KENT ON-LINE SYSTEM

Document: KUSE/SCAN/2

SCAN - a sub-system for document searching

P.J. Brown
Computing Laboratory
University of Kent at Canterbury
May 1972

Table of contents

Chapter 1 Introduction

The purpose of SCAN is to provide a relatively simple means for the layman to search and to analyze information, and thus find out for himself what computers can and cannot do in this area. Its aims are to be easy to understand (for a computer language) and to be easy to use for simple applications such as might be set as problems for students. It is hoped that SCAN will also be usable for some more sophicticated applications that may arise in research work. SCAN is, however, unlikely to be adequate for really sophisticated text analysis and information retrieval work and workers in these areas would do better to use a powerful list processing language (e.g. L6) or a string manipulation language (e.g. SNOBOL) or to explore the use of other text processing languages (e.g. PROTEXT).

High-sounding aims always tend to be compromised by practical realities, and this is certainly true of SCAN. For SCAN to be usable in an on-line environment the program controlling it must be small and must use limited space. This has meant that certain desirable features have been curtailed or simplified, and the notation for writing SCAN programs has been made very concise and terse.

This document contains a complete description of SCAN, as implemented under the Kent On-line System (KOS). SCAN has also been implemented on other computers, and these implementations have separate, though similar, User Manuals. A shorter, less detailed, description of SCAN can be found in the March 1972 issue of Computers and the Humanities.

1.1 The source document

The document that SCAN is to examine is called the source document. The source document may contain any sort of character information, for example a sonnet, a computer program or a list of names and addresses.

SCAN divides the source document into sentences. The user himself decides what constitutes the end of a sentence; for a piece of prose it would be a full stop but for a computer program it would be the end of a line. A sentence need not necessarily correspond to a sentence of English grammar. It is usually best to analyze poetry line by line, so that if the source document is a piece of poetry, a sentence should probably be defined as a line.

Each sentence is in turn split up into words and separators (i.e. commas, spaces, etc.).

The action of SCAN is to pass through the source document sentence by sentence from beginning to end. The user of SCAN specifies certain words that are to be specially recognized, and the action to be taken in each case. Typical actions might be to update a count, to print the sentence or to check if another word has also occurred. The specification of the words to be recognized and the action to be taken at each is written as a series of SCAN statements, which, taken as a whole, make up a SCAN program.

As well as words, SCAN can be made to recognize separators and take action on these.

In many cases SCAN statements have been made similar to statements in the BASIC programming language. There are, however, considerable differences between the two because BASIC generally deals with rational numbers whereas SCAN deals with characters, words and sentences. When learning about SCAN it is certainly a help if one knows BASIC but this is not essential.

1.2 Introductory examples

A full specification of SCAN is given in subsequent Chapters. However the reader may like to look at some examples first, in order to get an overall impression of the workings of SCAN.

Example 1

This program counts the number of occurrences of the word "it" in the source document, and, at the end, prints the total. Note that, unlike BASIC programs, SCAN programs do not need an END statement to terminate them.

```
10   AT     "IT"

20   LET    N1 = N1 + 1


50   AT     .END.; THIS MEANS AT END OF SOURCE

60   PRINT  N1, "IS THE COUNT OF 'IT' "
```

An occurrence of a semicolon terminates a statement and the text beyond it is a comment for the benefit of the reader. This is illustrated by statement 50.

Example 2

This program prints all sentences that contain words longer than seven letters that end in "-ing". It illustrates the use of some of the "system variables" for communicating with SCAN: the variable .LENGTH. is the length of the current word and the variable .PRINTOPTION. controls the printing of the current sentence - a value of one means print it.

```
10   AT    "-ING"

20   IF    .LENGTH.>7 LET    .PRINTOPTION.=1
```

Example 3

This is a more complicated example. It searches for all sentences that contain "if" followed subsequently by "then" and prints the text in between. SCAN stores the current sentence in an area called B and, on each word, it sets the variables .INITIAL. and .FINAL. to give the position of the beginning and end of the word.

```
10   AT    "IF"

20   LET    M2 = 1  ;  M2  IS ONE OF THE VARIABLES THAT IS ZEROIZED
                                AT THE START OF EACH SENTENCE

30   LET    M3 = .INITIAL. ; M3  POINTS  AT  START  OF  'IF'

50   AT    "THEN"

60   IF     M2>0  PRINT B(M3) TO B(.FINAL.); PRINT FROM 'IF' TO
                                                        'THEN'
```

1.3 KOS commands for using SCAN

The SCAN processor is entered by means of the command

&ENTER SCAN DR-spec

(DR-specs are only needed when disc files and other devices are to be used; for the ordinary console user all DR-specs can be omitted and the command &ENTER SCAN is all that need be typed.)

SCAN has the supplementary commands RUN, PROG and TEXT. These are identical to those found in BASIC, but for those readers unfamiliar with BASIC they are explained below. All the commands can be followed by DR-specs.

The RUN command is used to run the current program. The DR-spec specifies the source document and where the results are to go.

The TEXT command outputs the current program on the results device.

The PROG command introduces a series of corrections or additions to the program. Program lines are always taken from the data device.

SCAN uses the same editing system as BASIC. Each statement is preceded by an integer called the statement number. If a new statement has the same statement number as an existing one then the new one overwrites the existing one. If a statement number occurs on its own with no statement following it this deletes any existing statement with that statement number. Lines of a program may be input in any order; they are automatically sorted into numerical order of statement number before a program is run. Thus, for example, it is possible to insert extra statements into the middle of a program by means of lines added on to the end.

Blank lines or lines starting with a semicolon are completely ignored within a program. The TEXT command automatically inserts a blank line before each AT statement in order to improve readability.

