST FILE THAT AN I/O
STRING IS ASSIGNED TO.  TO RE-ASSIGN AN I/O STRING TO A SECOND FILE,
IT IS NECESSARY TO USE THE SNOBOL FUNCTIONS CLOSE() AND FILL().
   NOTE ESPECIALLY THAT TO ASSIGN THE INPUT FILE READ (WHICH CAN
ONLY BE DONE WITH TELETYPE JOBS, SINCE IT IS ASSIGNED TO THE PROGRAM
FILE AUTOMATICALLY FOR BATCH JOBS), THE CARD USED STARTS WITH
"? FILE PROGRAM", NOT WITH "? FILE READ".


%CP 30
```

```
SOME EXAMPLES OF BATCH JOBS:

TO RUN A SIMPLE BATCH JOB USING A CARD DECK, THE FOLLOWING MIGHT
BE USED:

   ? USER 1234/5678
   ? EXECUTE SNOBOL/SNOBOL
   ? FILE PROGRAM = AJONES
   ? PROCESS = 3
   ? IO = 7
   ? DATA AJONES
         PROGRAM DECK
          .
          .
          .
   END
         DATA DECK



TO RUN A SNOBOL PROGRAM FROM THE DISK FILE TMILLR/SDCK:

   ? USER 9876/5432
   ? EXECUTE SNOBOL/SNOBOL
   ? FILE PROGRAM = TMILLR/SDCK SERIAL
   ? PROCESS = 5
   ? IO = 10
   ? CORE = 18000



%CP 25
2.2. SNOBOL CONTROL CARDS.

   VARIOUS ACTIONS OF THE SNOBOL COMPILER ARE CONTROLED BY SNOBOL
CONTROL CARDS, WHICH ARE ANY CARDS WITH A "-" IN COLUMN 1.  IF SUCH
A CARD IS NOT A VALID CONTROL CARD, A DIAGNOSTIC MESSAGE WILL BE
PRINTED, AND THE CARD WILL BE IGNORED.  THE CONTROL CARDS THAT ARE
CURRENTLY RECOGNIZED ARE:



%CP 4
-LIST <INTEGER>
   THIS INITIATES LISTING OF THE PROGRAM ON THE LINE PRINTER FILE,
   PRINT.  THERE WILL BE <INTEGER> BLANK LINES BETWEEN EACH LINE
   OF THE LISTING.  IF <INTEGER> IS OMITTED, 0 (ZERO) IS ASSUMED.



%CP 2
-UNLIST
   THIS CARD STOPS THE LISTING.



%CP 3
```

```
-PCC
   THIS CAUSES ALL CONTROL CARDS TO BE LISTED, EVEN IF THE PROGRAM
   ITSELF IS NOT BEING LISTED.



%CP 2
-EJECT
   THIS EJECTS THE LISTING TO THE TOP OF THE NEXT PAGE



%CP 3
-SPACE <INTEGER>
   THIS PRODUCES <INTEGER> BLANK LINES IN THE LISTING.



%CP 6
-WIDTH <INTEGER>
   THIS SAYS THAT THE COMPILER IS TO USE THE FIRST <INTEGER>
   CHARACTERS OF EACH CARD READ FROM A CARD-IMAGE FILE.  IT DOES
   NOT APPLY TO INPUT FROM A TELETYPE.  THE COMPILER ASSUMES
   THAT 72 CHARACTERS ARE TO BE USED IF THIS CARD ISNT USED. THE
   MAXIMUM CARD WIDTH IS 80.



%CP 6
-26
   THIS SAYS THAT THE DECK WAS PUNCHED ON AN 026 KEYPUNCH, WITH
   THE IBM/CDC CHARACTER SET.  A CHARACTER-SET TRANSLATION WILL
   BE MADE BY THE COMPILER BEFORE COMPILING EACH CARD.  THE HANDLING
   OF "SPECIAL CHARACTERS" NOT ON THE 026 IS NOT VERY WELL-DEFINED.
   THIS CARD CAN ALSO BE WRITTEN "-026" IF DESIRED.



%CP 9
-3600
   THIS CAUSES THE SAME CONVERSION AS THE -26 CARD, AND ALSO TRANSLATES
   THE I/O STRING NAMES USED ON THE CDC3600 SNOBOL AS IMPLEMENTED
   AT THE UNIVERSITY OF WISCONSIN.  IN PARTICULAR, THE FOLLOWING
   SUBSTITUTIONS ARE MADE:

      "SYSPIT" BECOMES "READ  "
      "SYSLOK" BECOMES "LOOK  "
      "SYSPPT" BECOMES "PUNCH "



%CP 2
-B5500
   THIS CANCELS ANY PREVIOUS -26 OR -3600 CARD.



```

```
%CP 5
-PUNCH
   THIS CAUSES THE COMPILER TO PUNCH OUT THE PROGRAM DECK ON THE
   CARD PUNCH, STARTING WITH THE NEXT CARD.  THIS CAN BE USED WITH
   A -26 OR -3600 CARD TO PRODUCE A B5500 DECK FROM A SNOBOL PROGRAM
   WRITTEN FOR ANOTHER MACHINE.



%CP 8
-INFORM
   THIS TURNS ON A SET OF SYSTEM INFORMATIVE DIAGNOSTICS.  IT IS
   EQUIVALENT TO THE RUN-TIME USE OF MODE("INFORM").  SOME MESSAGES
   THAT ARE PRODUCED BY THE COMPILER IF THIS CARD IS USED ARE:
   LABELS REFERENCED THAT ARE UNDEFINED (AT THE END OF THE LISTING);
   INACCESSIBLE INSTRUCTIONS; USING -DEFINE TO RE-DEFINE A PREVIOUSLY
   DEFINED FUNCTION; USING NON-NUMERIC LITERALS IN ARITHMETIC;
   AND OTHER THINGS THAT MAY BE ADDED IN THE FUTURE.



%CP 2
-SILENCE
   THIS CANCELS ANY PREVIOUS -INFORM CARD.



%CP 4
-WAIT <INTEGER>
   THIS SETS THE WAITING TIME FOR TELETYPE I/O TO <INTEGER>
   SECONDS. IT IS EQUIVALENT TO USING WAIT("<INTEGER>") AT
   RUN TIME.



%CP 12
-LIMIT <QUANTITY> <INTEGER>
   THIS SETS A PROGRAM LIMIT TO THE THING NAMED BY <QUANTITY>
   TO THE VALUE OF <INTEGER>.  IF THIS LIMIT IS EXCEEDED, THE
   PROGRAM WILL BE TERMINATED ABNORMALLY.  SOME POSSIBLE VALUES
   FOR <QUANTITY> ARE:

      RULES   SETS A LIMIT TO THE NUMBER OF RULES THAT CAN BE
              EXECUTED.
      PROCESS SETS A TIME LIMIT (IN SECONDS) TO THE PROGRAMS
              CPU TIME.
      IO      SETS A TIME LIMIT (IN SECONDS) TO THE PROGRAMS
              I/O TIME.



%CP 9
-SIZE <INTEGER>
   THIS TELLS THE COMPILER HOW MANY INSTRUCTIONS TO EXPECT IN THE
   PROGRAM.  IT NEED NOT BE USED; IT ALLOWS THE COMPILER TO SET
   UP ITS TABLES TO TRY TO MINIMIZE THE TIME FOR BOTH COMPILATION
   AND EXECUTION.  THE VALUE OF <INTEGER> SHOULD BE ABOUT (SAY,
```

WITHIN 25 OF) THE NUMBER OF INSTRUCTIONS IN THE PROGRAM. THIS CARD CAN CAUSE SAVINGS OF 5-10% IN LARGE PROGRAMS. IF USED, A -SIZE CARD MUST APPEAR BEFORE ALL PROGRAM MATERIAL (INCLUDING -SET, -DEFINE, -COMPILE, AND -LOAD CARDS).

%CP 21

-SET <IDENTIFIER> <LITERALS>

THIS CARD ASSIGNS AN INITIAL VALUE TO THE VARIABLE NAMED <IDENTIFIER>. THE VALUE IS DETERMINED BY SCANNING THE CARD FOR ALL LITERALS, AND CONCATENATING THEM. ANYTHING BETWEEN THE LITERALS IS TREATED AS COMMENT. THUS, A LONG STRING CAN BE BROKEN UP BETWEEN SEVERAL CARDS IN ANY CONVENIENT WAY. SOME EXAMPLES OF -SET CARDS ARE:

```
-SET ALPHABET "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
-SET CARDNAME "GEORGE/POSDECK"
-SET SUITS "HEARTS,CLUBS,SPADES,DIAMONDS"
-SET CITIES  "NEW YORK,N.Y./CHICAGO,ILL./BOSTON,MASS./"
.        "SAN FRANCISCO,CALIF./DENVER,COL./CLEVELAND,OHIO/"
.        "NEW ORLEANS,LA./LOS ANGELES,CALIF./ATLANTA,GA./"
.        "PHOENIX,ARIZ./PITTSBURG,PENN./SEATTLE,WASH./"
-SET OPS  " + - x / * "
```

THE MAJOR REASON FOR USING A -SET CARD IS THAT IT SAVES THE SPACE THAT WOULD BE REQUIRED FOR THE EQUIVALENT RUN-TIME ASSIGNMENT STATEMENT. THIS CAN BE IMPORTANT WHEN THE SMALL MEMORY SIZE OF THE B5500 IS TAKEN INTO ACCOUNT.

%CP 34

-DEFINE <LIT1> , <LIT2> , <LIT3>

THIS SETS UP A FUNCTION DURING COMPILATION IN THE SAME WAY THAT A CALL OF DEFINE(<LIT1>,<LIT2>,<LIT3>) WOULD DURING EXECUTION. THE PARAMETERS MUST ALL BE SINGLE LITERALS (NO CONCATENATION WILL BE DONE BY THE COMPILER). LIKE THE -SET CARD, A -DEFINE CARD SAVES THE SPACE AND EXTRA TIME THAT WOULD BE REQUIRED FOR AN INSTRUCTION TO CREATE THE FUNCTION AT RUN TIME. IT USUALLY TURNS OUT EASIER TO USE THE -DEFINE CARD, THOUGH, SINCE THIS MAKES IT POSSIBLE TO PLACE THE DEFINITION TOGETHER WITH THE FUNCTION CODE WITHOUT THE NEED TO LINK THE DEFINITIONS TOGETHER WITH GO-TOS. THE PARAMETERS HAVE THE FOLLOWING MEANINGS:

<LIT1> IS THE "FUNCTION PROTOTYPE", OF THE FORM:

```
<FCT>( <PARAMS> )
```

WHERE <FCT> IS THE FUNCTION NAME, AND <PARAMS> IS A LIST OF FORMAL PARAMETERS, SEPARATED BY COMMAS.

<LIT2> CONTAINS A LABEL, WHICH IS THE ENTRY POINT OF THE FUNCTION. IF <LIT2> IS MISSING (I.E., THE COMMAS HAVE NO LITERAL BETWEEN THEM), THEN THE ENTRY POINT IS TAKEN TO BE SPELLED THE SAME AS <FCT>.

<LIT3> CONTAINS A LIST OF LOCAL VARIABLES--A LIST OF IDENTIFIERS, SEPARATED BY COMMAS.

```

   FOR FURTHER INFORMATION, SEE SECTIONS 3.2 (FUNCTION CALLS) AND 9
   (INTRINSIC FUNCTIONS--DEFINE()).  SOME EXAMPLES OF -DEFINE CARDS:

      -DEFINE "M(A1,A2)","M1","V,XY,ALPH"
      -DEFINE "MAX(A,B)"
      -DEFINE "MIN(MIN,X)"
      -DEFINE "FCT()",,"X"
      -DEFINE "MRGE(L)","MRGENTRY","L1,L2,L3"
      -DEFINE("GLORCH(A,B,C,D,E,F,G,H,I)","GLORK","J,K,L,M,N,O,P")



%CP 4
-DEBUG
   THIS TURNS ON "DEBUG MODE" (SEE SECTION 11.2) DURING COMPILATION, AND
   CAUSES THE JOB TO BE SUSPENDED JUST BEFORE THE FIRST INSTRUCTION IS
   EXECUTED (IF THERE IS A TELETYPE ATTACHED).



%CP 42
-COMPILE <NAME1>/<NAME2>
   THIS CARD CAUSES THE COMPILER TO COMPILE THE DISK FILE NAMED
   <NAME1>/<NAME2> INTO THE PROGRAM AT THE POINT WHERE THE
   CARD IS FOUND.  THE COMPILER WILL READ EVERYTHING IN THE FILE
   UP TO AN END CARD OR THE END-OF-FILE, AND THEN RETURN TO THE
   ORIGINAL FILE (OR TELETYPE).  IF AN END CARD IS FOUND, IT IS
   SEARCHED FOR A LABEL; IF ONE IS FOUND, IT BECOMES THE PROGRAM
   ENTRY POINT (UNLESS A LATER END CARD CHANGES IT).  IF THE FILE
   NAMED IS NOT ON THE DISK, A DIAGNOSTIC WILL BE PRINTED AND THE
   CARD WILL BE IGNORED.  ANY CARD-IMAGE FILE (WITH 80-CHAR LOGICAL
   RECORDS) CAN BE READ BY THE COMPILER.
      THE MOST OBVIOUS USE OF THE -COMPILE CARD IS TO ENABLE USERS TO
   BUILD UP A SNOBOL "LIBRARY" OF FUNCTIONS, WHICH CAN THEN BE USED
   BY ANYONE BY SIMPLY INCLUDING A -COMPILE CARD FOR EACH FUNCTION
   DESIRED.  ALTHOUGH STANDARDS FOR SUCH A LIBRARY ARE CLEARLY NOT
   ENFORCEABLE, SOME GUIDELINES ARE SUGGESTED HERE WHICH SHOULD MAKE
   IT EASIER TO USE LIBRARY FILES:

   1) THOROUGH DOCUMENTATION SHOULD EITHER BE INCLUDED IN THE CODE OR
      THERE SHOULD BE COMMENTS AT THE START OF THE FILE TELLING THE
      USER WHERE TO FIND DOCUMENTATION.
   2) FUNCTIONS SHOULD BE SET UP BY THE -DEFINE CONTROL CARD, RATHER
      THAN BY THE FUNCTION DEFINE(), SO THAT THE USER DOESNT HAVE TO
      SET UP A LOT OF GO-TOS INTO THE CODE.
   3) THE USER SHOULD GET INTO THE CODE BY CALLING A FUNCTION, RATHER
      THAN BY A GO-TO.  THIS MAKES IT MUCH EASIER TO INTEGRATE THE CODE
      WITH THE MAIN PROGRAM.
   4) NO INSTRUCTIONS IN THE LIBRARY FILE SHOULD TRANSFER OUT
      OF THE FILE, WITHOUT EXPLICIT DESCRIPTIONS OF ALL LABELS
      THAT THE CALLING PROGRAM SHOULD CONTAIN.  THIS SHOULD
      BE AVOIDED ENTIRELY IF POSSIBLE.
   5) THERE SHOULD BE VERY EXPLICIT COMMENTS DESCRIBING ALL GLOBAL
      VARIABLES USED OR ALTERED BY THE CODE.  WHENEVER POSSIBLE,
      THE CODE SHOULD DO ITS OWN INITIALIZATION BY MEANS OF
      THE -SET CONTROL CARD, RATHER THAN ASKING THE USER TO
```

```
         FIGURE OUT WHAT TO DO.
      6) LABELS SHOULD BE SUCH THAT USERS OF THE FILE ARE UNLIKELY
         TO DUPLICATE THEM IN THEIR OWN CODE.  THIS IS MOST EASILY
         HANDLED BY MAKING EACH LABEL INCLUDE EITHER THE NAME OF
         THE FILE OR THE NAME OF THE FUNCTION THEY ARE USED IN.  NOTE
         THAT IF TWO LIBRARY FILES USE THE SAME LABEL, THEY CANT
         BE USED IN THE SAME PROGRAM.



%CP 12
-LIBRARY <NAME1>/<NAME2>
    THIS CARD CREATES A "LIBRARY" FILE CONTAINING THE COMPILED
    VERSION OF THE PROGRAM.  THE FILE IS NAMED <NAME1>/<NAME2>.
    THIS LIBRARY FILE CAN THEN BE USED IN A LATER RUN WITHOUT
    RECOMPILING IT BY USING THE -LOAD CARD (SEE BELOW).  A -LIBRARY
    CARD IS NORMALLY THE LAST CARD IN THE PROGRAM DECK, COMING
    JUST BEFORE THE END CARD, BUT IT CAN APPEAR ANYWHERE IN A
    PROGRAM, AND THE PROGRAM STORED IN THE FILE WILL LOOK AS IF
    THE -LIBRARY CARD HAD BEEN FOLLOWED BY AN END CARD.  SEVERAL
    -LIBRARY CARDS CAN BE USED IN THE SAME PROGRAM, IF DESIRED.
    IN ADDITION TO THE PROGRAM ITSELF, ANY LIMITS SET BY -LIMIT
    CARDS ARE SAVED, AS IS THE ENTRY POINT AS OF THE -LIBRARY CARD.



%CP 13
-LOAD <NAME1>/<NAME2>
    THIS LOADS A LIBRARY FILE CREATED BY A -LIBRARY CARD IN A
    PREVIOUS RUN.  IF THE FILE NAMED IS NOT PRESENT OR IS NOT
    A VALID LIBRARY FILE, A DIAGNOSTIC MESSAGE WILL BE PRINTED.
    IF THE FILE WAS THERE, BUT CONTAINED DAMAGED CODE, THE COMPILER
    WILL OFTEN BE UNABLE TO RECOVER FROM THE BAD INFORMATION LOADED
    INTO ITS TABLES WHEN THE PROBLEM IS DISCOVERED; THIS WILL
    CAUSE THE COMPILER TO SIMPLY GIVE UP AND THE JOB WILL TERMINATE.
    ONLY ONE -LOAD CARD CAN BE USED IN A PROGRAM, AND IT MUST BE
    BEFORE ANY OTHER PROGRAM MATERIAL (INCLUDING INSTRUCTIONS, -SET,
    -DEFINE CONTROL CARDS).  THE -LOAD CARD CAN BE FOLLOWED BY MORE
    PROGRAM, WHICH WILL BE COMPILED AND ADDED ONTO THE PROGRAM
    LOADED FROM THE FILE.  (THE FILE, OF COURSE, WILL NOT BE ALTERED.)



%SECTION 3
%PAGE 1
3. PROGRAM ORGANIZATION.

    A SNOBOL PROGRAM CONSISTS OF A SET OF INSTRUCTIONS.  THE PROGRAM
CAN BE INPUT FROM A CARD-IMAGE FILE, OR FROM A TELETYPE.  EACH CARD
IMAGE FROM AN INPUT FILE STARTS A NEW INSTRUCTION, UNLESS THE FIRST
CHARACTER IS ".", IN WHICH CASE THE CARD IS A CONTINUATION OF THE
PREVIOUS INSTRUCTION.  IN B5500 SNOBOL, IT IS ALSO POSSIBLE TO FIT
SEVERAL INSTRUCTIONS ON ONE CARD, THOUGH THIS IS RARELY DONE.
    FOR JOBS RUN FROM A TELETYPE, THE PROGRAM IS READ FROM THE TELETYPE,
WITH EACH INPUT BEING AN INSTRUCTION.  CONTINUATIONS ARE NOT VALID
FROM A TELETYPE.  THIS IS NOT REALLY A RESTRICTION, SINCE FEW PEOPLE
ACTUALLY ATTEMPT TO INPUT ENTIRE PROGRAMS FROM A TELETYPE.  IT IS FAR
```

EASIER TO CREATE A DISK FILE CONTAINING THE PROGRAM (SAY, WITH AN EDITING PROGRAM), AND USE THE -COMPILE CARD TO DIRECT THE COMPILER TO THE FILE.

SCATTERED AMONG THE INSTRUCTIONS IN A PROGRAM CAN BE ANY NUMBER OF SNOBOL CONTROL CARDS, WHICH ARE INSTRUCTIONS TO THE COMPILER, BUT ARE NOT ACTUALLY PART OF THE PROGRAM. IN ADDITION TO THE USUAL CONTROL CARDS, THE B5500 HAS CARDS THAT CAN BE USED TO DEFINE FUNCTIONS DURING COMPILATION, AND TO ASSIGN INITIAL VALUES TO PROGRAM VARIABLES.

A PROGRAM DECK CAN ALSO CONTAIN ANY NUMBER OF COMMENT CARDS, WHICH ARE CARDS STARTING WITH "*" OR "%". SUCH CARDS ARE PRINTED ON A LISTING, BUT OTHERWISE IGNORED. USERS ARE ENCOURAGED TO MAKE LIBERAL USE OF COMMENTS IN THEIR PROGRAMS.

AT THE END OF A SNOBOL PROGRAM SHOULD BE AN "END" CARD, WHICH IS A CARD WITH "END " IN COLUMNS 1-4. THE FIRST STRING OF NON-BLANK CHARACTERS ON THE CARD IS TAKEN AS A LABEL WHICH IS THE "ENTRY POINT" TO THE PROGRAM. THAT IS, IT IS THE LABEL OF THE FIRST INSTRUCTION TO BE EXECUTED. IF THE ENTIRE CARD IS BLANK, THE FIRST INSTRUCTION IN THE PROGRAM IS ASSUMED TO BE THE ENTRY POINT.

INSTRUCTIONS CAN BE EITHER LABELED OR UNLABELED. LABELS ARE NEEDED WHENEVER AN INSTRUCTION MUST BE TRANSFERED TO BY ANOTHER ONE, AND OTHER INSTRUCTIONS ARE GENERALLY UNLABELED. A LABEL IS ANY STRING OF NON-BLANK CHARACTERS STARTING IN COLUMN 1. IF COLUMN 1 IS A BLANK, THE INSTRUCTION IS UNLABELED. LABELS ARENT REALLY A PART OF THEIR INSTRUCTIONS, BUT SIMPLY SERVE AS "NAMES" FOR THE INSTRUCTIONS.

A SNOBOL PROGRAM IS A SINGLE SET OF INSTRUCTIONS, WITH NO SUB-PARTS (UNLIKE MOST OTHER PROGRAMMING LANGUAGES THAT HAVE SEPARATE BLOCKS OF CODE FOR EACH SUB-ROUTINE OR FUNCTION OR WHATEVER THEY ARE CALLED). ONE CONSEQUENCE OF THIS IS THAT STATEMENT LABELS IN A SNOBOL PROGRAM MUST BE UNIQUE. THERE IS VERY LITTLE CONSISTENCY BETWEEN IMPLEMENTATIONS ON HOW DOUBLY-DEFINED LABELS ARE TREATED. THE METHOD USED ON THE B5500 IS THIS: THE SECOND LABEL OVERWRITES THE FIRST AND CAUSES IT TO BE RE-DEFINED, BUT A WARNING MESSAGE IS PRINTED. THIS CAN BE USEFUL AT TIMES ON INTERACTIVE JOBS TO RE-DEFINE AN INSTRUCTION PREVIOUSLY COMPILED.

%PAGE

3.1. INSTRUCTIONS.

