BLAISE, A preprocessor for PASCAL.
(Experimental program first written in March 1979.)

This program reads an input .PAS file and produces an output .TEX file that should make the pascal program look readable when printed with the macros in file ALGOL.TEX. The exact conversion is specified on the next page using an extended BNF grammar. As in the PASCAL manual, the symbols | and {...} in the BNF productions denote alternation and zero-or-more-times-repetition, while spaces are ignored. Each production appears on two (2) lines, however, where the material on the second line is to be inserted as part of the translation. In the second line the characters { and } denote themselves, <CR> denotes carriage-return-line-feed, and <SP> denotes space.

For example, consider the following two-line production:

```
<A> ::=    begin     +{ - }  end
       \&{     }<SP>  .   \3   \!<CR>
```

This means that the input string "begin+---end" when parsed as <A> would be translated into the output string "\&{begin}<SP>+.-.-.-\3end\!<CR>".

It will not be difficult to modify this program for other languages, since the main logic is simply a top-down transduction closely following the syntax: There is a recursive procedure corresponding to each production rule. In other words, if you don't like what BLAISE does, you can change it to your heart's content. The source file is BLAISE.SAI[tex,sys].

It will be clear from the syntax that BLAISE accepts a vast generalization of the PASCAL language, including many constructions whose semantics would be pointless to define. The author's intention was to find the simplest syntax-directed translation that would handle everything in PASCAL, without being any more restrictive than necessary about rules and regulations in the syntax.

The macros in ALGOL.TEX have the following general significance (more or less):

\\	identifier style of type
\&	boldface style of type
\.	typewriter (fixed width) style of type
_	produces the underline symbol
\0	optional beginning of new line
\1	indent one more unit
\2	compulsory beginning of new line
\3	indent one less unit
\4n	permissible break in middle of line with penalty 10n
\5	like \2 but with extra space between lines

When lines of program or comment are broken, the carryover lines appear two units indented with respect to lines broken at a \0 or a \2. The macros achieve this effect by using negative indentation on lines that begin at a \0 or a \2 break.

Note: The macros use \count7, so you had better not try to use this counter when you embed BLAISE output in a larger .TEX file.

Here now is the promised syntax.

Note: BLAISE treats many of PASCAL's reserved words as equivalent to each other, and only one representative of each equivalence class appears in the syntax. Other reserved words are converted as follows:

Reserved word	Equivalent appearing in the syntax
and	div
const	var
do	then
downto	to
file	array
function	procedure
goto	packed
in	to
initprocedure	procedure
loop	begin
mod	div
not	packed
or	div
segmented	packed
set	array
type	var
while	for
with	for