These concepts are best illustrated by an example. The example shows the use of SCAN at a console to count occurrences of "it" in two source documents, which are stored on disc files (DICKENS,CURO99) and (FIELDING,CURO99), both of which are on disc number 9. Characters typed by KOS are underlined.

```
&ENTER   SCAN
:10  AT  "IT"
:20  LET  N1 = N1+11
:30  AT  .END.
:40  PRINT  N1,  "IS THE COUNT OF 'IT'"
:50  PRINT  "FINISHED"
:20  LET  N1 = N1 + 1                          Correct line 20
:.
```

```
&RUN  FROM  DC 9 (DICKENS,CURO99)
23  IS THE  COUNT  OF  'IT'
FINISHED

&PROG
:50
:.
&TEXT

10  AT  "IT"
20  LET  N1 = N1+1

30  AT  .END.
40  PRINT  N1,  "IS THE COUNT OF 'IT'"
& RUN FROM DC 9(FIELDING,CURO99)
11  IS THE COUNT OF 'IT'
```

It is conventional to write statement numbers as multiples of ten, This leaves plenty of room to insert new statements.

1.4 The ASK facility

(This Section can be skipped on a first reading.)

The SCAN processor allocates a fixed number of each class of variable (see later). In the case of two of these classes the default number can be increased or decreased by the user when SCAN is entered. This is done by placing the argument ASK immediately after ENTER SCAN and before the DR-spec that follows. When this is done, the SCAN processor asks the two following questions

```
NUMBER  OF  C  VARIABLES =

NUMBER  OF  N  VARIABLES =
```

The user can supply any non-negative integer as answer to these questions, and, if there is enough room, he is given the number of variables he asks for. However, the more variables there are, the less room there is for storing the program or sentences of the source document.

The questions are in the form of optional data question-and-answers. This means that they come from the data device, not the command device. In non-conversational mode either or both question-and-answers may be omitted if the default allocation of ten variables is required; if the data device is not a console the ASK parameter is, in fact, redundant since SCAN always checks to see whether these question-and-answers are present.

If an answer is incorrect the message "EH" is output and the question repeated.

The TEXT command outputs these question-and-answers at the start of the program in cases when the number of C variables or N variables is not the default number of ten.

1.5 Breaks

An on-line user can "break" SCAN at any time by pressing the CONTROL G key. If a break occurs during initialization before any program lines have been specified then an exit is made. Otherwise the SCAN processor returns to command status within itself and the program remains intact.

1.6 Incremental compilation

(This Section is not of direct relevance in using SCAN and can be skipped by a reader unfamiliar with computers.)

The SCAN processor compiles each statement of the program into an internal code so that it can be run more speedily. This compiling process is incremental in the sense that statements are compiled one by one as they are supplied, thus ensuring that in a conversational environment syntactic errors are detected immediately they occur, and if one line of a program is changed only that line is recompiled. The source text is not preserved but is recreated from the internal code when the TEXT command is used. This means that there may be minor changes in format, though not in meaning, between the program as supplied and as printed by the TEXT command. In particular the spacing might be different.

Chapter 2 Elements of SCAN

2.1 Data types

SCAN deals with two types of data, namely integers and characters.

Integers may be positive or negative. There is an upper limit in the KOS implementation of SCAN of just over eight million on the size of integers (including those that arise in intermediate calculations) but this is so large that normal users will not be affected.

In addition to integers SCAN deals with characters. The full set of characters available in this implementation of SCAN is given in Appendix A. It includes the (upper-case) letters A to Z and the digits 0 to 9 together with a set of characters that are neither letters nor digits. These latter are called separators. Examples of separators are the comma, the plus sign and the full stop.

In addition to the ordinary visible characters the character set of SCAN includes some special separators called layout characters. These control the printing mechanism or indicate the state of scan. Layout characters affecting printing are the characters space, tab and newline. The newline character occurs at the end of each line of the source document. The remaining layout characters are

(a) start-of-source. This is an imaginary character inserted by SCAN at the very start of the source document.

(b) end-of-source. This is an imaginary character inserted by SCAN at the very end of the source document.

(c) null.

All these three are completely ignored on output. The purpose of (a) and (b) is to allow the user to specify actions to be taken when these imaginary characters come up, for example to perform initialization at "start-of-source" or to print out some statistics at "end-of-source". The main purpose of "null" is for blanking out previous information, since null characters are ignored on output.

2.2 Variables and their classes

There are two types of variable, corresponding to the two types of data, namely integer and character. Character variables can have as their value any single character in the character set of the implementation. (Note that in SCAN it is only possible to deal with one character at a time. It is not possible to deal with multi-character strings as a unit - they must be processed character by character. The only exceptions to this are statements concerned with matching or with output.)

Each type of variable is divided into the following classes: system variables, sentence variables and free variables. These are described below.

2.3 System variables

A principle of the SCAN processor is that it is highly parameterized. In layman's terms this means that it has a large number of control knobs for the user to adjust if he wants to. However, SCAN is set up so that it will work very well if the control knobs are left untouched. The control knobs are implemented by means of system variables. These variables have preset values which are maintained by the SCAN processor, but can be changed by the user at any time if he wishes to modify the action of the SCAN processor. For example the system character variable A2 contains the character that denotes the end of the sentence. Initially this is set to newline, so by default a sentence is a line. However the user can change the end-of-sentence character to a full stop at any time by supplying the statement

```
10 LET A2 = "."
```

Some system variables serve a slightly different purpose from acting as control knobs. Instead they provide information about how the scan is going. For example the system integer variable L30 contains the number of words so far scanned. Every time it encounters a new word, the SCAN processor increases L30 by one. L30 is useful in print statements, for example

```
30 PRINT " 'AND' OCCURRED AS THE ", L30, "TH WORD"
```

It is unlikely that the user would want to change the values of system variables like L30, but there is nothing to stop him from doing so if he wants to.

A complete list of system variables and the values they are given appears in Appendix B.

There are currently over forty different system variables used within SCAN, and the user will clearly have trouble in remembering which is which. To help alleviate this problem SCAN contains a facility for using mnemonic synonyms for the names of these variables. For example, the system integer variable that contains the number of words scanned can be called .WORDS. instead of L30. Appendix B shows the synonyms that are available. A number of them begin with "S:", which means "pertaining to the current sentence". For example .S:WORDS. is the number of words in the current sentence, and the synonym for A2 mentioned above is .S:END..

In detail, the working of the synonym feature is as follows; every time a dot is encountered in a position where a variable is expected, the SCAN processor looks ahead until it finds the closing dot. The first 3 characters of the text in between the dots (ignoring spaces and tabs) are compared with all the possible synonyms given in Appendix B. If no match is found an error message is given. Otherwise the corresponding variable name is substituted in place of the synonym.