```
   <INSTRUCTION>  ::= <LABELED INST>
                  ::= <UNLABELED INST>
   <LABELED INST> ::= <LABEL> <UNLABELED INST>
   <UNLABELED INST> ::= <BLANKS> <INST>
   <INST>           ::= <RULE>
                    ::= <GO-TO PART>
                    ::= <RULE> <GO-TO PART>
```

THE VARIOUS FORMS OF INSTRUCTIONS CAN BE TREATED AS SPECIAL CASES OF THE GENERAL FORM:

```
      <LABEL> <RULE> <GO-TO PART>
```

WHERE IT IS REMEMBERED THAT THERE IS ALWAYS AT LEAST ONE BLANK AFTER <LABEL>, IF THIS IS PRESENT, OR ELSE THE INSTRUCTION STARTS WITH AT LEAST ONE BLANK.

THE <LABEL>, OF COURSE, IS JUST A NAME FOR THE INSTRUCTION. A <LABEL> IS ONLY NEEDED IF SOME OTHER PART OF THE PROGRAM IS TO TO TRANSFER TO THE INSTRUCTION, THOUGH THERE IS NO OBJECTION TO USING LABELS WHICH ARE NEVER REFERENCED. LABELS ARE DESCRIBED IN SECTION 7.

THE <RULE> IS THE PART OF AN INSTRUCTION THAT OPERATES ON DATA; IT IS THE PART THAT "DOES THINGS". SEE SECTION 4 FOR DESCRIPTIONS OF THE VARIOUS TYPES OF RULES AVAILABLE.

THE <GO-TO PART> TELLS WHAT INSTRUCTIONS ARE TO FOLLOW NEXT. SNOBOL3 ALLOWS CONDITIONAL BRANCHING ON THE SUCCESS OR FAILURE OF THE RULE. AN INSTRUCTION CAN ALSO CALCULATE THE LABEL OF THE NEXT INSTRUCTION. OMISSION OF THE GO-TO PART CAUSES THE INSTRUCTION TO "FALL THROUGH" TO THE NEXT INSTRUCTION IN THE PROGRAM. SEE SECTION 7 FOR DETAILED DESCRIPTIONS OF GO-TO PARTS.

%CP 8

3.2 PROGRAM-DEFINED FUNCTIONS.

IN ADDITION TO THE FUNCTIONS PRE-DEFINED BY THE SNOBOL INTERPRETER, A PROGRAMMER CAN DEFINE HIS OWN FUNCTIONS CODED IN SNOBOL BY USING EITHER THE FUNCTION DEFINE() (SEE SEC. 9) OR THE -DEFINE CONTROL CARD (SEE SEC.2.2). IN SECTION 5.3, THE USE OF FUNCTIONS CALLS IS DESCRIBED; THIS SECTION DESCRIBES THE DEFINITION OF THE FUNCTIONS.

%CP 15

3.2.1. FUNCTION CALLS.

WHEN A PROGRAM-DEFINED FUNCTION IS CALLED, THE FOLLOWING OCCURS:

1) THE STRING WITH THE SAME NAME AS THE FUNCTION IS PUSHED DOWN, AND SET NULL.
2) THE FORMAL PARAMETERS ARE PUSHED DOWN, AND THEN ASSIGNED THE VALUES OF THE CORRESPONDING ACTUAL PARAMETERS. IF THERE ARENT ENOUGH ACTUAL PARAMETERS, THE EXTRA FORMAL PARAMETERS ARE SET NULL.
3) THE LOCAL VARIABLES ARE PUSHED AND SET NULL.
4) THE PROGRAM TRANSFERS TO THE FUNCTIONS ENTRY POINT.

%CP 15

3.2.2. FUNCTION RETURNS.

WHEN A PROGRAM TRANSFERS TO THE "RESERVED LABELS" RETURN AND FRETURN, THE LAST FUNCTION CALLED IS TERMINATED, AND THE FOLLOWING ACTIONS OCCUR:

1) THE VALUE OF THE STRING NAMED THE SAME AS THE FUNCTION IS GIVEN TO THE CALLING INSTRUCTION AS THE VALUE OF THE FUNCTION.
2) THIS STRING, THE FORMAL PARAMETERS, AND THE LOCAL VARIABLES ARE ALL POPPED UP, RETURNING THEM TO THEIR VALUES BEFORE THE CALL.
3) IF THE RETURN WAS MADE BY TRANSFERING TO "FRETURN", THE CALLING INSTRUCTION FAILS; IF THE RETURN WAS BY "RETURN", THE CALLING INSTRUCTION CONTINUES AS USUAL.

%CP 10

3.2.3. FUNCTION CODE.

IN CONTRAST WITH MOST OTHER PROGRAMMING LANGUAGES, A SNOBOL FUNCTION HAS NO "BODY" OF CODE THAT BELONGS TO THE FUNCTION ALONE. THAT IS, FROM ITS ENTRY POINT, A FUNCTION CAN TRANSFER TO ANY INSTRUCTION IN THE PROGRAM. RETURN OCCURS ONLY WHEN A TRANSFER TO RETURN OR FRETURN IS MADE. THIS MEANS THAT SEVERAL SNOBOL FUNCTIONS CAN SHARE ALL OR PART OF THEIR CODE. ALSO, NOTE THAT IF A STRING IS NOT ONE OF THE STRINGS PUSHED AT THE CALL OF A FUNCTION, THEN THAT FUNCTION CAN USE OR ALTER THE STRING FREELY.

%CP 12

3.2.4. CALLING DEFINE().

ONE VERY COMMON PROBLEM WITH NOVICE SNOBOL PROGRAMMERS IS THE FAILURE TO REALIZE THAT THE FUNCTION DEFINE() ACTUALLY CREATES A FUNCTION WHEN IT IS CALLED. IT IS NOT ENOUGH TO INCLUDE A CALL OF DEFINE() IN A PROGRAM--THE PROGRAM MUST EXECUTE THE STATEMENT CONTAINING THE CALL, OR THE FUNCTION WILL NOT BE DEFINED.

THIS PROBLEM CAN BE AVOIDED ON THE B5500 BY USING THE -DEFINE CONTROL CARD (SEC.2.2). SINCE MOST SNOBOL SYSTEMS DONT HAVE THIS CONTROL CARD, USING IT WILL MAKE THE PROGRAM INCOMPATIBLE WITH THESE OTHER SNOBOLS.

%CP 25

3.3. PROGRAM ENTRY POINTS.

IN ORDER TO START EXECUTION OF A SNOBOL PROGRAM, IT IS NECESSARY TO INDICATE WHERE THE EXECUTION IS TO BEGIN. THE FIRST INSTRUCTION TO BE EXECUTED, THE "ENTRY POINT", IS INDICATED BY THE END CARD AT THE END OF THE PROGRAM. THE FIRST FOUR CHARACTERS OF THIS CARD ARE "END "; THE COMPILER SCANS THE REST OF THE CARD FOR A NON-BLANK CHARACTER. IF ONE IS FOUND, THE STRING UP TO THE NEXT BLANK OR TO THE END OF THE CARD IS THE LABEL OF THE ENTRY POINT. FOR EXAMPLE:

END BEGIN

SAYS THAT EXECUTION IS TO START AT THE LABEL BEGIN. IF THERE IS NO LABEL ON THE END CARD, THE FIRST INSTRUCTION IN THE PROGRAM IS TAKEN TO BE THE ENTRY POINT.

IF THE PROGRAM CONTAINS -COMPILE CARDS, AND THE FILES THEMSELVES CONTAIN END CARDS, THE LABELS (IF ANY) ON THESE END CARDS ARE NOTED, AND THE LAST SUCH LABEL IS THE PROGRAM ENTRY POINT.

IF THE ENTRY POINT IS A LABEL THAT DOESNT OCCUR IN THE PROGRAM, A DIAGNOSTIC MESSAGE TO THAT EFFECT IS PRINTED, AND NO EXECUTION TAKES PLACE. THIS CAN BE USED TO GET COMPILES FOR SYNTAX CHECKING OR A COMPILE TO LIBRARY WITHOUT IMMEDIATE EXECUTION.

%SECTION 4

%PAGE 1

4. RULES.

```
<RULE>      ::= <STR REF>
            ::= <STR REF> <PATTERN>
            ::= <STR REF> <REPLACEMENT>
            ::= <STR REF> <PATTERN> <REPLACEMENT>
<STR REF> ::= <ELEMENT>
<PATTERN> ::= <ELEMENT>
            ::= <STR VAR>
            ::= <PATTERN> <PATTERN>
            ::= <PATTERN> <BACK REF>
<REPLACEMENT>    ::= <RPL DELIMITER> <EXPRESSION>
<RPL DELIMITER> ::= ←
                 ::= =
```

THERE ARE SEVERAL VARIETIES OF RULES, AS SHOWN ABOVE, BUT ALL CAN BE VIEWED AS SPECIAL CASES OF THE GENERAL FORM:

```
<STR REF> <PATTERN> <REPLACEMENT>
```

IF THE PATTERN IS OMITTED, AN ARBITRARY STRING VARIABLE WILL BE ASSUMED; THIS STRING VARIABLE WILL MATCH THE ENTIRE STRING REFERENCE, RESULTING IN AN "ASSIGNMENT STATEMENT". IF THE REPLACEMENT IS OMITTED, NO REPLACEMENT TAKES PLACE. IF BOTH PATTERN AND REPLACEMENT ARE OMITTED, THE STATEMENT MERELY EVALUATES THE STRING REFERENCE. IN THE LATTER CASE, THE STRING REFERENCE CAN ALSO BE OMITTED, AND THE STATEMENT IS A "NO-OP".

THE FOLLOWING SUB-SECTIONS DESCRIBE MORE THOROUGHLY THE VARIOUS PARTS OF A RULE.

%CP 20

4.1. THE STRING REFERENCE.

<STR REF>, THE "STRING REFERENCE", IS THE STRING THAT THE RULE IS TO OPERATE ON; IT IS THE "SUBJECT" OF THE STATEMENT. THE COMPILER ASSUMES THAT THE SHORTEST COMPLETE EXPRESSION AT THE START OF A RULE IS THE STRING REFERENCE; IF A STRING REFERENCE CONSISTING OF SEVERAL STRINGS CONCATENATED IS DESIRED, IT MUST BE SURROUNDED BY PARENTHESES. SOME EXAMPLES OF VALID STRING REFERENCES ARE:

```
X
(ALPHA BETA + GAMMA)
SIZE(A)
$("A" N)
"123456789"
""
A + B
```

EACH TYPE OF RULE INTERPRETS ITS STRING REFERENCE SLIGHTLY DIFFERENTLY, AND SOME RULES WILL NOT ACCEPT ALL OF THE ABOVE AS STRING REFERENCE. THE FOLLOWING DESCRIPTIONS OF INDIVIDUAL RULE TYPES DESCRIBES EACH USAGE IN DETAIL.

%CP 20

4.2. RULES WITH STRING REFERENCE ONLY.

```
   <RULE>    ::=  <STR REF>
```

THIS TYPE OF RULE SIMPLY EVALUATES THE STRING REFERENCE. THE USUAL USE OF SUCH A RULE IS TO CALL A FUNCTION. THIS CAN BE DONE EITHER FOR THE SIDE-EFFECTS OF THE FUNCTION, OR TO BRANCH ON THE SUCCESS OR FAILURE OF THE FUNCTION. AN EXAMPLE OF THE LATTER IS:

```
      .GT(SIZE(A),"5")             :S(RM)
```

THIS STATEMENT WILL TRANSFER TO RM IF THE STRING A CONTAINS MORE THAN FIVE CHARACTERS.

ANOTHER, RARELY USED, FORM OF THIS TYPE OF RULE IS WITH AN INPUT STRING AS THE STRING REFERENCE. THIS TYPE OF RULE WILL INPUT A RECORD (IF POSSIBLE), AND THE INPUT WILL BE LOST. FOR EXAMPLE, IF THE PROGRAM HAS USED THE STRING LOOK TO READ THE NEXT CARD FROM THE FILE READ, THE CARD CAN BE CLEARED FROM THE BUFFER BY:

```
      READ
```

AND THE NEXT INPUT WILL GET THE NEXT RECORD.

%CP 37

4.3. RULES WITH PATTERNS.

```
   <RULE>    ::=  <STR REF>  <PATTERN>
```

THIS CAUSES A "PATTERN MATCH" TO TAKE PLACE. A PATTERN IS A DESCRIPTION OF A STRING, AND A PATTERN MATCH IS AN ATTEMPT TO FIND A SUBSTRING OF THE STRING REFERENCE WHICH IS DESCRIBED BY THE PATTERN. THE VARIOUS MEANS OF BUILDING PATTERNS ARE DESCRIBED IN SECTION 6. THE GENERAL WAY IN WHICH THIS TYPE OF RULE IS EXECUTED IS AS FOLLOWS:

1) EVALUATE THE STRING REFERENCE. IF IT IS AN INPUT STRING, READ A RECORD.
2) EVALUATE THE PATTERN, FROM LEFT TO RIGHT.
3) ATTEMPT TO MATCH THE PATTERN ON SOME SUBSTRING OF THE STR REF.

OF COURSE, IF ANY PART OF THE RULE FAILS, THE RULE IS TERMINATED, AND THE FAILURE PART OF THE GO-TO PART IS EVALUATED.

FOR EXAMPLE, ASSUME THAT X = "ABCDE" AND E = "BC". THE RULE

```
      X  E "D"     /S(M)
```

WILL SUCCEED, AND THE PROGRAM WILL TRANSFER TO M. THE STRING REFERENCE, "ABCDE", CONTAINS THE PATTERN, E "D", OR "BCD". ON THE OTHER HAND, THE RULE

```
      X  E "C"     /S(M)
```

WILL FAIL, AND TRANSFER TO M WILL NOT BE MADE, BECAUSE THE STRING REFERENCE DOES NOT CONTAIN THE PATTERN E "C", OR "BCC".

FOR ANOTHER EXAMPLE, THE RULE

```
850000        "1357"  D       /S(X)
851000
852000 WILL SUCCEED (AND TRANSFER TO X) IF D IS ANY OF THE FOLLOWING: "",
853000 "1", "13", "135", "1357", "3", "35", "357", "5", "57", "7".
854000
855000
856000
857000 %CP 30
858000 4.4. RULES WITH REPLACEMENTS.
859000
860000    <RULE>            ::=  <STR REF> <REPLACEMENT>
861000    <REPLACEMENT>     ::=  <RPL DELIMITER> <EXPRESSION>
862000    <RPL DELIMITER> ::=  ←
863000                      ::=  =
864000
865000    THIS IS AN "ASSIGNMENT STATEMENT".  THE STRING REFERENCE MUST
866000 BE A STRING NAME (AN IDENTIFIER OR AN INDIRECT EXPRESSION).  THE
867000 VALUE OF THE <EXPRESSION> BECOMES THE NEW VALUE OF THE STRING
868000 NAMED.  IF THIS STRING IS AN OUTPUT STRING, OUTPUT OCCURS AFTER
869000 THE ASSIGNMENT.  NOTE THAT THERE ARE TWO DELIMITERS LISTED, "←"
870000 AND "=".  MANY COMPUTERS DO NOT HAVE THE "←" IN THEIR CHARACTER
871000 SETS, SO THEY WILL ONLY ACCEPT A "=".  THE B5500 HAS BOTH, AND
872000 EITHER CAN BE USED TO INDICATE A REPLACEMENT.  SOME EXAMPLES OF
873000 ASSIGNMENT STATEMENTS ARE:
874000
875000    X = "123"
876000    ALPH.1 = BET.3 + GAMM.7
877000    LIST ← NUM "/" ITEM "," LIST
878000    $("X" N)  ←  $("X" N) ";" E
879000    $$FR(A,$B N)  =  $A $("B" N)  FL(A1,A2 "," A3)
880000    $E  ← F G "," EXP
881000    I = J
882000    N = .LT(N,MAX) N + "1"
883000
884000 THE LAST EXAMPLE IS A "CONDITIONAL ASSIGNMENT".  IF THE CALL OF .LT()
885000 FAILS (I.E., IF N ≥ MAX), THE STATEMENT WILL FAIL DURING EVALUATION
886000 OF THE REPLACEMENT, AND THE ASSIGNMENT WILL NOT BE CARRIED OUT.
887000
888000
889000
890000 %CP 30
891000 4.5. RULES WITH BOTH PATTERN AND REPLACEMENT.
892000
893000    <RULE>  ::=  <STR REF> <PATTERN> <REPLACEMENT>
894000
895000   THIS IS THE MOST COMPLEX FORM OF A RULE, IN WHICH A PATTERN MATCH
896000 IS ATTEMPTED, AND IF IT SUCCEEDS, THE MATCHED SUBSTRING OF THE STRING
897000 REFERENCE IS REPLACED WITH THE VALUE OF THE REPLACEMENT EXPRESSION.
898000 SPECIFICALLY, THE FOLLOWING ACTIONS OCCUR IN THIS TYPE OF RULE:
899000
900000 1) THE STRING REFERENCE IS EVALUATED, DOING INPUT IF IT IS AN
901000    INPUT STRING.
902000 2) THE PATTERN IS EVALUATED FROM LEFT TO RIGHT.
903000 3) A PATTERN MATCH IS ATTEMPTED.  IF THIS FAILS (AS WITH ANY
904000    OTHER FAILURE), THE RULE IS TERMINATED AND THE FAILURE EXIT
905000    IS TAKEN.
906000 4) THE REPLACEMENT IS EVALUATED.
```

5) THE SUBSTRING OF THE STRING REFERENCE MATCHED BY THE PATTERN IS REPLACED BY THE VALUE OF THE REPLACEMENT EXPRESSION. IF THE STRING REFERENCE IS AN OUTPUT STRING, OUTPUT OCCURS.

AS USUAL, ANY PART OF THE RULE CAN FAIL. IF THIS HAPPENS, THE RULE IS IMMEDIATELY TERMINATED, AND THE FAILURE PART OF THE GO-TO PART IS EVALUATED.

FOR EXAMPLE, IF AB = "COME", THEN THE RULE

```
    AB  "OM" = "RAT"
```

WILL SUCCEED, AND GIVE AB THE NEW VALUE "CRATE". IF AB HAD BEEN, SAY, "CAME", THE PATTERN MATCH WOULD HAVE FAILED, AND AB WOULD NOT HAVE BEEN ALTERED.

NOTE THAT REPLACEMENT APPLIES ONLY TO THE SUBSTRING MATCHED BY THE PATTERN, NOT TO THE WHOLE STRING REFERENCE, AND THAT THE NEW VALUE THAT REPLACES THE SUBSTRING NEED NOT BE THE SAME SIZE AS THE SUBSTRING.

AS A SPECIAL CASE, THE REPLACEMENT PART CAN BE EMPTY (SEE THE BNF DEFINITION OF <EXPRESSION>). IN THIS CASE, THE REPLACEMENT EXPRESSION IS THE NULL STRING, AND THE MATCHED SUBSTRING IS DELETED. FOR EXAMPLE, THE RULE:

```
L      A  "," =     :S(L)F(M)
```

WILL DELETE ALL PERIODS FROM A, AND THEN TRANSFER TO M.

%SECTION 5
%PAGE 1

5. EXPRESSIONS.

```
    <EXPRESSION>  ::=  <ELEMENT>
                  ::=  <EXPRESSION> <CONCATENATE OP> <ELEMENT>
                  ::=  <EMPTY>
    <ELEMENT>     ::=  <NAMED EXPR>
                  ::=  <VALUE EXPR>
    <NAMED EXPR>  ::=  <IDENTIFIER>
                  ::=  $  <ELEMENT>
    <VALUE EXPR>  ::=  <LITERAL>
                  ::=  <GROUPING>
                  ::=  <FCT CALL>
                  ::=  <ARITH EXPR>
    <CONCATENATE OP> ::=  <BLANKS>
    <IDENTIFIER>  ::=  <LETTER>
                  ::=  <IDENTIFIER><LETTER>
                  ::=  <DIGIT>
                  ::=  <IDENTIFIER><DIGIT>
                  ::=  .
                  ::=  <IDENTIFIER>  .
    <LITERAL>     ::=  "  <STRING>  "
    <GROUPING>    ::=  (  <EXPRESSION>  )
    <FCT CALL>    ::=  <FCT NAME>(  <PARAMS>  )
    <FCT NAME>    ::=  <IDENTIFIER>
    <PARAMS>      ::=  <EXPRESSION>
                  ::=  <PARAMS>  ,  <EXPRESSION>
```

```
<ARITH EXPR>  ::=  <A-TERM>
              ::=  -  <A-TERM>
<A-TERM>      ::=  <A-TERM>  <ADD AP>  <M-TERM>
              ::=  <M-TERM>
<M-TERM>      ::=  <M-TERM>  <MUL OP>  <E-TERM>
              ::=  <E-TERM>
<E-TERM>      ::=  <E-TERM>  <EXP OP>  <ARITH ELT>
              ::=  <ARITH ELT>
<ARITH ELT>   ::=  <NAMED EXPR>
              ::=  <GROUPING>
              ::=  <FCT CALL>
              ::=  <LITERAL>
<ADD OP>      ::=  +
              ::=  -
<MUL OP>      ::=  *
              ::=  x
              ::=  /
<EXP OP>      ::=  **
```

EXPRESSIONS ARE USED TO GENERATE STRING VALUES, AND TO FORM NEW STRINGS FROM OLD. THE VARIOUS TYPES OF EXPRESSIONS ARE DESCRIBED IN THE FOLLOWING SECTIONS.

%CP 12

5.1. NAMES AND VALUES.

IN SNOBOL3, BOTH NAMES AND VALUES OF DATA ARE STRINGS, AND SO IT IS QUITE NATURAL THAT THERE SHOULD BE SOME CONFUSION BETWEEN THE TWO AMONG PROGRAMMERS; EXPECIALLY SINCE THE INDIRECTION OPERATOR ("$") PROVIDES A WAY TO MANIPULATE STRING NAMES AS WELL AS THEIR VALUES. IN GENERAL, ANY STRING CAN BE USED AS BOTH A NAME AND AS A VALUE. ONE RESTRICTION TO THIS IS THAT NAMES THAT APPEAR AS SUCH IN SNOBOL CODE CAN ONLY BE THOSE THAT ARE "IDENTIFIERS"-- NON-NULL STRINGS OF LETTERS, DIGITS, AND PERIODS. OTHER NAMES MUST BE HANDLED INDIRECTLY.

%CP 25

5.1.1. LITERALS.

THE SIMPLEST WAY TO DESCRIBE A STRING IS TO GIVE THE STRING ITSELF. THIS IS DONE BY USING A "LITERAL", WHICH IS A STRING OF CHARACTERS BOUNDED BY QUOTES. QUOTES THEMSELVES CAN NOT OCCUR IN LITERALS, AND MUST BE HANDLED DIFFERENTLY. SOME EXAMPLES OF LITERALS ARE:

```
"1"
"A+B"
"THIS IS A STRING."
"ABCDEFGHIJKLM"
" GEORGE Q. SMITH "
"          "
""
```

THE LAST LITERAL IS THE NULL STRING, WHICH HAS A POSITION IN STRING

MANIPULATION SIMILAR TO ZERO IN ARITHMETIC (IN FACT, IT IS THE IDENTITY ELEMENT FOR THE OPERATION OF CONCATENATION).

NOTE THAT, AS IN THE NEXT-TO-LAST EXAMPLE, A BLANK IS AS "GOOD" A CHARACTER AS ANY OTHER, AND THE STRING " " IS NOT THE SAME AS THE STRING "   ". THEY CONTAIN DIFFERENT NUMBERS OF BLANKS.

TO HANDLE QUOTES, SNOBOL3 PROVIDES A SPECIAL STRING NAME, QUOTE, WHICH IS INITIALIZED TO CONTAIN A SINGLE QUOTE AS ITS VALUE. THIS CAN, OF COURSE, BE CHANGED BY THE PROGRAM.

*CP 25

5.1.2. NAMES

IT IS NECESSARY, OF COURSE, TO TALK ABOUT STRINGS WITHOUT SPECIFYING (OR KNOWING) THEIR ACTUAL VALUES. SO SNOBOL PROVIDES THE ABILITY TO GIVE A STRING A NAME. A NAME IS ITSELF A STRING, WHOSE VALUE IS ANOTHER (NOT NECESSARILY DIFFERENT) STRING. IN GENERAL, ANY STRING CAN BE USED AS A NAME, BY USING INDIRECTION (SEE THE NEXT SECTION). NAMES ARE ASSIGNED VALUES IN TWO DIFFERENT WAYS: BY AN "ASSIGNMENT", AND BY USING THE NAME WITH A STRING VARIABLE. SOME EXAMPLES OF ASSIGNMENT STATEMENTS ARE:

```
      X = "1234"
      I = J
      ALPHA = A B (C+D) FF("=")
      N = .GT(N,"0")  N - "1"
```

ANOTHER WAY TO GIVE A STRING NAME A VALUE IS TO ATTACH THE NAME TO A STRING VARIABLE IN A PATTERN. IF THE PATTERN MATCH SUCCEEDS, THE SUBSTRING MATCHED BY THE STRING VARIABLE WILL BECOME THE NEW VALUE OF THE STRING NAME. SOME EXAMPLES ARE:

```
      "1234"  *X*
      TR   ","  *I*  "/"  *J*  ","
      C  *CH/"1"*  *C*
      CARD  *R/"72"*  *SEQ*
      EXP  *(E1)*
```

NOTE THAT THE FIRST EXAMPLE IN THIS SET MEANS THE SAME AS THE FIRST EXAMPLE IN THE PREVIOUS SET.

NAMES THAT ARE USED IN SNOBOL CODE MUST SATISFY THE RESTRICTION OF BEING AN "IDENTIFIER"--A NON-NULL STRING OF LETTERS, DIGITS, AND PERIODS. ANY OTHER STRING WHICH IS TO BE USED AS A NAME MUST BE HANDLED BY INDIRECTION.

*CP 40

5.1.3. INDIRECTION.

SINCE NAMES AND VALUES IN SNOBOL3 ARE BOTH STRINGS, IT IS USEFUL TO BE ABLE TO CONNECT THE TWO. THIS IS DONE BY MEANS OF "INDIRECTION", WHICH CONSISTS OF TREATING THE VALUE OF A STRING AS A NAME. THE OPERATOR WHICH DOES THIS IS "$", WHICH IS A UNARY PREFIX OPERATOR. WHEN "$" IS APPLIED TO AN EXPRESSION, THE VALUE OF THE RESULTING EXPRESSION IS THE VALUE OF THE STRING NAMED BY THE ORIGINAL EXPRES-

SION. PUT ANOTHER, MORE INTUITIVE WAY, THE "$" OPERATOR REPLACES ITS ARGUMENT WITH THE THING THAT IT NAMES. A THIRD INTERPETATION IS: INDIRECTION REMOVES ONE LAYER OF QUOTES FROM AROUND ITS ARGUMENT.

FOR EXAMPLE, IF A = "B", B = "C", AND C = "D", THEN $A = $"B" = B = "C". SIMILARLY, $$A = $B = C = "D". FOR ANOTHER EXAMPLE, SUPPOSE THAT I = "37" AND C37 = "NEW YORK". THEN:

```
      $("C" I) = "NEW YORK"
```

FOR YET ANOTHER EXAMPLE, IT WILL BE SHOWN HOW "ARRAYS" CAN BE HANDLED IN SNOBOL3, A LANGUAGE THAT DOESNT HAVE ARRAYS AS SUCH AMONG ITS DATA TYPES. ARRAY HANDLING CAN BE SIMULATED, HOWEVER, BY USING INDIRECTION ON AN EXPRESSION THAT CONTAINS THE "ARRAY NAME" AND ALL THE RELEVANT SUBSCRIPTS, SEPARATED BY ANY APPROPRIATE DELIMITERS. TO LOOK ALGOL-LIKE, THE DELIMITERS CAN BE BRACKETS AND COMMAS. THUS, TO USE THE I, J, K ELEMENT OF THE ARRAY AB, THE FOLLOWING EXPRESSION COULD BE USED:

```
      $("AB[" I "," J "," K "]")
```

IF I = "3", J = "-2", AND K = "47", THIS WOULD GIVE:

```
      AB[3,-2,47]
```

SOME FURTHER EXAMPLES OF EXPRESSIONS INVOLVING INDIRECTION ARE:

```
      $I
      $$$C
      $$$$""
      $"A=I1"
      $($X "/" $B "/" $("N" I))
      $$F(A,"3")
      $$SIZE(E)
      $("FATHER/" NAME)
      $("NAME-" SOC.SEC.NO.)
```

%CP 45

5.2. CONCATENATION.

THE BASIC STRING OPERATOR IS CONCATENATION, WHICH SIMPLY JOINS ITS ARGUMENTS END-TO-END. CONCATENATION IS ASSUMED IN SNOBOL3 WHEN TWO STRINGS (LITERALS, NAMES, EXPRESSIONS, ETC.) ARE PLACED NEXT TO EACH OTHER WITHOUT AN OPERATOR BETWEEN THEM. SINCE THE OPERATION OF CONCATENATION IS ASSOCIATIVE, IT CAN BE USED AS IF IT WERE AN OPERATION WITH ANY NUMBER OF OPERANDS, AND SO CONCATENATE EXPRESSIONS CONSIST SIMPLY OF A LIST OF STRINGS TO BE CONCATENATED, WITH JUST BLANKS BETWEEN THEM IF A DELIMITER IS NEEDED. SOME EXAMPLES OF EXPRESSIONS INVOLVING CONCATENATION ARE:

```
      A "," B
      (A "," B)
      ("ABC" X) "DE"
      "    " ALPHA "    "
      LIST ELT "/" PROPS ","
```

```
     X1 "," X2 "," X3 "," X4 "," X5 "," X6
```

A "GROUPING" IS SIMPLY A STRING EXPRESSION SURROUNDED BY PARENTHESES. SOME GROUPINGS ARE INCLUDED IN THE ABOVE EXAMPLES. TO EVALUATE A GROUPING, ALL THE ELEMENTS WITHIN IT ARE EVALUATED, AND A TEMPORARY STORAGE LOCATION IS GIVEN THEIR CONCATENATED VALUE. IF THERE IS ONLY ONE OBJECT WITHIN THE PARENTHESES, A NEW TEMPORARY VALUE IS CREATED ANYWAY. THIS FACT CAN BE USED IN CASES WHERE LATER PART OF THE INSTRUCTION MAY ALTER A VALUE OF A STRING, AND THE OLDER VALUE IS THE ONE THAT SHOULD BE USED. FOR EXAMPLE, IF THE FUNCTION F() ALTERS THE STRING X, THEN THE EXPRESSION (X F()) WILL USE THE NEW VALUE IN THE CONCATENATION. TO FORCE THE GROUPING TO USE THE VALUE THAT X HAD BEFORE THE CALL, THE EXPRESSION SHOULD BE:

```
     ((X)  F())
```

SIMILARLY, IF A PROGRAM CALLS F(READ,READ), THE SECOND PARAMETER WILL ASSIGN A NEW VALUE TO READ, AND THE FIRST INPUT CARD WILL BE LOST (THE SAME VALUE WILL BE PASSED TO BOTH PARAMETERS). TO CAUSE THE FUNCTION TO USE THE OLDER VALUE OF ITS FIRST PARAMETER, THE CALL COULD BE F((READ),READ).

IF A PARAMETER TO A FUNCTION IS AN EXPRESSION INVOLVING CONCATENATION, THEN AN IMPLICIT PAIR OF PARENTHESES SURROUNDS THE PARAMETER, AND THE CONCATENATION IS PERFORMED BEFORE THE NEXT PARAMETER IS EVALUATED. THUS IN F(READ "*",READ), THE PROBLEM MENTIONED IN THE PREVIOUS PARAGRAPH IS AVOIDED, SINCE THE FIRST PARAMETER IS EVALUATED BEFORE THE SECOND READ IS PERFORMED, AND THE RIGHT VALUE WILL BE USED FOR THE FIRST READ.

THE GROUPING () IS EQUIVALENT TO "".

%PAGE

5.3. FUNCTION CALLS.

