Phrase structures in FORTRAN

R. J. W. Housden
Computing Centre, University of East Anglia, Norwich

This paper describes an extension to ICL 1900 FORTRAN which provides phrase structures for defining the syntax of data. FORTRAN statements embedded in the syntax definitions define the action to be taken when data is recognised. This extended FORTRAN facilitates the writing of complex input routines in applications programs and in compiler compiler work.

(Received July 1970)

Many computer programs involve the input and output of structured data and high level languages such as FORTRAN, ALGOL and COBOL do not provide adequate facilities for the definition of such data. Atlas Autocode provides phrase structures by means of which the syntax of data is easily specified (Brooker, Morris and Rohl, 1967), and it is therefore a suitable language in which to write syntax recognisers and complex input routines. Unfortunately, Atlas Autocode is not generally available on small machines. Both FORTRAN and ALGOL have been used as syntax languages. Leavenworth (1964) describes the use of logical functions in the implementation of a syntax analyser, with code generation as a possible side effect. However, the transformation of a given syntax into ALGOL or FORTRAN is not a trivial job and involves much detailed programming. Several special purpose systems have been developed for compiler compiler type of work but, with the possible exception of BCL (Hendry, 1968), none of these is small enough or sufficiently general to satisfy the needs of the ordinary programmer. Consequently many programs are still being written in assembly level languages for the simple reason that there is no suitable alternative.

This paper describes an extension to ICL 1900 FORTRAN which provides phrase structures for the purpose of defining the syntax of data. Embedded in the syntax definitions are semantic commands, in the form of FORTRAN statements, which define the action to be taken on recognition of input data. In this extended FORTRAN, input routines and syntax recognisers can be written and debugged quickly and efficiently.*

The following sections contain a description of the format of phrase definitions, examples of their use, and a brief description of their implementation.

Phrase definitions

A phrase consists of a named sequence of phrase elements commencing with the declaration

PHRASE ⟨phrasename⟩ ⟨newline⟩
and terminated by
ENDPHRASE ⟨newline⟩

The elements which constitute a phrase are of two main types:

passive elements which define the syntax of data to be input or output,

and

active elements, or FORTRAN statements, which are obeyed when they are encountered during the matching of input data. These FORTRAN statements define the semantic parts of phrases.

It is intended that any executable FORTRAN statement should be accepted as a valid semantic element, although in the present implementation there are some minor restrictions on the FORMAT specifications for READ statements both in the phrases, where they are very rarely required, and in the main body of the program.

The syntactic, or passive elements in a phrase definition may be either:

literal strings in quotes, which when encountered in 'output mode' are copied into the output buffer and in 'input mode' are to be matched with characters in the input stream,

or

phrase names, i.e. reference to phrases, including recursive references.

A sequence of one or more elements may be compounded to form a single compound element by enclosing the sequence in parentheses. A 'branch' element, which is a set of two or more alternative elements, possibly compound, is specified in terms of the EITHER . . . OR . . . OR . . . notation of BCL. There are a number of system defined phrases including NULL, the null phrase, OSP for matching zero or more optional spaces, NL for matching a single newline, REJECT which is used to create a 'non-match' condition and results in backtracking, and EXIT for returning from the body of a phrase before ENDPHRASE is reached. A formal definition of phrases is given in **Appendix 1.**

Phrase definitions are collected together to form a special program segment called BLOCK PHRASES which is terminated in the usual way by an END statement. The type and scope of any variables used in the semantic elements of phrases are defined by declaration statements at the head of the BLOCK PHRASES segment. These declarations may include COMMON and EQUIVALENCE statements, so providing a means of communication with other program segments. A number of system defined variables and COMMON areas are made available to the user in this segment.

Phrases are entered from other program segments, either in input mode or in output mode, by means of INPUT and OUTPUT statements of the form

$$\left\{\begin{matrix}\text{INPUT}\\ \text{OUTPUT}\end{matrix}\right\}\langle\text{phrasename}\rangle$$

An example of phrase structures

The details of phrases are best described in terms of an example. **Table 1** shows a complete program illustrating the use of phrases in which semantic elements are embedded in the syntax definitions so avoiding the necessity to construct any analysis records. The MASTER segment simply calls for input of a phrase called DATA which is defined in BLOCK PHRASES in terms of other user defined phrases, namely INTEGER, IDENTIFIER and ANSTRING, and the system defined phrase OSP.