When a program is output using the TEXT command, the reverse applies. Synonyms are substituted in place of all relevant system variable names, even if the synonyms were not used in the original program.

Note that the matching process only takes account of the first three characters in a synonym. Thus the following are all equivalent:

.WORDS.., .WOR. ,. W O R . ,. WORDS THAT HAVE BEEN SCANNED (INCLUDING THIS ONE).

Throughout this document synonyms have normally been used wherever possible.

2.4 Sentence variables

A number of SCAN variables is local to each sentence. The sentence character variables B1, B2, B3, etc. contain the current sentence. Note that this includes separators, so that if

a sentence starts with a space (as it usually will if a full stop ends the previous sentence), then B1 will be a space and the first word of the sentence will start in B2 or, if there is more than one separator at the start of the sentence, at some subsequent B variable. For each word and separator in the source document the system integer variables .INITIAL. and .FINAL. are set to give the position within B of the initial and final characters in the word or separator (see Example 3 in Chapter 1).

There is also a set of sentence integer variables. They are set to zero at the start of each sentence. They have no set meanings and are available for the user to employ as he wishes (see Example 3 of Chapter 1).

2.5 Free variables

Free variables are variables that have no inherent meaning but are entirely at the discretion of the user. Initially they are all set to zero by the SCAN processor, but thereafter the SCAN processor never touches them.

2.6 Naming of variables

A name of a variable consists of a letter followed by a subscript. The subscript may be an integer constant, the name of a variable or an expression in parentheses. The initial letter determines the class of variable as follows

A means system character variable.

B means sentence character variable.

C means free character variable.

L means system integer variable.

M means sentence integer variable.

N means free integer variable.

Examples of variable names are therefore

A1, B16, CL1, M2, N(L4+6),

N(L(L3+LM2)*3)

If the subscript is not a constant its value is calculated and this value acts as the subscript. Thus if M2 has value 6 then

BM2 means B(M2) means B(6) means B6,

and N(M2*2 - 10) means N(2) means N2.

(Note that this is a different naming convention from many programming languages. In BASIC, for example, if a variable is called A1 the value "1" does not act as a subscript and A1 is not the same as A(1). In SCAN it is.)

The above names of variables (and their synonyms) are the only ones available in SCAN. The user cannot make up his own names. He does not need to delcare or reserve his variable names. These exist whether he makes use of them or not.

The numbers of each class of variable, apart from the sentence character variables, are fixed as follows

A1, A2, ...,A20
C1, C2, ...,C10*
L1, L2, ...,L40
M1, M2, ...,M10
N1, N2, ...,N10*

(In the cases marked by an asterisk the number can be changed when SCAN is entered by using the ASK facility mentioned earlier.)

The sentence character variables contain the current sentence, and hence the number that exists is the number of characters in the current sentence (which is given by the system variable .S:FINAL.).

The value of a subscript must be within the range of the variable class concerned. This includes the sentence character variables, i.e. it is not possible to look beyond the last character of the current sentence.

2.7 Constants

Constants may be integers or single characters. Integer constants are written simply as a string of decimal digits, for example

23 2 0 32000

The highest permissible integer constant in the KOS implementation of SCAN is 32767. This is rather smaller than the maximum permitted integer value (which is over eight million) and integers greater than 32767 can be formed by multiplying other integers together.

Character constants are represented by a single character enclosed within double quotes, for example

```
"A",   "+",   """
```

In the last example above the character itself is a double quote.

Character constants of more than one character are allowed only within AT, PRINT and WARN statements. These are called long character constants and are written just like ordinary character constants. the entire string being enclosed within double quotes, for example

```
"ANSWER IS",      "/+      +-%"
```

If the string itself contains a double quote this can only occur at the start of a string. This may sometimes necessitate a string being split into two, for example

```
20   PRINT    ""AND",   "" OCCURS  3  TIMES"
```

would print

```
"AND"  OCCURS  3  TIMES
```

2.8 Specification of layout characters

It is often necessary to use character constants representing layout characters such as tab, start-of-source or null. Apart from space and tab, none of the layout characters can be represented as character constants in the way described above. Hence another mechanism has been provided for specifying layout characters. The SCAN processor initializes certain system variables to contain the values of these layout characters. These system variables have synonyms that are the same as the layout characters they represent (e.g. .NULL.,.NEWLINE.) or abbreviations of them (e.g. .START. and .END. contain start-of-source and end-of-source).

Examples of the use of these are

```
10  AT  .START.

20  LET  C3=  .NULL.

30  LET  .S:END. = .NEWLINE.

40  PRINT  "NAME", .TAB., "AGE",.TAB., "SEX"
```

2.9 Expressions

Expressions consist of variables and constants connected by the arithmetic operations plus, minus, multiply and divide. These operators are represented as

+, -, *, /

respectively. Parentheses may be freely used.

As regards precedence the normal rules of arithmetic apply unless parentheses override them, i.e. multiply and divide are done before plus and minus but otherwise operators are executed from left to right. For example

```
2 + 3 * 2      is  2 + (3*2)        is      8.

2 - 6 - 4      is  (2 - 6) - 4      is    - 8.

2 - (6 - 4)                         is      0.

2 + 64/8/2     is  2 + ((64/8)/2)       is      6.
```

The division operator is such that the result is truncated. Any remainder is ignored. For example

```
 7/4  and  6/4  and  5/4  and 4/4  are  all  1.

-7/4  and -6/4  and -5/4  and -4/4 }
                                   } are all  -1
and   7/-4  and  6/-4 and 5/-4  and 4/-4 }
```

Minus may be used as a unary operator (i.e. without an operand to the left), and when used in this way has top precedence. It is possible (though not very useful) to write several unary minus signs in sequence. Examples of expressions involving unary minus are

-1, Y/-X, -A*-3, A----6

2.10 Mixed data types

(This Section may be skipped on a first reading.)

It is possible to mix data types, i.e. to treat characters as integers and vice-versa. To each character there corresponds an integer that is its internal code. Appendix A shows the internal codes for the KOS implementation of SCAN. When a character variable or constant is used within an expression or as a subscript its internal code is used in its place. For example the statement

```
LET N1= "A"+1
```

would, if "A" had the internal code 33, set N1 to 34.