```
   <FCT CALL>   ::=  <FCT NAME>(  <PARAMS>  )
   <FCT NAME>   ::=  <IDENTIFIER>
   <PARAMS>     ::=  <EXPRESSION>
                ::=  <PARAMS> ,  <EXPRESSION>
```

FUNCTION CALLS ARE USED TO CALL FUNCTIONS. SNOBOL3 PRE-DEFINES A LARGE NUMBER OF FUNCTIONS (SEE SECTION 9), AND PROGRAMMERS CAN DEFINE OTHERS CODED IN SNOBOL (SEE THE FUNCTION DEFINE() IN SEC. 9 AND THE -DEFINE CARD IN SEC. 2.2).

THERE ARE THREE BASIC REASONS FOR CALLING A FUNCTION:

1) TO CALCULATE AND RETURN A VALUE.
2) TO TEST A CONDITION, AND SUCCEED OR FAIL.
3) TO CAUSE "SIDE EFFECTS"--THAT IS, TO ALTER GLOBAL INFORMATION
   THAT IS TO BE USED BY OTHER PARTS OF THE PROGRAM, OR TO DO
   INPUT OR OUTPUT.

A SNOBOL FUNCTION CAN DO ANY COMBINATION OF THESE THINGS. THE LATTER IS ESPECIALLY EASY IN SNOBOL, SINCE A PROGRAM-DEFINED FUNCTION HAS ACCESS TO ANY VARIABLES IN THE PROGRAM THAT ARE NOT EXPLICITLY DECLARED TO BE LOCAL TO IT (THE LIST OF FORMAL PARAMETERS AND LOCAL VARIABLES). THIS MEANS ESPECIALLY THAT A FUNCTION GENERALLY

HAS ACCESS TO ANY OF THE DATA IN USE BY THE PIECE OF CODE THAT CALLED IT, WHICH MAY ITSELF BE PART OF A HIGHER-LEVEL FUNCTION.

FOR DETAILS OF WHAT HAPPENS WHEN A USER-DEFINED FUNCTION IS CALLED, SEE SECTION 3.2. SOME SPECIAL DETAILS CONCERNING THE CALL ITSELF ARE:

1) THE "(" MUST APPEAR IMMEDIATELY AFTER THE FUNCTION NAME. IF THERE IS A BLANK BETWEEN THEM, THE COMPILER WILL INTERPRET IT AS THE CONCATENATION OF A STRING WITH A GROUPING.

2) OMITTED PARAMETERS ARE NULL STRINGS. EXTRA PARAMETERS ARE EVALUATED BUT NOT PASSED TO THE FUNCTION. FOR EXAMPLE:

```
    F(,X)              = F("",X)
    MW(A,,B,C,)        = MW(A,"",B,C,"")
```

IN THE SECOND EXAMPLE, THE LAST COMMA IS NOT NEEDED.

3) IF THE FUNCTION BEING CALLED IS NOT DEFINED, THE PROGRAM HAS COMMITTED A FATAL ERROR. IF IN DEBUG MODE, THE PROGRAM WILL BE SUSPENDED WITH AN EXPLANATORY MESSAGE. IF NOT IN DEBUG MODE, A MESSAGE GIVING THE STATEMENT NUMBER AND THE FUNCTION NAME WILL BE PRINTED ON THE LINE PRINTER AND TELETYPE, AND THE JOB WILL BE TERMINATED ABNORMALLY.

SOME EXAMPLES OF FUNCTION CALLS ARE:

```
    MV("A",B L,("," X ","))
    ARRGH(,,B " " X1, "CARD")
    LEVEL()
    .NE(A,"375")
    EQUALS(STR,RES)
```

NOTE ESPECIALLY THE FOLLOWING:

```
    F(A)             IS A FUNCTION CALL.
    F (A)            IS A CONCATENATION.
    F(B ",","1")     IS A FUNCTION CALL.
    F (B ",","1")    IS A SYNTAX ERROR. (WHY?)
```

%PAGE

5.4. ARITHMETIC.

```
<ARITH EXPR>  ::=  <A-TERM>
              ::=  -  <A-TERM>
<A-TERM>      ::=  <A-TERM>  <ADD AP>  <M-TERM>
              ::=  <M-TERM>
<M-TERM>      ::=  <M-TERM>  <MUL OP>  <E-TERM>
              ::=  <E-TERM>
<E-TERM>      ::=  <E-TERM>  <EXP OP>  <ARITH ELT>
              ::=  <ARITH ELT>
<ARITH ELT>   ::=  <NAMED EXPR>
              ::=  <GROUPING>
              ::=  <FCT CALL>
              ::=  <LITERAL>
<ADD OP>      ::=  +
              ::=  -
<MUL OP>      ::=  *
              ::=  ×
```

```
                  ::=  /
   <EXP OP>       ::=  **
```

A NUMBER IN SNOBOL3 IS A STRING WHICH CONSISTS ONLY OF DIGITS, WITH THE POSSIBLE EXCEPTION OF THE FIRST CHARACTER WHICH CAN BE "-". ONLY INTEGER ARITHMETIC IS PROVIDED IN SNOBOL3. SOME EXAMPLES OF NUMERIC STRINGS ARE:

```
      "12"  = "00012"
      "-3"  = "-00000003"
      "0"   = "0000"  =  "-0"  = "-"  =  ""
```

SOME IMPLEMENTATIONS OF SNOBOL3 INSIST THAT ARITHMETIC BE FULLY PARENTHESIZED. THIS IS NOT NECESSARY ON THE B5500, THOUGH USING UNPARENTHESIZED ARITHMETIC MAY MAKE A B5500 SNOBOL3 PROGRAM INCOMPATIBLE WITH ANOTHER IMPLEMENTATION. THE PRECEDENCE RELATIONS ARE BASICALLY THOSE OF FORTRAN OR ALGOL, WITH THE RANKING:

EXPONENTIATION	**	(HIGH)
MULTIPLICATION, DIVISION	*, x, /	
ADDITION, SUBTRACTION	+, -	
CONCATENATION	BLANKS	(LOW)

AMONG OPERATORS OF EQUAL PRECEDENCE, LEFT-ASSOCIATION IS USED. SOME ARITHMETIC EXPRESSIONS, WITH PARENTHESISED EQUIVALENTS, ARE:

```
      A + B - C       = ((A + B) - C)
      A / B * C       = ((A / B) * C)
      A - B x C       = (A - (B x C))
      - X / R         = (- (X / R))
      A x B ** "2"    = (A x (B ** "2"))
      AL AR - AM      = (AL (AR - AM))
      C + I "0"       = ((C + I) "0")
      P ** Q ** R     = ((P ** Q) ** R)     = P ** (Q x R)
```

%CP 25

5.4.1. DIVISION MODES.

THE B5500 HAS THREE WAYS TO HANDLE DIVISION, DEPENDING ON HOW REMAINDERS ARE TREATED. THE METHOD USED IS SET BY THE FUNCTION MODE(), AS FOLLOWS:

```
   MODE("ROUND") CAUSES DIVISION TO BE ROUNDED.
   MODE("TRUNCATION") CAUSES REMAINDERS TO BE DISCARDED.
   MODE("INTEGER") CAUSES DIVISION TO FAIL IF THE REMAINDER ≠ 0.
```

THE SYSTEM ASSUMES ROUNDING UNLESS MODE() IS USED, SINCE THAT IS HOW THE HARDWARE OPERATES, AND IS FASTEST. SOME EXAMPLES OF THE DIFFERENCES BETWEEN THE THREE ARE:

EXPRESSION	ROUNDED	TRUNCATED	INTEGER
"4" / "2"	"2"	"2"	"2"
"5" / "2"	"3"	"2"	FAILS
"5" / "3"	"2"	"1"	FAILS
"12" / "4"	"3"	"3"	"3"

```
     "13" / "4"          "3"       "3"      FAILS
     "14" / "4"          "4"       "3"      FAILS
     "15" / "4"          "4"       "3"      FAILS
     "-3" / "2"          "-2"      "-1"     FAILS
     "9" / "-6"          "-2"      "-1"     FAILS
     "-5" / "3"          "-2"      "-1"     FAILS
```

%CP 13

5.4.2. FAILURE OF ARITHMETIC.

THE FOLLOWING CONDITIONS WILL CAUSE FAILURE OF AN ARITHMETIC EXPRESSION:

1) A NON-NUMERIC ARGUMENT. A STRING IS NUMERIC IF ITS FIRST CHARACTER IS "-" OR A DIGIT AND ALL OTHER CHARACTERS ARE DIGITS. BOTH "" AND "-" ARE NUMERIC, AND BOTH EQUAL ZERO.
2) A NUMBER OCCURS WHOSE MAGNITUDE IS GREATER THAN 2**39-1, THE LARGEST INTEGER THAT THE B5500 CAN HANDLE. THE DECIMAL VALUE OF THIS NUMBER IS 549,755,813,887.
3) DIVISION BY ZERO.
4) THE DIVISION MODE IS "INTEGER" AND DIVISION PRODUCES A NON-ZERO REMAINDER.

%CP 30

5.4.3. ARITHMETIC FUNCTIONS.

THIS SECTION DESCRIBES A NUMBER OF FUNCTIONS PRE-DEFINED BY SNOBOL WHICH ARE USED WITH NUMBERS. ALL OF THESE FUNCTIONS WILL FAIL IF A NON-NUMERIC PARAMETER IS GIVEN THEM.

.EQ(A,B) SUCCEEDS IF A AND B REPRESENT THE SAME INTEGER. THIS FUNCTION SHOULD BE COMPARED WITH EQUALS(A,B), WHICH DOES A STRING COMPARE. SOME EXAMPLES SHOWING THE DIFFERENCES ARE:

```
    .EQ("1","01") SUCCEEDS          EQUALS("1","01") FAILS
    .EQ("00","")  SUCCEEDS          EQUALS("00","")  FAILS
    .EQ("A","A")  FAILS             EQUALS("A","A")  SUCCEEDS
    .EQ("7","7")  SUCCEEDS          EQUALS("7","7")  SUCCEEDS
```

.EQ() ALWAYS RETURNS A NULL VALUE, AS DO THE FOLLOWING:

.NE(A,B) SUCCEEDS IF A ≠ B.
.LT(A,B) SUCCEEDS IF A < B.
.LE(A,B) SUCCEEDS IF A ≤ B.
.GT(A,B) SUCCEEDS IF A > B.
.GE(A,B) SUCCEEDS IF A ≥ B.
.NUM(A) SUCCEEDS IF A IS NUMERIC.

.REMDR(A,B) RETURNS THE REMAINDER OF A / B. THIS IS DEFINED AS:

```
    A - ((A / B) × B)
```

WHERE THE DIVISION IS TRUNCATED.

```
%CP 10

5.4.4. ADDITIONAL COMMENTS ON ARITHMETIC.

   THERE ARE TWO WAYS TO INDICATE MULTIPLICATION, WITH "*" OR
WITH "x".  WHEN USED IN A PATTERN OUTSIDE OF PARENTHESES, "*" WILL
ALWAYS BE ASSUMED TO BE A STRING VARIABLE ASTERISK, SO EITHER
"x" OR PARENTHESES SHOULD BE USED IF MULTIPLICATION IS DESIRED.
ALSO, IF "**" APPEARS OUTSIDE OF PARENTHESES IN A PATTERN, IT WILL
BE AN UNNAMED ARBITRARY STRING VARIABLE, NOT EXPONENTIATION.
   NOTE THAT THE ONLY UNARY ARITHMETIC OPERATOR IS "-".
   BLANKS ARE NOT NEEDED AROUND ARITHMETIC OPERATORS IN B5500
SNOBOL3, UNLIKE IN MANY OTHER IMPLEMENTATIONS OF THE LANGUAGE.



%SECTION 6
%PAGE 1
6. PATTERNS.

   <PATTERN>      ::=  <ELEMENT>
                  ::=  <STR VAR>
                  ::=  <PATTERN> <PATTERN>
                  ::=  <PATTERN> <BACK REF>
   <STR VAR>      ::=  <ARB VAR>
                  ::=  <BAL VAR>
                  ::=  <FIXED-LENGTH VAR>
   <ARB VAR>      ::=  **
                  ::=  *  <NAMED EXPR>  *
   <BAL VAR>      ::=  *()*
                  ::=  *(  <NAMED EXPR>  )*
   <FIXED-LENGTH VAR> ::=  */  <EXPRESSION>  *
                      ::=  *  <NAMED EXPR>  /  <EXPRESSION>  *
   <BACK REF>     ::=  <NAMED EXPR>

   PROBABLY THE MOST IMPORTANT PART OF A SNOBOL INSTRUCTION IS THE
PATTERN, WHICH IS USED TO IDENTIFY AND ALTER PARTS OF STRINGS ACCORDING
TO CERTAIN PROPERTIES OF THE STRINGS BEING EXAMINED.  A PATTERN
IS MADE UP OF A LIST OF "PATTERN ELEMENTS", EACH OF WHICH WILL
"MATCH" A PART OF A STRING IF THAT PART HAS THE PROPERTY DESCRIBED BY
THE PATTERN ELEMENT.  A PATTERN WILL MATCH A STRING IF EACH OF ITS
ELEMENTS MATCHES A SUBSTRING, AND THE MATCHED SUBSTRINGS ARE ADJACENT.
IN OTHER WORDS, THE PATTERN IS MATCHED ONE ELEMENT AT A TIME, FROM
LEFT TO RIGHT, AND EACH PATTERN ELEMENT MUST MATCH A SUBSTRING THAT
STARTS JUST AFTER THE SUBSTRING MATCHED BY THE PREVIOUS ELEMENT.
   THE BASIC PATTERN ELEMENTS DEFINED IN SNOBOL3 ARE CONSTANTS, STRING
VARIABLES OF THREE TYPES (ARBITRARY, BALANCED, AND FIXED-LENGTH),
AND BACK REFERENCES.  THE FOLLOWING SUB-SECTIONS DESCRIBE EACH OF
THESE PATTERN ELEMENTS IN DETAIL.



%CP 20
6.1. CONSTANT PATTERN ELEMENTS.

   A "CONSTANT" PATTERN ELEMENT IS ONE WHICH WILL MATCH ONLY ONE
STRING.  THAT IS, IT IS AN ELEMENT WHICH EVALUATES TO A STRING OF
```

CHARACTERS, AND IT WILL MATCH ONLY THAT STRING. EXAMPLES OF CONSTANT PATTERN ELEMENTS ARE LITERALS, STRING NAMES, FUNCTION CALLS, AND ANY OTHER STRING-VALUED EXPRESSION. FOR EXAMPLE, THE PATTERN

```
      A  "+"  (B C)
```

CONSISTS OF THREE CONSTANT PATTERN ELEMENTS: A, "+", AND (B C). IF A = "IS", B = "3", AND C = "K", THEN THIS PATTERN WILL MATCH THE STRING "IS+3K". IF EXP = "SQRT(S)=IS+3K-J", THEN THE RULE

```
      EXP  A "+" (B C)
```

WILL SUCCEED, WITH THE PATTERN MATCHING STARTING AT THE 9TH CHARACTER OF THE STRING REFERENCE.

%CP 45

6.2. STRING VARIABLES.

A STRING VARIABLE IS A PATTERN ELEMENT WHICH WILL MATCH MORE THAN ONE STRING, DEPENDING ON THE PROPERTIES OF THE STRING. THERE ARE THREE TYPES IN SNOBOL3: "ARBITRARY", "BALANCED", AND "FIXED-LENGTH". SEPARATE SUB-SECTIONS BELOW ARE DEVOTED TO EACH TYPE. THIS SECTION DESCRIBES SOME PROPERTIES COMMON TO ALL TYPES OF STRING VARIABLES.

A STRING VARIABLE CAN BE "NAMED" OR "UNNAMED". A NAMED VARIABLE IS ONE WHICH CONTAINS A <NAMED EXPR> (SEE THE BNF DEFINITIONS). BOTH FORMS ACT THE SAME DURING A PATTERN MATCH; THE DIFFERENCE IS THAT IF THE PATTERN MATCH IS SUCCESSFUL, THE STRING NAMED BY THE THE STRING VARIABLES ARE ASSIGNED THE MATCHED SUBSTRINGS AS NEW VALUES. FOR EXAMPLE, IF X = "1234567", THEN THE RULE

```
      X  "2" ** "5"
```

WILL SIMPLY SUCCEED AND HAVE NO SIDE-EFFECT, WHILE

```
      X  "2" *I* "5"
```

WILL SUCCEED AND ASSIGN TO I THE VALUE "34", WHICH IS THE SUBSTRING MATCHED BY THE STRING VARIABLE *I*.

THE ASSIGNMENT OF STRING VARIABLE VALUES IS CARRIED OUT IMMEDIATELY AFTER THE PATTERN MATCH, BEFORE THE REPLACEMENT PART (IF ANY) IS EVALUATED, SO THE NEW VALUES WILL BE USED IF THE STRING VARIABLE NAMES APPEAR IN THE REPLACEMENT. FOR EXAMPLE, IF L = "A,LN,AR,XYZ,CB," THEN THE RULE

```
      L  "," *R* "," *S* "," ← RPL(R,S)
```

WILL DO A PATTERN MATCH (WHICH WILL SUCCEED), SET R = "LN", SET S = "AR", AND THEN CALL RPL("LN","AR"), WHOSE VALUE WILL THEN REPLACE THE MATCHED SUB-STRING ",LN,AR,".

THERE MAY BE SEVERAL STRING VARIABLES WITH THE SAME NAME. THE ASSIGNMENTS ARE CARRIED OUT IN A STRICT LEFT-TO-RIGHT ORDER, SO EACH STRING WILL END UP WITH THE LAST VALUE ASSIGNED TO IT. FOR EXAMPLE, WITH L AS ABOVE, THE RULE

```
      L  "," *X* "," *X* "," *X* ","
```

WILL SUCCEED AND GIVE X THE VALUES "LN", "AR", AND "XYZ". THE LAST VALUE, "XYZ", WOULD BE THE VALUE OF X AT THE END OF THE INSTRUCTION. IF X WERE AN OUTPUT STRING, OUTPUT WOULD OCCUR THREE TIMES DURING THIS RULE.

%CP 40

6.2.1. ARBITRARY STRING VARIABLES.

```
   <ARB VAR>    ::=  **
                ::=  *  <NAMED EXPR>  *
```

AN ARBITRARY STRING VARIABLE WILL MATCH ANY STRING (INCLUDING THE NULL STRING). AT THE BEGINNING OF A PATTERN (THE LEFT END), AN ARBITRARY VARIABLE MATCHES THE LEFT END OF THE STRING REFERENCE, AT THE RIGHT END OF A PATTERN, IT MATCHES THE "REST" OF THE STRING REFERENCE. FOR EXAMPLE, IF SENT = "WHEN I SAW HER ", THEN

```
      SENT  *A* " " *B* " " *C*
```

WILL SUCCEED, AND ASSIGN A = "WHEN", B = "I", AND C = "SAW HER ". NOTE ESPECIALLY THAT AN ARBITRARY STRING VARIABLE WILL MATCH A NULL STRING IF POSSIBLE, SO FOR EXAMPLE, IF X = "++", THEN

```
      X  *I* "+" *J* "+"
```

WILL SUCCEED AND SET BOTH I AND J TO "".

IF SEVERAL ARBITRARY STRING VARIABLES ARE ADJACENT TO EACH OTHER, ALL BUT THE LAST ONE WILL ALWAYS MATCH THE NULL STRING. AN EXAMPLE OF A USE FOR THIS IS:

```
      STR  *V*  **  "-"
```

THIS INSTRUCTION WILL SEARCH STR FOR A MINUS SIGN, AND IF IT FINDS ONE, IT WILL SET V TO NULL. FOR ANOTHER EXAMPLE, TO ASSIGN A TO B AND SIMULTANEOUSLY SET I, J, K, L, AND M NULL, THE FOLLOWING STATEMENT COULD BE USED:

```
      A  *I* *J* *K* *L* *M* *B*
```

THE GENERAL RULE FOR MATCHING AN ARBITRARY VARIABLE IS:

1) MATCH THE PRECEDING ELEMENT, IF THERE IS ONE, AS FAR TO THE LEFT AS POSSIBLE.
2) MATCH THE FOLLOWING ELEMENT, IF THERE IS ONE, AS CLOSE AFTER THE PRECEDING ONE AS POSSIBLE.
3) THE ARBITRARY VARIABLE THEN MATCHES EVERYTHING BETWEEN THE STRINGS MATCHED BY THE SURROUNDING ELEMENTS, OR IF EITHER OF THESE IS MISSING, THE VARIABLE EXTENDS TO THE APPROPRIATE END OF THE STRING.

%CP 45

6.2.2. BALANCED STRING VARIABLES.

```
   <BAL VAR>  ::=  *()*
```

```
            ::=  *(  <NAMED EXPR>  )*
```

A BALANCED STRING VARIABLE WILL MATCH ANY STRING WHICH IS NON-NULL AND HAS BALANCED PARENTHESES. THAT IS, THERE MUST BE THE SAME NUMBER OF RIGHT PARENTHESES AS THERE ARE LEFT PARENTHESES (POSSIBLY NONE OF EITHER), AND THE FIRST PARENTHESIS MUST BE A "(". THUS A BALANCED VARIABLE WILL MATCH ANY OF THE FOLLOWING:

```
     "ABCD"
     " "
     "()"
     "A(BLMX)+D"
     "A + (B - C)"
     "(F(1,2) + (ALPHA ** 2))"
     "HE SAID (VERY SOFTLY) THAT "
     "((())(()))(()(()))"
```

A BALANCED VARIABLE WILL NOT MATCH ANY OF:

```
     ""
     "("
     ")("
     "A+B)"
     "(A))"
     "VERY SOFTLY) THAT "
     " ) + (A - M) "
```

UNLIKE ARBITRARY STRING VARIABLES, BALANCED VARIABLES ARE NOT EXTENDED TO THE RIGHT WHEN THEY END A PATTERN, THOUGH AT THE START OF A PATTERN THEY EXTEND AS FAR LEFT AS POSSIBLE. FOR EXAMPLE, IF

```
     EXPR  =  "(A+(LIM(P/Q,Q,1)-SIN(LN(1-ALPH)))x(2**N)x3)-N"
```

THEN THE RULE

```
     EXPR  *(L)* "x" *(R)*
```

WILL SUCCEED, AND SET L = "A+(LIM(P/Q,Q,1)-SIN(LN(1-ALPH)))", AND R = "(2**N)". L WILL INCLUDE THE INITIAL "A+", BUT R WILL NOT BE EXTENDED TO INCLUDE THE "x3", THOUGH "(2**N)x3" WOULD BE A VALID MATCH FOR A BALANCED VARIABLE.

%CP 30

6.2.3. FIXED-LENGTH STRING VARIABLES.

