**2.1 Input Elements and Tokens**

< input character *> -> one of*

ASCII characters *but not* SEOF

<input> ->

<input elements> SEOF

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

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

<token> ->  
 <variable identifier> | <keyword> | <floating-point literal >

| <integer literal > | <string literal> | <separator> | <operator>

**2.2 White Space**

<white space> ->  
 the ASCII SP character, also known as “space”

|  the ASCII HT character, also known as “horizontal tab”

|  the ASCII VT character, also known as “vertical tab”

|  the ASCII FF character, also known as “form feed”

|  <line terminator>

<line terminator> ->

CR | LF | CR LF

**2.3 Comments**

<comment> ->  
 !< <opt\_characters in line> <line terminator>

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

which actually means (see 1.4):

<comment> ->  
 !< <opt\_characters in line> <line terminator>

<opt\_characters in line>

<characters in line> | ∈

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

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

**2.4 Variable Identifiers**

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

<arithmetic variable identifier> ->

<letter> <opt\_letters or digits>

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

**2.5 Keywords**

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

**2.6 Integer Literals**

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

< decimal integer literal> ->  
 0 | <non zero digit> <opt\_digits>

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

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

<non zero digit> one of

123456789

<octal integer literal> ->  
​ 0 <octal digit> < octal digits>

**<octal integer literal> ->**

**0 0 | 0 <octal positive> <opt octal digits>**

<octal digit> ->  
 0 | 1 | 2 |3 | 4 | 5 | 6 | 7

**<octal positive>**

**1 | 2 |3 | 4 | 5 | 6 | 7**

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

**2.7 Floating-point Literals**

<floating-point literal> ->  
 <decimal integer literal> **.** <opt\_digits>

**2.8 String Literals**

<string literal> ->  
 “<opt\_string characters>”

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

**2.9 Separators**

<separator> *-> one of*

() {},;“ **.**

**2.10 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> ->

=

**3.1 PLATYPUS Program**

<program> ->  
 PLATYPUS {<opt\_statements>} SEOF

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

**3.2 Statements**

<statement> ->  
<assignment statement>

|  <selection statement>

|  <iteration statement>

|  <input statement>

|  <output statement>

**3.2.1 Assignment Statement**

<assignment statement> -> <assignment expression> ;

< assignment expression> ->  
 AVID = <arithmetic expression>

| SVID = <string expression>

**3.2.2 Selection Statement( the if statement)**

<selection statement> ->  
 IF (<conditional expression> ) THEN <statements> ELSE { <opt\_statements> } ;

<selection statement> ->  
 IF (<conditional expression> ) THEN <opt\_statements> ELSE { <opt\_statements> } ;

**3.2.3 Iteration Statement (the loop statement)**

<iteration statement> ->  
 <*complete this production*>

<iteration statement> ->  
 USING (<assignment statement>, <conditional expression>, <assignment statement> ) REPEAT {

<opt\_statements>

} ;

**3.2.4 Input Statement**

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

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

**3.2.5 Output Statement**

<output statement> ->  
 <*complete this production*>

<output statement> ->  
 OUTPUT (<opt\_variable list> | <string literal>) ;

**3.3.1 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>)

<primary arithmetic expression> -> <AVID>

|  <floating-point literal>

|  <integer literal>

|  (<arithmetic expression>)

**3.3.2 String Expression**

<string expression> ->

<primary string expression>  
| <string expression> <> <primary string expression>

<primary string expression> -> <string variable identifier>

| <string literal>

**3.3.3 Conditional Expression**

<conditional expression> ->

<logical OR expression>

<logical OR expression> ->

<*complete this production*>

<logical OR expression> ->

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

<logical AND expression>

<logical AND expression> ->

<*complete this production*>

<logical AND expression> ->

<logical AND expression> .AND. <conditional expression> |

<relational expression>

**3.3.4 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 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 a\_relational expression> ->

<floating-point literal>

| <integer literal>  
 | <variable identifier>

| <AVID>

<primary s\_relational expression> -> <*complete this production*>

<primary s\_relational expression> ->

<string literal>

| <SVID>