Assignment 3.1 – Language Grammar

Velvet Language Specification

Team – Megan Clinch – Id: 041043369 / Cameron Ewing – Id: 041037946

Grammar, which knows how to control even kings . . .
Molière, *Les Femmes Savantes* (1672)
Act II, Scene VI

1. The Velvet Language Lexical Specification

1.1. White Space

White space is defined as the ASCII space, horizontal and vertical tabs, and form feed characters, as well as line terminators. White space is discarded by the scanner.

```
<white_space> → one of { SPACE, TAB, FF, NL, CR, NLCR }
```

1.2. Comments

Velvet supports only single-line comments: all the text from the ASCII characters to the end of the line is ignored by the scanner.

```
<comments> → // { sequence of ASCII chars } NL | NULL | NBSP
```

1.3. Variable Identifiers (Prefix)

The following variable identifier (VID) tokens are produced by the scanner: one kind of arithmetic token **ENID_T** (entero and decimal) and one kind of character token **CNID_T** (chain).

```
<variable_identifier> → ENID_T | CNID_T
```

1.4. Method Identifiers (Prefix)

The following method identifier (MID) token is produced by the scanner: **MNID_T**

```
<method_identifier> → MNID_T
```

1.5. Keywords

The scanner produces a single token: **KEY_T**. The type of the keyword is defined by the attribute of the token (the index of the **keywordTable[ ]**). Remember that the list of keywords in Velvet is given by:

```
ent, decimal, chain, if, elseif, else, for, when, true, false, send, print,
input, AND, OR
```

1.6. Entero Literals

The scanner produces a single token: **ENL_T** with an integer value as an attribute.

```
<entero_literal> → ENL_T
```

1.7. Decimal Literals

DECI_T token with a real decimal value as an attribute is produced by the scanner.

```
<decimal_literal> → DECI_T
```

1.8. Chain Literals

CHN_T token is produced by the scanner.

```
<chain_literal> → CHN_T
```

1.9. Separators

Some different tokens are produced by the scanner – **LPR_T**, **RPR_T**, **LBR_T**, **RBR_T**, **COMA_T**, **EOS_T**.

```
<separator> → one of { (, ), {, }, ,, ; }
```

1.10. Operators

Arithmetic Operators

A single token is produced by the scanner: **ART_OP_T**. The type of the operator is defined by the attribute of the token.

```
<arithmetic_operator> → one of { +, -, *, / }
```

Relational Operators

A single token is produced by the scanner: **REL_OP_T**. The type of the operator is defined by the attribute of the token.

```
<relational_operator> → one of { >, <, == }
```

Assignment Operator

A single token is produced by the scanner: EQ_T.

```
<assignment_operator> → =
```

2. The Velvet Language Syntactic Specification

2.1. Velvet Language Program

2.1.1. Program

Velvet is composed of one special function: **_main** (method name) defined as follows.

```
<program> → _main() {
                <opt_data_declarations>
              | <opt_code_statements>
                }
```

2.1.2. Data

Variable Lists

The optional variable list declarations is used to define several datatype declarations.

```
<opt_data_declarations> → <varlist_declarations>
                         | ϵ
```

Variable Declarations

```
<varlist_declarations> →
                  <varlist_declaration>
                | <varlist_declaration><varlist_declarationsPrime>
```

```
<varlist_declarationsPrime> →
                 <varlist_declaration><varlist_declarationsPrime>
               | ϵ
```

```
<varlist_declaration> → <entero_varlist_declaration>
                       | <decimal_varlist_declaration>
                       | <chain_varlist_declaration>
```

2.1.3. Declaration of Lists

The variables list declaration is defined here.

```
<entero_varlist_declaration>  → ent <entero_variable>;

<decimal_varlist_declaration> → decimal <decimal_variable>;

<chain_varlist_declaration>   → chain <chain_variable>;
```

2.1.4. List of Variables

The list of variables is defined here.

Enteros:

```
<entero_variable> → ENID_T
```

Decimals:

```
<decimal_variable> → ENID_T
```

Chains:

```
<chain_variable>  → CNID_T
```

2.1.5. Code Session

The second part (CODE) is the place we have statements.

