SCAN: A Simple Conversational Programming Language for Text Analysis

P. J. Brown

Among the new programming languages designed to exploit the opportunities of conversational mode when it was first developed, the most important were JOSS (Shaw) and BASIC (Kemeny and Kurtz). These were intended primarily for numerical work and, in their original forms, they were much simpler (and less powerful) than numerical programming languages developed primarily for batch mode, such as ALGOL and FORTRAN. They had the merit that they were especially designed for conversational use. In particular, the languages had the following five properties:

(1) there were built-in facilities for making corrections during a run;
(2) whenever possible, errors were pointed out immediately they were made;
(3) programs were easy to type;
(4) compilers for the languages were small, and could be shared by any number of simultaneous users, thus minimizing the demands on the storage space available on the computer;
(5) compilers were incremental. This means a program can be changed statement by statement without the time-consuming process of recompiling the whole program each time.

It is possible to adapt FORTRAN or ALGOL for conversational use, but since these languages were not designed with this in mind they are not totally suitable and are only used for historical reasons or reasons of compatibility.

The programming language SCAN described in this paper represents an attempt to provide an equivalent, in the field of text processing, to JOSS and BASIC. SCAN has the five properties outlined above, and, compared with a powerful text processing language such as SNOBOL4 (Griswold, Poage, and Polonsky), it is much simpler and much less powerful. Its compiler, which consists of under 2,000 words of 24 bits each on the ICL 4130 computer, is perhaps twenty times smaller than the compiler for SNOBOL4 would be. It is very suitable for use by students or any other computing laymen. It can be used by research workers for simple tasks, but a user trying to do a complicated stylistic analysis would stretch it beyond its limits.

No attempt will be made here to describe SCAN in detail, as a full description is given elsewhere (Brown). Instead the main principles of its operation will be described and some examples presented.

P. J. Brown is a lecturer at the University of Kent at Canterbury. *He expresses his gratitude to Mr. David Shaw and Miss Eveline Wilson for their useful suggestions after using the early pilot versions of SCAN.*

Method of scanning

The text that SCAN is to process is called the *source document,* and each time he runs a program, the user specifies which source document is to be used. It will normally be stored on a disc. The source document may contain any kind of information in character form. It may, for example, consist of a poem, a piece of prose, some historical records, a computer program, a bank statement, a list of names and addresses, or even, perhaps, a musical score in a suitably encoded form.

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 other material it might be the end of a line. A sentence need not correspond to a sentence of English grammar. It is often preferable to analyze poetry line by line, so that if the source document is a piece of poetry, a sentence would 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 specifies certain words and separators 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 whether another word has also occurred. The specification of the words and separators 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.*

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 real or floating point numbers, whereas SCAN deals with characters, words, and sentences.

Example of a SCAN program

The following is a simple SCAN program

```
10   AT  "HE", "SHE"
20   LET N1 = N1 + 1

40   AT  .END.
50   PRINT "THE COUNT FOR 'HE' AND 'SHE' IS ",N1
```

This program causes a variable, N1, to be increased by one at each occurrence of 'he' or 'she' in the source document. At the end the total value of N1 is printed and a typical output from the program might be

```
THE COUNT FOR 'HE' AND 'SHE' IS 16
```

The purpose of the numbers preceding each program statement is explained later.

As can be seen, a program consists of a number of AT statements specifying words or separators to be recognized, and each AT statement is followed by one or more statements specifying the action to be taken on recognition.

Variables

Names of variables in SCAN consist of a letter followed by a number or a parenthesized subscript. Examples of names of variables are N1, N20, B3, B(3), or B(N1+3). There are two types of variables: *character variables* can take any single character as their value, and *integer variables* can take any integer as their value.

