

## Production Rules

<i>program</i> ->	<b>program id ;</b> <i>declarations</i> <i>subprogram_declarations</i> <i>compound_statement</i> <b>.</b>
<i>identifier_list</i> ->	<b>id</b>   <b>id , identifier_list</b>
<i>declarations</i> ->	<b>var identifier_list : type ; declarations</b>   $\lambda$
<i>type</i> ->	<i>standard_type</i>   <b>array [ num : num ] of standard_type</b>
<i>standard_type</i> ->	<b>integer</b>   <b>real</b>
<i>subprogram_declarations</i> ->	<i>subprogram_declaration ;</i> <i>subprogram_declarations</i>   $\lambda$
<i>subprogram_declaration</i> ->	<i>subprogram_head</i> <i>declarations</i> <i>subprogram_declarations</i> <i>compound_statement</i>
<i>subprogram_head</i> ->	<b>function id arguments : standard_type ;</b>   <b>procedure id arguments ;</b>
<i>arguments</i> ->	<b>( parameter_list )</b>   $\lambda$
<i>parameter_list</i> ->	<i>identifier_list : type</i>   <i>identifier_list : type ; parameter_list</i>
<i>compound_statement</i> ->	<b>begin optional_statements end</b>
<i>optional_statements</i> ->	<i>statement_list</i>   $\lambda$

*statement\_list* ->     *statement* |  
                              *statement* ; *statement\_list*

*statement* ->            *variable* **assignop** *expression* |  
                              *procedure\_statement* |  
                              *compound\_statement* |  
                              **if** *expression* **then** *statement* **else** *statement* |  
                              **while** *expression* **do** *statement* |  
                              *read* ( *id* ) |  
                              *write* ( *expression* )

*variable* ->            **id** |  
                              **id** [ *expression* ]

*procedure\_statement* ->            **id** |  
    **id** ( *expression\_list* )

*expression\_list* ->     *expression* |  
                              *expression* , *expression\_list*

*expression* ->            *simple\_expression* |  
                              *simple\_expression* **relop** *simple\_expression*

*simple\_expression* ->            *term* *simple\_part* |  
    *sign* *term* *simple\_part*

*simple\_part* ->            **addop** *term* *simple\_part* |  
                               $\lambda$

*term* ->                    *factor* *term\_part*

*term\_part* ->            **mulop** *factor* *term\_part* |  
                               $\lambda$

*factor* ->                **id** |  
                              **id** [ *expression* ] |  
                              **id** ( *expression\_list* ) |  
                              **num** |  
                              ( *expression* ) |  
                              **not** *factor*

*sign* ->                + |  
                              -

## Lexical Conventions

1. Comments are surrounded by `/*` and `*/`. They may not contain a `{`. Comments may appear after any token.
2. Blanks between tokens are optional.
3. Token **id** for identifiers matches a letter followed by letter or digits:  
**letter** -> **[a-zA-Z]**  
**digit** -> **[0-9]**  
**id** -> **letter (letter | digit)\***

The `*` indicates that the choice in the parentheses may be made as many times as you wish.

1. Token **num** matches numbers as follows:  
**digits** -> **digit digit\***  
**optional\_fraction** -> **. digits |  $\lambda$**   
**optional\_exponent** -> **(E (+ | - |  $\lambda$ ) digits) |  $\lambda$**   
**num** -> **digits optional\_fraction optional\_exponent**
2. Keywords are reserved.
3. The relational operators (**relop**'s) are:  
**=, <>, <, <=, >=, and >.**
4. The **addop**'s are **+, -, and or.**
5. The **mulop**'s are **\*, /, div, mod, and and.**
6. The lexeme for token **assignop** is **:=.**