PLATYPUS - Language Specification

The PLATYPUS Lexical Specification

Program Start:

```
<program> ->
      PLATYPUS { <opt_statements> }

<opt_statements> ->
      <statements> | ϵ
```

Input Elements and Tokens:

```
<input character> -> one of
      ASCII characters but not SEOF (Source End Of File)

<input> ->
      <input elements> SEOF   Does this go in?

<input elements> ->
      <input elements> <input element> | <input element>

<input element> ->
      <white space > | <comment> | <token>

<token> ->
      <variable identifier> | <keyword> | <floating-point literal > |
      <integer literal > | <string literal> | <separator> | <operator>
```

White Space:

```
<white space> ->
      ASCII SP | ASCII HT | ASCII VT | ASCII FF | <line terminator>

<line terminator> ->
      CR | LF | CR LF
```

Comments:

```
<comment> ->
      !< <opt_characters in line> <line terminator>

<opt_characters in line>
      <characters in line> | ϵ

<characters in line> ->
      <characters in line> <comment character> | <comment character>

<comment character> ->
      <input character> but not <line terminator>
```

Variable Identifiers:

```
<variable identifier> ->
      <arithmetic variable identifier> | <string variable identifier>

<arithmetic variable identifier> ->
      <letter> <opt_letters or digits>

<opt_letters or digits> ->
      <letters or digits> | ϵ

<letters or digits> ->
      <letters or digits> <letter or digit> | <letter or digit>

<letter> -> one of
      a...z A...Z ( uppercase and lowercase ASCII Latin letters A – Z)

<letter or digit> -> one of
      a...z A...Z 0...9 ( ASCII digits 0-9)

<string variable identifier> ->
      <arithmetic variable identifier>%
```

Keywords:

```
<keyword> ->
      PLATYPUS | IF | THEN | ELSE | USING | REPEAT | INPUT | OUTPUT
```

Operators:

```
<operator> ->
      <arithmetic operator> | <string concatenation operator> |
      <relational operator> | <logical operator> | <assignment operator>

<arithmetic operator> -> one of
      +  -  *  /

<string concatenation operator> ->
      #

<relational operator> -> one of
      >  <  ==  <>

<logical operator> ->
      .AND. | .OR.

<assignment operator> ->
      =
```

Separators:

```
<separator> -> one of
      ( )  { } , ; “  .
```

Floating-point Literals:

```
<floating-point literal> ->
      <whole decimal number> . <opt_digits>

<whole decimal number> ->
      0 | <decimal integer literal>
```

String Literals:

```
<string literal> ->
      “<opt_string characters>” but not SEOF

<opt_string characters> ->
      <string characters> | ϵ

<string characters> ->
      <string characters> <input character> | <input character>
```

Integer Literals:

```
<integer literal> ->
      0 | <decimal integer literal> | <octal integer literal>

<decimal integer literal> ->
      <non zero digit> <opt_digits>

<opt_digits> ->
      <digits> | ϵ

<digits> ->
      <digits> <digit> | <digit>

<digit> ->
      0 | <non zero digit>

<non zero digit> ->
      1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<octal integer literal> ->
      0 <non zero octal digit> <opt_octal digits>

<opt_octal digits> ->
      <octal digits> | ϵ

<octal digits> ->
      <octal digits> < octal digit> | <octal digit>

<octal digit> ->
      0 | <non zero octal digits>

<non zero octal digits> ->
      1 | 2 |3 | 4 | 5 | 6 | 7
```

The PLATYPUS Syntactic Specification

PLATYPUS Program:

```
<program> ->
      PLATYPUS { <opt_statements> } SEOF

<opt_statements> ->
      <statements> | ϵ

<statements> ->
      <statements> <statement> | <statement>
```

Statements:

```
<statement> ->
      <assignment statement> | <selection statement> | <iteration statement> |
      <input statement> | <output statement>
```

Assignment Statement:

```
<assignment statement> ->
      <assignment expression>;

<assignment expression> ->
      AVID = <arithmetic expression> | SVID = <string expression>
```

Selection Statement(the if statement):

```
<selection statement> ->
      IF ( <conditional expression> ) THEN <opt_statements>
      ELSE { <opt_statements> };
```

Iteration Statement (the loop statement):

```
<iteration statement> ->
      USING ( <assignment expression>, <conditional expression>,
      <assignment expression> ) REPEAT { <opt_statements> };
```

Input Statement:

```
<input statement> ->
      INPUT ( <variable list> );

<variable list> ->
      <variable list>,<variable identifier> | <variable identifier>
```

Output Statement:

```
<output statement> ->
      OUTPUT ( <optional variable list> ); | OUTPUT ( <string literal> );

<optional variable list> ->
      <variable list> | ϵ
```

Expressions:

Arithmetic Expression:

```
<arithmetic expression> - >
      <unary arithmetic expression> | <additive arithmetic expression>


<unary arithmetic expression> ->
      - <primary arithmetic expression> | + <primary arithmetic expression>


<additive arithmetic expression> ->
      <additive arithmetic expression> + <multiplicative arithmetic expression> |
      <additive arithmetic expression> - <multiplicative arithmetic expression> |
      <multiplicative arithmetic expression>


<multiplicative arithmetic expression> ->
      <multiplicative arithmetic expression> * <primary arithmetic expression> |
      <multiplicative arithmetic expression> / <primary arithmetic expression> |
      <primary arithmetic expression>


<primary arithmetic expression> ->
      <variable identifier> | <floating-point literal> | <integer literal> |
      ( <arithmetic expression> )
```

String Expression:

```
<string expression> ->
      <primary string expression> | <string expression>  #  <primary string expression>


<primary string expression> ->
      <string variable identifier> | <string literal>
```

Conditional Expression:

```
<conditional expression> ->
      <logical OR expression>


<logical OR expression> ->
      <logical AND expression> |
      <logical OR expression> .OR. <logical AND expression>


<logical AND expression> ->
      <relational expression> |
      <logical AND expression> .AND. <relational expression>
```

Relational Expression:

```
<relational expression> ->
      <primary a_relational expression>  ==  <primary a_relational expression> |
      <primary a_relational expression>  <>  <primary a_relational expression> |
      <primary a_relational expression>  >   <primary a_relational expression> |
      <primary a_relational expression>  <   <primary a_relational expression> |
      <primary s_relational expression>  ==  <primary s_relational expression> |
      <primary s_relational expression>  <>  <primary s_relational expression> |
      <primary s_relational expression>  >   <primary s_relational expression> |
      <primary s_relational expression>  <   <primary s_relational expression>

<primary a_relational expression> ->
      <arithmetic variable identifier> | <floating-point literal> | <integer literal>


<primary s_relational expression> ->
      <primary string expression>
```