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
\& 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 \& or a \2. The macros achieve this
effect by using negative indentation on lines that begin at a \& 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:

<table>
<thead>
<tr>
<th>Reserved word</th>
<th>Equivalent appearing in the syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>div</td>
</tr>
<tr>
<td>const</td>
<td>var</td>
</tr>
<tr>
<td>do</td>
<td>then</td>
</tr>
<tr>
<td>downto</td>
<td>to</td>
</tr>
<tr>
<td>file</td>
<td>array</td>
</tr>
<tr>
<td>function</td>
<td>procedure</td>
</tr>
<tr>
<td>goto</td>
<td>packed</td>
</tr>
<tr>
<td>in</td>
<td>to</td>
</tr>
<tr>
<td>initprocedure</td>
<td>procedure</td>
</tr>
<tr>
<td>loop</td>
<td>begin</td>
</tr>
<tr>
<td>mod</td>
<td>div</td>
</tr>
<tr>
<td>not</td>
<td>packed</td>
</tr>
<tr>
<td>or</td>
<td>div</td>
</tr>
<tr>
<td>segmented</td>
<td>packed</td>
</tr>
<tr>
<td>set</td>
<td>array</td>
</tr>
<tr>
<td>type</td>
<td>var</td>
</tr>
<tr>
<td>while</td>
<td>for</td>
</tr>
<tr>
<td>with</td>
<td>for</td>
</tr>
</tbody>
</table>

A few other equivalences are also made at the character level, e.g., @ :=, ' * '.

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

<fragment> ::= program <gen exp> <SP> 1
  | label { <simple token> | , <comment> } <SP> 8 <45<SP> <48<SP>
  | var <gen exp> | procedure <gen exp> | <compound statement> <noncompound statement> <SP> 132 [8-2]

<gen exp> ::= { <outer token> }

$ $ $ [Note: If there are no outer tokens, the $'s are omitted.]

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

| \mathrel\{!! \, \!\} | \, \! \mathrel
| record <special list> end <SP>[2-8] [8-2] 2\&{ $\null\null\3
| \null\null\null\null<SP> $null
| ' { <character> } '
| \{<character> \}

[Note: Here <character> is anything except ''''. A space is converted to "\SP". ]
<inner token> ::= <outer token> | ; 
\42, \mathop{\&}\{ \}

| \procedure 
\mathop{\&}\{
<open> ::= ( | [ | \mathop{\&}\{ \array \mathop{\&}\{ <SP>\}} \mathop{\&}\{ \}!!) 
<close> ::= ) | ] | \mathop{\&}\{ \of <SP> \}!!)
<token> ::= <simple token> | \packed 
\mathop{\&}\{
| <nonblank character not mentioned elsewhere in this syntax>
| \to | \uparrow | * | \div | \nil
\mathrel{\{\&\{ \}} \{\uparrow} \mathbin{\&}\{ \} \&\{ \} \} \&\{ \}
[Note: the symbols \ast and \* will not appear in the output.]
<simple token> ::= <identifier> | <unsigned integer>

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

<unsigned integer> ::= \digit { \digit } { \mathopen{\hbox{ \}
| \" { \letter or digit } \} \\
\-

<comment> ::= (* { <character> } *) \\
$\{\;\}$ $\{\;\}$$\cr$$\[a-2]$
[Note: the delimiters (*) and *) will not appear in the output.]
<comments> ::= \48\cr \l2 <CR>
<main comment> ::= <comment> | <comment> \\
| \3\5 \l1 \48\sp \\
[Note: The first form is used if the comment is preceded by a blank line.]
<special list> ::= <gen exp> { ; <comments> <gen exp> } \\
\l2 \l2
[Note: The \l2 is omitted if the \gen exp is empty.]
{ case <gen exp> of <variant> { ; <variant> } } \\
\l2\l1\&\{ <sp> <sp>&\{ \} \} \l3
<variant> ::= <comments> { <simple token> | , | <comment> } \\
\l2\l1$s \\
\l45 <sp>$\48\sp \\
\mathrel{\null}\$ [2-8]
| <comments>
<compound statement> ::= begin <statement 1>
   \\{  \}⟨SP⟩
   { ; <comments> <statement> } <comments> end
   [0-2]\\2\\&{  }[0-2]

<statement 1> ::= <comment> { <comment> } <statement>
   \4\0⟨SP⟩
   \2
   { <simple token>: <statement 1> }
   \2 ⟨SP⟩
   { <compound statement> | <noncompound statement> }
   \4 [2-0]

<noncompound statement> ::= exit if <gen exp>
   \2\\&{  }⟨SP⟩ }⟨SP⟩
   { \2\1\\&{  }⟨SP⟩ }⟨SP⟩\\&{  }[0-2]
   <else clause>
   { \2\1\\&{  }⟨SP⟩ }⟨SP⟩\\&{  }[0-2]
   <for <gen exp> then <comments><statement> <comments>
   \2\1\\&{  }⟨SP⟩ }⟨SP⟩\\&{  }[0-2]
   <else clause>
   { \2\1\\&{  }⟨SP⟩ }⟨SP⟩\\&{  }[0-2]
   <case > <gen exp> of <case> { ; <case> }<case>
   \2\1\\&{  }⟨SP⟩ }⟨SP⟩\\&{  }[0-2]
   end
   [0-2]\\2\\&{  }[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> ::= <comment> { <simple token> | <comment> }
   { <character> } { <character> }  { \.
   \} \{ <comment> }<comment>
   \4\5⟨SP⟩ \4\0⟨SP⟩
   { <statement 1> }
   ⟨SP⟩ \3
   { <comments>
In these formulas the code "[0-2]" means that the next \0 or \2 will be changed to \2, and the code "[2-0]" means vice versa.
The sequence "[2-0][0-2]\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 \0 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) (algo1.TEX 1) [1] [2])
*\vfill\end
[3]
End of SAIL execution
.xspool fou.xgp/head/q/xgp/ntn=33