*A more efficient implementation of the phrase preprocessor with additional built-in phrases, conditional phrase elements and debugging facilities has become available since this paper was first submitted. Details of these enhancements may be obtained from the author.

```
    MASTER TESTPHRASES
    INPUT   DATA
    CALL PROCESS DATA
    STOP
    END

    BLOCK PHRASES
    COMMON NAME(2), I, J
    PHRASE DIGIT
    CALL NEXTCH;          IF(CHVAL .LE. 9) GO TO 11;REJECT
 11 CONTINUE
    ENDPHRASE

    PHRASE LETTER
    CALL NEXTCH
    IF(CHVAL .GE. 33 .AND. CHVAL .LE. 58) GO TO 10
    REJECT
 10 CONTINUE
    ENDPHRASE

    PHRASE INTEGER
    DIGIT  ;        I = CHVAL
 12 CONTINUE
    EITHER (DIGIT ;        I = I * 10 + CHVAL ; NULL)
    OR      EXIT
    GO TO 12
    ENDPHRASE

    PHRASE IDENTIFIER
    ICOUNT = 1
    CALL COPY(8,NAME(1),1,8H        ,1)
    LETTER
 15 IF (ICOUNT .GT. 8) GO TO 14
    CALL COPY(1,NAME(1),ICOUNT,CHVAL,4)
    ICOUNT = ICOUNT + 1
 14 CONTINUE
    EITHER LETTER OR DIGIT OR EXIT
    GO TO 15
    ENDPHRASE

    PHRASE ANSTRING
    EITHER (LETTER ; ANSTRING )
    OR     (DIGIT  ; ANSTRING )
    OR     NULL
    ENDPHRASE

    PHRASE DATA
    OSP;INTEGER;        J=I;OSP;IDENTIFIER
    OSP;ANSTRING;OSP;INTEGER
    ENDPHRASE
    END

    SUBROUTINE PROCESS DATA
    COMMON NAME(2), I, J
    K = I + J
    WRITE(2,100) NAME, J, I, K
100 FORMAT(1X, 2A4, 5X, 3I6)
    RETURN
    END

    FINISH
```

Table 1. A complete program using phrase structures

During execution of the input command, the data elements defined by DATA are stored in COMMON variables, so making them available for processing by other program segments. In this example the subroutine PROCESS DATA could have been called directly from PHRASE DATA with the data elements NAME, I and J as arguments.

BLOCK PHRASES commences with a COMMON declaration which is followed by phrase definitions. The most basic of these are DIGIT and LETTER which could have been defined more explicitly as

```
PHRASE DIGIT
EITHER ('0';        CHVAL = 0)
OR     ('1';        CHVAL = 1)
   :
OR     ('9';        CHVAL = 9)
ENDPHRASE
```

and similarly for LETTER, but for compactness and efficiency DIGIT and LETTER call the character input routine, NEXTCH, which leaves the internal value of the next input character in CHVAL. If the character value lies in the appropriate range for digits (or letters) we have a successful match and control returns from DIGIT (or LETTER) to the calling phrase, otherwise the system defined phrase REJECT is entered so creating a 'non-match' condition which causes the input stream pointer to be reset to its value at the last branch point encountered and control is transferred to the next alternative in sequence in the current set of alternatives.

The phrase INTEGER defines an integer as a sequence of one or more digits terminated by a non-digit. The digits are assembled by the semantic elements I = CHVAL and I = I*10 + CHVAL leaving the input integer value in I.

An identifier is defined as a string of one or more alphanumeric characters the first of which is alphabetic. Up to eight such characters are packed, left justified, into the two element array NAME, using the library routine COPY. Any further alphanumeric characters are insignificant and are skipped.

ANSTRING is included as an example of phrase which is defined recursively. It defines an alphanumeric string as any combination of letters and digits terminated by a non-alphanumeric character. No semantic action is defined in this phrase, as each character is recognised it is skipped until eventually a non-alphanumeric character is found. Note that the phrase ANSTRING can never fail, since in the event that the first character is non-alphanumeric a null string is 'matched' by the alternative NULL.