The reverse conversion occurs when an expression is assigned to a character variable. Thus

```
LET C1 = 33
```

would set C1 as the character "A".

In the case of PRINT and WARN statements, a potential ambiguity arises. Consider the statement

```
PRINT "A" + 1
```

Should this print the number 34 or the character whose internal code is 34? The ambiguity is resolved by taking the data type as that of the first variable or constant that occurs in the expression. Thus the above statement would print "B" (the character whose internal code is 34) whereas

```
PRINT 1 + "A"
```

would print 34. As a more complicated example

```
LET L("+") = 1
PRINT L("+") + "A"
```

would print 34.

Character variables in the KOS implementation of SCAN can, in fact, take on any value that an integer variable can but the effect of statements such as

```
LET C1 = -1
PRINT C1
```

is undefined (since -1 is not a correct internal code).

Chapter 3 SCAN Statements

Statements in SCAN are divided into statements defining the words and separators that are to be matched and statements that are executed when a match has been made. The former are called scan-mode statements and the latter execution-mode statements. The format of all statements is as follows

1) Number field. Each statement must be preceded by a positive integer (not exceeding 32767 in the KOS implementation of SCAN). The integer is called the statement number. It is used in the GOTO statement (q.v.), and for editing.

2) Statement field. This is the statement itself. Execution-mode statements may be preceded by IF clauses, see Chapter 4.

3) Terminator. The terminator is either the end of the line or a semicolon. In the latter case a comment may appear between the semicolon and the end of the line. (A semicolon within a character constant or a long character constant does not count as a terminator.)

The following are examples of statements

```
100   LET   N3=N3+1;     COUNT OF ALLITERATIONS
150   AT "PIG"
200   LET   C3= ";"
```

3.1 Spaces and tabs

SCAN statements have a completely free format and any spaces or tabs within them are completely ignored (except where they occur within character constants or long character constants). For example the following two statements are exactly equivalent

```
100          LET     N1 = M2   +      36

100LETN1=M2+36
```

3.2 AT statements

This Chapter concludes by describing scan-mode statements. All the execution-mode statements are described in the next Chapter.

There is, in fact, only one type of scan-mode statement. This is the AT statement, and has the form

AT AT element 1, AT element 2, ... , AT element N

The list of AT elements must contain at least one item, and can be arbitrarily long. Each AT element specifies a word or separator to be matched. If the item to be matched is a separator it can be specified either by a character constant (e.g. "+") or by the name of a system character variable or free character variable. In the latter case the subscript must be an unparenthesized integer. Hence A3, C10 or .TAB. would be permissible but A(3), B1 or AN1 would not.

If the item to be matched is a word, this is specified by an AT element consisting of a word-pattern enclosed within double quotes. A word-pattern consists of a series of letters (upper or lower case) and digits with dashes (minus signs) interspersed. It must not contain any characters other than these; in particular, spaces are not allowed. The letters and digits represent characters to be matched and the dashes stand for any sequence of letters or digits, possibly a null one. Thus CA-T will match CAT, CART or CATARACT. However it will not match SCAT or CATS, but a dash can be put at the start and/or the end of a word-pattern if matches such as these are required. Thus CA-T- will match CAT, CATS or CARTHORSE but will not match SCAT. On the other hand -CA-T- will match all these, together with SCAT or even LMNCAPQR123T789. Two dashes in sequence are not allowed. A word-pattern consisting simply of a single dash will match every word. For example

```
20    AT   "-"

25    LET  M2 = M2+.LENGTH.
```

might be part of a program to calculate the average word length in each sentence.

The arbitrary character string represented by a dash is considered to end when the letters following the dash are found. If these letters occur at the end of the word-pattern then they only end an arbitrary string if they occur at the end of the word to be matched. For example if the word-pattern is D-G and the word to be matched is DIGGING then the arbitrary string is matched with IGGIN, not with I.

The following examples show typical AT statements

```
10  AT  ",", ";",  ":", C3, "."

20  AT  "UNIVERSIT-", "POLY-", "COLLEGE"

30  AT  .TAB., .SPACE.

40  AT  "-S-", "-E-"; MATCHES WORDS CONTAINING 'E' OR 'S'
```

Note that the values of variables used as AT elements can change dynamically. For example the program

```
10  AT  .START.

20  LET  C2=";"

30  AT  C2

40  PRINT  "FIRST SEMICOLON IS AFTER THE", .WORDS., "TH
                                                    WORD"
50  LET  C2=.NULL.
```

would print a message after the first semicolon in the source document was found. At this stage C2 would be reset to a null value, so that no subsequent separator in the source document would ever match it.

Note also that values of variables specified as AT elements are only compared with separators in the source document and never with words. Hence if the value of C2 was a letter or digit it would never be matched.

There is a potential ambiguity with the AT element "-", which qualifies as a word-pattern or as a character constant. This is resolved in favour of the former; hence if it is required to match dashes (minus signs, hyphens) this should be done by a technique such as

```
10  AT  .START.
20  LET  C1="-"
 .
 .
 .
50  AT  C1
```

3.3 Initialization

It often happens that the user wishes to give initial values to some of his variables at the very start of a process. (These values override the initial values set by the SCAN processor.) To facilitate this, the SCAN processor automatically assumes that

```
AT .START.
```

is appended to the very start of each program. Hence the occurrences of AT .START. in examples in the previous Section are actually redundant.

Chapter 4 Execution-mode statements

This Chapter lists all the execution-mode statements in SCAN. Any execution-mode statement can be preceded by one or more IF clauses, for example

```
(a)   10 IF L3 = 8 PRINT "EIGHT"

(b)   20 IF N6/2 > NM6 IF B7 = "X" LET M1 = 1
```

The next Section describes IF clauses and the remaining Sections describe the execution-mode statements.

4.1 IF clauses

The exact form of an IF clause is

IF expression relational operator expression

where a relational operator is one of the following

=	meaning	"equal to".
>	meaning	"greater than".
<	meaning	"less than".
NE	meaning	"not equal to".
GE	meaning	"greater than or equal to".
LE	meaning	"less than or equal to".

The meaning of an IF clause should be self-evident.