Several of the variables are used by SCAN for special purposes. The current sentence is stored character by character in B1, B2, B3, etc. Variables with prefix A or L are called *system variables.* Each system variable has a predefined purpose and is used to communicate information between SCAN and the user. For example, A2 is the character

that indicates the end of a sentence. Thus the statement

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

would cause full stops to end a sentence. The variables L1 and L2 give the position of the current word within the current sentence. B(L1) is the first character of the current word and B(L2) the last. Further system variables are used for such purposes as relaying statistical information from SCAN to the user (e.g., how many words, sentences, etc., so far) or for the user to tell SCAN what printing options he needs. Since the user cannot be expected to remember the numbers of all the system variables—there are about fifty of them in all—a facility for mnemonic synonyms is provided. Thus, for example, .INITIAL. and .FINAL. can be used for L1 and L2, respectively, and .END., as used in the above example, means A3, which is the imaginary end-of-document character.

To aid the user, all variables that do not have predefined uses are initially set to zero or null, and certain variables, in particular those starting with the letter 'M,' are set to zero at the start of each sentence. These latter are useful for counts within sentences.

The following example illustrates some of these concepts. It is a program that searches for instances of 'if' followed, within the same sentence, by 'then.' At each occurrence it prints the text in between. An explanation follows the program.

```
10   LET   .S:END. = "."
20   AT   "IF"
30   LET   M1 = .INITIAL.
40   AT   "THEN"
50   IF   M1 > 0   PRINT   B(M1)   TO   B(.FINAL.)
```

In statement 10, .S:END. is the synonym for A2, the sentence terminator. Thus this statement defines a sentence to end with a full stop.

In statement 30, M1 is set as the subscript of the start of the 'if,' thus remembering it in case it needs to be printed later.

In statement 50, the prefix 'IF M1>0' eliminates the case where M1 has never been changed from its initial value of zero, i.e., where no 'if' has preceded the 'then.' When an 'if' has occurred, the text from the 'i' of 'if' up to and including the 'n' of 'then' is printed.

Word-patterns

Words that are to be recognized are specified by means of *word-patterns* in AT statements. In the examples so far the words to be recognized have been explicitly specified, and word-patterns have been literals such as "HE" and "SHE." It is possible to specify word-patterns involving arbitrary elements. This is done by placing a hyphen in the word-pattern at the place where the arbitrary string is to be allowed. A hyphen will match any string, including a null one. The following examples illustrate the use of hyphens.

(a) "P-" will match all words beginning with 'P'.
(b) "SPR-ING" will match all words beginning with 'SPR' and ending with 'ING', including the word 'SPRING'.
(c) "-" matches every word.
(d) "-S-S-S-" will match all words containing three or more occurrences of the letter 'S'.

When a word-pattern containing arbitrary strings is matched, certain system variables are set to indicate the position of the first and last letter in each arbitrary string within the word, so that the program may examine them. These system variables have the mnemonic names .1INITIAL., .1FINAL., .2INITIAL., .2FINAL., etc., corresponding to the first, second, etc., arbitrary strings within the current word.

The following sample program prints all palindromes in the source document that start with 'R,' 'S,' or 'T' and consist of at least four letters. An explanation is given at the end.

```
10  AT  "R-R", "S-S", "T-T"
20  IF  .LENGTH.  <4  GOTO  0
30  IF  B(.1INITIAL.)  NE  B(.1FINAL.)  GOTO  0
40  LET  .1INITIAL. = .1INITIAL. + 1
50  LET  .1FINAL. = .1FINAL. — 1
60  IF  .1INITIAL. < .1FINAL.  GOTO  30
70  PRINT  B(.INITIAL.)  TO  B(.FINAL.)
```

Statement 10 means AT any of the three alternatives.

Statement 20 uses the system variable .LENGTH., which gives the length of the current word. 'GOTO 0' means "abandon the action for the current match."

Statement 30 causes abandonment if the initial letter in the arbitrary string is not equal to (NE) the final letter.

Statement 40 moves .1INITIAL. forward one letter and statement 50 moves .1FINAL. back one letter. They thus point at the next pair of letters to be compared, and statement 60 goes back to repeat the comparison process unless the two pointers have met, thus indicating the word is indeed a palindrome. In this latter case, statement 70 prints the palindrome.