Finally the phrase DATA is defined as an integer preceded by optional spaces, the integer value is stored in J, it is followed by further optional spaces, an identifier which is stored in NAME, more optional spaces, an alphanumeric string, optional spaces and finally a second integer whose value is left in I. After successful matching of data specified by this phrase, control is returned to the master segment which then calls PROCESS DATA. This program when tested on the data

```
▽▽▽▽42▽▽JOHNSON▽▽20JULY1970▽▽▽12▽
```

gave the expected results, namely

```
JOHNSON▽▽▽▽▽▽▽▽42▽▽▽▽12▽▽▽▽54
```

One may ask what the result would be in the event of failure to match input data with that specified by the phrase DATA. In this case a non-match condition would arise and the system would normally backtrack to the next alternative but in this program no alternative data specification has been defined. A better definition of DATA which would deal with this possibility is the following.

```
    PHRASE DATA
    EITHER (OSP; INTEGER;        J = I
            OSP; IDENTIFIER
            OSP; ANSTRING
            OSP; INTEGER)
    OR      WRITE(2,101)
101 FORMAT(1X,20H DATA NOT RECOGNISED)
    ENDPHRASE
```

The reader will have noticed a number of details concerning the syntax and layout of the program.

First, more than one element may be punched per line using a semi-colon as element separator. Thus an element separator is either newline (end of card) or semi-colon. Note, however, that the first six character positions of any semantic elements (FORTRAN statements) represent the statement number and continuation fields of the statement and care should be taken not to omit these when they follow a semi-colon. For example the element . . .; J = I; . . . is not valid and should be punched as . . .; ▽▽▽▽▽▽ J = I; . . .

FORTRAN statements may not appear as the last elements of a compound element, i.e. a compound element must finish with a passive element. This can always be arranged by using NULL, the dummy passive element.

FORTRAN statement numbers may be used in the usual way but only with FORTRAN statements. There is no labelling facility for passive elements. However, the same effect can be achieved by using CONTINUE statements. Statement numbers in the range 90000-99999 are reserved for the system and should not be used. Transfer of control into or out of an alternative is not allowed as links saved on the system work stack have to be initialised and tidied up at the start and end of a set of alternatives. The only valid transfer out of an alternative is by means of the EXIT statement which is effectively a return from the phrase.

Only the first eight characters of phrase names are significant and no embedded space-characters are allowed. Particular care is needed to ensure that certain valid FORTRAN statements are not misinterpreted as references to phrases. Any element which consists of a letter followed by an alphanumeric string and is terminated by optional spaces and a separator is interpreted as a phrase reference. Thus the elements

```
     CONTINUE
     RETURN
     END
     GOTO19
and  CALLNEXTCH
```

are all assumed to be phrase references whereas

```
11  CONTINUE
    GOTO     19
    CALL     NEXTCH
```

cannot be phrase references because they contain embedded spaces. Clearly no CONTINUE statement should appear without a statement number and spaces should be used whereever they are meaningful. The FORTRAN statements END and RETURN have no meaning as phrase elements. Control is transferred from a phrase only on encountering ENDPHRASE or EXIT.

The method of implementation

A preprocessor, which was originally written in BCL but has now been implemented in the extended FORTRAN described in this paper, translates the segment BLOCK PHRASES into a subroutine called PHRASES. Any FORTRAN declarations are passed straight across to the subroutine and several declarations of system defined variables and common areas are inserted at the head of the subroutine by the preprocessor so making them available to the user. A symbol table is compiled giving the names and 'addresses' of phrases including the names of all system defined phrases. The address of a phrase defines the start of the object code into which the phrase is compiled. The present implementation restricts the number of phrases to a maximum of 100.

All passive elements and branch points (EITHERs and ORs) are compiled into an object code represented by an array of integers which is interpreted at execution time by a syntax analysis routine (SAR). At each occurrence of one or more passive elements the preprocessor generates a set of FORTRAN statements for entry to the SAR segment together with parameters specifying the entry point in the array of integers. Any semantic element (FORTRAN) is passed across to the subroutine PHRASES and if it follows a passive element causes the compilation of object code representing a return from the interpreter SAR to active mode. Thus the subroutine PHRASES consists of the semantic elements and calls to the SAR to deal with the syntax defined by passive phrase elements. SAR returns control to PHRASES to deal with the semantic phrase elements.