A statement is executed only if all the IF clauses attached to it hold. Hence in Example (a) above "EIGHT" is printed if L3 has the value 8, and in Example (b) above M1 is set to one only if the value of N6/2 is greater than the value of NM6 and also the value of B7 is the character "X"

Note that relational operators such as "greater than" can be applied to character variables. The collating sequence for the KOS implementation of SCAN is given in Appendix A. In particular, digits and upper-case letters both collate in the natural order. Thus, for example, "B" is greater than "A" and "6" is greater than "5".

4.2 The null statement

A statement consisting of a statement number on its own is ignored, except that, as part of the editing system, it serves to delete any existing line of the same number.

A statement consisting of a statement number followed by a comment is taken as a null statement in the program. Such null statements are remembered as part of the program and are output by the TEXT command. They are therefore useful for placing comments, e.g.

```
10; THIS PROGRAM ANALYSES USE OF PUNCTUATION
```

4.3. The LET statement

General form LET variable = expression

Examples

```
10 LET   N1 = N(L3+2)/6
20 LET   C3 = "A"
```

Action The expression to the right of the equals sign is evaluated and assigned to the variable on the left.

4.4 The PRINT statement

General form PRINT print element1, print element2,...,print elementN

where each print element is either an expression, a long character constant or a compound element of form

variable 1 TO variable 2

where variable 1 and variable 2 belong to the same class (i.e. have the same initial letter).

The list of print elements may be arbitrarily long. It may, for instance, contain zero, one or ten elements.

Examples

```
10  PRINT  "ANSWER IS",  M6+3

20  PRINT  B1,B6 TO B(N9-1),M1 TO M6, "."
```

Action The print elements are output on the results device in the order in which they are written. In the case of a compound element all the variables from the first up to and including the last are output. Hence for example

```
10  PRINT  M3 TO M6
```

has an identical effect to

```
10  PRINT  M3, M4, M5, M6
```

(If the subscript of the first element is greater than that of the last element or if either of these subscripts is out of range then nothing is printed. Such cases are not treated as errors.)

The format of printing is as follows. Character strings are printed exactly as they stand; nothing is added to the beginning or the end. Integers are printed as a string of decimal digits, preceded by a minus sign if the integer is negative, with no redundant leading zeros. A space is printed before and after the integer.

A newline is printed at the end of a PRINT statement. Thus the statement

```
25   PRINT
```

on its own, produces a blank line of output.

The characters printed at the end of each PRINT statement and the characters printed before and after each integer can be changed by the user at any time. They correspond to the system variables .PREINTEGER., .POSTINTEGER. and .CLOSEOUTPUT., respectively.

Thus

```
10  LET  .POSTINT.  =  .NEWLINE.

20  LET  .CLOSE.    =  .NULL.
```

would cause a newline to be output after each number but no newline at the end of a PRINT statement.

4.5 Printing the current sentence

SCAN contains a special automatic facility for printing the current sentence. This is done by setting the variable .PRINTOPTION. to the value one or two. The value two causes the sentence to be prefixed by the value of .SENTENCES., which is a count of the number of sentences so far (- the effect is exactly as if the user had written

```
PRINT .SENTENCES., B1 TO B.S:FINAL.
```

except that the final .CLOSEOUTPUT character is not printed). The value one, on the other hand, causes the current sentence to be copied identically over to the output, with no added characters. .PRINTOPTION. is set to zero at the start of each sentence, so printing of each sentence has to be explicitly asked for.

Example 2 of Chapter 1 shows a simple use of .PRINTOPTION..

4.6 The WARN statement

General form WARN print element1, print element2,...,print elementN

Example 70 WARN "HYPHEN FOLLOWS 'DOG' IN LINE",.LINES.

Action The WARN statement is identical to the PRINT statement except that the output goes to the message device, not to the results device. The purpose of the WARN statement is to allow the user to supply his own diagnostic messages.

4.7 The STOP statement

General form 250 STOP

Action SCAN ignores the rest of the source document. It does, however, supply an end-of-source character (unless there has already been one) and hence performs any actions specified for AT .END..

4.8 The GOTO statement

General form (GOTO) integer
(THEN)

where the integer is zero or corresponds to a statement number occurring in the current program.

Examples

```
 50  GOTO 0
100  THEN 26
 78  GOTO 250
```

Action The statement with the given statement number is taken as the next statement. If this statement is an AT statement then a switch to scan-mode is made (see next Chapter).

A special convention applies if the statement number is zero. This means return to scan-mode and continue the scan where it left off, i.e. try to match the remaining AT statements if any. Thus the example

```
 80  IF  M3 = 0  THEN  100

 90  LET  M1 = 1

100  AT  ...
```

is equivalent to

```
 80  IF  M1 = 0  GOTO  0

 90  LET  M1 = 1

100  AT  ...
```

There is a built-in mechanism to prevent endless loops. This is controlled by the variable .GOLIMIT., which gives the maximum permissible number of GOTO statements that can be executed on any given match. If this number is exceeded then an error message is printed and execution stops (see Appendix C). .GOLIMIT. can be changed by the user if he wishes; such changes come into effect on the next match and remain in effect until .GOLIMIT. is changed again.

Chapter 5 Scanning and matching

5.1 Sequence of operation

In the normal sequence of operation of SCAN, a program is specified and then run. The program may then be run again, possibly after making some changes to it using the PROG command.

Each line of program is checked for syntatic accuracy. Any errors that are found at this stage are called syntactic errors. When a syntactic error is found a message (see Appendix C) is output, and the offending statement is ignored. In non-conversational mode the offending statement is listed with the error message. A program can still be run even if syntactic errors have occurred in it.

When the program is run, the source document is scanned sentence by sentence until the end is reached or a STOP statement is executed.

Certain programming errors may be detected at this stage, for example division by zero. These errors are called execution errors, and cause an informatory message to be produced and the run to stop immediately. See Appendix C for details.

One type of execution error is to GOTO a non-existent statement. All GOTO statements are checked at the very start of a run, and if there are any errors the run does not take place.