```
   <FIXED-LENGTH VAR>  ::=  */  <EXPRESSION>  *
                       ::=  *  <NAMED EXPR>  /  <EXPRESSION>  *
```

A FIXED-LENGTH STRING VARIABLE WILL MATCH ANY STRING WITH A GIVEN NUMBER OF CHARACTERS. THE <EXPRESSION> MUST BE NUMERIC AND NON-NEGATIVE, OR THE PATTERN WILL FAIL DURING EVALUATION AND THE PATTERN MATCH WILL NOT TAKE PLACE. ONE SORT OF USE OF A FIXED-LENGTH VARIABLE IS TO PICK OFF INITIAL STRINGS, USUALLY ONE CHARACTER:

```
     STR  *CH/"1"* =    /F(OUT)
```

THIS WILL ASSIGN TO CH THE FIRST CHARACTER OF STR AND DELETE THAT CHARACTER FROM STR. IF STR IS NULL, THE STATEMENT WILL FAIL AND TRANSFER TO OUT. FOR ANOTHER EXAMPLE, TO GET THE LAST N CHARACTERS OF A STRING S:

```
      S  */SIZE(S)-N*  *TAIL*   /F(TOO,SMALL)
```

THE UNNAMED FIXED-LENGTH VARIABLE WILL MATCH ALL BUT THE LAST N CHARACTERS OF S, AND THE ARBITRARY VARIABLE WILL GET THE LAST N. IF S HAS FEWER THAN N CHARACTERS, THE FIXED-LENGTH VARIABLE WILL HAVE A NEGATIVE LENGTH, AND WILL FAIL DURING EVALUATION OF THE PATTERN.

NOTE THAT THE FIRST EXAMPLE ABOVE COULD BE WRITTEN:

```
      STR  *CH/"1" *STR*    /F(OUT)
```

%CP 30

6.3. BACK REFERENCES.

ANOTHER TYPE OF PATTERN ELEMENT IS THE "BACK REFERENCE". THIS IS A STRING NAME WHICH IS THE SAME AS THE NAME OF A PREVIOUS STRING VARIABLE. A BACK REFERENCE WILL MATCH A STRING WHICH IS IDENTICAL TO THE STRING MATCHED BY THE STRING VARIABLE. FOR EXAMPLE, IN

```
      X  *H* "E" H
```

THE BACK REFERENCE, H, MUST MATCH THE SAME STRING AS THE ARBITRARY VARIABLE, *H*. IF X = "ABEBCA", THE RULE WILL SUCCEED AND SET H = "B". SIMILARLY, IF EXP IS AN ALGEBRAIC EXPRESSION, THEN

```
      EXP  *(P)* ** P    :F(NONE)
```

WILL LOOK FOR A SUB-EXPRESSION THAT OCCURS TWICE, AND IF THERE ARE NONE, IT WILL TRANSFER TO NONE.

THERE CAN BE SEVERAL BACK REFERENCES TO THE SAME STRING VARIABLE. EACH SUCH BACK REFERENCE MUST MATCH THE SAME STRING. FOR EXAMPLE

```
      X  *S/"1"* S S
```

WILL SUCCEED ONLY IF X CONTAINS THREE CONSECUTIVE IDENTICAL CHARACTERS.

IF THERE ARE SEVERAL STRING VARIABLES WITH THE SAME NAME, A BACK REFERENCE WILL MATCH THE LAST SUCH VARIABLE THAT PRECEDES IT. FOR EXAMPLE, THE RULE

```
      AL  *X*  " "  *X*  " "  X
```

WILL SUCCEED FOR THE STRING AL = "AB CD CDEF", AND SET X = "CD". THE FIRST STRING VARIABLE WILL MATCH THE STRING "AB".

%CP 40

6.4. PATTERN MATCH MODES.

NORMALLY, A PATTERN WILL MATCH ANYWHERE IN THE STRING REFERENCE THAT IT FINDS AN APPROPRIATE SUB-STRING. SOMETIMES IT IS DESIRABLE TO

FORCE THE PATTERN TO MATCH STARTING AT THE FIRST CHARACTER OF THE STRING REFERENCE. A PATTERN THAT CAN ONLY MATCH AN INITIAL SUB-STRING IS CALLED "ANCHORED". PATTERNS THAT CAN MATCH ANYWHERE (THE USUAL KIND) ARE CALLED "UNANCHORED".

THERE ARE TWO METHODS OF INDICATING WHETHER OR NOT A PATTERN IS TO BE ANCHORED. ONE IS BY USING THE MODE FUNCTION: MODE("ANCHOR") SETS A SWITCH AT CAUSES ALL PATTERNS TO BE ANCHORED; MODE("UNANCH") RESETS THE SWITCH SO THAT PATTERNS ARE UNANCHORED. EITHER OF THESE MODES CAN BE CANCELED FOR A SINGLE INSTRUCTION BY USING THE FUNCTIONS ANCHOR() AND UNANCH(). ANCHOR() CAUSES THE INSTRUCTION THAT IT IS IN TO BE ANCHORED, NO MATTER WHAT THE MODE IS. SIMILARLY, UNANCH() CAUSES ITS PATTERN TO BE UNANCHORED, NO MATTER WHAT THE MODE. IF MODE() IS NOT USED, SNOBOL ASSUMES THAT THE MODE IS UNANCHORED.

TO SHOW THE DIFFERENCE, SUPPOSE THAT E = ")(A+B)(". THEN

```
      E  *(F)*
```

WILL SUCCEED, AND SET F = "(A+B)". HOWEVER, THE RULE

```
      E  ANCHOR()  *(F)*
```

WILL FAIL, SINCE THE PATTERN IS RESTRICTED TO MATCHING AN INITIAL SUBSTRING, AND NO BALANCED STRING CAN START WITH ")". IF MODE("ANCHOR") HAD BEEN CALLED, THEN THE FIRST RULE WOULD ALSO HAVE FAILED.

FOR ANOTHER EXAMPLE, SUPPOSE X = "ABCDE". THEN THE RULE

```
      X  "C" *I*
```

WILL SUCCEED (IF THE MODE IS UNANCHORED), AND SET I = "DE". BUT THE ANCHORED RULE:

```
      X  ANCHOR() "C" *I*
```

WILL FAIL, BECAUSE THE FIRST CHARACTER OF X IS NOT "C".

%CP 50

6.5. ASSIGNMENT OF STRING VARIABLES.

IF A PATTERN MATCH SUCCEEDS, THE STRING NAMES ASSOCIATED WITH ITS STRING VARIABLES ARE ASSIGNED THE VALUES OF THEIR SUBSTRINGS IN A STRICT LEFT-TO-RIGHT ORDER. THIS MEANS THAT IF SEVERAL STRING VARIABLES HAVE THE SAME NAME, THE RIGHT-MOST ONE WILL DETERMINE THE VALUE ASSIGNED TO THE NAME; THE OTHERS WILL BE ASSIGNED THEIR APPROPRIATE VALUES, AND THEN RE-ASSIGNED. IF THE STRINGS HAVE ANY OUTPUT USE, OUTPUT WILL OCCUR FOR EACH ASSIGNMENT. FOR EXAMPLE, IF THE RULE

```
      X  *PRINT* "," *PRINT* "," *PRINT*
```

SUCCEEDS, THE THREE STRINGS ASSIGNED TO PRINT WILL ALL BE WRITTEN ON THE LINE PRINTER.

FURTHERMORE, THE ASSIGNMENT IS DONE IMMEDIATELY AFTER THE PATTERN MATCH, BEFORE THE REPLACEMENT AND GO-TO PARTS ARE EVALUATED, TO ANY STRING NAMES USED, SAY, BOTH AS A STRING VARIABLE NAME AND IN THE REPLACEMENT WILL HAVE THE NEW VALUE WHEN THE REPLACEMENT IS EVALUATED.

FOR EXAMPLE, IF AST = "XZ.AB", THEN IN THE RULE

```
      AST  "."  *L*  =  "."  DBL(I,L)
```

THE VALUE PASSED TO THE SECOND PARAMETER OF DBL() WILL BE "AB".

A RATHER INTERESTING SITUATION OCCURS WHEN THE STRING REFERENCE HAS THE SAME NAME AS A STRING VARIABLE. FOR EXAMPLE, THE RULE

```
      I  *I* "+" *J*
```

WILL FIRST ASSIGN A NEW VALUE TO I, THEN ATTEMPT TO ASSIGN A VALUE TO J. BUT BY THIS TIME, I HAS BEEN CHANGED, AND I WAS THE STRING REFERENCE. THERE IS CLEARLY NO LONGER ANYTHING MEANINGFUL TO ASSIGN TO J, AND SUCH INSTRUCTIONS SHOULD BE AVOIDED. IN GENERAL, J WOULD BE EXPECTED TO CONTAIN RATHER ARBITRARY GARBAGE. A SIMILAR RULE THAT ACHIEVES WHAT MOST PEOPLE WOULD ASSUME TO BE THE BEST MEANING OF THIS RULE IS

```
      (I)  *I* "+" *J*
```

IN THIS CASE, THE STRING REFERENCE IS NOT I ITSELF, BUT A GROUPING THAT HAS THE SAME VALUE AS I. SINCE THE STRING VARIABLES ARE NO LONGER NAMED THE SAME AS THE STRING REFERENCE (THE STRING REFERENCE HAS NO NAME AT ALL), THE PROBLEM DOESNT ARISE.

IT IS AT TIMES USEFUL TO HAVE THE LAST STRING VARIABLE IN A PATTERN NAMED THE SAME AS THE STRING REFERENCE. FOR EXAMPLE, THE FOLLOWING TWO STATEMENTS ARE EQUIVALENT:

```
      ST  *WD*  " "  =       /F(E)
      ST  *WD*  " "  *ST*    /F(E)
```

%CP 35

6.6. PATTERN MATCH ALGORITHM.

THERE ARE USUALLY MANY WAYS FOR A PATTERN MATCH TO SUCCEED. WHEN THIS IS TRUE, THERE IS A CLEAR NEED FOR A WAY TO DETERMINE WHICH OF THE POSSIBLE MATCHES WILL BE THE ONE CHOSEN BY SNOBOL, SO THE PROGRAMMER CAN PREDICT WHAT VALUES WILL BE ASSIGNED TO THE STRING VARIABLES AND WHICH SUBSTRING WILL BE REPLACED.

AS A GENERAL, INTUITIVE RULE, THE PROGRAMMER SHOULD REMEMBER THAT A PATTERN MATCH PROCEEDS FROM LEFT TO RIGHT, AND EACH ELEMENT IS MATCHED WITH THE SHORTEST SUBSTRING POSSIBLE CONSISTENT WITH THE SUCCESS OF THE REST OF THE PATTERN. THAT IS, THE PREFERRED MATCH WILL BE THE ONE FOR WHICH AN "EARLIER" PATTERN ELEMENT MATCHES AS FAR LEFT AS POSSIBLE.

THE GENERAL RULES GOVERNING A PATTERN MATCH ARE:

1) TRY TO MATCH THE FIRST PATTERN ELEMENT STARTING AT THE FIRST CHARACTER OF THE STRING REFERENCE. AN ARBITRARY VARIABLE STARTS BY MATCHING THE NULL STRING. IF THIS MATCH FAILS, GO TO STEP 5.
2) IF THERE ARE NO MORE ELEMENTS, GO TO STEP 6.
3) IF THERE IS A NEXT ELEMENT, TRY TO MATCH IT AT THE CHARACTER AFTER THE SUBSTRING MATCHED BY THE PRECEDING ELEMENT. IF THIS SUCCEEDS, GO TO STEP 2.
4) IF THERE IS NO PREVIOUS ELEMENT, GO TO STEP 5. IF THERE IS A PREVIOUS ELEMENT, TRY TO EXTEND IT TO MATCH A LONGER STRING (ONLY ARBITRARY AND BALANCED STRING VARIABLES CAN BE

```
   EXTENDED).  IF THIS SUCCEEDS, GO BACK TO STEP 3 AND TRY TO RE-MATCH
   THE CURRENT ELEMENT.  IF THE EXTENSION CANT BE MADE, GO TO STEP 4.
5) IF THE PATTERN IS ANCHORED, THE PATTERN MATCH FAILS.  IF IT IS
   NOT ANCHORED, THEN TRY TO FIND ANOTHER MATCH FOR THE FIRST ELEMENT
   STARTING AT A LATER CHARACTER.  IF THIS CANT BE DONE, THE MATCH
   FAILS.  IF IT CAN BE DONE, GO TO STEP 2.
6) IF THE LAST PATTERN ELEMENT IS AN ARBITRARY VARIABLE, EXTEND IT
   TO INCLUDE THE REST OF THE STRING REFERENCE.  THE PATTERN MATCH
   HAS SUCCEEDED, AND THE ASSIGNMENT TO THE STRING VARIABLE NAMES
   CAN BE MADE.



%SECTION 7
%PAGE 1
7. GO-TO PARTS, LABELS.

   <GO-TO PART>       ::=  <GO-TO DELIMITER>  <GO-TOS>
   <GO-TO DELIMITER> ::=  <BLANKS>  /
                      ::=  :
   <GO-TOS>          ::=  <UNCONDITIONAL GO-TO>
                     ::=  <SUCCESS GO-TO>
                     ::=  <FAILURE GO-TO>
                     ::=  <SUCCESS GO-TO>  <FAILURE GO-TO>
                     ::=  <FAILURE GO-TO>  <SUCCESS GO-TO>
   <UNCONDITIONAL GO-TO> ::=  (  <LABEL EXPR>  )
   <SUCCESS GO-TO>   ::=  S(  <LABEL EXPR>  )
   <FAILURE GO-TO>   ::=  F(  <LABEL EXPR>  )
   <LABEL EXPR>      ::=  <RESERVED LABEL>
                     ::=  <LABEL>
                     ::=  <COMPUTED LABEL>
   <RESERVED LABEL>  ::=  END
                     ::=  RETURN
                     ::=  FRETURN
   <COMPUTED LABEL>  ::=  $  <ELEMENT>



%CP 10
7.1 LABELS.

   ANY INSTRUCTION IN A SNOBOL PROGRAM CAN BE LABELED.  A LABEL IS
A NON-NULL STRING PRECEDING THE FIRST BLANK OF AN INSTRUCTION.  IT
FOLLOWS FROM THIS DEFINITION THAT LABELS CANT CONTAIN BLANKS.  IF
THE FIRST CHARACTER OF AN INSTRUCTION IS A BLANK, THE INSTRUCTION
IS UNLABELED.  ASIDE FROM THE SPECIAL CHARACTERS IN COLUMN ONE, THERE
ARE NO RESTRICTIONS ON THE CHARACTERS THAT CAN BE USED IN LABELS.



%CP 45
7.2. SIMPLE GO-TO PARTS.

   THE GO-TO PART OF AN INSTRUCTION DETERMINES WHICH INSTRUCTION IS
TO BE EXECUTED NEXT.  IF THERE IS NO GO-TO PART, OR IF THE GO-TO
PART DOESNT APPLY WHEN THE INSTRUCTION IS EXECUTED, THEN THE NEXT
INSTRUCTION TO BE EXECUTED IS THE NEXT ONE IN THE PROGRAM.
```

BRANCHING IN SNOBOL3 CAN BE EITHER UNCONDITIONAL OR CONDITIONED ON THE SUCCESS OR FAILURE OF THE INSTRUCTION. THE GO-TO PART CAN BE SET OFF BY EITHER A "/" (WHICH MUST BE PRECEDED BY A BLANK) OR BY A ":". MANY IMPLEMENTATIONS OF SNOBOL3 ONLY USE THE "/", USUALLY BECAUSE THEIR CHARACTER SETS DONT INCLUDE ":". THE B5500 WILL ACCEPT EITHER DELIMITER.

A SIMPLE GO-TO PART CONSISTS OF A LABEL WITHIN PARENTHESES. FOR EXAMPLE:

```
     :(L)              MEANS "GO TO L".
     :S(X1)            MEANS "IF THE RULE SUCCEEDED, GO TO X1".
     :F(AX)            MEANS "IF THE RULE FAILED, GO TO AX".
     :S(BY)F(AGAIN)    MEANS "IF SUCCEEDED, GO TO BY, OTHERWISE GO
                       TO AGAIN".
     /F(E)S(BACK)      MEANS "IF FAILED, GO TO E, OTHERWISE GO TO BACK".
     /S(BLXGQ)         MEANS "IF SUCCEEDED, GO TO BLXGQ".
     /(A+T$)           MEANS "GO TO A+T$".
```

THE B5500 ALLOWS THE USER TO OMIT THE "S" OR "F" ON THE SECOND PART WHEN BOTH SUCCESS AND FAILURE EXITS ARE USED. THUS

```
     :S(A)(B)          MEANS  :S(A)F(B)
     /F(L1)(L2)        MEANS  /F(L1)S(L2)
```

THERE ARE A FEW RESTRICTIONS ON LABELS THAT CAN BE USED IN A SIMPLE GO-TO PART. THESE VARY FROM ONE IMPLEMENTATION TO ANOTHER; ON THE B5500 THEY ARE:

1) THE FIRST CHARACTER MUST BE A LETTER OR A DIGIT.
2) THE LABELS CANT CONTAIN PARENTHESES.

FOR EXAMPLE, THE FOLLOWING LABELS, THOUGH VALID LABELS, CANT BE USED IN SIMPLE GO-TO PARTS:

```
     AB)
     +25
     $$A
     AB(2)
```

%CP 20

7.3. RESERVED LABELS.

THERE ARE THREE "RESERVED LABELS" THAT CAN BE USED IN SIMPLE GO-TO PARTS. THESE ARE NOT REALLY LABELS, BUT RATHER COMMANDS TO PERFORM CERTAIN ACTIONS. THEY ARE:

```
   END      CAUSES THE PROGRAM TO BE TERMINATED NORMALLY.
   RETURN   CAUSES A RETURN FROM THE LAST PROGRAMMER-DEFINED FUNCTION
            WHICH WAS CALLED. THE RETURN IS SUCCESSFUL.
   FRETURN  CAUSES A RETURN ALSO, BUT THE FUNCTION FAILS.
```

A PROGRAM CAN USE RETURN AND FRETURN AS LABELS, BUT THEY CANT BE TRANSFERED TO BY A SIMPLE GO-TO; A COMPUTED GO-TO MUST BE USED. IF THE PROGRAMMER DOESNT DEFINE THESE LABELS, THE COMPILER WILL

DEFINE THEM AS FOLLOWS:

```
RETURN    :(RETURN)
FRETURN   :(FRETURN)
```

%CP 12

7.4. COMPUTED GO-TOS.

A PROGRAM CAN COMPUTE THE LABEL THAT AN INSTRUCTION IS TO TRANSFER TO BY USING A CONSTRUCT THAT LOOKS EXACTLY LIKE INDIRECTION INSTEAD OF A LABEL IN A GO-TO PART. THE VALUE OF THE EXPRESSION FOLLOWING THE "$" IS THE LABEL TRANSFERED TO. FOR EXAMPLE, IF X = "1" AND Y = "AL", THEN:

```
      /S($("A" X))                MEANS /S(A1)
      :F($Y)                      MEANS :F(AL)
      :S($(X Y))F($(Y X))         MEANS :S(1AL)F(AL1)
      :($("M" X "," SIZE(Y)))     MEANS :(M1,2)
```

%CP 15

7.5. ERRORS IN GO-TO PARTS.

THERE ARE TWO FATAL ERRORS THAT CAN OCCUR IN A GO-TO PART. IF A PROGRAM ATTEMPTS TO TRANSFER TO A LABEL THAT ISNT DEFINED, THE PROGRAM IS TERMINATED ABNORMALLY WITH A MESSAGE GIVING THE ERROR, THE STATEMENT NUMBER, AND THE ERRONEOUS LABEL.

IF A COMPUTED GO-TO CONTAINS A FUNCTION CALL, AND THE FUNCTION FAILS, THE PROGRAM IS AGAIN TERMINATED ABNORMALLY, AND A MESSAGE IS PRINTED SAYING THAT THE GO-TO PART FAILED, GIVING THE STATEMENT NUMBER.

OF COURSE, IF THE USER IS USING DEBUG MODE, THE PROGRAM WILL NOT BE TERMINATED. INSTEAD, THE ERROR MESSAGES WILL BE PRINTED ON THE LINE PRINTER AND TELETYPE AND THE PROGRAM WILL BE SUSPENDED.

%SECTION 8

%PAGE 1

8. FAILURE.

AN EXTREMELY IMPORTANT CONCEPT IN SNOBOL IS THAT OF "FAILURE". A SNOBOL INSTRUCTION CAN BE VIEWED AS A COMMAND TO ATTEMPT CERTAIN ACTIONS; IF THE INSTRUCTION CANT BE CARRIED OUT, IT IS TERMINATED AT THE POINT WHERE IT HAS TO BE STOPPED. THE GO-TO PART OF AN INSTRUCTION PROVIDES THE PROGRAMMER WITH A WAY TO BRANCH ON THE SUCCESS OR FAILURE OF THE INSTRUCTION. MOST BRANCHING IN SNOBOL PROGRAMS IS DONE THIS WAY; THE OTHER METHOD (COMPUTING THE LABEL OF THE NEXT INSTRUCTION) IS USUALLY RATHER RARE.

IT SHOULD BE EMPHASIZED THAT FAILURE CAN OCCUR AT VIRTUALLY ANY PART OF AN INSTRUCTION, AND WHEN FAILURE OCCURS, THE INSTRUCTION TERMINATES IMMEDIATELY AND THE FAILURE EXIT (IF ONE EXISTS) IS TAKEN.

THERE ARE TWO RATHER VITAL POINTS NECESSARY TO SUCCESSFUL USE OF FAILURE: WHAT CAUSES FAILURE, AND WHAT WILL BE LEFT UNDONE WHEN

```
AN INSTRUCTION FAILS AT A PARTICULAR POINT.



%CP 18
8.1. CAUSES OF FAILURE.

THE FOLLOWING CONDITIONS CAUSE FAILURE OF AN INSTRUCTION:

1) FAILURE OF THE PATTERN MATCH.
2) FUNCTION FAILURE.  INTRINSIC FUNCTIONS FAIL WHEN THEIR DESCRIPTIONS
   SAY THEY SHOULD (ASSUMING NO BUGS); PROGRAM-DEFINED FUNCTIONS
   FAIL WHEN THEY USE FRETURN IN A GO-TO PART.
3) FAILURE OF ARITHMETIC.  THERE ARE SEVERAL REASONS FOR THIS:
   NON-NUMERIC ARGUMENTS, VALUES TOO BIG FOR THE HARDWARE TO HANDLE;
   DIVISION BY ZERO, DIVISION WITH INTEGER MODE THAT GIVES A
   NON-ZERO REMAINDER.
4) I/O FAILS WHEN THE OPERATION CANT BE COMPLETED.  FOR FILES, THIS
   MEANS END-OF-FILE OR PARITY ERRORS.  FOR TELETYPES, FAILURE IS
   CAUSED BY EXCEEDING THE WAIT TIME OR USING THE BREAK KEY DURING
   OUTPUT, OR FOR VARIOUS ABNORMAL CONDITIONS.
5) A FIXED-LENGTH VARIABLE FAILS DURING EVALUATION IF ITS LENGTH
   IS NON-NUMERIC OR NEGATIVE.



%CP 25
8.2. ORDER OF EXECUTION.

   IN ORDER TO KNOW HOW MUCH WILL BE DONE WHEN AN INSTRUCTION FAILS,
IT IS NECESSARY TO KNOW THE EXACT ORDER IN WHICH AN INSTRUCTION IS
EXECUTED.  THE RULES GOVERNING THE ORDER OF EXECUTION ARE:

1) ALL EXPRESSIONS ARE EVALUATED FROM LEFT TO RIGHT.  FUNCTIONS ARE
   CALLED AS SOON AS ALL PARAMETERS HAVE BEEN EVALUATED.  ARITHMETIC
   OPERATORS ARE APPLIED AS SOON AS BOTH OPERANDS HAVE BEEN EVALUATED.
2) THE STRING REFERENCE IS THE FIRST PART OF A RULE THAT IS EVALUATED.
3) AFTER THE STRING REFERENCE, THE PATTERN IS EVALUATED FROM LEFT TO
   RIGHT.  NAMES OF STRING VARIABLES ARE EVALUATED WHEN THE STRING VAR-
   IABLE IS ENCOUNTERED.
4) AFTER THE PATTERN IS EVALUATED, THE PATTERN MATCH TAKES PLACE.
5) AFTER THE PATTERN MATCH, STRING VARIABLES (IF ANY) ARE ASSIGNED
   THEIR MATCHING SUBSTRINGS AS NEW VALUES.  FAILURE CAN OCCUR HERE
   IF A STRING VARIABLE NAME IS AN OUTPUT STRING, AND THE OUTPUT
   OPERATION FAILS.
6) THE REPLACEMENT PART (IF ONE EXISTS) IS THEN EVALUATED.
7) THE REPLACEMENT OF THE MATCHED SUBSTRING THEN OCCURS.
8) LASTLY, THE APPROPRIATE PART OF THE GO-TO PART IS EVALUATED, AND THE
   PROGRAM TRANSFERS TO THE NEXT INSTRUCTION.  FAILURE OF A GO-TO PART
   IS A FATAL ERROR.