Any references to other phrases are dealt with by the SAR. Also, in the event of a non-match condition arising, transfer of control to the next alternative in sequence is effected interpretively and any remaining semantic elements in the alternative which failed are skipped. We find therefore, that as a result of backtracking and references to other phrases, the flow of control through the semantic elements may be out of step with that in the passive elements and on return from the SAR to subroutine PHRASES it is necessary to switch control to the appropriate semantic statements. This switching is implemented by using the ASSIGNED GO TO command. A typical call to the SAR is compiled into the following statements.

```
       ASSIGN 90006 TO IZZZ
       CALL SAR(123)
       GO TO IZZZ
90006  CONTINUE
```

Before the return from SAR to PHRASES the appropriate value is assigned to the variable IZZZ which is COMMON to both SAR and PHRASES. This particular feature is dependent upon the implementation of the ASSIGNED GO TO in 1900 FORTRAN but could easily be modified if any changes in the FORTRAN compiler make this necessary.

Appendix 2 gives a full listing of the FORTRAN segments compiled from the source statements in **Table 1.** It will be seen that an INPUT statement is compiled into a call to subroutine PHRASES, as is an OUTPUT statement also. Of the two parameters specified in this call the first is a locator for the phrase to be entered and the second is either zero or one, representing input and output mode respectively. Although the listing of subroutine PHRASES is very long it should be remembered that most of these statements compile into not more than two or three machine orders each. The segment PHRASES given here compiles into 280 words together with about 70 integers to be interpreted by the SAR.

The Input/Output mechanism

When a non-match condition arises backtracking results and the input pointer is moved back, possibly over more than one input record. It is therefore necessary to keep several of the previous input records so that they are available for rescanning should the need arise. To facilitate this the input data is held in a set of four cyclic buffers, allowing backtracking over at most four input records. This is usually quite sufficient. Characters are fetched one at a time from the cyclic buffers and their internal values looked up in an array called CHSET by the subroutine NEXTCH. A subroutine PEEP is available for picking up the value of the next input character without incrementing the input buffer pointer CHPOINT. The character set consists of 1900 α-shift characters with values in the range 0-63, plus newline which is represented by the value 64. By planting flags in the array CHSET any character can be declared as insignificant and is then ignored by subroutines NEXTCH and PEEP.

In addition to the normal WRITE statements, routines called DECPRINT and OCTPRINT are available for outputting integers in decimal and octal form respectively. Character strings of specified lengths can be output by a subroutine called OUTCHS. These three subroutines send characters to an output buffer which is printed either when full or when the phrase NL is encountered in output mode or when the routine NLPRINT is called. For most compiler compiler work it is found that the FORTRAN input/output routines are not required.

Acknowledgements
The interpretive implementation of passive phrase elements used in this work is in some ways similar to that developed by D. F. Hendry at the University of London Institute of Computer Science. The author is indebted to Mr. Hendry and other members of the BCL group for many helpful discussions.

Appendix 1

The function of the preprocessor is to translate input source statements into a FORTRAN program. This involves the recognition of:

(*a*) INPUT and OUTPUT commands,
(*b*) the segment BLOCK PHRASES,
and (*c*) the program terminator, FINISH.

Other statements are not analysed in detail but are assumed to be valid FORTRAN statements which require no processing at this stage. Thus the phrase ⟨FORTRAN statement⟩, used in the definitions below, represents any string of characters which has not already been recognised by the preprocessor and is terminated by a separator (⟨sepr⟩). Similarly, in the definition of BLOCK PHRASES the phrase ⟨any FORTRAN declarations⟩ is any statement which is not recognised as a phrase declaration.