5.2 Sentences

The variable .S:END. contains the character that terminates a sentence. Initially this has the value "newline", which means that each line of the source document is taken as a sentence. If he wishes, the user can change this to any other separator. This is normally done once and for all at the start, but it is possible for it to be changed dynamically during a run. In the case of a dynamic change, this comes into effect for the next sentence. In the unlikely case of .S:END. being changed back to newline in the middle of a run, the rest of the current line of input is ignored and a fresh line is taken.

If .S:END. is set to a character that is not a separator this is detected as an error when the next sentence is scanned.

When the end of a sentence has been found, the sentence is placed character by character in the variables B1, B2, etc. up to B.S:FINAL. and then split up into words and separators as described below. The user can, if he wishes, subsequently overwrite any of these variables during the processing of the sentence except for B.S:FINAL., which contains the terminating character and cannot be changed. If a user overwrites part of the sentence beyond the current point of scan - not a recommended thing to do - he can change subsequent scanning.

If there is an incomplete sentence at the end of the source document, and if this sentence contains anything (i.e. characters other than spaces and newlines), then the message

LAST SENTENCE INCOMPLETE

is output. In any case the incomplete sentence is ignored.

5.3 Words and separators

Each sentence is split up into words and separators. Any character that is neither a letter nor a digit is taken as a separator. A word is any sequence of letters and/or digits bounded on each side by a separator. Thus for example if .S:END. is a newline and a line of input reads

"IT'S EX-ARMY", SAID J.SMITH.

then this consists of: the separator double-quote, the word IT, the separator quote, the word S, the separator space, the word EX, the separator minus (or hyphen), the word ARMY, the separator double quote, the separator comma, the separator space, the word SAID, the separator space, the word J, the separator dot, the word SMITH, the separator dot and, lastly, the separator newline.

Special points to note are

(a) hyphen is a separator.

(b) more than one separator can occur together. In some documents there might, for instance, be long sequences of spaces. These are treated separately.

(c) if, in the above example, dot was the end-of-sentence character then the source would consist of the two sentences

"IT'S EX-ARMY", SAID J. and SMITH.

SCAN does not have any intelligence - it just follows the rules, sometimes with inconvenient results, as the above illustrates.

5.4 Scan-mode and execution-mode

Each word and separator in a sentence is compared successively with all the AT elements in the program. (Words are compared with word-patterns and separators with the values of variables and character constants.) If there is a match, the SCAN processor jumps to the statement immediately following the AT statement that made the match, and starts executing the program. On each match several system variables are set in order to give information about the match (see Appendix B). In particular the subscript within the variable class B of the initial and final characters of the matched item and of any substrings matching dashes in a word-pattern are given. The variable .ATPOSITION. gives the position of the matched AT element in the AT statement it belongs to. The first AT element is position 1, the second 2, etc. Hence the program

```
10  AT "+", "/", "*", "$"

20  LET  N.ATPOSITION. = N.ATPOSITION. + 1
```

would increment N1 at a plus sign, N2 at a divide sign, etc.

When executing statements in the program, the SCAN processor is said to be in execution-mode; when it is scanning the source document trying to find a match it is said to be in scan-mode. When in execution-mode the SCAN processor executes statements in sequence, performing any GOTOs it encounters on the way, until one of the following happenings causes it to go back to scan-mode:

(a) it runs into an AT statement.

(b) it executes a

```
GOTO  0
```

statement.

(c) it runs off the end of the program. In this case scanning resumes with the next word or separator.

(d) it executes a STOP statement.

5.5 Returning to scan-mode

Before considering what happens in cases (a) and (b) above, it is best to consider a general example.

Once a word has been matched with one AT, it does not preclude it from matching subsequent ATs. For example consider the program

```
10  AT "-ING"
20  LET  M8 = M8 + 1
40  AT  "S-"
50  LET  M9 = M9 + 1
70  AT  .S:END. ; AT END OF SENTENCE
80  PRINT  M8,  "WORDS  END  IN  -ING"
90  PRINT  M9,  "WORDS  START  WITH  'S'"
```

Now the word STICKING would match the first two ATs.

Hence when SCAN matches an AT statement, it remembers the position of the next AT statement, which is called the resumption AT, so that it can resume there when the action of the current AT is finished. Execution of the statement

```
GOTO  0
```

causes scanning to resume at the resumption AT. When execution-mode ends because an AT statement has been encountered, scanning resumes with the resumption AT or at the AT just encountered, whichever is the later in the program. This allows the user to inhibit certain possible matches by employing forward GOTO statements. Note that if an AT statement contains several AT elements, then if one is matched no attempt is made to match subsequent ones with the same word or separator. The following three examples illustrate these rules.

Example 1

```
10  AT  "-X-"
20  GOTO  100
50  AT  "-Y-"
60  LET N1=N1+1
100 ...:
```

In this case if a word contains the letter "X" it matches the first AT. This causes it to GOTO 100 and the second AT is missed out. Thus the second AT matches words that contain "Y" but do not contain "X".

Example 2

```
100   AT "-Z-"
110   LET  N1=1
200   AT "-X-"
210   GOTO   100
300   AT   "-Y-"
```

This example does not represent a very sensible program, but just shows how the scanning rules apply. If the second AT is matched, scanning resumes with the third AT, since the GOTO statement goes to a point further back in the program than the resumption AT. The purpose of this rule is to stop the user putting the scan into an endless repetitive loop.

Example 3

```
10   AT  "-E-", "-S-";  MATCHES WORDS CONTAINING 'E' OR 'S'
20   LET  N1 = N1 + 1
```

In this case the word "MESSAGE" would cause N1 to be incremented only once although it matches both AT elements - indeed it matches both of them in two possible ways.

5.6 Repetition

It is quite valid to have two AT statements with identical patterns. There are examples where this is very useful, for instance

```
10  AT "-A-"
20  GOTO  100
30  ; THE FOLLOWING MATCHES WORDS CONTAINING "E" BUT NOT "A"
40  AT  "-E-"
    .
    .
80  GOTO  200
90  ; THE FOLLOWING MATCHES WORDS CONTAINING "E" AND "A"
100 AT  "-E-"
    ...
200 ...
```

5.7 The start and end of the source document

As mentioned earlier, the SCAN processor supplies an imaginary .START. character before it starts scanning, so that any AT action for this can be performed. Similarly it supplies an .END. character when scanning is complete. Neither of these characters counts as a sentence or as part of a sentence; each is entirely divorced from sentence scanning, and no sentence character variables exist when either is matched.