%SECTION 9
%PAGE 1
9. INTRINSIC FUNCTIONS.
   THIS SECTION DESCRIBES THE FUNCTIONS THAT ARE PRE-DEFINED BY THE
```

B5500 SNOBOL3 SYSTEM. OF COURSE, THE NAMES OF THESE FUNCTIONS ARE IN NO SENSE "RESERVED", AND A PROGRAM CAN RE-DEFINE ANY OF THEM IF THE USER WISHES TO USE THEIR NAMES FOR HIS OWN FUNCTIONS. THE FUNCTIONS ARE LISTED IN ALPHABETICAL ORDER.

%CP 8

ANCHOR(N) WHEN USED IN A PATTERN, CAUSES THE PATTERN TO "ANCHORED" AT CHARACTER N. THAT IS, THE FIRST PATTERN ELEMENT MUST MATCH A SUBSTRING THAT STARTS AFTER AN INITIAL SUBSTRING OF N CHARACTERS. THE USUAL FORM, ANCHOR(), THEREFORE FORCES THE PATTERN TO MATCH AN INITIAL SUBSTRING. N MUST BE A NON-NEGATIVE INTEGER, OR ANCHOR(N) WILL FAIL. NOTE THAT MOST IMPLEMENTATIONS OF SNOBOL3 DO NOT HAVE A PARAMETER FOR ANCHOR(); IF A PROGRAM IS TO BE USED ON OTHER SYSTEMS, THE PARAMETER SHOULD NEVER BE USED.

%CP 17

ASSIGN(A,B) IS A SPECIAL FAST, "EFFICIENT" ASSIGNMENT FUNCTION THAT TAKES SPECIAL ADVANTAGE OF THE B5500 STRING REPRESENTATION. THIS FUNCTION CAN GENERALLY BE ASSUMED TO BE LACKING IN OTHER VERSIONS OF SNOBOL3. IT IS EQUIVALENT TO THE FOLLOWING TWO STATEMENTS:

```
    A ← B
    B ←
```

NO CHECKING OF PARAMETERS IS DONE, INCLUDING CHECKING FOR OUTPUT USE. (CHECKS FOR INPUT USE OCCUR BEFORE THE FUNCTION IS CALLED, SO THEY WILL NOT BE SUPPRESSED.) THE FUNCTION SIMPLY MOVES POINTERS AROUND WITHOUT LOOKING AT WHAT THEY POINT TO. NOTE THE FOLLOWING WARNINGS: IF EITHER PARAMETER IS A LITERAL, THE VALUE OF THAT LITERAL WILL BE DESTROYED (THROUGHOUT THE PROGRAM)--LITERALS SHOULD NEVER EVER EVER BE USED AS PARAMETERS TO ASSIGN(). ALSO, ALTHOUGH B CAN BE OMITTED (IT WILL THEN BE ASSUMED NULL), A MUST BE PRESENT, OR THE FUNCTION WILL FAIL.

%CP 13

CALLF(F,P1,P2,P3,...) CALLS THE FUNCTION NAMED BY F, WITH PARAMETERS P1, P2, P3, ETC. CALLF RETURNS THE VALUE RETURNED BY THIS FUNCTION, OR FAILS IF THE FUNCTION FAILS. F CAN NAME ANY FUNCTION, INTRINSIC OR PROGRAM-DEFINED. FOR EXAMPLE:

```
    CALLF("FX",B,"1",AL)             MEANS  FX(B,"1",AL)
    CALLF("SIZE",B)                  MEANS  SIZE(B)
    CALLF("CALLF","MODE","INTEGER") MEANS  MODE("INTEGER")
```

IF CALLF IS BEING TRACED, NO OUTPUT WILL APPEAR AT ITS RETURN. IT IS NECESSARY TO TRACE THE FUNCTION BEING CALLED TO GET THE RETURN VALUE. IF THE FUNCTION IS UNDEFINED, AN ABNORMAL TERMINATION WILL OCCUR AS WITH ALL ØTHER UNDEFINED FUNCTION CALLS.

%CP 24

CLEAR()

THIS FUNCTION SETS ALL STRINGS TO NULL. THIS IS ONLY DONE TO THE TOP VALUE ON A PUSH-DOWN STACK, SO VALUES THAT

```
2047000    HAVE BEEN PUSHED DOWN WILL NOT BE LOST.  AS AN EXAMPLE OF ITS USE,
2048000    THE FOLLOWING FUNCTION CLEARS ALL STRINGS EXCEPT THOSE WHOSE NAMES
2049000    ARE ON A CERTAIN LIST (WHICH IS THEN CALCULATED BY THE PROGRAM,
2050000    OF COURSE):
2051000
2052000 *****************
2053000 -DEFINE  "RESET(L)", ,"N"
2054000 * L IS A LIST OF NAMES OF VARIABLES.  THE NAMES ARE SEPARATED BY
2055000 * SEMICOLONS (";"), AND L SHOULD END WITH A SEMICOLON.  ALL THE
2056000 * VARIABLES LISTED IN L ARE PUSHED, EVERYTHING IS SET NULL, AND
2057000 * THE VARIABLES ARE THEN POPPED TO THEIR FORMER VALUES.
2058000 * FOR EXAMPLE, RESET("ALPHA;RESULTS;MEM;") WILL SET EVERYTHING TO
2059000 * NULL EXCEPT FOR ALPHA, RESULTS, AND MEM.  NOTE THAT THE STRINGS
2060000 * RESET, L, AND N WILL ALSO BE SAVED.
2061000 RESET  PUSH(L,L)
2062000 RESET1 L  *N* ";" =        :F(RESET2)
2063000        PUSH($N)            :(RESET1)
2064000 RESET2 CLEAR()
2065000        POP(L)
2066000 RESET3 L  *N* ";" =        :F(RETURN)
2067000        POP($N)             :(RESET3)
2068000 *****************
2069000
2070000
2071000
2072000 %CP 24
2073000 CLOSE(F,V)  CLOSES THE FILE ASSOCIATED WITH THE I/O STRING $F, IN
2074000    THE MANNER INDICATED BY V.  CLOSE FAILS IF $F IS NOT A FILE I/O
2075000    STRING OR V ISNT ONE OF THE VALUES LISTED BELOW:
2076000
2077000    "SAVE"  CLOSES THE FILE AND SAVES THE I/O UNIT FOR THE PROGRAM.
2078000       DISK FILES CREATED BY THE PROGRAM ARE LOST; OTHER DISK FILES
2079000       REMAIN ON THE DISK.  TAPES ARE REWOUND AND THE OPERATOR IS
2080000       TOLD TO REMOVE AND SAVE THEM.
2081000    "LOCK"  CLOSES AND LOCKS THE FILE.  DISK FILES CREATED BY THE
2082000       PROGRAM ARE ENTERED IN THE DISK DIRECTORY, AND ARE SAVED.
2083000       TAPES ARE REWOUND AND LOCKED (THE UNIT IS "NOT READY"  UNTIL
2084000       THE OPERATOR READIES IT).
2085000    "RELEASE"  CLOSES THE FILE AND RELEASES THE UNIT AND ALL BUFFER
2086000       AREAS TO THE SYSTEM.  TAPES ARE REWOUND AND MARKED AS SCRATCH.
2087000       DISK FILES CREATED BY THE PROGRAM ARE LOST.
2088000    "PURGE"  CLOSES, PURGES, AND RELEASES THE FILE.  DISK FILES ARE
2089000       REMOVED FROM THE DIRECTORY IF THEIR SECURITY STATUS ALLOWS
2090000       THE USER TO PURGE THEM.  (IF HE CANT, THE MCP WILL KILL THE
2091000       PROGRAM.)
2092000    "*"  IS USED ONLY WITH TAPE FILES.  FOR INPUT FILES, THE END-OF-FILE
2093000       FOR THE CURRENT FILE IS LOCATED, AND THE TAPE IS POSITIONED
2094000       JUST BEYOND IT.  FOR OUTPUT FILES, AN END-OF-FILE IS WRITTEN.
2095000       IN EITHER CASE, THE TAPE IS NOT REWOUND AND REMAINS UNDER
2096000       CONTROL OF THE PROGRAM.
2097000
2098000
2099000 %CP 12
2100000 COPY(S,N)  RETURNS A STRING CONSISTING OF N COPIES OF S.  FAILURE
2101000    OCCURS IF N IS NON-NUMERIC OR NEGATIVE.  THIS FUNCTION IS QUITE
2102000    USEFUL AS AN AID TO FORMATTING OUTPUT.  FOR EXAMPLE:
2103000
```

```
      PRINT ← COPY(" ","10"-SIZE(A))   A   :F(TOOBIG)

   WILL PRINT A RIGHT-JUSTIFIED IN A 10-CHARACTER FIELD.



%CP 3
DATE()  RETURNS THE CURRENT DATE IN THE FORM:

      MM/DD/YY




%CP 25
DEFINE(A,B,C)  CREATES A PROGRAM-DEFINED FUNCTION AT RUN-TIME.
   THE PARAMETERS DESCRIBE THE FUNCTION AS FOLLOWS:

   A  IS THE "FUNCTION PROTOTYPE", WHICH GIVES THE FUNCTION NAME
      AND ITS FORMAL PARAMETERS IN THE FORM:

         <FCT NAME>(  <PARAMS>  )

      WHERE <FCT NAME> IS THE FUNCTION NAME, AND <PARAMS> IS
      THE LIST OF FORMAT PARAMETERS (POSSIBLY NULL) SEPARATED BY
      COMMAS.
   B  CONTAINS A LABEL WHICH IS THE ENTRY POINT FOR THE FUNCTION.
      IF B IS NULL, THE ENTRY POINT IS TAKEN TO SPELLED THE SAME
      AS <FCT NAME>.
   C  CONTAINS A LIST OF LOCAL VARIABLES FOR THE FUNCTION, IF ANY,
      SEPARATED BY COMMAS.

   SEE THE DESCRIPTIONS OF FUNCTION CODE AND CALLS FOR FURTHER INFO
   ON PROGRAM-DEFINED FUNCTIONS.  SOME EXAMPLES OF CALLS OF DEFINE():

      DEFINE("M(AB,AC)","M1","V,XY,ALPH")
      DEFINE("MAX(A,B)")
      DEFINE("MIN(MIN,X)")
      DEFINE("FCT()",,"X")
      DEFINE("MRGE(LA,LB)","MRGENTRY","L1,L2,L3")



%CP 13
DETACH(S,T)  TURNS OFF ANY I/O USE OF THE OBJECT NAMED BY S (A STRING
   NAME, FUNCTION, OR LABEL).  THE TYPE OF THE OBJECT IS INDICATED
   BY THE FIRST CHARACTER OF T AS FOLLOWS:

      "S" MEANS THAT S NAMES A STRING.
      "F" MEANS THAT S NAMES A FUNCTION.
      "L" MEANS THAT S CONTAINS A LABEL.

   IF T IS NULL, "S" IS ASSUMED, SINCE THE MOST COMMON USE OF THIS
   FUNCTION IS TO TURN OFF TRACING.  FOR EXAMPLE, TO TERMINATE ANY
   I/O USE OF THE STRING ALPHA, EITHER DETACH("ALPHA") OR DETACH(
   "ALPHA","S") WOULD WORK.

```

```
%CP 5
DUMP() CAUSES A DUMP OF ALL STRINGS AND LABELS IN USE TO BE PRINTED
   ON THE LINE PRINTER FILE, PRINT.  THE OUTPUT IS IN THE SAME
   FORM AS THAT USED FOR TRACING.  FOR EACH STRING, ITS NAME AND
   CURRENT VALUE IS PRINTED.  FOR EACH LABEL, THE NUMBER OF TIMES
   IT HAS BEEN EXECUTED IS GIVEN.



%CP 2
EOF(S)  SUCCEEDS IF $S IS A FILE I/O STRING AND THE LAST I/O OPERATION
   ON THAT FILE FAILED DUE TO AN END-OF-FILE.



%CP 2
.EQ(A,B)  SUCCEEDS IF A AND B ARE BOTH NUMERIC AND REPRESENT
   THE SAME INTEGER.



%CP 2
EQUALS(A,B)  SUCCEEDS IF A AND B HAVE THE SAME STRINGS AS VALUES.
   IT CAN BE DEFINED IN SNOBOL AS:

      A B      :F(FRETURN)
      B A      :F(FRETURN)S(RETURN)



%CP 15
EXECUTE(S) "EXECUTES" THE STRING S.  AT PRESENT, THIS FUNCTION IS
   RATHER LIMITED, AND WORKS FOR THE FOLLOWING CASES ONLY:

   1) S IS AN I/O STRING TO A FILE.  THE FILE IS GIVEN TO THE MCP
      TO EXECUTE.  THIS IS THE SAME AS THE "ZIP WITH <FILE PART>"
      CONSTRUCT OF ALGOL.  FOR DETAILS OF FILE ORGANIZATION AND
      USE OF CONTROL CARDS IN THE FILE, SEE THE ALGOL MANUAL.
   2) S STARTS WITH A "-".  THE VALUE IS PASSED TO THE SNOBOL CONTROL
      CARD ROUTINE, AND INTERPRETED AS A CONTROL CARD.  MOST OF
      THE CONTROL CARDS ARE RATHER USELESS DURING EXECUTION, BUT
      A FEW (LIKE -COMPILE AND -LIBRARY) CAN AT TIMES BE USEFUL.
   3) S STARTS WITH "?".  THE VALUE OF S IS PASSED TO THE MCP AS
      AN MCP CONTROL CARD.  EXECUTE() THEN RETURNS, AND THE MCP
      GIVES NO INDICATION WHAT IT DID WITH THE CARD.  THE STRING
      SHOULD ALWAYS END WITH ";END.".



%CP 42
FILL(F,MFID,FID,UNIT,DATE,REEL,CYCLE)  ALTERS CERTAIN MCP INFORMATION
   ASSOCIATED WITH THE I/O STRING $F.  THIS FUNCTION IS USUALLY
   A NO-OP IF THE FILE IS OPEN (WHEN THE MCP WILL ALLOW CHANGES
   TO AN OPEN FILE IS VERY UNCLEAR).  IF $F IS NOT AN I/O STRING TO
   A FILE, FILL() WILL FAIL.  IF ANY OF THE OTHER PARAMETERS IS NULL,
```

THEN THE INFORMATION ASSOCIATED WITH THAT PARAMETER IS NOT ALTERED. NON-NULL PARAMETERS WILL CAUSE THEIR ASSOCIATED QUANTITIES TO BE ASSIGNED THE VALUE OF THE PARAMETER. THE MEANINGS OF THE PARAMETERS ARE AS FOLLOWS:

MFID IS THE "MULTI-FILE I.D." OF THE FILE; THAT IS, IT IS THE FIRST HALF OF THE FILE NAME. FOR SINGLE-NAME CARD READER FILES, MFID IS "0000000". IF SIZE(MFID) > "7", ONLY THE FIRST SEVEN CHARACTERS WILL BE USED.

FID IS THE "FILE I.D." OF THE FILE--THE LAST HALF OF ITS NAME.

UNIT IS AN INTEGER DESCRIBING THE I/O DEVICE. THE VALID UNIT NUMBERS ARE GIVEN IN TABLE 9.1. NOTE THAT FOR INPUT FILES, THE ONLY DISTINCTION THE PROGRAM NEED MAKE IS BETWEEN DISK FILES AND NON-DISK FILES. THUS, IF UNIT = "0" AND THE FILE IS ON A MAGNETIC TAPE, THE PROGRAM WILL FIND IT CORRECTLY. FOR OUTPUT FILES, HOWEVER, IT IS IMPORTANT TO GET THE RIGHT INTEGER.

DATE IS THE DATE ASSOCIATED WITH THE FILE (NORMALLY ONLY USED WITH TAPES). TWO FORMS CAN BE USED: MM/DD/YY OR YYDDD (WHICH IS HOW IT IS STORED IN A FILE LABEL).

REEL IS THE REEL NUMBER, FOR MULTI-REEL TAPES.

CYCLE IS THE CYCLE NUMBER, USED WHEN SEVERAL TAPES HAVE THE SAME NAME AND DATE.

FOR EXAMPLE, TO ASSIGN THE I/O STRING CARD TO THE DISK FILE SMITH/CRD3, THE PROGRAM WOULD USE:

```
   FILL("CARD","SMITH","CRD3","12")
```

TO GIVE THE FILE NEWDISK THE NAME GEO/CALIBR, YOU COULD USE:

```
   FILL("NEWDISK","GEO","CALIBR")
```

NOTE THAT THE UNIT NEED NOT BE USED HERE; IT IS ONLY NEEDED WHEN A DIFFERENT I/O DEVICE IS DESIRED THAN WAS USED THE LAST TIME THE FILE WAS OPENED. FOR ANOTHER EXAMPLE, TO ASSIGN THE FILE READ TO THE CARD READER FILE INPT, THE FOLLOWING WOULD WORK:

```
   FILL("READ","0000000","INPT","0")
```

%PAGE

TABLE 9.1.

UNIT	DESCRIPTION
0	CARD READER/PUNCH
1	LINE PRINTER (NO BACK-UP)
2	LABELED MAGNETIC TAPE
3	UNUSED
4	LINE PRINTER OR TAPE BACK-UP
5	LABELED "DESIGNATED" FILE (?)
6	PRINTER BACK-UP TAPE
7	UNLABELED "DESIGNATED" FILE (?)
8	UNLABELED PAPER TAPE
9	UNLABELED MAGNETIC TAPE
10	RANDOM DISK FILE

11	SUPERVISORY PRINTER (SPO)
12	SERIAL DISK FILE
13	UPDATE DISK FILE
14	DATA COMMUNICATIONS
15	PRINTER BACK-UP DISK
16	PRINTER BACK-UP TAPE OR DISK
17	PRINTER OR BACK-UP DISK
18	PRINTER OR BACK-UP DISK OR BACK-UP TAPE
32	SPECIAL FORMS FILE.

%PAGE

.GE(A,B) SUCCEEDS IF A AND B ARE BOTH NUMERIC AND A ≥ B.

.GT(A,B) SUCCEEDS IF A AND B ARE BOTH NUMERIC AND A > B.

.LE(A,B) SUCCEEDS IF A AND B ARE BOTH NUMERIC AND A ≤ B.

LEVEL() RETURNS THE LEVEL OF FUNCTION CALLS WHICH THE PROGRAM HAS REACHED. THE LEVEL COUNTER STARTS AT 1, INCREASES WHEN A FUNCTION IS CALLED, AND DECREASES WHEN A FUNCTION RETURNS.

%CP 7

LOOK(S,F) DEFINES THE STRING $S TO BE A "NON-READING" INPUT STRING FOR THE FILE ASSOCIATED WITH THE I/O STRING $F. THIS MEANS THAT USING $S AS AN INPUT STRING WILL CAUSE THE "NEXT" RECORD TO BE READ FROM THE FILE, BUT THIS RECORD WILL REMAIN IN THE BUFFER, SO IT WILL BE THE NEXT ONE READ. THUS, CONSECUTIVE USES OF $S WILL GIVE THE SAME INPUT RECORD. THE SYSTEM DEFINES THE I/O STRING LOOK AS: LOOK("LOOK","READ").

.LT(A,B) SUCCEEDS IF A AND B ARE BOTH NUMERIC AND A < B.

%CP 52

MODE(V,P) SETS VARIOUS GLOBAL SYSTEM FLAGS, AS INDICATED BY V. P IS USUALLY NOT USED, BUT FOR SOME VALUES OF V IT INDICATES A VALUE TO BE USED WITH THE FLAG. V MUST BE ONE OF THE VALUES BELOW, OR MODE() WILL FAIL. THE CURRENTLY RECOGNIZED VALUES OF V ARE AS FOLLOWS:

MODE("ANCHOR",P) CAUSES ALL PATTERNS TO BE ANCHORED AT CHARACTER P. THAT IS, ALL PATTERNS MUST MATCH WITH THE FIRST PATTERN ELEMENT MATCHING IMMEDIATELY AFTER THE P-TH CHARACTER. FOR P = "0" (OR

```
   P NULL), THE PATTERN MUST MATCH AN INITIAL SUBSTRING.  NOTE THAT
   MOST VERSIONS OF SNOBOL DONT ALLOW THE EXTRA PARAMETER P, SO
   PROGRAMS THAT ARE TO BE RUN ON OTHER SNOBOL SYSTEMS SHOULD
   ONLY USE MODE("ANCHOR").  IF P IS NON-NUMERIC OR NEGATIVE,
   FAILURE OCCURS.
MODE("UNANCH") OR MODE("UNANCHOR") RETURNS THE PATTERN MATCH MODE
   TO ITS NORMAL, UNANCHORED MODE.  THAT IS, THE FIRST ELEMENT
   OF A PATTERN CAN MATCH STARTING ANYWHERE IN THE STRING REFERENCE.
MODE("DUMP") CAUSES DUMP() TO BE CALLED WHEN THE PROGRAM TERMINATES.
MODE("ERRDUMP") CAUSES DUMP() TO BE CALLED IF THE PROGRAM TERMINATES
   ABNORMALLY.
MODE("INFORM") CAUSES WARNING DIAGNOSTIC MESSAGES TO BE PRINTED
   WHEN CERTAIN CONDITIONS OCCUR THAT ARE VALID IN SNOBOL BUT ARE
   OFTEN SIGNS OF PROGRAM ERRORS WHEN THEY OCCUR.  SOME SUCH THINGS
   ARE: USING NON-NUMERIC ARGUMENTS TO ARITHMETIC OPERATORS OR
   ARITHMETIC FUNCTIONS; INDIRECTION WITH THE NULL STRING AS THE
   STRING NAME; USING POP() TO WIPE OUT THE PUSH-DOWN STACK FOR
   A FUNCTIONS FORMAL PARAMETERS OR LOCAL VARIABLES; USING A
   FIXED-LENGTH VARIABLE WITH NON-NUMERIC OR NEGATIVE LENGTH;
   ETC.  THIS LIST MAY BE EXTENDED IN THE FUTURE, AS MORE SUCH
   CONDITIONS ARE BROUGHT TO THE ATTENTIONS OF THE SYSTEM AUTHORS
   WHICH CAN EASILY BE CHECKED BY THE SNOBOL INTERPRETER.
MODE("SILENCE") TERMINATES MODE("INFORM").
MODE("DEBUG") TURNS ON THE CHECKING FOR DEBUG MODE.  THIS IS
   THE INTERACTIVE DEBUGGING TOOL THAT IS PART OF THE INTERPRETER;
   IT IS DESCRIBED IN SECTION 11.2.
MODE("RUN") TERMINATES DEBUG MODE, AND TURNS OFF THE ASSOCIATED
   LIMIT TESTS.
MODE("ROUND") SAYS THAT ALL DIVISION IS TO BE ROUNDED.  THAT IS,
   IF THE REMAINDER IS ≥ 1/2 OF THE DIVISOR, THE QUOTIENT IS
   INCREASED BY 1.
MODE("TRUNCATION") SAYS THAT DIVISION IS TO BE TRUNCATED--THE
   REMAINDER IS TO BE DISCARDED.
MODE("INTEGER") SAYS THAT DIVISION IS TO FAIL IF THE REMAINDER
   IS NOT ZERO.
MODE("OVERFLOW",P) WHERE $P IS A FILE OUTPUT STRING, SAYS THAT
   IF A STRING IS WRITTEN THAT IS LONGER THAN A RECORD, THE
   EXTRA CHARACTERS ARE WRITTEN ON THE NEXT RECORD (THIS WILL
   BE REPEATED IF THE REMAINING STRING IS STILL TOO LONG).
MODE("TRUNCATE",P) CANCELS MODE("OVERFLOW",P).  THIS MEANS THAT
   STRINGS THAT ARE TOO LONG FOR ONE RECORD WILL BE TRUNCATED,
   AND THE TAIL END WILL NOT BE OUTPUT.  THIS IS THE ASSUMED MODE
   FOR ALL OUTPUT FILES UNLESS MODE("OVERFLOW",P) IS USED.



.NE(A,B) SUCCEEDS IF A AND B ARE BOTH NUMERIC AND A ≠ B.



%CP 3
.NUM(A) SUCCEEDS IF A IS NUMERIC.  THAT IS, THE FIRST CHARACTER
   MUST BE A DIGIT OR "-"; ALL OTHER CHARACTERS MUST BE DIGITS.
   THE NULL STRING IS NUMERIC, AS IS "-"; BOTH = ZERO.
```

%CP 34

OPSYN(A,B,T) SETS THE OBJECT (STRING, FUNCTION, LABEL) NAMED BY A TO BE THE SAME AS THE ONE NAMED BY B. THE TYPE OF THE TWO OBJECTS IS GIVEN BY THE FIRST CHARACTER OF T, AS FOLLOWS:

```
    "S" MEANS THAT A AND B ARE STRING NAMES.
    "F" MEANS THAT A AND B ARE FUNCTION NAMES.
    "L" MEANS THAT A AND B CONTAIN LABELS.
```

IF T IS NULL, "F" IS ASSUMED--IN ORDER TO BE COMPATIBLE WITH THE MANY SNOBOL SYSTEMS THAT ONLY ALLOW THIS FUNCTION TO BE APPLIED TO FUNCTION NAMES. SOME EXAMPLES OF THE USE OF OPSYN() ARE:

OPSYN("XA","XB","STR") ASSIGNS THE VALUE OF XB TO XA, AND ALSO GIVES XA ANY I/O USE OF XB.

OPSYN("FL","RXA","F") DEFINES THE FUNCTION FL TO BE THE SAME AS THE FUNCTION RXA (INCLUDING TRACING, IF RXA IS BEING TRACED). IF RXA IS UNDEFINED, THEN FL IS ALSO UNDEFINED.

OPSYN("REDO","DO","LABEL") DEFINES THE LABEL REDO TO BE THE SAME INSTRUCTION AS DO. ANY INSTRUCTION PREVIOUSLY LABELED REDO IS LOST UNLESS IT HAS BEEN GIVEN ANOTHER NAME.

NOTE THAT OPSYN ACTS AS AN ASSIGNMENT, NOT A PERMANENT DEFINITION, SO A FURTHER CHANGE IN ONE OF THE VARIABLES WILL NOT AFFECT THE OTHER. IF, AFTER THE FIRST EXAMPLE ABOVE, XB WERE ASSIGNED A NEW VALUE, XA WOULD REMAIN UNALTERED; IF TRACEL("DO") WERE CALLED, TRACING WOULD BE STARTED FOR DO BUT NOT FOR REDO. A TRIVIAL EXAMPLE OF THE USE OF OPSYN IS:

```
    OPSYN("NULL","EQUALS")
```

WHICH WILL DEFINE A FUNCTION NULL() WHICH, WHEN CALLED WITH ONLY ONE ARGUMENT, WILL SUCCEED IF THE ARGUMENT IS NULL. THIS MAY BE DESIRABLE BECAUSE THE MEANING OF NULL(X) IS A BIT MORE OBVIOUS THAN THAT OF EQUALS(X).

%CP 16

PAGE(N,F) CAUSES A PAGE-EJECT TO BE DONE ON THE FILE $F, WHICH SHOULD BE A LINE PRINTER FILE. THE EJECTION IS TO "CHANNEL" N, WHICH IS A POSITION SOMEWHERE ON THE PAGE DETERMINED BY A PAPER TAPE CONTROL WITHIN THE PRINTER. THE EJECT WILL OCCUR ON THE NEXT LINE OF OUTPUT, AFTER THE OUTPUT OCCURS. N MUST BE A VALUE BETWEEN "0" AND "11"; "0" AND "1" ARE BOTH THE TOP OF THE NEXT PAGE. IF F IS OMITTED, THEN "PRINT" IS ASSUMED; IF N IS NULL, THE TOP OF THE PAGE ("1") WILL BE ASSUMED. THUS, TO SIMPLY DO A PAGE-EJECT ON THE PRINT FILE, THE FOLLOWING WOULD SUFFICE:

```
    PRINT ← PAGE()
```

ACTUALLY, PAGE() IS A SPECIAL CASE OF THE FUNCTION RECORD(), AND IS MEANT TO BE USED ONLY WITH LINE PRINTERS (THUS THE RESTRICTION ON THE RANGE OF N). THE SAME EFFECT COULD BE ACHEIVED BY USING RECORD(F,N), WITH 1 ≤ N ≤ 11.

```

%CP 12
POP(S) REMOVES THE TOP OF THE PUSH-DOWN STACK FOR THE STRING S.
   EVERY STRING NAME IN SNOBOL CAN BE CONSIDERED A PUSH-DOWN
   STACK, WITH THE TOP OF THE STACK THE CURRENT VALUE OF THE
   STRING.  ONE WARNING: IF THE FORMAL PARAMETERS OR LOCAL VAR-
   IABLES OF A FUNCTION ARE POPPED WITHIN THE FUNCTION, THE RETURN
   FROM THE FUNCTION WILL TRY TO POP THEM AGAIN; THIS CAN CAUSE
   SOME GRIEF IF CARE IS NOT TAKEN TO RESTORE THE STACKS TO THEIR
   CORRECT "DEPTH" IN SUCH CASES.  IN SUCH A SITUATION, THE RETURN
   WOULD NOT ALTER THE STRING (IF THE INFORM FLAG IS ON, A MESSAGE
   WILL BE PRINTED--SEE THE FUNCTION MODE()).  POP(S) WILL FAIL
   IF S IS NOT A STRING NAME, OR IF S CANT BE POPPED BECAUSE THERE
   IS ONLY ONE VALUE IN ITS STACK.



%CP 8
PUSH(S,V) PUSHES THE VALUE V ONTO THE PUSH-DOWN STACK FOR S.
   IF THIS IS FOLLOWED BY POP(S), S WILL RETURN TO ITS FORMER
   VALUE.  PUSH(S,V) WILL FAIL IF S ISNT A NAMED STRING.  ALSO,
   EACH CALL OF PUSH() REQUIRES ONE MORE ENTRY IN THE SYMBOL
   TABLE; THERE IS ROOM IN THE TABLE FOR SOMEWHAT OVER 4000
   SYMBOLS (STRINGS, LITERALS, FUNCTIONS, LABELS, TEMPORARY
   STORAGE FOR GROUPINGS, ETC.) AND EXHAUSTING THIS SPACE WILL
   CAUSE THE PROGRAM TO DIE ABNORMALLY.