The phrases ⟨space⟩, ⟨newline⟩, ⟨digit⟩ and ⟨letter⟩ have their usual meanings, ⟨null⟩ represents the null phrase and ⟨not prime⟩ represents any string of characters which does not include prime.

```
⟨osp⟩ ::= ⟨space⟩ ⟨osp⟩ | ⟨null⟩
⟨EOL⟩ ::= ⟨osp⟩ ⟨newline⟩
⟨sepr⟩ ::= ⟨osp⟩; ⟨sepr⟩ | ⟨osp⟩; | ⟨EOL⟩ ⟨sepr⟩ | ⟨EOL⟩
⟨ospr⟩ ::= ⟨sepr⟩ | ⟨null⟩
⟨anstring⟩ ::= ⟨letter⟩ ⟨anstring⟩ | ⟨digit⟩ ⟨anstring⟩ | ⟨null⟩
⟨phrasename⟩ ::= ⟨letter⟩ ⟨anstring⟩
⟨literal⟩ ::= '⟨notprime⟩'
⟨element⟩ ::= ⟨osp⟩ NULL | ⟨osp⟩ ⟨literal⟩ | ⟨osp⟩
              ⟨phrasename⟩ | ⟨osp⟩ ⟨branch⟩ | ⟨osp⟩
              ⟨compound⟩ | ⟨FORTRAN statement⟩
⟨branch⟩ ::= EITHER ⟨element⟩ ⟨alternative⟩
⟨alternative⟩ ::= ⟨ospr⟩ ⟨osp⟩ OR ⟨element⟩ ⟨alternative⟩ |
                  ⟨ospr⟩ ⟨osp⟩ OR ⟨element⟩
⟨compound⟩ ::= (⟨element⟩ ⟨elements⟩ ⟨osp⟩)
⟨elements⟩ ::= ⟨sepr⟩ ⟨element⟩ ⟨elements⟩ | ⟨null⟩
⟨any further elements⟩ ::= ⟨sepr⟩ ⟨osp⟩ ENDPHRASE
                           ⟨sepr⟩ | ⟨sepr⟩ ⟨element⟩
                           ⟨any further elements⟩
⟨phrase declaration⟩ ::= ⟨osp⟩ PHRASE ⟨osp⟩
                         ⟨phrasename⟩ ⟨sepr⟩ ⟨element⟩
                         ⟨any further elements⟩
⟨block phrases⟩ ::= ⟨osp⟩ BLOCK ⟨osp⟩ PHRASES ⟨sepr⟩
                    ⟨any FORTRAN declarations⟩
                    ⟨phase declarations⟩
                    ⟨any further phrase declarations⟩
                    ⟨osp⟩ END ⟨sepr⟩
⟨any further phrase declarations⟩ ::= ⟨phrase declaration⟩
                                      ⟨any further phrase
                                      declarations⟩ | ⟨null⟩
⟨input command⟩ ::= ⟨osp⟩ INPUT ⟨osp⟩ ⟨phrasename⟩
                    ⟨sepr⟩
⟨output command⟩ ::= ⟨osp⟩ OUTPUT ⟨osp⟩ ⟨phrasename⟩
                     ⟨sepr⟩
⟨source statements⟩ ::=
        ⟨input command⟩ ⟨source statements⟩ |
        ⟨output command⟩ ⟨source statements⟩ |
        ⟨block phrases⟩ ⟨source statements⟩ |
        ⟨osp⟩ FINISH ⟨EOL⟩ |
        ⟨FORTRAN statement⟩ ⟨source statements⟩
```

```
      MASTER TESTPHRASES
      CALL PHRASES(      25,       0)
      CALL PROCESS DATA
      STOP
      END

      SUBROUTINE PHRASES(ILOC,MODE)
      INTEGER WS(2000),INBFR(88),CHPOINT,CHVAL,EODSET,EODRESET
      INTEGER OUTBFR(32),OUTBPTR,CURELR,MAPP,GROUPP,FREESP,ASP
      INTEGER CHSET(64)
      COMMON /IN/ INBFR,CHPOINT,CHVAL,EODSET,EODRESET
      COMMON /OUT/ OUTBFR,OUTBPTR
      COMMON /%SAR/ CURELR,MAPP,GROUPP,FREESP,ASP
      COMMON /MAPS/ WS,IOFLAG,IZZZ
      COMMON /%CHSET/ CHSET
      COMMON NAME(2), I, J
      IF(INITSAR.EQ.0)READ(1,99999)N,(WS(I),I=1,N)
99999 FORMAT(2000I0)
      INITSAR = 1
      WS(101) = ILOC
      IOFLAG = MODE
      ASSIGN 90001 TO IZZZ
      CALL SAR(  101)
      GO TO IZZZ
90001 CONTINUE
      RETURN

C     PHRASE DIGIT

      ASSIGN 90002 TO IZZZ
      CALL SAR(  104)
      GO TO IZZZ
90002 CONTINUE
      CALL NEXTCH
      IF(CHVAL .LE. 9) GO TO 11
      ASSIGN 90003 TO IZZZ
      CALL SAR(  106)
      GO TO IZZZ
90003 CONTINUE
   11 CONTINUE
      ASSIGN 90004 TO IZZZ
      CALL SAR(  109)
      GO TO IZZZ
90004 CONTINUE

C     PHRASE LETTER

      ASSIGN 90005 TO IZZZ
      CALL SAR(. 110)
      GO TO IZZZ
90005 CONTINUE
      CALL NEXTCH
      IF(CHVAL .GE. 33 .AND. CHVAL .LE. 58) GO TO 10
      ASSIGN 90006 TO IZZZ
      CALL SAR(  112)
      GO TO IZZZ
90006 CONTINUE
   10 CONTINUE
      ASSIGN 90007 TO IZZZ
      CALL SAR(  115)
      GO TO IZZZ
90007 CONTINUE
C     PHRASE INTEGER

      ASSIGN 90008 TO IZZZ
      CALL SAR(  116)
      GO TO IZZZ
90008 CONTINUE
      I = CHVAL
   12 CONTINUE
      ASSIGN 90009 TO IZZZ
      CALL SAR(  119)
      GO TO IZZZ
90009 CONTINUE
      I = I * 10 + CHVAL
      ASSIGN 90010 TO IZZZ
      CALL SAR(  124)
      GO TO IZZZ
90010 CONTINUE
      GO TO 12
      ASSIGN 90011 TO IZZZ
      CALL SAR(  130)
      GO TO IZZZ
90011 CONTINUE

C     PHRASE IDENTIFI

      ASSIGN 90012 TO IZZZ
      CALL SAR(  131)
      GO TO IZZZ
90012 CONTINUE
      ICOUNT = 1
      CALL COPY(8,NAME(1),1,8H        ,1)
      ASSIGN 90013 TO IZZZ
      CALL SAR(  133)
      GO TO IZZZ
90013 CONTINUE
   15 IF (ICOUNT .GT. 8) GO TO 14
      CALL COPY(1,NAME(1),ICOUNT,CHVAL,4)
      ICOUNT = ICOUNT + 1
   14 CONTINUE
      ASSIGN 90014 TO IZZZ
      CALL SAR(  136)
      GO TO IZZZ
```

```
90014 CONTINUE
      GO TO 15
      ASSIGN 90015 TO IZZZ
      CALL SAR(  149)
      GO TO IZZZ
90015 CONTINUE

C     PHRASE ANSTRING

      ASSIGN 90016 TO IZZZ
      CALL SAR(  150)
      GO TO IZZZ
90016 CONTINUE
C     'HRASE DATA

      ASSIGN 90017 TO IZZZ
      CALL SAR(  163)
      GO TO IZZZ
90017 CONTINUE
      J=I
      ASSIGN 90018 TO IZZZ
      CALL SAR(  167)
      GO TO IZZZ
90018 CONTINUE
      END

      SUBROUTINE PROCESS DATA
      COMMON NAME(2), I, J
      K = I + J
      WRITE(2,100) NAME, J, I, K
  100 FORMAT(1X, 2A4,3X, 3I6)
      RETURN
      END

      FINISH
```