A few other equivalences are also made at the character level, e.g., @ ≡ ↑, ' ≡ '.

```
<program> ::=              { { ; | . | <main comment> }    <fragment> }
            <initialinfo>                                <CR>          <endinfo>

<fragment> ::=        program     <gen exp>
                \3\2\&{       }<SP>        \1

        |           label      {  <simple token> | ,        |        <comment> }
          \3\2\1\&{      }<SP>   \0                  \45\<SP>  \40<SP>

        |           var      <gen exp> |           procedure     <gen exp> |
          \3\2\1\&{    }<SP>            \3\2\1\1\&{         }<SP>

        |     <compound statement> |      <noncompound statement>
          \3\2                        [0→2]

<gen exp> ::=  { <outer token > }
              $                $
[Note: If there are no outer tokens, the $'s are omitted.]

<outer token> ::= <open> { <inner token> } <close> | <token> | . <token>

        |              . .    | ,    |         :
          \mathrel{\! \, \!}     \45    \mathrel

        |              record          <special list>          end
          \null$\1\2\&{       }<SP>[2→0]             [0→2] \2\&{    }$\null\3

        |              <comment>
          \null$\40\<SP>          $\null

        |    ' { <character> } '
          \.{                  }
[Note: Here <character> is anything except "'". A space is converted to "\<SP>".]
```

```
<inner token> ::= <outer token> | ;       |              var
                                  \42\,    \mathop{\&{    }}

        |             procedure
          \mathop{\&{           }}

<open> ::= ( | [ |              array        |              file
                  \mathop{\&{     <SP>}}    \mathop{\&{     }\!}

<close> ::= ) | ] |                   of
                  \mathop{\&{\<SP>    }\!}

<token> ::= <simple token> |              packed
                             \mathop{\&{        }\!}

        | <nonblank character not mentioned elsewhere in this syntax>


        |              to    |   ↑    |  *    |              div    |    nil
          \mathrel{\&{   }}    {\up}    \ast    \mathbin{\&{    }}    \&{    }
[Note: the symbols ↑ and * will not appear in the output.]

<simple token> ::= <identifier> | <unsigned integer>


<identifier> ::=    <letter> { <letter> | <digit> |  _ }
                  \\{                                \_  }

<unsigned integer> ::= <digit> { <digit> } {                    <letter>    }
                                             \mathopen{\hbox{          }}


        |     " { <letter or digit> }
          \.{                       }

<comment> ::= (*       { <character> }     *)
                $\{\;$                    $\;\}$<CR>[0→2]
[Note: the delimiters (* and *) will not appear in the output.]

<comments> ::=          <comment> {   <comment> } | <empty>
                \40\<CR>            \2                      <CR>

<main comment> ::=      <comment>    |          <comment>
                     \3\5          \1    \40\<SP>
[Note: The first form is used if the comment is preceded by a blank line.]

<special list> ::=   <gen exp> { ; <comments>  <gen exp> }
                       \2                      \2
[Note: The \2 is omitted if the <gen exp> is empty.]

        {         case      <gen exp>        of <variant> { ; <variant> }    }
          \2\1\&{     }<SP>          <SP>\&{  }                            \3

<variant> ::= <comments>      { <simple token> | ,     |               <comment> }
                           \2\1$                     \45  <SP>$\40\<SP>          $

                           :      <comments> (     <special list> <close> <comments>
                  \mathrel \null$             [2→0]                                  \3

        | <comments>
```

```
<compound statement> ::=   begin     <statement 1>
                         \&{      }<SP>

        { ; <comments> <statement> } <comments>          end
                                                [0→2]\2\&{    }[0→2]

<statement 1> ::=           <comment> {   <comment> } <statement>
                    \40\<SP>            \2

        |   <simple token>:    <statement 1>
          \2               <SP>

        |   <compound statement>   |      <noncompound statement>
          \1                     \3   [2→0]

<noncompound statement> ::=       exit    if     <gen exp>
                                \2\&{    <SP>  }<SP>

        |         if     <gen exp>       then <comments><statement>       <comments>
          \2\1\&{   }<SP>          <SP>\&{     }                          \3[0→2]

                <else clause>

        |         for     <gen exp>       then <comments><statement>       <comments>
          \2\1\&{    }<SP>          <SP>\&{     }                          \3[0→2]

        |         repeat     <statement 1> { ; <comments> <statement> }
          \2\1\&{       }<SP>

                <comments>            until     <gen exp>
                          [0→2]\3\2\&{      }<SP>         [0→2]

        |         case     <gen exp>       of <case> { ; <case> }
          \2\1\&{     }<SP>          <SP>\&{  }

                          end
                [0→2]\2\&{    }\3[0→2]

        |   <gen exp>
          \0
[Note: The "\0" is omitted if the <gen exp> is empty.]

<else clause> ::=       else     <statement 1>     <comments> | <empty>
                  \2\&{     }<SP>             [0→2]

<statement> ::=   <simple token>:    <statement 1>
                \2               <SP>

        |   <compound statement> | <noncompound statement>
          \2

<case> ::= <comments>          { <simple token> |     ' { <character> } '   |
                    [0→2]\2\1                    \.{                    }

                                                ,          |          <comment> }
                                             \45\<SP>    \40\<SP>

                :    <statement 1>
                 <SP>         \3

        | <comments>
```

In these formulas the code "[Ø→2]" means that the next \Ø or \2
will be changed to \2, and the code "[2→Ø]" means vice versa.
The sequence "[2→Ø][Ø→2]\2" results in "\2".
Two or more successive <CR>'s in the output will be replaced by a single <CR>.
When the input contains a blank line, the next \Ø or \2 in the output (including
the text, if any, inserted just before this part of the input, will be
changed to \5.

The syntax is slightly ambiguous: A <statement> or <statement 1> beginning
with "<simple token>:", where the ":" is not followed immediately by an
"=" sign, is not considered to be a special case of <gen exp> (one of
the possibilities in <noncompound statement). Identifiers, unsigned integers, and
statements are made as long as possible from a given starting position, as is
customary when doing top-down "recursive descent" parsing.

Typical use of BLAISE

```
.r blaise
Input file: fou
Output file (default = fou.TEX):
(fou.PAS 1 2 3)
End of SAIL execution

.r tex
*\input fou
(fou.TEX 1 2 (basic.TEX 1 2 3 4) (algol.TEX 1) [1] [2])
*\vfill\end
 [3]
End of SAIL execution
.xspool fou.xgp/head/q/xgp/ntn=33
```