%CP 8

.RANF(N) PRODUCES A PSEUDO-RANDOM NUMBER IN THE RANGE:

      1 ≤ .RANF(N) ≤ N

   THE GENERATOR IS ALWAYS INITIALIZED TO THE SAME NUMBER, SO
   THE VALUES RETURNED ARE "PREDICTABLE" IN THE SENSE THAT REPEATING
   A RUN WITH THE SAME VALUES FOR N WILL PRODUCE THE SAME SEQUENCE
   OF NUMBERS.



%CP 10
RECORD(F,N) SETS THE "RECORD POINTER" FOR THE FILE ASSOCIATED
   WITH $F TO N.  $F MUST BE A FILE I/O STRING; N MUST BE A
   NON-NEGATIVE NUMBER, OR RECORD(F,N) WILL FAIL.  THIS FUNCTION
   IS REALLY ONLY USEFUL FOR DISK FILES.  FOR LINE PRINTER FILES,
   IT IS THE SAME AS THE FUNCTION PAGE(); FOR OTHER NON-DISK FILES
   RECORD() IS A NO-OP.  FOR DISK FILES, USING RECORD(F,N) SAYS
   THAT THE NEXT I/O OPERATION IS TO BE ON RECORD N.  THE RECORDS
   OF A DISK FILE ARE NUMBERED STARTING AT 0.  IF N IS GREATER THAN
   THE NUMBER OF THE LAST RECORD IN THE FILE, THE NEXT I/O OPERATION
   WILL FAIL DUE TO END-OF-FILE.



%CP 5
RELEASE(A) WHERE A IS A TELETYPE TERMINAL ADDRESS, REMOVES THE
```

```
   TELETYPE FROM THE LIST OF TELETYPES ATTACHED TO THE PROGRAM.
   FURTHER I/O ON THIS TELETYPE BECOMES IMPOSSIBLE UNTIL THE PERSON
   AT THE TELETYPE RE-ATTACHES HIMSELF.  RELEASE(A) WILL FAIL IF A
   IS NOT A VALID ADDRESS OR THE TELETYPE IS NOT ATTACHED.



%CP 4
REWIND(F) CAUSES THE FILE ASSOCIATED WITH $F TO BE REWOUND.  THE
   FILE REMAINS OPEN, AND I/O CAN CONTINUE AS USUAL, STARTING
   FROM THE FIRST RECORD OF THE FILE.  FOR DISK FILES, REWIND(F)
   IS EQUIVALENT TO RECORD("0",F).



%CP 7
RULES(N) RETURNS THE NUMBER OF RULES EXECUTED AT LEVEL N (≥ 1)
   SINCE THE LAST TIME LEVEL N WAS REACHED.  IF N = "0" (OR NULL)
   THE TOTAL FOR THE ENTIRE PROGRAM IS RETURNED.  IF N ≤ LEVEL(),
   THE CURRENT INSTRUCTION IS INCLUDED IN THE TOTAL.  NOTE THAT IF
   RULES(N) = "0", THEN LEVEL N HAS NEVER BEEN REACHED.  IF N IS
   NON-NUMERIC OR NEGATIVE, RULES(N) WILL FAIL.



%CP 30
SEARCH(FILE) WHERE $FILE IS AN I/O STRING TO A DISK FILE, DOES A
   SEARCH OF THE DISK DIRECTORY FOR THE FILE.  IF IT IS PRESENT,
   A DESCRIPTION OF THE FILE IS RETURNED.  FAILURE OCCURS IF $FILE
   IS NOT A DISK FILE, OR IF THE FILE IS NOT IN THE DIRECTORY.
   IF THE FILE IS IN THE DIRECTORY, THE RETURN VALUE IS OF THE
   FOLLOWING FORM:

      <MFID> / <FID> : <SECURITY> : <REC SIZE> : <BLCK SIZE> :
         <EOF PNTR> : <OPEN COUNT>

   THE EXTERNAL NAME OF THE FILE IS <MFID>/<FID>.  THE SECURITY
   STATUS, <SECURITY>, IS A STRING OF 3 "0"-S OR "1"-S WITH THE
   FOLLOWING MEANING:

      1ST = "1" IF THE USER CAN ALTER THE SECURITY STATUS OR
         NAME, OR CAN REMOVE THE FILE.
      2ND = "1" IF THE USER CAN READ FROM THE FILE.
      3RD = "1" IF THE USER CAN WRITE ON THE FILE.

   THE SIZE OF THE LOGICAL AND PHYSICAL RECORDS OF THE FILE, IN
   CHARACTERS, ARE GIVEN BY <REC SIZE> AND <BLCK SIZE>.  <EOF PNTR>
   IS THE RELATIVE ADDRESS OF THE LAST RECORD OF THE FILE (THAT
   IS, ONE LESS THAN THE NUMBER OF RECORDS IN THE FILE).  <OPEN COUNT>
   IS THE NUMBER OF PROGRAMS DOING I/O ON THE FILE.  NOTE THAT,
   UNLIKE ALGOL WHICH DEALS ENTIRELY WITH WORDS, THE RECORD SIZE
   AND BLOCK SIZE ARE GIVEN TO A SNOBOL PROGRAM IN CHARACTERS.
   IF <SECURITY> = "000", THEN <REC SIZE>, <BLCK SIZE>, ETC.
   WILL ALL BE "0".
```

SIZE(S) RETURNS THE NUMBER OF CHARACTERS IN S. THUS, SIZE("ABC") = "3".

%CP 13

SPACE(F,N) SETS A COUNTER FOR THE FILE ASSOCIATED WITH $F TO N, SO THAT EACH I/O OPERATION ON THE FILE IS PRECEDED BY SKIPPING N RECORDS (ON INPUT) OR WRITING N BLANK RECORDS (ON OUTPUT). FOR LINE PRINTER FILES, N = "1" IS NORMAL, N = "0" CAUSES OVERPRINTING, N = "2" CAUSES DOUBLE SPACING, ETC. FOR OTHER FILES, A SKIP TO THE NEXT RECORD IS PART OF THE I/O OPERATION, AND N = "0" IS THE NORMAL SETTING. SPACE(F,N) WILL FAIL IF $F IS NOT AN I/O STRING FOR A FILE, OR IF N IS NON-NUMERIC. FOR EXAMPLE,

```
      SPACE("PRINT","2")
```

WILL CAUSE DOUBLE-SPACING ON THE LINE PRINTER FILE PRINT UNTIL ANOTHER CALL OF SPACE() CHANGES THE SPACING.

%CP 6

SUSPEND(N) HAS TWO USES. IF N IS NON-NULL, IT MUST BE AN INTEGER; THE PROGRAM WILL BE SUSPENDED FOR N SECONDS AND THEN WILL RESUME EXECUTION FROM THE FUNCTION CALL, WHICH WILL RETURN A NULL STRING. IF N ≤ 0, THE CALL IS A NO-OP. IF N IS NULL (OR MISSING), THE PROGRAM IS SUSPENDED AND THE INTERACTIVE DEBUGGING ROUTINE IS ACTIVATED. SEE SECTION 11.2 FOR DETAILS.

%CP 50

STATUS(A,I) GIVES THE STATUS OF ATTACHED TELETYPES. A IS EITHER THE ADDRESS OF AN ATTACHED TELETYPE, OR IS NULL, WHICH MEANS THAT ALL ATTACHED TELETYPES ARE TO BE CONSIDERED. I IS A NUMBER, EITHER "0" OR "1", DEPENDING ON THE TYPE OF STATUS STRING DESIRED. THE MEANINGS OF THE TWO VALUES OF I ARE:

I = "0" IS A "PASSIVE" STATUS. THAT IS, THE STATUS AS OF THE LAST HARDWARE INTERRUPT ASSOCIATED WITH THE TELETYPE IS RETURNED. INTERRUPTS OCCUR FOR I/O OPERATIONS AND FOR STATUS INTERROGATES.

I = "1" IS AN "ACTIVE" STATUS. THIS MEANS THAT A HARDWARE STATUS INTERROGATE IS PERFORMED, THE MCP STATUS WORD IS UPDATED, AND THE NEW VALUE IS THE VALUE RETURNED. UNFORTUNATELY, THE MCP ALSO INTERPRETS AN ACTIVE INTERROGATE AS A COMMAND TO START CHARGING THE PROJECT NUMBER OF THE PERSON AT THAT TELETYPE FOR THE JOB TIME. FOR SINGLE-USER JOBS, THIS HAS NO EFFECT, BUT FOR MULTI-USER JOBS IT CAN BE RATHER ANNOYING.

THE FORM OF A STATUS STRING FOR A SINGLE TERMINAL IS:

```
      <STATUS> = ( <PROJ#> / <USER#> : <ADDR> : <CONDITIONS> )
```

WHERE <PROJ#>/<USER#> IS THE USERS PROJECT AND I.D. NUMBER. (THIS IS THE SECURITY SYSTEM IN USE AT THE UNIVERSITY OF WISCONSIN; USERS AT OTHER INSTALLATIONS WILL PROBABLY HAVE TO ALTER THIS

```
   INTRINSIC TO HANDLE THEIR LOCAL ACCOUNTING PROCEDURES.)
   <ADDR> IS THE ADDRESS OF THE TELETYPE, AND <CONDITIONS> IS
   A STRING OF "0"-S AND "1"-S WITH THE FOLLOWING MEANINGS:

      1ST = "1" IF UNIT IS READY.
      2ND = "1" IF UNIT IS READ-READY (I.E., THERE IS INPUT TO READ).
      3RD = "1" IF UNIT IS WRITE-READY (OUTPUT IS POSSIBLE)
      4TH = "1" IF WRITE IS IN PROGRESS.
      5TH = "1" IF BREAK KEY USED ON LAST OUTPUT.
      6TH = "1" IF HARDWARE DETECTS ABNORMAL CONDITION.
      7TH = "1" IF UNIT IS BUSY.
      8TH = "1" IF SOME PROGRAM HAS EXCLUSIVE USE OF UNIT.

   MORE CONDITIONS MAY BE ADDED IN THE FUTURE, AS THEY ARE MADE
   AVAILABLE BY THE HARDWARE AND BY THE MCP.
   WHEN A IS NULL, THE STATUS STRINGS OF ALL ATTACHED TELETYPES
   IS RETURNED IN THE FORM:

      <STATUS> <STATUS> <STATUS> ... <STATUS>

   WITH ONE <STATUS> FOR EACH TELETYPE.  IF THERE ARE NO TELETYPES
   ATTACHED, OR IF A IS NON-NULL AND NOT THE ADDRESS OF AN ATTACHED
   TELETYPE, STATUS() WILL FAIL.  FAILURE ALSO OCCURS IF I ISNT
   NUMERICALLY "0" OR "1" (NULL OR MISSING IS "0").



%CP 20
TIME(N) RETURNS VARIOUS TIMES ASSOCIATED WITH THE PROGRAM, AS FOLLOWS:

   TIME("0") OR TIME() RETURNS THE TIME OF DAY IN THE FORM:
            HH:MM:SS

   TIME("1") RETURNS THE ELAPSED TIME SINCE MIDNIGHT IN SIXTIETHS
      OF A SECOND.

   TIME("2") RETURNS THE PROGRAMS TOTAL CPU TIME, IN SIXTIETHS OF
      A SECOND.

   TIME("3") RETURNS THE PROGRAMS TOTAL I/O CHANNEL TIME, AGAIN IN
      SIXTIETHS OF A SECOND.

   TIME("4") RETURNS THE VALUE OF A SIX-BIT TIMER IN THE FORM OF
      A SINGLE CHARACTER.  THE TIMER USED CHANGES EVERY SIXTIETH OF
      A SECOND.  WHEN CALLED AT WIDELY SPACED INTERVALS (THAT IS,
      WITH SEVERAL HUNDRED SNOBOL INSTRUCTIONS IN BETWEEN), IT
      BEHAVES MUCH LIKE A RANDOM CHARACTER.



%CP 2
TRACEF(F1,F2,F3,...) TURNS ON TRACING FOR THE FUNCTIONS NAMED BY
   F1, F2, F3, ETC.  SEE SECTION 11.1 FOR DETAILS.



%CP 2
```

```
TRACEL(L1,L2,L3,...) TURNS ON TRACING FOR THE LABELS CONTAINED IN L1,
   L2, L3, ETC.  SEE SECTION 11.1 FOR DETAILS.



%CP 2
TRACES(S1,S2,S3,...) TURNS ON TRACING FOR THE STRINGS NAMED BY
   S1, S2, S3, ETC.  SEE SECTION 11.1 FOR DETAILS.



%CP 6
TRIM(S) RETURNS THE VALUE OF S WITH ALL TRAILING BLANKS (IF ANY)
   REMOVED.  SOME EXAMPLES ARE:

   TRIM(" AB C   ")  =  " AB C"
   TRIM("      "     =  ""
   TRIM(" XYZ")      =  " XYZ"



%CP 3
UNEQL(A,B) SUCCEEDS IF A AND B HAVE DIFFERENT STRINGS AS VALUES.
   THE USER SHOULD COMPARE UNEQL() WITH THE FUNCTIONS EQUALS() AND
   .NE().


%CP 7
USER(A) HAS TWO USES.  IF A IS NULL, THE VALUE IS THE ADDRESS OF
   THE TELETYPE BEING USED FOR I/O WITH THE STRING TELETYPE.  IF
   A IS NON-NULL, IT MUST BE THE ADDRESS OF AN ATTACHED TELETYPE.
   THIS TELETYPE BECOMES THE ONE BEING USED FOR I/O, AND ITS ADDRESS
   IS ALSO THE VALUE RETURNED.  NOTE THAT IN THE LATTER CASE, USER(A)
   IS NOT NECESSARILY THE SAME AS A, SINCE THE ADDRESS IS PUT INTO
   A STANDARD FORMAT.  FOR EXAMPLE, USER("01/03") = "1/3".



%CP 6
WAIT(N) SETS THE WAITING TIME FOR TELETYPE I/O TO N SECONDS, IF N
   IS NON-NULL AND NUMERIC.  IF N IS NON-NUMERIC, WAIT(N) FAILS.
   IF N IS NULL, THE WAITING TIME IS NOT ALTERED.  IN EITHER CASE,
   THE RETURN VALUE IS THE WAITING TIME AS IT WAS BEFORE THE FUNCTION
   WAS CALLED.  THIS FUNCTION IS EQUIVALENT TO THE -WAIT CONTROL
   CARD; IF NEITHER IS USED, 300 SECONDS (5 MINUTES) IS ASSUMED.



%SECTION 10
%PAGE 1
10. INPUT/OUTPUT.

   I/O ON THE B5500 IS NORMALLY ASSOCIATED WITH A FILE, WHICH IS AN
ORDERED SET OF I/O "RECORDS".  A FILE IS IDENTIFIED BY ITS NAME,
WHICH IS OF THE FORM <MULTI-FILE I.D.>/<FILE I.D.>, WHERE BOTH
<MULTI-FILE I.D.> AND <FILE I.D.> ARE STRINGS OF FROM 1 TO 7
CHARACTERS.  NORMALLY, ONLY ALGOL-LIKE IDENTIFIERS ARE USED IN FILE
```

NAMES, SINCE MANY MCP FILE-HANDLING COMMANDS CAN ONLY HANDLE SUCH NAMES. A SPECIAL CASE IS MADE FOR CARD-READER FILES, WHICH MAY HAVE ONLY A <FILE I.D.>--THE <MULTI-FILE I.D.> IS THEN ASSUMED TO BE "0000000".

TELETYPES ARE HANDLED DIFFERENTLY THAN NON-INTERACTIVE I/O DEVICES. THERE IS NO "FILE" ASSOCIATED WITH A TELETYPE (AS FAR AS SNOBOL USERS ARE CONCERNED). TELETYPE I/O IS DISCUSSED IN SECTION 10.4.

%CP 21

10.1. I/O STRINGS.

TO DO I/O, SNOBOL USES "I/O STRINGS" THESE ARE STRING NAMES THAT ARE ASSOCIATED WITH I/O OPERATIONS IN ONE OF TWO WAYS: IF A STRING HAS "INPUT USE", THEN ATTEMPTING TO USE ITS VALUE WILL CAUSE INPUT OF A RECORD FROM ITS ASSOCIATED FILE; IF THE STRING HAS "OUTPUT USE", THEN ASSIGNING A VALUE TO THE STRING WILL CAUSE THE STRING TO BE OUTPUT TO ITS ASSOCIATED FILE. FOR EXAMPLE, THE STRING NAMED "PRINT" IS AN OUTPUT STRING TO THE LINE PRINTER; THE INSTRUCTION:

```
      PRINT = "**A = " QUOTE A QUOTE
```

WILL OUTPUT THE VALUE OF A, IDENTIFIED AS SUCH, TO THE PRINTER.

BEFORE AN I/O STRING CAN BE USED, IT IS NECESSARY TO ASSIGN IT TO A FILE. THERE ARE TWO WAYS TO DO THIS IN SNOBOL. THE WAY THAT IS RECOMMENDED IS TO USE THE INTRINSIC FUNCTION FILL(), WHICH IS DESCRIBED IN SECTION 9. AN ASSIGNMENT CAN BE MADE BY USING THE MCP "?FILE" CONTROL CARD, THOUGH THIS IS RATHER RESTRICTED. SEE SECTION 2.1 FOR A DESCRIPTION OF THIS CONTROL CARD.

%CP 24

10.2. OPEN AND CLOSED FILES.

A FILE IS EITHER "OPEN" OR "CLOSED" WITH RESPECT TO A GIVEN PROGRAM. A FILE IS OPEN IF THE PROGRAM HAS DONE SOME I/O ON IT AND HAS NOT CLOSED IT. IN SNOBOL, THERE ARE TWO WAYS TO CLOSE A FILE: BY CALLING THE INTRINSIC FUNCTION CLOSE(), AND BY TERMINATING THE PROGRAM.

WHEN THE FILE ASSOCIATED WITH A GIVEN I/O STRING HAS BEEN CLOSED, THE PROGRAM CAN USE THE FILL() INTRINSIC TO RE-ASSIGN THE I/O STRING TO A DIFFERENT FILE. THE NEW FILE CAN THEN BE OPENED BY DOING I/O ON IT. A PROGRAM CAN CLOSE A FILE, RE-ASSIGN ITS I/O STRING, AND OPEN THE NEW FILE AS OFTEN AS DESIRED. FOR EXAMPLE, IF A PROGRAM HAS BEEN USING THE I/O STRING CARD, WHICH IS ASSIGNED TO, SAY, THE FILE A/B, THIS FILE CAN BE DROPPED AND I/O RE-INITIATED ON THE DISK FILE C/D BY THE FOLLOWING CODE:

```
      CLOSE("CARD")
      FILL("CARD","C","D","12")
      C = CARD
```

THIS WILL ASSIGN C THE FIRST RECORD IN THE FILE C/D.

SOME I/O STRINGS (AND THEIR ASSOCIATED FILES) ARE OPEN AT THE

START OF A SNOBOL JOB, WHILE OTHERS ARE CLOSED. SEE THE NEXT SECTION THE SUCH INFORMATION ON PARTICULAR I/O STRINGS

%CP 10

10.3. STANDARD I/O STRINGS.

THERE ARE A NUMBER OF STANDARD I/O STRINGS PROVIDED BY THE B5500 SNOBOL SYSTEM, WHICH CAN BE ASSIGNED TO FILES OF THE APPROPRIATE TYPES. UNFORTUNATELY, THE B5500 OPERATING SYSTEM MAKES IT IMPOSSIBLE TO PROVIDE THE USER WITH A WAY TO DEFINE HIS OWN FILE TYPES (HOPEFULLY THIS MAY BE REMEDIED SOMETIME IN THE FUTURE); THE I/O STRINGS THAT ARE PROVIDED BY SNOBOL SHOULD BE SUFFICIENT FOR THE NEEDS OF MOST USERS.

THE FOLLOWING I/O STRINGS ARE PRE-DEFINED BY B5500 SNOBOL3:

%CP 8

READ IS INPUT FROM A CARD-IMAGE FILE--THAT IS, ANY FILE (CARD, TAPE, DISK, ETC.) WITH 80-CHAR LOGICAL RECORDS AND 240-CHAR PHYSICAL RECORDS. CARD-READER FILES ACTUALLY DO NOT HAVE 240-CHAR PHYSICAL RECORDS, BUT THE SOFTWARE MAKES IT LOOK LIKE THEY DO. FOR BATCH JOBS, READ IS OPEN AT THE START OF EXECUTION, AND IS ASSIGNED TO THE FILE THAT CONTAINS THE PROGRAM. THE FIRST RECORD READ WILL BE THE ONE FOLLOWING THE END CARD. FOR REMOTE JOBS, READ IS CLOSED AT THE START OF EXECUTION.

%CP 13

LOOK IS A "NON-READING" INPUT STRING FOR THE SAME FILE THAT THE STRING READ IS ASSIGNED TO. THAT IS, REPEATED USES OF LOOK WILL GIVE THE SAME RECORD OF THE FILE; USING READ WILL THEN GIVE THIS SAME RECORD, AND ADVANCE TO THE NEXT ONE. THUS:

```
A = LOOK
B = LOOK
C = READ
D = READ
```

WILL ASSIGN THE SAME RECORD TO A, B, AND C; D WILL GET THE NEXT ONE. THE FUNCTION LOOK() CAN BE USED TO DEFINE STRINGS OF THIS TYPE FOR OTHER INPUT FILES. THE STRING LOOK CAN IN FACT BE DEFINED BY LOOK("LOOK","READ").

%CP 5

CARD IS BOTH INPUT AND OUTPUT FOR A CARD-IMAGE FILE WITH 80-CHAR LOGICAL RECORD AND 240-CHAR PHYSICAL RECORD. OF COURSE, IF IT IS ASSIGNED TO A CARD-READER FILE, OUTPUT IS IMPOSSIBLE (AND WILL CAUSE TERMINATION BY THE MCP). CARD IS CLOSED AT THE START OF EXECUTION.

%CP 28

NEWDISK IS AN I/O STRING FOR A DISK FILE THAT WILL BE CREATED WHEN THE FILE IS OPENED. THIS FILE CAN BE VIEWED AS A TEMPORARY DISK FILE, WHICH WILL BE SAVED (PUT IN THE DISK DIRECTORY) WHEN CLOSE("NEWDISK") OR CLOSE("NEWDISK","LOCK") IS CALLED, OR WHEN THE PROGRAM TERMINATES. IF ANY OTHER SECOND PARAMETER

```
    IS USED, THE FILE WILL BE CLOSED AND DISCARDED.  THE FILE ASSOC-
    IATED WITH NEWDISK HAS 80-CHAR LOGICAL RECORDS AND 240-CHAR
    PHYSICAL RECORDS.  WHEN NEWDISK IS CLOSED, IT CAN THEN BE
    GIVEN A NEW NAME, IF DESIRED, AND RE-OPENED, AND ANOTHER
    DISK FILE WILL BE CREATED.  IF THERE IS ALREADY A FILE IN THE
    DIRECTORY WITH THE NAME ASSIGNED TO NEWDISK, THEN ATTEMPTING
    TO ENTER IT IN THE DIRECTORY WILL CAUSE THE PROGRAM TO BE SUSPENDED
    UNTIL THE OPERATOR TAKES ACTION--USUALLY THIS ACTION CONSISTS OF
    KILLING THE PROGRAM.  THE MAXIMUM NUMBER OF RECORDS THAT CAN
    BE PUT IN A FILE ASSOCIATED WITH NEWDISK IS 12000.  THE END-OF-
    FILE IS ALWAYS JUST AFTER THE LAST RECORD THAT HAS BEEN WRITTEN
    ON THE FILE.  THE SAVE FACTOR GIVEN TO FILES CREATED BY
    NEWDISK IS 15 DAYS.


%CP 9
PRINT IS AN OUTPUT FILE TO THE LINE PRINTER.  IT IS OPEN AT THE
    START OF EXECUTION, SINCE THIS FILE IS USED BY THE COMPILER
    TO PRODUCE THE LISTING AND THE OTHER OUTPUT THAT THE COMPILER
    GENERATES.  ON MOST B5500 SYSTEMS, THE LINE PRINTER HAS 132
    CHARACTERS PER LINE; OUTPUT OF A STRING LONGER THAN THIS WILL
    CAUSE TRUNCATION UNLESS MODE("OVERFLOW","PRINT") IS CALLED
    (SEE SECTION 9, THE MODE() FUNCTION).  CARRIAGE CONTROL IS
    HANDLED BY THE FUNCTIONS SPACE() AND PAGE(), WHICH ARE DESCRIBED
    IN SECTION 9.

%CP 12
SYSPOT IS AN ATTEMPT TO SIMULATE THE FIRST-CHARACTER CARRIAGE CONTROL
    OF IBM, CDC, ETC. LINE PRINTERS.  THE FIRST CHARACTER OF THE
    OUTPUT IS DELETED AND TRANSLATED INTO THE APPROPRIATE CALLS OF
    SPACE() AND PAGE(); THE REST OF THE STRING IS THEN PASSED TO
    THE OUTPUT STRING, PRINT.  THE CARRIAGE CONTROL CHARACTERS
    CURRENTLY RECOGNIZED ARE:

       " "  SINGLE SPACING
       "0"  (ZERO) DOUBLE SPACING
       "1"  PAGE EJECT
       "*"  NO SKIP AT END OF PAGE
       "+"  NO SPACING--OVERPRINTING


%CP 3
PUNCH IS OUTPUT TO THE CARD PUNCH.  THE LOGICAL AND PHYSICAL
    RECORD SIZES ARE BOTH 80 CHARACTERS.  PUNCH IS CLOSED AT THE
    START OF EXECUTION.

%CP 2
TELETYPE IS AN I/O STRING TO ALL ATTACHED TELETYPES.  IT IS
    DESCRIBED IN THE NEXT SECTION.



%CP 12
10.4. TELETYPES.