Searching for alliterations

The following example shows a SCAN program to search for alliterations in the source document. To do this it is necessary to match every word and then to compare the first letter of each with the first letter of the previous word within the current sentence.

```
10  LET .S:END. = "."
20  AT "-"
30  IF M1>0 IF B(M1) = B(.INITIAL.) PRINT B(M1) TO B(.FINAL.)
40  LET M1 = .INITIAL.
```

Statement 10 defines sentences to end with a full stop.

Statement 30 has two IF clauses: the first eliminates the case where the current word is the first in a sentence; the second compares the first letter of the current word with the first letter of the previous word (which is pointed at by M1 because of the action of statement 40 for the previous word).

The program is, in fact, rather oversimplified, as in practice it would be desirable to eliminate such trivial alliterations as 'to the' or 'an axe.'

Summary of examples

These examples have shown most of the important features of SCAN. It can be seen that the accent in the design of SCAN was to provide the user with a few simple and adaptable building bricks rather than with a comprehensive range of facilities.

SCAN programs could be expressed more concisely in many other programming languages, but it is hoped that as they stand they are in a form that is fairly easy to read and understand. If a complete layman were shown a SCAN program and given, say, five minutes of explanation, there is a reasonable chance he would have some glimmer of an understanding of what was going on.

How SCAN is used

SCAN runs under the KOS operating system (Brown) on the ICL 4130. When a user wishes to use SCAN he simply types

```
ENTER SCAN
```

and then types his program. When his program is complete and syntactically correct he types 'RUN' to run it, specifying what source document to use.

SCAN uses the same scheme as BASIC for making corrections and additions to a program. This scheme uses the numbers that precede each program statement. If a new statement is typed with the same number as a previous one then the previous one is overwritten. If the new statement number comes between two existing ones then it is inserted between these statements. The current form of the program can be printed out by typing 'TEXT.' For example, if the statement

```
25 IF .LENGTH. < 3 GOTO 0
```

were added to the program in the previous example, this new statement would be included between the existing statements 20 and 30. The effect would be to eliminate alliterations involving one-letter or two-letter words.

Practical usage

SCAN was used with fair success in a programming course for humanities students during the past year. Miss Eveline Wilson, who ran the course, thought out a very imaginative example of its use. Three texts were stored on disc as follows:

(1) A chapter of one of Ian Fleming's James Bond books.
(2) A chapter of an imitation James Bond book, written by Kingsley Amis under a pseudonym.
(3) A chapter of Kingsley Amis writing under his own name.

The students were asked to investigate whether Amis pretending to be Fleming was more like Amis or more like Fleming. It is doubtful whether any of the students produced an irrefutable answer to the question, but by working on the problem they learned a great deal about what computers can and cannot do.

SCAN cannot be claimed to be a powerful programming language, and it clearly will not help any research workers to break new horizons in methods for stylistic analysis by computer. It is, however, hoped that in its role as a conversational programming language for the layman who wants to learn about text analysis SCAN will fill a gap in the range of available programming tools. If SCAN makes some contribution to alleviating the massive ignorance of computer techniques in the humanities, or if it acts as a stepping stone for a potential research worker, it will have succeeded in its aims.

References

Brown, P.J. "The Kent On-Line System." *Software–Practice and Experience* 1,iii(July 1971):269.

Brown, P.J. *SCAN–A Sub-System for Document Searching.* Computing Laboratory, University of Kent at Canterbury, 1971.

Griswold, R.E., J. E. Poage, and I. P. Polonsky. *The SNOBOL4 Programming Language.* Englewood Cliffs, N.J.: Prentice-Hall, 1969.

Kemeny, J.G., and T.E. Kurtz. *BASIC.* 6th ed., ed. Stephen V. F. Waite and Diane G. Mather, Hanover, N.H.: University Press of New England, 1971.

Shaw, J.C. "JOSS: A Designer's View of an Experimental On-Line Computing System," *Proc. AFIPS 1964 Fall Joint Computer Conference,* Vol. 26, p. 455.