When SCAN returns to command status after a run it is quite possible to perform another run. Subsequent runs start again entirely from scratch, all variables being set back to their initial values.

In KOS, a line of output is only sent to the relevant device when a newline is output to terminate the line. If, at the end of a SCAN run, there remains an incomplete line of output, a newline is automatically added so that the line will be output in the normal way.

Chapter 6 Examples of SCAN programs

6.1 Changing the source document

The main use of SCAN is to extract information from the source document. It can, however, be used to make changes to the source document. In this case the results device for the run will normally be a disc file, so that "printing" means writing to disc. As an example consider the following program

```
&ENTER  SCAN
10  AT  ":"
20  LET  B.INITIAL. = ";" ; REPLACE IT BY A SEMICOLON
30  AT  .S:END.
40  LET  .PRINTOPTION. = 1 ; PRINT EACH SENTENCE
&RUN  FROM  DC  (A)  TO  DC  (B)
```

The results output from this program will be an identical copy of the source document but with every colon replaced by a semicolon.

A similar technique can be used to replace one word by another, provided that the new word is the same length or shorter than the old. (If it is shorter it can be padded out with null characters so that it is the same length as the old word and hence completely overwrites it.)

If the new word is longer than the old it is best to control the changes by the use of *PRINT* statements. When a replacement is to be made, the sentence should be printed up to where the change is to be made and then the new word should be printed. At the end of each sentence the remaining unprinted part should be dealt with. A program to replace PEEWIT by LAPWING might read

```
&ENTER  SCAN
10  LET  .CLOSEOUTPUT. = .NULL.; SUPPRESS NEWLINE AT END OF PRINT
20  AT  "PEEWIT"
30                    PRINT  B(M2+1)  TO  B(.INITIAL.-1)
40  PRINT "LAPWING"
50  LET  M2 = .FINAL.

70  AT  .S:END.
80  PRINT  B(M2+1) TO  B.S:FINAL,
&RUN FROM DC (BIRDS.CURO99) TO DC  (NEWBIRDS,CURO99)
```

The program works even if a sentence contains the word PEEWIT several times. The variable M2 is used to give the subscript of the latest character of the current sentence that has been printed. Note that, being a sentence variable, M2 is automatically set to zero at the beginning of each new sentence.

The technique illustrated by this program will work irrespective of the relative lengths of the replacer and the replacee, and hence is a good general method. It can be made to work for plurals and other derivatives by the following edits

```
&PROG
20  AT  "PEEWIT-"
50  LET  M2 = .1INITIAL. -1; M2 POINTS AT 'T' of PEEWIT
```

6.2 Searching for multiple words

It is often required to search for the simultaneous occurrence of several different words or separators within the same sentence. An earlier example showed a search for IF followed subsequently by THEN within the same sentence. Two examples below show how use can be made of the fact that it is possible to look ahead or behind within the current sentence.

Example 1

```
&ENTER  SCAN
10;  THIS PROGRAM PRINTS OUT ALL LINES THAT
20;  CONTAIN THE WORD "DIABECTIC" AND ALSO HAVE
30;  AN  'F'  IN COLUMN 20 AND A  'U' IN COLUMN 7
50   AT  "DIABETIC"
60   IF  B20 = "F" IF B7 = "U" LET .PRINTOPTION. = 1
```

(If there existed lines of fewer than twenty characters that contained the word DIABETIC the reference to B20 would cause an error. Hence an extra clause IF .S:FINAL. GE 20 at the start of statement 60 would be a useful safety precaution.)

Example 2

```
&ENTER  SCAN

10; THIS PROGRAM PRINTS OUT ALL LINES CONTAINING 'GUN-DOG'

20  AT  "GUN"

30  IF B(.FIN.+1)="-" IF B(.FIN.+2)="D" IF B(.FIN.+3)="O" GOTO 50

40; GOTO  0

50  IF B(.FIN.+4)="G" LET .PRINTOPTION.=2
```

(The program is longer than it otherwise need be as it is impossible to get all the IF clauses onto one line.)

6.3 Use of arrays

The following program prints out a histogram of sentence length. Sentences of zero words (resulting from a sequence of dots in the original) or more than 80 words are ignored. The histogram consists of a sentence length on the left with an asterisk against it for each sentence that has been found to have that length. For example part of the histogram might read

```
20      *
21      ***
22
23      *****
24      ***
```

From a programming point of view, this example is of interest as it shows the use of the N variables as an array: N1 counts sentences of length one, N2 sentences of length 2, etc.

```
&ENTER  SCAN  ASK

NUMBER  OF  C VARIABLES = 0

NUMBER  OF  N VARIABLES = 80

10  LET  .S:END.= "."

20  AT  .S:END.

30  IF .S:WORDS.> 0 IF .S:WORDS.<81 LET N.S:WORDS.=N.S:WORDS.+1
```

```
 40  AT  .END.
 50  LET  .CLOSEOUTPUT. = .NULL.
 60  LET  M1 = 1 ; LOOP  M1 = 1 TO 80
 70  PRINT .NEWLINE., M1,.TAB.;  SENTENCE LENGTH
 80 ; NOW  PRINT  THE  ASTERISKS
 90  LET  NM1 = NM1-1
100  IF  NM1 < 0  GOTO 130
110  PRINT  "*"
120  GOTO 90
130  LET  M1=M1+1
140  IF M1 < 81 GOTO 70
```

Appendix A Character set

The following table gives the character set of the KOS implementation of SCAN and the internal code for each character. These internal codes, which are decimal integers, determine the collating sequence when an IF clause tests whether one character is greater than another.

Code	Character	Code	Character
0	space		
1	tab	27	;
2	newline	28	<
3	½	29	=
4	$	30	>
5	%	31	↑
6	&	32	@
7	´(acute)	33 to 58	A to Z
8	(	59	[
9	)	60	£
10	*	61	]
11	+	96	` (grave)
12	,	97 to 122	a to z
13	-	123	"
14	.	124	↑
15	/	128	null
16 to 25	0 to 9	129	end-of-source
26	:	130	start-of-source

Most input/output devices only permit a sub-set of these characters. Few devices, for example, allow for lower-case letters. Teleprinters vary slightly as to their representation of some of the more unusual characters. Some teleprinters, for example, use the character @ where others use a grave accent.