    TELETYPES ARE HANDLED SOMEWHAT DIFFERENTLY FROM THE NON-INTERACTIVE
I/O DEVICES.  THERE IS NO "FILE" OF PRE-DEFINED "RECORDS" WHEN
```

USING A TELETYPE. ALSO, TELETYPE I/O OPERATIONS, ESPECIALLY INPUT, CANT ALWAYS BE DONE IMMEDIATELY; IT IS OFTEN NECESSARY FOR THE PROGRAM TO WAIT, ESPECIALLY FOR INPUT. THE INTRINSIC FUNCTIONS THAT ARE USED WITH FILE I/O STRINGS WILL FAIL IF A TELETYPE I/O STRING IS GIVEN TO THEM, AND THERE ARE A FEW FUNCTIONS THAT ARE USED ONLY WITH TELETYPES.

THIS SECTION (AND ITS SUB-SECTIONS) GIVES A GENERAL PICTURE OF HOW I/O IS DONE ON TELETYPES.

%CP 28

10.4.1. TELETYPE INPUT AND OUTPUT.

THERE IS A SINGLE I/O STRING USED WITH TELETYPES; IT IS CALLED "TELETYPE". WHEN TELETYPE IS USED AS AN INPUT STRING, A STRING IS READ FROM A TELETYPE. THE INPUT IS GIVEN TO THE PROGRAM WHEN THE USER TYPES THE "END-OF-MESSAGE" CHARACTER, AN ARROW, "←". (THE CTRL-Q COMBINATION IS ALSO END-OF-MESSAGE.) IF NO INPUT IS RECEIVED WITHIN THE WAITING TIME FOR TELETYPE I/O, THEN TELETYPE WILL FAIL.

WHEN A PROGRAM ASSIGNS A VALUE TO TELETYPE, THE VALUE IS WRITTEN ON THE TELETYPE. THE OUTPUT OPERATION CAN FAIL IF, FOR SOME REASON, THE OUTPUT CANT BE INITIATED WITHIN THE WAITING TIME, OR IF THE USER PUSHES THE BREAK KEY DURING OUTPUT. THE OUTPUT STARTS WHEREVER THE WRITE HEAD MAY HAPPEN TO BE, AND IF THE END OF THE LINE IS REACHED, OVERPRINTING WILL OCCUR. THE PROGRAM MUST INDICATE THE "CARRIAGE CONTROL" BY USING THE FOLLOWING CHARACTERS:

"≤" (SINGLE QUOTE ON THE TELETYPE) CAUSES A RETURN TO THE FIRST CHARACTER POSITION OF THE CURRENT LINE.

"≠" (UP-ARROW ON THE TELETYPE) CAUSES THE PAPER TO ADVANCE ONE LINE ("LINE FEED").

IT SHOULD BE NOTED THAT ASSIGNING A VALUE TO TELETYPE ONLY INITIATES OUTPUT. AS SOON AS THE OUTPUT STARTS, THE OUTPUT ROUTINE RETURNS TO THE PROGRAM AND EXECUTION CONTINUES. SINCE A TELETYPE IS SUCH A SLOW OUTPUT DEVICE, THE PROGRAM CAN GET QUITE A LOT OF COMPUTING DONE BY THE TIME THE OUTPUT IS FINISHED. IT IS QUITE COMMON FOR A PROGRAM TO HAVE A SECOND OUTPUT STRING READY BY THE TIME A PREVIOUS OUTPUT IS FINISHED TYPING.

THERE CAN BE SEVERAL TELETYPES ATTACHED TO A SNOBOL PROGRAM AT THE SAME TIME. AT A GIVEN INSTANCE, THE PROGRAM CAN DO I/O WITH ONLY ONE OF THEM, BUT THE ONE BEING TALKED TO CAN BE CHANGED BY USING THE INTRINSIC FUNCTION USER(). FOR OTHER FUNCTIONS RELEVANT TO TELETYPE, SEE THE DESCRIPTIONS OF STATUS(), WAIT(), RELEASE(), AND SUSPEND().

%CP 7

10.4.2. TELETYPE WAITING TIMES.

AN I/O OPERATION ON A TELETYPE WILL FAIL IF IT DOES NOT SUCCEED WITHIN A FIXED PERIOD OF TIME CALLED THE "WAITING TIME". THIS TIME LIMIT CAN BE SET BY USING THE -WAIT CONTROL CARD OR THE INTRINSIC FUNCTION WAIT(). IF NEITHER IS USED, A WAITING TIME OF 300 SECONDS

(5 MINUTES) IS ASSUMED BY SNOBOL.

%CP 40

10.4.3. TELETYPE ADDRESSES.

THIS SECTION IS ONLY OF INTEREST TO USERS WISHING TO USE SEVERAL TELETYPES AT THE SAME TIME. OTHER USERS SHOULD PROBABLY IGNORE IT.

SINCE IT IS POSSIBLE FOR A PROGRAM TO BE COMMUNICATING WITH SEVERAL TELETYPES, AND ALL TELETYPE I/O IS DONE THROUGH THE ONE I/O STRING, TELETYPE, THERE MUST BE A WAY TO DISTINGUISH BETWEEN THE INDIVIDUAL TELETYPES. THIS IS DONE BY GIVING EACH TELETYPE AN ADDRESS. THE FUNCTION USER() CAN THEN BE USED TO SWITCH BACK AND FORTH BETWEEN THE VARIOUS TELETYPES ATTACHED.

A TELETYPE ADDRESS CONSISTS OF A PAIR OF INTEGERS SEPARATED BY A "/". THE FIRST INTEGER IS CALLED THE "TERMINAL UNIT" NUMBER; THE SECOND IS CALLED THE "BUFFER ADDRESS" (THE USER NEED NOT REMEMBER THESE TERMS, SINCE THEY ARE RATHER WORTHLESS FOR MOST PURPOSES). SOME POSSIBLE ADDRESSES, DEPENDING ON THE HARDWARE AVAILABLE, ARE: "1/5", "1/15", "3/7", "10/13", "2/2", ETC.

A SNOBOL PROGRAM CAN ONLY COMMUNICATE WITH TELETYPES THAT ARE "ATTACHED" TO IT. A TELETYPE BECOMES ATTACHED TO A SNOBOL PROGRAM IN ONE OF TWO WAYS:

1) THE USER INITIATES THE JOB FROM A TELETYPE, WITH AN MCP CONTROL CARD STARTING:

```
    ??EXECUTE SNOBOL/SNOBOL ...
```

2) THE USER ATTACHES HIS TELETYPE TO A RUNNING JOB WITH ONE OF THE FOLLOWING EQUIVALENT COMMANDS:

```
    ??RUN SNOBOL/SNOBOL←
    ??RUN SNOBOL←
```

A PROGRAM CAN FIND OUT WHICH TELETYPES, IF ANY, ARE ATTACHED BY USING THE STATUS() FUNCTION. ANY OF THE TELETYPES GIVEN BY THIS FUNCTION CAN BE USED AS AN I/O DEVICE BY CALLING USER(A), WHERE A IS THE TELETYPE ADDRESS. FOR EXAMPLE, IF IT IS KNOWN THAT A TELETYPE WITH THE ADDRESS "1/5" IS ATTACHED, I/O CAN BE DONE ON IT IN THE FOLLOWING MANNER:

```
      USER("1/5")
      TELETYPE = "≤≠" MESSAGE "≤≠"
      ANSWER = TELETYPE       /F(NO.RESPONSE)
```

OF COURSE, IF THERE IS ONLY ONE TELETYPE ATTACHED (AS IS ALMOST ALWAYS THE CASE), THERE IS NO NEED TO WORRY ABOUT WHICH ONE IT IS, SINCE ALL TELETYPE I/O WILL BE WITH THAT ONE TELETYPE.

%CP 40

10.4.4. SPECIAL CHARACTERS FOR THE TELETYPE.

THE CHARACTERS LISTED BELOW REQUIRE SPECIAL ATTENTION WHEN USED WITH TELETYPE I/O. SOME ARE INTERPRETED AS CONTROL CHARACTERS BY THE HARDWARE, AND OTHERS ARE PRINTED DIFFERENTLY ON THE LINE PRINTER

```
THAN ON THE TELETYPE.


  PRINTER TELETYPE         DESCRIPTION

    "←"     "←"            "GROUP MARK", OR "END-OF-MESSAGE" CHARACTER.
                           THIS CAN NOT BE READ OR WRITTEN.  THE
                           CTRL-Q COMBINATION IS THE SAME CHARACTER,
                           BUT DOESNT PRINT.

    "≤"     APOSTROPHE     THIS CAN BE INPUT, BUT WHEN OUTPUT, IT
                           CAUSES A "CARRIAGE RETURN"--THE WRITE
                           HEAD RETURNS TO COLUMN 1, AND NO CHAR
                           IS PRINTED.

    "≠"     UP-ARROW       THIS CAN ALSO BE INPUT, BUT WHEN OUTPUT,
                           IT CAUSES THE PAPER TO ADVANCE BY ONE LINE.

    "≥"     EXCLAMATION    WHEN TYPED, THIS CAUSES ERASURE OF THE
                           THE CURRENT "BUFFER"; WHEN OUTPUT, IT
                           CAUSES A DISCONNECT.
    "<"    "<"             IS IGNORED ON OUTPUT; WHEN TYPED, IT
                           SOMETIMES CAUSES ERASURE OF THE PREVIOUS
                           CHARACTER TYPED, AND SOMETIMES ACTS
                           LIKE A "←".  THE RULES THAT SAY WHICH
                           WILL OCCUR ARE TOO COMPLICATED TO BE
                           OF ANY PRACTICAL USE.

    "×"    BACK-SLASH      NORMAL EXCEPT FOR THE DIFFERENT CHARACTER
                           USED ON THE TELETYPE.  MOST TELETYPES
                           ALSO DONT INDICATE THAT THIS CHARACTER
                           IS TYPED BY USING SHIFT-L.

NOTE THAT "[" AND "]" ARE ON THE TELETYPE (SHIFT-J AND SHIFT-M);
THEY JUST ARENT INDICATED ON THE KEYS OF MOST TELETYPES.



%CP 50
10.5. CAUSES OF I/O.

   TO EFFECTIVELY USE SNOBOL I/O, IT IS NECESSARY TO KNOW WHAT
CONSTRUCTS CAUSE INPUT OR OUTPUT TO OCCUR.  THESE ARE DESCRIBED
HERE IN DETAIL:

INPUT OCCURS WHEN:
   1) AN INPUT STRING IS USED AS A STRING REFERENCE IN RULES
      OF THE FORMS:

         <STR REF>
         <STR REF>  <PATTERN>
         <STR REF>  <PATTERN>  <REPLACEMENT>

      INPUT OCCURS IMMEDIATELY AFTER THE STRING REFERENCE IS
      EVALUATED.  THUS IN THE RULE:

         CARD  F(B)    /F(L3)

    THE INPUT OCCURS BEFORE F(B) IS CALLED.
 2) AN EXPRESSION USES THE VALUE OF AN INPUT STRING.  INPUT
    OCCURS WHEN THE NAME IS ENCOUNTERED.  THUS IN THE RULE:

       STR F(READ "*",AL("1"))

    THE INPUT OCCURS BEFORE AL("1") IS CALLED.

OUTPUT OCCURS WHEN:
 1) AN OUTPUT STRING IS USED AS A STRING REFERENCE IN RULES
    OF THE FORMS:

       <STR REF>  <REPLACEMENT>
       <STR REF>  <PATTERN>  <REPLACEMENT>

    OUTPUT OCCURS IMMEDIATELY AFTER THE REPLACEMENT IS DONE.
    FOR EXAMPLE, TO OUTPUT A LINE TO THE PRINTER:

       PRINT  =  "CRD = " QUOTE CRD QUOTE

    NOTE THAT IN RULES OF THE FORM <STR REF> <PATTERN> <REPLACEMENT>
    BOTH INPUT AND OUTPUT WILL OCCUR IF THE STRING REFERENCE IS
    BOTH AN INPUT STRING AND AN OUTPUT STRING.
 2) A STRING VARIABLE HAS AN OUTPUT STRING FOR ITS NAME, AND
    THE PATTERN MATCH SUCCEEDS.  OUTPUT OCCURS WHEN THE ASSIGNMENT
    TO THAT VARIABLE IS MADE.
 3) A FUNCTION IS CALLED, AND AN OUTPUT STRING IS EITHER THE
    STRING NAMED THE SAME AS THE FUNCTION, A FORMAL PARAMETER,
    OR A LOCAL VARIABLE.  OUTPUT OCCURS AT BOTH THE CALL AND
    THE RETURN OF THE FUNCTION.



%SECTION 11
%PAGE 1
11. DEBUGGING AIDS.

   THERE ARE CURRENTLY TWO TYPES OF DEBUGGING AIDS PROVIDED IN
B5500 SNOBOL3.  THERE IS A TRACING FACILITY THAT CAN BE USED TO
MAKE A RECORD OF VALUES ASSIGNED TO VARIABLES, CALLS OF FUNCTIONS,
AND TRANSFERS TO LABELS.  THERE IS ALSO AN INTERACTIVE DEBUGGING
TOOL THAT ALLOWS A USER AT A TELETYPE TO SUSPEND A PROGRAM, ASK
QUESTIONS ABOUT ITS STATE, CORRECT ERRORS MADE BY THE PROGRAM,
AND RESUME EXECUTION.



%CP 5
11.1. TRACING.

   TRACING IS PROVIDED IN B5500 SNOBOL FOR STRINGS, FUNCTIONS, AND
LABELS.  OUTPUT FROM TRACING GOES TO THE LINE PRINTER FILE, PRINT.
THE FUNCTIONS THAT CONTROL TRACING ARE:


%CP 21
```

```
TRACES(S1,S2,S3,...)  TURNS ON TRACING FOR THE STRINGS NAMED BY
   S1, S2, S3, ETC.  THIS MEANS THAT WHENEVER A VALUE IS ASSIGNED
   TO ONE OF THESE STRINGS, OUTPUT OCCURS GIVING THE STATEMENT
   NUMBER, THE STRING NAME, AND ITS VALUE, IN THE FORM:

      *S <STMT#> <NAME> = " <VALUE> "

   FOR EXAMPLE, TO TURN ON TRACING FOR ALPHA, XV2, AND QVAR, CALL:

      TRACES("ALPHA","XV2","QVAR")

   IF THE VARIABLE ALPHA WERE ASSIGNED THE VALUE "THIS ONE" IN
   STATMENT 135, THE FOLLOWING WOULD BE PRINTED:

      *S  135 ALPHA = "THIS ONE"

   STRING TRACING IS CONSIDERED AN OUTPUT USE OF THE STRING, AND
   TURNING ON TRACING WILL TERMINATE ANY PREVIOUS I/O USE THE STRING
   MAY HAVE HAD.  TRACING OUTPUT OCCURS AT ALL PLACES WHERE OUTPUT
   OCCURS FOR STRINGS WITH OUTPUT USE--THAT IS, WHEREVER A VALUE
   IS ASSIGNED.


%CP 27
TRACEF(F1,F2,F3,...)  TURNS ON TRACING FOR THE FUNCTION NAMED BY
   F1, F2, F3, ETC.  WHEN ANY OF THESE FUNCTIONS IS CALLED, OUTPUT
   OCCURS GIVING THE FUNCTION NAME, THE STATEMENT NUMBER, AND THE
   VALUES OF THE PARAMETERS, IN THE FORM:

      *C  <STMT#>  <FCT NAME>(  <PARAMS>  )

   WHEN THE FUNCTION RETURNS, MORE OUTPUT OCCURS (STARTING WITH
   "*R" RATHER THAN "*C") GIVING THE RETURN VALUE OR INDICATING
   FAILURE.  FOR EXAMPLE, SUPPOSE A PROGRAM HAD CALLED:

      TRACEF("BLOCK","TIME","FLXGQ")

   THIS WILL START TRACING FOR THE FUNCTIONS BLOCK(), TIME(), AND
   FLXGQ().  SUPPOSE THAT STATEMENT 93 THEN CALLS BLOCK(A,"3"), AND
   A = "ITO"; AND THE VALUE RETURNED IS "X.3".  THEN THE FOLLOWING
   OUTPUT WILL OCCUR:

      *C   93 BLOCK("ITO","3")
      *R   93 BLOCK() = "X.3"

   IF BLOCK HAD FAILED, THE LAST LINE WOULD HAVE BEEN:

      *R   93 BLOCK() FAILED.

   NOTE THAT TRACING CAN BE USED WITH BOTH SYSTEM-DEFINED AND
   PROGRAM-DEFINED FUNCTIONS.


%CP 13
TRACEL(L1,L2,L3,...)  INITIATES TRACING FOR THE LABELS CONTAINED
   IN L1, L2, L3, ETC.  THIS MEANS THAT WHENEVER ANY OF THESE LABELS
   IS ENCOUNTERED, OUTPUT OCCURS GIVING THE LABEL, THE NUMBER OF
```

```
    TIMES IT HAS BEEN EXECUTED (INCLUDING THE CURRENT ONE), AND
    THE NUMBER OF THE PREVIOUS INSTRUCTION.  THUS, IF A PROGRAM
    HAD CALLED:

        TRACEL("BACK","NONE","B1","B2")

    THEN IF STATEMENT 216 TRANSFERS TO BACK, WHICH HAS PREVIOUSLY
    BEEN EXECUTED 17 TIMES, THE FOLLOWING WILL BE PRINTED:

        *L  BACK 18 FROM 216


%CP 17
DETACH(N,T)  CAN BE USED TO TURN OFF TRACING.  THE NAME OF THE
    OBJECT CONCERNED IS CONTAINED IN N; T GIVES THE TYPE AS FOLLOWS:

        "S"  MEANS THAT N CONTAINS A STRING NAME.
        "F"  MEANS THAT N CONTAINS A FUNCTION NAME.
        "L"  MEANS THAT N CONTAINS A LABEL.

    ONLY THE FIRST CHARACTERS OF T IS LOOKED AT; IF T IS NULL, "S"
    IS ASSUMED.  THUS, TO TURN OFF TRACING FOR THE STRING NAMED ALPHA,
    THE PROGRAM COULD CALL ANY OF:

        DETACH("ALPHA")
        DETACH("ALPHA","S")
        DETACH("ALPHA","STRING")
        DETACH("ALPHA","SLORKLE")



%PAGE
11.2. INTERACTIVE DEBUGGING.

    AN INTERACTIVE DEBUGGING AID HAS BEEN IMPLEMENTED FOR USE WITH
A TELETYPE.  BASICALLY, IT CONSISTS OF SUSPENDING A PROGRAM AND
ASKING THE INTERPRETER QUESTIONS ABOUT ITS STATE.  IF ERRORS HAVE
BEEN MADE, VALUES OF STRINGS CAN BE CHANGED, AND THE PROGRAM CAN
BE RE-INITIATED EITHER WHERE IT WAS SUSPENDED OR AT ANY OTHER
LABELED INSTRUCTION.
    A PROGRAM CAN BE SUSPENDED, AND THE DEBUGGING ROUTINE INITIATED,
BY ANY OF THE FOLLOWING METHODS:

    1) THE PROGRAM CAN CALL SUSPEND()--WITH NULL PARAMETER.
    2) IF A -DEBUG CARD OCCURRED DURING COMPILATION, SUSPENSION
       WILL OCCUR JUST BEFORE THE FIRST INSTRUCTION STARTS.
    3) SUSPENSION OCCURS WHEN THE PROGRAM IS RUNNING IN "DEBUG"
       MODE AND THE PROGRAM ENCOUNTERS A LIMIT SET DURING A PREVIOUS
       SUSPENSION.
    4) THE PROGRAM COMMITS ANY OF THE VARIOUS PROGRAM ERRORS THAT
       ARE NORMALLY FATAL.  THESE ARE ERRORS SUCH AS CALLING UNDEFINED
       FUNCTIONS, TRYING TO EXECUTE A STATEMENT WITH A SYNTAX ERROR,
       TRANSFERRING TO AN UNDEFINED LABEL, ETC.

    A PROGRAM RUNS EITHER IN NORMAL ("RUN") MODE OR IN "DEBUG" MODE.
"DEBUG" MODE CAUSES CHECKING FOR A NUMBER OF PROGRAM CONDITIONS
WHOSE OCCURENCE WILL CAUSE THE PROGRAM TO BE SUSPENDED.  FOR A
```

```
LIST OF THE LIMITS THAT CAN BE SET BY THE PROGRAMMER, SEE THE "RUN"
COMMAND BELOW.  A PROGRAM CAN BE IN DEBUG MODE FOR ANY OF THE
FOLLOWING REASONS:

    1) THE PROGRAM CONTAINED A -DEBUG CONTROL CARD.
    2) THE PROGRAM WAS SUSPENDED EARLIER.
    3) THE PROGRAM HAS CALLED MODE("DEBUG").

DEBUG MODE IS TURNED OFF BY CALLING MODE("RUN").  IF THIS IS DONE,
THE ONLY WAY TO GET BACK TO THE DEBUGGING ROUTINE IS FOR THE PROGRAM
TO CALL SUSPEND() OR MODE("DEBUG"), SINCE MODE("RUN") TURNS OFF ALL
THE LIMIT CHECKING ASSOCIATED WITH DEBUG MODE.
   WHEN A PROGRAM IS SUSPENDED, THE SYSTEM TYPES OUT "SUSPENDED"
AND IS THEN WAITING FOR A COMMAND.  THE COMMANDS THAT HAVE BEEN
IMPLEMENTED AS OF THIS WRITING ARE:


%CP 23
TYPE <NAME>←
   THIS CAUSES THE VALUE OF THE VARIABLE <NAME> TO BE TYPED IN
   THE FOLLOWING FORM:

      <NAME> = " <VALUE> "

   IF THE PROGRAM DOESNT CONTAIN <NAME>, AND IT HASNT BEEN CREATED
   BY INDIRECTION, THE RESPONSE IS:

      <NAME> NOT IN USE

   IN ORDER TO ALLOW THE USER TO USE ANY STRING NAME THAT CAN
   BE ENTERED FROM A TELETYPE, THE FOLLOWING CONVENTION IS MADE:
   THERE IS EXACTLY ONE BLANK AFTER "TYPE"; THE REST OF THE INPUT
   STRING IS THE <NAME>.  THUS ANY CHARACTER READABLE FROM A
   TELETYPE CAN BE USED IN <NAME>, INCLUDING BLANKS.  FOR EXAMPLE,
   THE COMMAND:

      TYPE *AA/3 B1←

   MIGHT GET THE RESPONSE:

      *AA/3 B1 = "437 A+B 4"



%CP 19
SET <NAME>←
<VALUE>←
   THIS COMMAND, REQUIRING TWO INPUT STRINGS, ASSIGNS THE STRING
   <VALUE> TO <NAME>.  THE FIRST INPUT, SET <NAME>←, PRODUCES
   A RETURN/LINE FEED FROM SNOBOL, AND IT IS THEN WAITING FOR THE
   <VALUE>.  THE CONVENTIONS FOR <NAME> ARE THE SAME AS FOR THE
   TYPE COMMAND.  FOR EXAMPLE, THE THREE COMMANDS:

      SET AB3←
      N.47.-27.C1←
      TYPE AB3←

   WILL GET THE RESPONSE:

      AB3 = "N.47.-27.C1"

   BECAUSE TRANSMISSION ERRORS DO OCCUR FAIRLY OFTEN WITH TELETYPES,
   IT IS ADVISABLE TO FOLLOW SET COMMANDS WITH TYPE COMMANDS TO
   VERIFY THAT THE CORRECT VALUE HAS BEEN ASSIGNED.



%CP 9
LABEL <LABEL>←
   THIS COMMAND SIMPLY TELLS THE USER HOW MANY TIMES THE LABEL HAS
   BEEN EXECUTED SO FAR.  IF THE LABEL ISNT DEFINED, THE RESPONSE
   WILL BE:

      <LABEL> NOT DEFINED

   LIKE WITH THE TYPE AND SET COMMANDS, THERE IS EXACTLY ONE BLANK
   BETWEEN "LABEL" AND <LABEL>.



%CP 41
RUN <INTEGER> FROM <LABEL> TO <LABEL> <LABEL> <LABEL> ...←
   THIS COMPLICATED COMMAND RE-INITIATES EXECUTION, AND ALSO SETS
   UP THE CONDITIONS THAT ARE TO CAUSE SUSPENSION LATER.  ALL OF
   THE COMMAND AFTER "RUN" IS OPTIONAL, AND BREAKS DOWN INTO SEVERAL
   SECTIONS, THAT ARE REALLY PARAMETERS.
      IF <INTEGER> IS USED, THEN THE PROGRAM WILL BE SUSPENDED
   AFTER IT EXECUTES <INTEGER> INSTRUCTIONS, IF IT MANAGES TO
   GET THAT FAR AT ALL.
      IF FROM <LABEL> IS USED, THE CURRENT INSTRUCTION WILL BE
   TERMINATED (AS IF IT HAD FAILED), AND EXECUTION WILL RESUME AT
   THE STATEMENT LABELED <LABEL>.  IF <LABEL> ISNT DEFINED, THE
   RESPONSE WILL BE:

      <LABEL> NOT DEFINED

   AND THE PROGRAM WILL REMAIN SUSPENDED.
      IF THE TO <LABEL> <LABEL> ... PART IS INCLUDED, IT IS
   INTERPRETED AS A LIST OF LABELS THAT ARE TO CAUSE SUSPENSION IF
   THE PROGRAM TRIES TO EXECUTE THEM.  NOTE THAT THE LABELS ARE
   SEPARATED BY BLANKS--ANY OTHER DELIMITERS (SUCH AS COMMAS) WILL
   BE INTERPRETED AS PART OF A LABEL.  IF A LABEL IS USED THAT
   ISNT DEFINED, THE "NOT DEFINED" RESPONSE WILL BE GIVEN, AND
   THE PROGRAM WILL REMAIN SUSPENDED.  THERE IS A LIMIT TO THE
   NUMBER OF LABELS THAT CAN BE GIVEN.  THIS LIMIT IS CURRENTLY
   SET TO 16, THOUGH IT MAY BE INCREASED IN THE FUTURE IF THERE
   IS DEMAND FOR MORE.  IF TOO MANY LABELS ARE GIVEN, THE RESPONSE
   WILL BE "TOO MANY LABELS", AND THE PROGRAM WILL REMAIN SUSPENDED.
   AS A SPECIAL CASE, IF "TO" IS USED WITHOUT ANY LABELS FOLLOWING
   IT, THE LIMITS WILL BE THE SAME AS THOSE USED THE LAST TIME (IF
   ANY).
      SOME EXAMPLES OF RUN COMMANDS ARE:

      RUN←
      RUN 75←
      RUN FROM HNT3 TO HNT5 HNT8←
      RUN 40 FROM GIN,7←
      RUN 500 FROM A3 TO B1 AA1 APPLY3 ERROR EOJ ALTRY←
      RUN FROM FORR1 TO←
      RUN 1000 TO←
      RUN 1←
      RUN FROM START←



%CP 2
END←
   THIS CAUSES THE PROGRAM TO TERMINATE NORMALLY.



