SLang LRM

Ray Siu (rls2121)

1. Lexical Conventions

1.1 Identifiers (Names)

Identifiers in SLang must begin with an upper or lower case letter and must be comprised of only letters, numbers, or the underscore character.

1.2 Reserved words

The following are reserved words in SLang and cannot be used as identifier names:

```
aclose   close    day    double   else
  for     high     if      int     load
  low     open   sdata   string  volume
while
```

1.3 Literals

1.3.1 String Literals

String literals must be prefaced with a double quote and must end in a double quote and may be composed of letter or digits. String literals may have any special character but may not have a double quote character in the body.

1.3.2 Integer Literals

Integer literals are composed entirely digits and may be prefaced by the ‘-‘ character denoting a negative value.

1.3.3 Double Literals

Double literals must be composed entirely of digits and must have a ‘.’ character. Doubles, however, must not begin or end in the ‘.’ character and may be prefaced by the ‘-‘ character denoting a negative value.

1.4 Other Characters

The follow characters are also used by SLang:

```
+       -       *       %       =
!=      :=      ::      ;       >
<       >=      <=      ++      --
}       {       ]       [       )
(       &&      ||
```

1.5 Comments

SLang supports only single line comments, lines that begin with '//' will be assumed to be comments and ignored by the SLang compiler.

2. Types

SLang supports a few basic data types and one complex object data type. For simplicity, SLang does not allow any type of type casting.

```
int       32 bit two's compliment
double    64 bit IEEE 754
string    String of Unicode characters
sdata     Stock data object
```

2.1 sdata type

The sdata is meant to represent the Y! finance data read in from file. There are a set of data elements which can be referenced by index:

```
Name      Type      Description
open      double    Opening price for the day
close     double    Closing price for the day
high      double    High price for the day
low       double    Low price for the day
volume    double    Volume for the day
aclose    double    Adjusted closing price
day       String    The day
```

In order to access these elements you must use '::' to index a specific element in a sdata object. For example given an sdata element, **s**, and an integer index, **index** :

```
s::open(index);
```

The snippet above would evaluate to the opening price for s on the day referenced by index.

3. Expressions

In SLang expressions are defined to be anything that evaluates. The order of precedence is determined by the precedence of the operators, a detailed precedence chart by operators can be found at the end of this section. Expressions grouped by the '(' and ')' characters are considered a single expression and have the highest possible precedence level.

3.1 Basic Expressions

3.1.1 Identifiers

By definitions an identifier is an expression that returns the value of the variable referred by the identifier.

3.1.2 Constants

A double or integer literals are expressions that evaluate to the value of the constant.

3.2 Unary Operators

An expression may be prefixed by '-', negating the value of the value that the expression evaluates to. An expression may be followed by '++' to increment the value that the expression evaluates to, or by '--' to decrement the value.

3.3 Binary Operators

3.3.1 Additive operators

The operators '+' and '-' represent addition and subtraction respectively. They are on the same precedence level and are applied to non-string expressions from left to right.

3.3.2 Multiplicative operators

The operators '*', '/', and '%' represent multiplication, division, and modulus respectively. They are on the same precedence level and are applied to non-string expressions from left to right.

3.3.3 Relational operators

The operators '<', '>', '<=', '>=', '!=', and '=' represent, less than, greater than, less than or equal to, greater than or equal to, not equal, and equal respectively. They are on the same precedence level and are applied to non-string expressions from left to right. Relational operators evaluate to a numeric value 0 if the statement is false and some non-negative, non-zero value if the statement is true.

3.3.4 Logical operators

The operators '&&' and '||' represent logical AND and logical OR respectively. They are on the same precedence level and are applied to non-string expressions from left to right. Logical operators evaluate to zero if the expression evaluates to false and a non-zero, non-negative value otherwise.

3.3.5 Order of Precedence

The following is a chart of the order of precedence of operators in SLang.

Unary sign operator	-
Multiplicative operators	* / %
Additive operators	+ -
Relational operators	< > <= >= = !=
Logical operators	&& ||

3.4 SDATA type expressions

The ‘load’ command is considered an expression which evaluates to an sdata object given a string literal which gives the URI pointing to stock data. The syntax of this operation is as follows.

```
load(“<URI_OF_STOCK_DATA>”);
```

The sdata data type allows the user to index values from the Y! finance data read from file. Specifically, the methods referenced in section {2.1} are expressions that evaluate to the value specified by the field name and index.

4. Statements

Statements in SLang are segments of code that do not evaluate. These are the base components of the language and constitute the logic control of the language.

4.1 Assignment Operators

Assignment operators are used to map a literal value or expression evaluation to an identifier. The basic format of an expression is as follows.

```
ID := EXPR;
```

The above snippet would assign the evaluation of the expression of EXPR to the identifier ID.

4.2 Conditional statement

SLang implements one conditional statement, the if-statement. The if statement branches on some expression, if the expression evaluates to some non-zero and non-negative value then the code in the ‘{..}’ block following the ‘if’ is executed. Otherwise the following block of code is omitted.

```
if(expr)
{
     stmt1;
     …
     stmtn;
}
stmtx;
```

In the above code if expr evaluates to zero, then the code would branch to stmtx, otherwise stmt1 would be executed. Note that in both cases stmtx would be executed. There may be conditional situations where the user may wish to execute that bit of code only when the expression evaluates to zero. This is allowed by using the optional 'else' statement.

```
if(expr)
{
     stmt1;
} else {
     stmt2;
}
```

In the above snippet if expr evaluates to zero the code would branch to stmt2, otherwise the code would branch to stmt1.

4.3 Iterative statements

SLang provides two mechanisms for iterative looping, the 'while' and the 'for' loops.

4.3.1 The FOR loop

There are two parts to the structure of a for loop, the condition and the body. The condition is in turn comprised of three parts delimited by ';' as follows.

```
for( stmt1 ; expr ; stmt2 )
{
     body_stmt;
}
```

The first statement, stmt1, is meant to be the initialization portion of the condition. This is where the initial value of the loop variable gets set. The second expression, expr, is the test for the loop. If the expression evaluates to a non-zero, non-negative value then the body of the loop gets executed, otherwise the loop is omitted. The final statement, stmt2, is where the loop variable gets incremented.

After the body_stmt is executed the code branches back to the condition and revaluate the condition expression expr and the final statement stmt2. Note that the body_stmt will continue to be re-evaluated until the expr in the condition evaluates to a zero. Also note that the initialization statement, stmt1, will only be executed the first time and not on subsequent iterations.

4.3.2 The WHILE loop

The while-loop is comprised of two parts the condition and the body.

```
while(expr)
{
     stmt;
```

```
}
```

If the expr evaluates to a non-zero, non-negative value then body, stmt, will be evaluated otherwise the code will skip the body of the while loop. After the body has been executed the code will then branch back to the condition expression to determine whether the body should be computed again.

4.4 Print statement

The print statement prints some parameters to the standard output of the system. Note that print is not analogous to the java "println" and will no automatically add a line feed. Furthermore print will only be able to take ONLY string literals or only a identifier.

```
Legal print statements:
     print("hello world");
     print(variable);

Not legal print statements:
     print("Hello world" + variable);
     print(variable + variable2);
```