Appendix B List of Variables

This Appendix gives a complete list of the available variables, together with the meaning and system setting of each. In the "When set" column, I means initially, S means at the start of each sentence, W means on each word, SEP means on each separator, M means on each match and L means on each line.

Synonym	Name	Usage	When set	System setting
1. System character				
.START.	A1	start-of-source	I	start-of-source
.S:END.	A2	end of sentence	I	newline
.END.	A3	end-of-source	I	end-of-source
.NULL.	A4	null	I	null
.TAB.	A5	tab	I	tab
.NEWLINE.	A6	newline	I	newline
.SPACE.	A7	space	I	space
	A8-A13	reserved		
.PREINTEGER.	A14	PRINT statement: character at start of integer	I	space
.POSTINTEGER.	A15	PRINT statement: character at end of integer	I	space
.CLOSEOUTPUT.	A16	PRINT statement character at end	I	newline
	A17-A20	reserved		
2. Sentence character				
	B1 - B.S:FINAL.	current sentence	S	current sentence

3. Free character

	C1-C10	available to user	I null

4. System integer

(L1 - L11 describe the current word or separator. L4 - L11 are set only on an AT statement where the word-pattern contains dashes. These values and the values of L12 - L19 should not be changed. The effect of doing so is undefined.)

.INITIAL.	L1	subscript of initial character	M
.FINAL.	L2	subscript of final character	M
.LENGTH.	L3	length	M
.1INITIAL.	L4	subscript of initial character in first arbitrary string	M
.1FINAL.	L5	subscript of final character in first arbitrary string	M
.2INITIAL.	L6	subscript of initial character in second arbitrary string	M
.2FINAL.	L7	subscript of final character in second arbitrary string	M
.3INITIAL.	L8	subscript of initial character in third arbitrary string	M
.3FINAL.	L9	subscript of final character in third arbitrary string	M
.4INITIAL.	L10	subscript of initial character in fourth arbitrary string	M
.4FINAL.	L11	subscript of final character in fourth arbitrary string	M
	L12-L15	reserved	
.S:FINAL.	L16	subscript of final character of current sentence	S
.ATPOSITION.	L17	position in list of matched AT element	M
	L18-L19	reserved	
.PRINTOPTION.	L20	0 means do not PRINT current sentence 1 means PRINT current sentence 2 means PRINT current sentence with number	S 0
GOLIMIT.	L21	limit on GOTOs	I 1000

(L30-L35 give information on the progress of the scan. The statistics always include the current word or separator and the current sentence. Changing the statistics will not change the scan. For example increasing L30 by one will not advance the scan by one word.)

.WORDS.	L30	number of words so far	W
.S:WORDS.	L31	" " " " " in current sentence	W
.SEPARATORS.	L32	" " separators so far	SEP
.S:SEPARATORS.	L33	" " " " " in current sentence	SEP
.LINES.	L34	" " lines so far	L
.SENTENCES.	L35	" " sentences so far	S
	L36-L40	reserved	

5. Sentence integer

M1-M10	available to user	S	O

6. Free integer

N1-N10	available to user	I	O

Appendix C Errors

C.1 Syntax errors

The following errors can arise when a SCAN program is being compiled.

(a) SYNONYM ERROR. Illegal synonym or missing dot at end of synonym.

(b) SUBSCRIPT ERROR. Subscript too large or too small. (This error is only detected during compilation if the subscript is an integer constant.)

(c) INTEGER TOO BIG. An integer constant exceeds the implementation maximum.

(d) SYNTAX ERROR. A statement has not been specified in the correct way, e.g. the statement number is missing, quotes round a message or dots round a synonym are wrong, a comma is missing between AT or PRINT elements, etc.

(e) UNMATCHED PARENTHESES ERROR.

C.2 Execution errors

On detecting an error when running a program, the SCAN processor abandons the run immediately. It gives a message to the user to help him find out what has gone wrong. A typical message might be

```
VARIABLE M(11) IS NON-EXISTENT IN LINE 30 OF THE PROGRAM
CURRENT SENTENCE IS: 16  HE OPENED THE DOOR.
23  LINES OF THE SOURCE DOCUMENT HAVE BEEN SCANNED
RUN ABANDONED
```

The number preceding the print-out of the current sentence is its sentence number, as given by .SENTENCES.. If an error is detected at the start or end of the source document the current sentence is not printed (since it does not exist).

The following is a list of all the types of execution errors that can occur.

(a) VARIABLE ... IS NON-EXISTENT. The subscript on the given variable is too large or too small. This includes the case of a reference to a character beyond the end of the current sentence or a reference to the current sentence when it does not exist (i.e. during AT.START. and AT.END.).

(b) NO PROGRAM TO RUN.

(c) SUSPECTED ENDLESS LOOP. See Section 4.8.

(d) DIVISION BY ZERO.

(e) UNDEFINED GOTO IN LINE ... OF THE PROGRAM. The statement number used in a GOTO statement does not exist.

(f) ILLEGAL SENTENCE TERMINATOR. The end-of-sentence character (.S:END.) is not a separator.

(g) END OF SENTENCE OVERWRITTEN. An attempt has been made to overwrite the last character in the current sentence.

(h) LAST SENTENCE INCOMPLETE. This is a warning message rather than an error message. It is described in Section 5.2.

(i) TOO MUCH. The amount of program between two AT elements is too much for the implementation.

(j) LOGICAL ERROR. This error should never occur. If it does, consult the implementor - it is his fault.

C.3 Storage exhausted

The message

LACK OF STORAGE CAUSES ABANDONMENT

may occur at any time. If it occurs during compilation the source program is lost and an exit is made from SCAN. Possible causes of the error are too many C or N variables, too big a program, a very long sentence or a combination of all these three factors. On a KOS console the available storage is usually very limited, and it may be necessary to switch to batch working if this error occurs. Before abandoning hope, however, the user should check that his end-of-sentence character is reasonable. If the end-of-sentence character is, say, a question mark, it may be that a sentence will cover several hundred lines of the source document, and, since the current sentence is kept in store, store is almost certain to be used up.