Optional Statements

```
<opt_code_statements> → <statements>;
                         | ϵ
```

2.1.6. Statements

```
<statements> → <statement>
               | <statement><statementsPrime>
```

```
<statementsPrime> → <statement><statementsPrime>
                   | ϵ
```

2.2. Statement

```
<statement> → <assignment_statement>
             | <selection_statement>
             | <iteration_statement>
             | <input_statement>
             | <output_statement>
```

2.2.1. Assignment Statement

```
<assignment_statement> → <assignment_expression>;
```

2.2.2. Assignment Expression

```
<assignment_expression> → <entero_variable> = <arithmetic_expression>
                          | <decimal_variable> = <arithmetic_expression>
                          | <chain_variable> = <chain_expression>
```

2.2.3. Selection Statement (if statement)

```
<selection_statement> →
     if (<conditional_expression>){<opt_code_statements> }
     ?(elseif (<conditional_expression>){ <opt_code_statements> })
     ?(else { <opt_code_statements> })
```

2.2.4. Iteration Statement (loop statements)

```
<iteration_statement> →
       for (<relational_expression>){ <opt_code_statements> }
     | when (<conditional_expression>){ <opt_code_statements> }
```

2.2.5. Input Statement

```
<input_statement> → input(<variable_list>)
```

Variable List

```
<variable_list> → <variable_identifier>
                | <variable_list>,<variable_identifier>
```

Variable Identifier

```
<variable_identifier> → <entero_variable>
                      | <decimal_variable>
                      | <chain_variable>
```

2.2.6. Output Statement

```
<output_statement> → print(<opt_variable_list>)
```

Optional Variable List

```
<opt_variable_list> → <variable_list>
                    | ϵ
```

2.3. Expressions

2.3.1. Arithmetic Expressions

```
<arithmetic_expression> → <unary_arithmetic_expression>
                        | <arithmetic_expressions_ADD_SUB>
```

Unary Arithmetic Expressions

```
<unary_arithmetic_expression> → - <primary_arithmetic_expression>
                              | + <primary_arithmetic_expression>
```

Arithmetic Expressions – Add and Subtract

```
<arithmetic_expressions_ADD_SUB> →
   <arithmetic_expressions_ADD_SUB> + <arithmetic_expressions_MUL_DIV>
 | <arithmetic_expressions_ADD_SUB> - <arithmetic_expressions_MUL_DIV>
 | <arithmetic_expressions_MUL_DIV>
```

Arithmetic Expression – Multiply and Divide

```
<arithmetic_expressions_MUL_DIV> →
   <arithmetic_expressions_MUL_DIV> * <primary_arithmetic_expression>
 | <arithmetic_expressions_MUL_DIV> / <primary_arithmetic_expression>
 | <primary_arithmetic_expression>
```

Primary Arithmetic Expression

```
<primary_arithmetic_expression> → <entero_variable>

                                | <decimal_variable>

                                | DECI_T

                                | ENL_T

                                | (<arithmetic_expression>)
```

2.3.2. String Expression

```
<chain_expression> → <chain_variable>
                   | <chain_literal>
```

2.3.3. Conditional Expression

```
<conditional_expression> → <logical_OR_expression>
```

Logical OR Expression

```
<logical_OR_expression> →
    <logical_AND_expression>
  | <logical_OR_expression> OR <logical_AND_expression>
```

Logical AND Expression

```
<logical_AND_expression> →
     <relational_expression>
   | <logical_AND_expression> AND <relational_expression>
```

2.3.4. Relational Expression

```
<relational_expression> →
                         <relational_a_expression>
                       | <relational_s_expression>
```

Relational Arithmetic Expression

```
<relational_a_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>
```

Relational String Expression

```
<relational_s_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 Arithmetic and String Relational Expressions

```
<primary_a_relational_expression> → <entero_variable>
                                   | <decimal_variable>
                                   | ENL_T
                                   | DECI_T
```

```
<primary_s_relational_expression> → <primary_string_expression>
```