INPUT and OUTPUT commands may occur in any segment except BLOCK DATA and BLOCK PHRASES. BLOCK PHRASES on the other hand will occur only once, and once dealt with need not be looked for again. The efficiency of the preprocessor is therefore improved if BLOCK PHRASES is the first segment of the program.

Appendix 2

In subroutine PHRASES the preprocessor provides access to the input and output buffers and to their character pointers. A number of stack pointers are also made available to the user. Perhaps the most useful of these is FREESP which points to the next available location on the system work stack where integer values may be saved, as required, in recursive calls on phrases. Such local work space is reclaimed in the normal way on return from a phrase.

Entry to PHRASES is always by means of a call representing an INPUT or OUTPUT command. The two parameters involved in such a call specify a phrase locator and the mode of entry (input or output). Entry to the specified phrase is achieved by planting the phrase locator in WS(101) and calling the SAR with this address as parameter. On exit from this phrase, control will return to the statement labelled 90001 which is followed by a RETURN statement.

The source program in Table 1 is compiled into the following FORTRAN program.

References

Brooker, R. A., Morris, D., and Rohl, J. S. (1967). Compiler Compiler facilities in Atlas Autocode, *The Computer Journal*, Vol. 9, p. 350.
Hendry, D. F., and Mohan, B. (1968). *A BCL Manual*, University of London Institute of Computer Science.
Leavenworth, B. M. (1964). FORTRAN IV as a Syntax Language, *CACM*, Vol. 7, pp. 72-79.