%CP 2
ABORT←
   THIS CAUSES ABNORMAL TERMINATION OF THE PROGRAM.



%CP 7
WHY←
   THIS CAUSES THE REASON(S) FOR SUSPENSION TO BE TYPED. SOME
   POSSIBLE RESPONSES TO THIS ARE:

      RULE LIMIT HIT
      LABEL LIMIT HIT
      SUSPEND() CALLED
      FATAL ERROR



%CP 4
WHERE←
   THIS CAUSES THE SYSTEM TO TYPE OUT WHERE THE PROGRAM WAS SUSPENDED
   (STATEMENT NUMBER AND LAST LABEL) AND HOW MANY RULES HAVE BEEN
   EXECUTED SO FAR.



%SECTION 12
%PAGE 1
12. MISCELLANEOUS.

%CP 6
THE FOLLOWING CHARACTERS HAVE SPECIAL MEANING IN COLUMN 1:
  "?"  MCP CONTROL CARD
  "-"  SNOBOL CONTROL CARD
  "."  CONTINUATION CARD (INVALID FROM TELETYPE)
  "*"  COMMENT CARD
  "%"  COMMENT CARD



```

```
%CP 3
A LABEL IS ANY NON-NULL STRING PRECEDING THE FIRST BLANK ON A CARD.
THE FIRST CHARACTER CAN BE ANY EXCEPT THOSE LISTED ABOVE; THE REST
OF THE LABEL CAN USE ANY CHARACTERS BUT BLANKS.



%CP 4
WHEN ENTERING A PROGRAM FROM A TELETYPE, ANY INPUT ENDING WITH
"#" WILL BE TREATED AS AN ERROR--THE COMPILER WILL DISCARD IT
TYPE "TRY AGAIN", AND ASK FOR MORE INPUT.



%CP 5
THE GO-TO PART CAN BE SET OFF BY EITHER " /" OR ":".  IF " /" IS
USED (NOTE THE BLANK), THE NEXT CHARACTER MUST BE EITHER "(", "S",
OR "F".  A COLON (":") CAN BE SURROUNDED BY ANY NUMBER OF BLANKS,
INCLUDING NONE, AND CAN STAND ALONE WITH GO GO-TO PART FOLLOWING
IT, IN WHICH CASE IT IS IGNORED.



%CP 2
TO SEPARATE A PATTERN OR STRING REFERENCE FROM A REPLACEMENT, EITHER
"=" OR "←" CAN BE USED.  THE COMPILER THINKS THEY ARE EQUIVALENT.



%CP 7
THE "ILLEGAL CHARACTER","?", CANT BE READ FROM A CARD READER EXCEPT
IN COLUMN 1, IN WHICH CASE IT INDICATES AN MCP CONTROL CARD.  ANY
CARD CONTAINING "?" ANYWHERE ELSE WILL BE THROWN OUT BY THE CARD
READER, AND THE JOB WILL USUALLY NOT BE RUN.  THIS CHARACTER CANT
BE INPUT FROM A TELETYPE, EITHER, SINCE ANY INPUT CONTAINING "?"
WILL BE INTERPRETED AS A MESSAGE TO THE MCP.  THERE IS NO PROBLEM
WITH WRITING A "?" ON THE LINE PRINTER OR ON A TELETYPE.



%CP 9
THERE ARE THREE STRINGS WHICH B5500 SNOBOL3 INITIALIZES TO HAVE
NON-NULL VALUES. THEY ARE:

   QUOTE CONTAINS A QUOTE
   QMARK = "?"
   ARROW = "←"

THE STRINGS QMARK AND ARROW ARE PROVIDED MAINLY FOR TELETYPE USERS;
SINCE NEITHER CAN BE INPUT FROM A TELETYPE.



%CP 2
THE SYSTEM LIMIT ON STRING SIZE IS 8181 CHARACTERS.  ATTEMPTING TO
CREATE A STRING LONGER THAN THIS WILL CAUSE ABNORMAL TERMINATION.



%CP 9
THE CONCATENATE OPERATOR IS A BLANK.  BLANKS ARE USUALLY OPTIONAL,
WITH THE FOLLOWING EXCEPTIONS:

   1) BEFORE THE STRING REFERENCE.
   2) BETWEEN TWO IDENTIFIERS.
   3) BETWEEN AN IDENTIFIER AND A FOLLOWING "(", TO DISTINGUISH
      A CONCATENATION OF A STRING AND A GROUPING FROM A FUNCTION
      CALL (WHICH HAS NO BLANK).
   4) BEFORE A "/" USED AS A GO-TO DELIMITER.



%CP 4
ANY STRING CAN BE USED AS A STRING NAME, INCLUDING THE NULL STRING.
IN PARTICULAR, THE FOLLOWING IS A VALID EXPRESSION:

      $""



%CP 5
IF A GO-TO PART HAS BOTH A SUCCESS AND A FAILURE EXIT, THEN THE "S"
OR "F" CAN BE OMITTED FROM THE SECOND PART. THUS:

   /S(A)(B)     MEANS /S(A)F(B)
   :F(XA)(X7)   MEANS :F(XA)S(X7)



%CP 11
SEVERAL STATEMENTS CAN BE PUT ON ONE CARD.  THEY ARE SEPARATED BY
SEMICOLONS (";").  CONTINUATION CARDS CAN BE USED AS USUAL.  OF
COURSE, SINCE A LABEL MUST START IN COLUMN 1, ONLY THE FIRST STATEMENT
ON A CARD CAN BE LABELED.  FOR EXAMPLE,

   L  A ".." = "." /S(L); B = A "," B /(LOOP)

MEANS THE SAME AS:

   L  A ".." = "." /S(L)
      B  = A "," B /(LOOP)



%CP 15
NO INTERNAL DISTINCTION IS MADE BETWEEN FORMAL PARAMETERS AND LOCAL
VARIABLES OF PROGRAM-DEFINED FUNCTIONS.  IF A FUNCTION IS CALLED WITH
MORE ACTUAL PARAMETERS THAN THERE ARE FORMAL PARAMETERS, THE EXTRA
PARAMETERS WILL BE ASSIGNED TO LOCAL VARIABLES.  FOR EXAMPLE, IF F()
IS DEFINED BY:

   DEFINE("F(A,B)","F1","C,D")

THEN IF THE PROGRAM CALLS F("+","-","X"), THE VALUES AT THE ENTRY
```

```
POINT, F1, WILL BE:

   A = "+"
   B = "-"
   C = "x"
   D = ""



%APPENDIX A
%PAGE 1
APPENDIX A. BNF DEFINITIONS FOR SNOBOL3.

   <INSTRUCTION>    ::=  <LABELED INST>
                    ::=  <UNLABELED INST>
   <LABELED INST>   ::=  <LABEL> <UNLABELED INST>
   <UNLABELED INST> ::=  <BLANKS> <INST>
   <INST>           ::=  <RULE>
                    ::=  <GO-TO PART>
                    ::=  <RULE> <GO-TO PART>
   <RULE>           ::=  <STR REF>
                    ::=  <STR REF> <PATTERN>
                    ::=  <STR REF> <REPLACEMENT>
                    ::=  <STR REF> <PATTERN> <REPLACEMENT>
   <STR REF>        ::=  <ELEMENT>
   <PATTERN>        ::=  <ELEMENT>
                    ::=  <STR VAR>
                    ::=  <PATTERN> <PATTERN>
                    ::=  <PATTERN> <BACK REF>
   <REPLACEMENT>    ::=  <RPL DELIMITER> <EXPRESSION>
   <RPL DELIMITER>  ::=  ←
                    ::=  =
   <EXPRESSION>     ::=  <ELEMENT>
                    ::=  <EXPRESSION> <CONCATENATE OP> <ELEMENT>
                    ::=  <EMPTY>
   <ELEMENT>        ::=  <NAMED EXPR>
                    ::=  <VALUE EXPR>
   <NAMED EXPR>     ::=  <IDENTIFIER>
                    ::=  $  <ELEMENT>
   <VALUE EXPR>     ::=  <LITERAL>
                    ::=  <GROUPING>
                    ::=  <FCT CALL>
                    ::=  <ARITH EXPR>
   <CONCATENATE OP> ::=   <BLANKS>
   <IDENTIFIER>     ::=  <LETTER>
                    ::=  <IDENTIFIER><LETTER>
                    ::=  <DIGIT>
                    ::=  <IDENTIFIER><DIGIT>
                    ::=  .
                    ::=  <IDENTIFIER>  .
   <LITERAL>        ::=  "  <STRING>  "
   <GROUPING>       ::=  (  <EXPRESSION>  )
   <FCT CALL>       ::=  <FCT NAME>(  <PARAMS>  )
   <FCT NAME>       ::=  <IDENTIFIER>
   <PARAMS>         ::=  <EXPRESSION>
                    ::=  <PARAMS>  ,  <EXPRESSION>
   <ARITH EXPR>     ::=  <A-TERM>
```

```
                    ::=  -  <A-TERM>
   <A-TERM>         ::=  <A-TERM>  <ADD AP>  <M-TERM>
                    ::=  <M-TERM>
   <M-TERM>         ::=  <M-TERM>  <MUL OP>  <E-TERM>
                    ::=  <E-TERM>
   <E-TERM>         ::=  <E-TERM>  <EXP OP>  <ARITH ELT>
                    ::=  <ARITH ELT>
   <ARITH ELT>      ::=  <NAMED EXPR>
                    ::=  <GROUPING>
                    ::=  <FCT CALL>
                    ::=  <LITERAL>
   <ADD OP>         ::=  +
                    ::=  -
   <MUL OP>         ::=  *
                    ::=  x
                    ::=  /
   <EXP OP>         ::=  **
   <PATTERN>        ::=  <ELEMENT>
                    ::=  <STR VAR>
                    ::=  <PATTERN> <PATTERN>
                    ::=  <PATTERN> <BACK REF>
   <STR VAR>        ::=  <ARB VAR>
                    ::=  <BAL VAR>
                    ::=  <FIXED-LENGTH VAR>
   <ARB VAR>        ::=  **
                    ::=  *  <NAMED EXPR>  *
   <BAL VAR>        ::=  *()*
                    ::=  *(  <NAMED EXPR>  )*
   <FIXED-LENGTH VAR> ::=  */  <EXPRESSION>  *
                      ::=  *  <NAMED EXPR>  /  <EXPRESSION>  *
   <BACK REF>         ::=  <NAMED EXPR>
   <GO-TO PART>     ::=  <GO-TO DELIMITER>  <GO-TOS>
   <GO-TO DELIMITER> ::=  <BLANKS>  /
                    ::=  :
   <GO-TOS>         ::=  <UNCONDITIONAL GO-TO>
                    ::=  <SUCCESS GO-TO>
                    ::=  <FAILURE GO-TO>
                    ::=  <SUCCESS GO-TO>  <FAILURE GO-TO>
                    ::=  <FAILURE GO-TO>  <SUCCESS GO-TO>
   <UNCONDITIONAL GO-TO> ::=  (  <LABEL EXPR>  )
   <SUCCESS GO-TO> ::=  S(  <LABEL EXPR>  )
   <FAILURE GO-TO> ::=  F(  <LABEL EXPR>  )
   <LABEL EXPR>     ::=  <RESERVED LABEL>
                    ::=  <LABEL>
                    ::=  <COMPUTED LABEL>
   <RESERVED LABEL> ::=  END
                    ::=  RETURN
                    ::=  FRETURN



%APPENDIX B
%PAGE 1
APPENDIX B. BNF NOTATION.

   FOR THOSE NOT FAMILIAR WITH "BNF" (BACKUS-NAUR FORM OR
BACKUS NORMAL FORM), A BRIEF DESCRIPTION FOLLOWS.  BNF IS A
```

SOMEWHAT STANDARDIZED METHOD OF PRECISELY NOTATING SYNTACTIC DEFINITIONS OF CERTAIN TYPES.

QUANTITIES WHICH MUST APPEAR AS A CERTAIN STRING OF CHARACTERS ARE REPRESENTED BY THAT STRING OF CHARACTERS. VARIABLE STRINGS ARE GIVEN A NAME, WHICH IS SURROUNDED BY "BNF BRACKETS", "<" AND ">". FOR EXAMPLE, THE MCP CONTROL CARD:

```
      ?DATA <NAME>
```

MUST CONTAIN THE LITERAL STRING "?DATA" FOLLOWED BY A <NAME>, WHICH MUST BE DEFINED ELSEWHERE. SOME EXAMPLES OF CARDS THAT SATISFY THIS FORM ARE:

```
      ?DATA ALPHA
      ?DATA XXZZYY
      ?DATA DATA
      ?DATA AB13X47
```

A VARIABLE ITEM MAY BE DEFINED BY USING A "BNF EQUATION" TO DEFINE IT IN TERMS OF OTHER ITEMS. A DEFINITION IS OF THE FORM:

```
      <NAME>  ::=  <DEFINITION>
```

FOR EXAMPLE, A FUNCTION CALL MAY BE DEFINED BY:

```
      <FCT CALL>  ::=  <FCT NAME>(  <PARAMS>  )
```

WHERE <FCT NAME> AND <PARAMS> ARE DEFINED BY OTHER EQUATIONS. IF THERE ARE SEVERAL ALTERNATE DEFINITIONS OF AN OBJECT, SEVERAL EQUATIONS ARE GIVEN. IF A VARIABLE HAS SEVERAL DEFINITIONS, THEN THAT VARIABLE HAS ALTERNATE DEFINITIONS, ANY OF WHICH CAN APPLY IN A GIVEN INSTANCE. THIS WAY, RECURSIVE DEFINITIONS ARE SIMPLE (AND QUITE COMMON). FOR EXAMPLE:

```
      <PARAMS>  ::=  <EXPRESSION>
      <PARAMS>  ::=  <PARAMS>  ,  <EXPRESSION>
```

THIS MEANS THAT <PARAMS> CAN BE A SINGLE EXPRESSION, OR IT CAN BE ANY NUMBER OF EXPRESSIONS SEPARATED BY COMMAS. WHEN SEVERAL CONSECUTIVE DEFINITIONS ARE GIVEN FOR THE SAME VARIABLE, IT IS RATHER CUSTOMARY (TO MAKE READING EASIER) TO OMIT THE LEFT-HAND SIDE OF THE EQUATION IN ALL BUT THE FIRST, SO THE ABOVE DEFINITION COULD HAVE BEEN:

```
      <PARAMS>  ::=  <EXPRESSION>
                ::=  <PARAMS>  ,  <EXPRESSION>
```

FOR LOTS OF EXAMPLES OF BNF DEFINITIONS, SEE THE PREVIOUS APPENDIX. NOTE THAT A FEW ITEMS ARENT DEFINED--IN PARTICULAR, <BLANKS> AND <EMPTY> ARE CONSIDERED TOO OBVIOUS TO BOTHER DEFINING. IT IS RATHER COMMON FOR SOME TERMS TO BE DEFINED BY ENGLISH-LANGUAGE DEFINITIONS, WHEN BNF IS UNSUITED FOR THE JOB. THIS IS DONE FOR A FEW SYMBOLS IN THE SNOBOL3 DEFINITIONS--SEE THE APPROPRIATE SECTION OF THE MANUAL FOR THEIR DEFINITIONS.

```
3700000 %APPENDIX C
3701000 %PAGE 1
3702000 APPENDIX C. B5500 CHARACTER SET.
3703000
3704000 CHAR   OCT  DEC  PUNCH    COMMENTS
3705000  0      0    0   0
3706000  1      1    1   1
3707000  2      2    2   2
3708000  3      3    3   3
3709000  4      4    4   4
3710000  5      5    5   5
3711000  6      6    6   6
3712000  7      7    7   7
3713000  8     10    8   8
3714000  9     11    9   9
3715000  #     12   10   3-8
3716000  @     13   11   4-8
3717000  ?     14   12   2-8      OR ANY ILLEGAL PUNCH
3718000  :     15   13   5-8
3719000  >     16   14   6-8
3720000  ≥     17   15   7-8      TELETYPE: DISCONNECT
3721000  +     20   16   12-2-8
3722000  A     21   17   12-1
3723000  B     22   18   12-2
3724000  C     23   19   12-3
3725000  D     24   20   12-4
3726000  E     25   21   12-5
3727000  F     26   22   12-6
3728000  G     27   23   12-7
3729000  H     30   24   12-8
3730000  I     31   25   12-9
3731000  .     32   26   12-3-8
3732000  [     33   28   12-4-8
3733000  &     34   29   12
3734000  (     35   30   12-5-8
3735000  <     36   30   12-6-8
3736000  ←     37   31   12-7-8    TELETYPE: END-OF-MESSAGE
3737000  ×     40   32   11-2-8    TELETYPE: BACKWARD SLASH
3738000  J     41   33   11-1
3739000  K     42   34   11-2
3740000  L     43   35   11-3
3741000  M     44   36   11-4
3742000  N     45   37   11-5
3743000  O     46   38   11-6
3744000  P     47   39   11-7
3745000  Q     50   40   11-8
3746000  R     51   41   11-9
3747000  $     52   42   11-3-8
3748000  *     53   43   11-4-8
3749000  -     54   44   11
3750000  )     55   45   11-5-8
3751000  ;     56   46   11-6-8
3752000  ≤     57   47   11-7-8    TELETYPE: APOSTROPHE, CARRIAGE RETURN
3753000 BLANK  60   48   NONE
3754000  /     61   49   0-1
3755000  S     62   50   0-2
3756000  T     63   51   0-3
3757000   U    64    52    0-4
3758000   V    65    53    0-5
3759000   W    66    54    0-6
3760000   X    67    55    0-7
3761000   Y    70    56    0-8
3762000   Z    71    57    0-9
3763000   ,    72    58    0-3-8
3764000   %    73    59    0-4-8
3765000   ≠    74    60    0-2-8    TELETYPE: UP-ARROW, LINE FEED
3766000   =    75    61    0-5-8
3767000   ]    76    62    0-6-8
3768000   "    77    63    0-7-8
3769000
3770000 %APPENDIX D
3771000 %PAGE 1
3772000 APPENDIX D. RUNNING JOBS FROM A TELETYPE.
3773000
3774000    THIS APPENDIX IS INTENDED TO DESCRIBE THE SOMEWHAT INTRICATE
3775000 METHODS OF RUNNING A PROGRAM (IN PARTICULAR, A SNOBOL PROGRAM) FROM
3776000 A TELETYPE, AND HOPEFULLY TO SOMEWHAT AMELIORATE THE INITIAL SHOCK
3777000 OF TRYING TO USE THIS RATHER PRIMITIVE AND FRUSTRATING INTERACTIVE
3778000 I/O DEVICE.  THIS DESCRIPTION IS FAR FROM COMPLETE, AND DUE TO THE
3779000 CONSTANT REVISION (AND OCCASIONALLY IMPROVEMENT) OF THE MCP, DETAILS
3780000 CAN BE EXPECTED TO CHANGE AT ANY TIME, USUALLY WITHOUT WARNING.  THE
3781000 NOVICE IS WARNED THAT THE MCP WAS DESIGNED FOR THE USE OF PEOPLE WHO
3782000 ARE VERY FAMILIAR WITH IT, AND DOES VERY LITTLE TO HELP THOSE WHO DONT
3783000 KNOW HOW IT BEHAVES.
3784000    THE FIRST THING NECESSARY IS TO "LOG IN", THAT IS, TO TELL THE MCP
3785000 WHO YOU ARE.  THIS IS DONE BY TYPING:
3786000
3787000       ?LI <I.D.>←
3788000
3789000 AT THE UNIVERSITY OF WISCONSIN, <I.D.> IS THE USER AND PROJECT
3790000 NUMBERS, IN THE FORM:
3791000
3792000       <I.D.>  =  <USER#> / <PROJ#>
3793000
3794000 THE MCP WILL RESPOND TO THE LOG-IN MESSAGE WITH SOME SORT OF ACKNOWLED-
3795000 GEMENT, AND THE USER CAN THEN INDICATE WHAT PROGRAM HE WANTS TO RUN
3796000 BY TYPING A ??EXECUTE CONTROL CARD.  THIS MESSAGE CONTAINS THE SAME
3797000 INFORMATION AS THE MCP CONTROL CARDS DESCRIBED IN SECTION 2.1, WITH
3798000 THE FOLLOWING CHANGES:
3799000
3800000 1) ALL CONTROL CARDS MUST BE SENT TOGETHER AS ONE MESSAGE.  THIS
3801000    MESSAGE CAN BE LONGER THAN ONE LINE, SINCE THE RETURN AND LINE-FEED
3802000    KEYS ON THE TELETYPE DO NOT TRANSMIT ANYTHING.
3803000 2) THE FIRST CONTROL CARD IS THE EXECUTE CARD, WHICH IS PRECEDED
3804000    BY TWO "?"-S.  ALL OTHER CONTROL CARDS ARE PRECEDED BY ";"
3805000    RATHER THAN "?".  THE USER CARD IS NOT USED, SINCE THAT INFORMA-
3806000    TION WAS GIVEN IN THE LOG-IN MESSAGE.
3807000 3) THE FILE PROGRAM NEED NOT BE ASSIGNED, SINCE THE TELETYPE IS THE
3808000    INPUT DEVICE FOR THE COMPILER.
3809000
3810000 %CP 9
3811000 SOME EXAMPLES OF EXECUTE MESSAGES ARE:
3812000
3813000       ??EXECUTE SNOBOL/SNOBOL←
```

```
     ??EXECUTE SNOBOL/SNOBOL;PROCESS=5;IO=10;
     FILE CARD=BLEEK/GRUNK SERIAL←

     ??EXECUTE SNOBOL/SNOBOL;CORE=19000;PROCESS=20;IO=30←
```

%CP 21

WHEN THE EXECUTE MESSAGE IS TYPED, THE MCP WILL RESPOND IN ONE OF TWO WAYS. IF THERE IS ROOM FOR THE PROGRAM, THE RESPONSE WILL BE:

```
     I:SNOBOL/SNOBOL=J  BOJ  <TIME>
```

IF THERE IS NOT ENOUGH ROOM, WHICH IS MUCH MORE LIKELY, THE RESPONSE WILL BE OF THE FORM:

```
     I:SNOBOL/SNOBOL=J  SCHEDULED  <TIME>
```

THE USER MUST THEN FIND SOMETHING ELSE TO OCCUPY HIS TIME UNTIL SPACE BECOMES AVAILABLE AND THE "BOJ" MESSAGE IS TYPED BY THE MCP. USERS ARE ADVISED TO BRING SOMETHING TO READ WHEN ATTEMPTING TO RUN DURING BUSY PARTS OF THE DAY.

WHEN THE BOJ MESSAGE APPEARS, THE COMPILER IS RUNNING AND WAITING FOR INPUT. THE USER CAN THEORETICALLY TYPE ANYTHING THAT IS VALID PROGRAM MATERIAL EXCEPT FOR CONTINUATION CARDS. IN PRACTICE, USERS ARE ADVISED TO FIRST GET THEIR PROGRAM IN A DISK FILE (THERE ARE SEVERAL EDITING PROGRAMS AROUND, AS WELL AS A PROGRAM TO COPY CARD DECKS TO DISK FILES), AND NOT ATTEMPT TO TYPE THE PROGRAM ITSELF DIRECTLY TO THE COMPILER.

%CP 21

WHEN A PROGRAM "CARD" IS TYPED, THE COMPILER WILL COMPILE IT AND THEN RESPOND WITH A RETURN/LINE FEED, AND THE USER CAN THEN TYPE THE NEXT LINE. ONE EXCEPTION TO THIS IS THAT WHEN A -COMPILE CONTROL CARD IS TYPED, THE COMPILER TYPES OUT A NUMBER EVERY TENTH INSTRUCTION AS IT COMPILES, AND AT THE END TYPES A MESSAGE SAYING IT IS DONE. THIS WAY THE USER CAN TELL HOW FAST THE COMPILER IS RUNNING, AND IF IT IS VERY SLOW, HE CAN DO SOMETHING ELSE (LIKE GET A CUP OF COFFEE) WHILE HE WAITS FOR IT TO FINISH.

USUALLY, THE USER WILL TYPE IN A SERIES OF SNOBOL CONTROL CARDS THAT WILL CAUSE THE PROGRAM TO BE LOADED OR COMPILED FROM ONE OR MORE DISK FILES, AND THEN INITIATE EXECUTION BY TYPING AN END CARD. AS SOON AS THE END CARD IS TYPED, THE COMPILER TRANSFERS TO THE INTERPRETER, AND THE PROGRAM IS RUNNING. SEE SECTION 10.4 FOR METHODS OF DOING I/O WITH A TELETYPE.

IF A MISTAKE IS MADE AND DISCOVERED BEFORE IT IS TRANSMITTED TO THE COMPILER, THE WHOLE THING CAN BE ERASED AND RETYPED BY TYPING "#←". THE COMPILER WILL RESPOND WITH "TRY AGAIN", AND THE CARD CAN THEN BE TYPED OVER. THIS USUALLY WORKS BETTER THAN TRYING TO USE THE TELETYPES RATHER AWKWARD (AND NOT ALWAYS SUCCESSFUL) METHODS FOR ERASING MISTAKES.

%CP 25

SOME EXAMPLES OF INPUTS TO THE COMPILER ARE:

```
-LIST←
-LOAD MRG/GROUP←
-COMPILE SLIB/DIFF←
-LIMIT RULES TO 5000←
END INIT←



-LIST←
-SIZE 370←
-COMPILE MARK/SINT←
-LIST 5←
-COMPILE MARK/INT1←
-COMPILE MARK/DIF3←
-DEBUG←
END←



-LIST 2←
-COMPILE GEO/TRAJ3←
-LIBRARY GEO/TRJ3←
END GO←



-LIST 1←
-COMPILE ML/TRY←
-DEBUG←
INIT TRACES("N","J","AL1","PN")←
      TRACEF("HINT","REF")          :(BEGIN)←
END INIT←




%SECTION "INDEX."
%PAGE 1
INDEX.

THIS IS RATHER INCOMPLETE; USERS ARE CORDIALLY INVITED TO SUBMIT
THEIR SUGGESTIONS AS TO WHAT SHOULD BE ADDED.


ARITHMETIC...5.4
BACK REFERENCING...6.3
CONTROL CARDS...2
    MCP...2.1
       FROM TELETYPE...APP.D
    SNOBOL...2.2
DEBUGGING AIDS...11
    INTERACTIVE...11.2
    TRACING...11.1
ENTRY POINTS
    FUNCTIONS...2.2(-DEFINE),9(DEFINE),3.2.1
    PROGRAM...3.2,3.0
FAILURE...8
```

```
LABEL   000000000LPAPRNT00176177?USER=SPO    ; EXECUTE COPY   /SPO                    COPY   /SPO
```