IOE Language v1.0

Programming Language Assignment

5/11/2013

Scott Arcuri

Contents

1. Introduction

In One Ear (IOE) is a language designed to be syntactically similar to the way we speak while modeling the proper grammatical structure of literature. IOE is different than other languages because they were designed to be written, thus wielding large structures with commands that are difficult to memorize. These languages were conceived based on programming languages representing mathematical computations, and mathematical computations are generally clearer and better represented in writing. Nowadays, programming languages are more common, and are used for more purposes and tasks than when they were first theorized and indoctrinated. Mathematics can be done, but the purpose of IOE is to replace the complicated syntax with intrinsic functions that simplify the code increasing usability. Due to IOE's specific purpose, and attempt to represent words and sentences, make it similar to COBOL. IOE is an object oriented language.

1.1 Hello World Program Example

```
1  Display Hello World.
```

1.2 Program Structure

The Key organizational concepts of IOE are:

1. Sentences end with a ",", blocks end with a "." or ";". IOE prefers using a ";" to end a sub-block and a "." to end a block. If you always use a "." IOE will check for correctness. If you always use a ";" you will receive errors. When in doubt, use a ".", it will always work. The use of ";" allows more readable code. Using only "." would mean the last sentence in a block would have two "."'s.
2. Variables are defined with prepositions separating the name and value.
3. Comments are declared with the "Describe" keyword.
4. No namespaces/packages, other resources need to be present and linked to the compiler.
5. Strings are special types of single dimensional character arrays.

This example

```
1  Class shape is:
2      height is 0,
3      width is 0,
4      function area is return width times height;
5      function perimeter is return width times 2 plus height times 2;
6      function set height with parameter x is height is now x;
7      function set width with parameter x is width is now x.
8
9  rectangle is a shape.
10 set height of rectangle with 4.
11 set width of rectangle with 8.
12 x is area of rectangle.
13 y is perimeter of rectangle.
14 Display x.
15 Display y.
```

declares a class named shape that has four functions (area, perimeter, set height, and set width)and two variables. The output of the program is the area and perimeter of the rectangle. Notice to access the functions of the class we use the “of” preposition, to declare variables we use the “is” preposition, and to have functions use and pass parameters we use the “with” preposition. Since the “.” The entire block, sub-blocks are ended with a “;”, which can be seen in the class. Since a function’s name will always be followed by a preposition, they can be numerous words in length.

1.3 Types and Variables

IOE is weakly typed, similar to languages JavaScript and Perl. This decision was made because IOE is designed around ease instead of flexibility; the casting or conversion of types is implied and performed when the operation is called. As shown in the code example, IOE also has type inference; the automatic deduction of the type for expressions. The variables are also alive based upon static scope, and IOE has both primitive and reference type variables.

1.4 Statements Differing From Other Languages

Statement	Example
Expression	Function double with parameter x is x plus x.
Conditional	Function feet with parameter inches is: If inches is greater than 12, x is inches divided by 12; Otherwise x is 0; Display x.
Loop	x is 0. Loop until x is 10, Display x; x is now x plus 1.
For each	x is “Hello World”. For each element in x, x is now “e”.

1.5 Classes and Objects

Classes are created and defined using the “class” keyword. An example class can be seen below.

```
1  Class shape with parameters height and width is:
2      function area is width times height;
3      function perimeter is width times 2 plus height times 2.
4
5  rectangle is a shape with 3 as height and 5 as width.
```

To create an instance of a class, you use the “is” preposition followed by an “a” to indicate it is an instance. Parameters are defined with the “as” keyword, and the variable names need to be explicitly matched.

1.5.1 Accessibility

IOE has private, public, and protected accessibility.

Accessibility Type	Definition	Example
Public	Access is not limited. IOE defaults to public.	Public function double with parameter x is x plus x.
Private	Only accessible in current scope.	Private function double with parameter x is x plus x.
Protected	Only accessible in current scope or to inheriting objects.	Protected function double with parameter x is x plus x.

1.5.2 Fields

A field is a variable that is associated with a class, or an instance of one. Fields in IOE are declared with the preposition “is”.

```
1  x is 32.
```

Strings are one-dimensional character arrays in IOE to allow for iteration and access to its base methods.

1.5.3 Constructors

IOE implements constructors into its instance creation statement. You can pass parameters to the class, and any function call inside the root of the class will be called on initialization.

```
Class shape with parameters height and width is:
    function area is return width times height;
    function perimeter is return width times 2 plus height times 2;
    x is area;
    function get x is return x.
```

When this class gets initialized, the variable x is set to the area. This is because it is directly encapsulated in shape.

1.5.4 Arrays

An array is a data structure that contains a number of variables called elements, and is accessed via indices. All elements must be of the same element type. The following are examples of a single and multidimensional array.

```
lunch specials are chicken salad, roast beef sandwhich, decomissioned tank missile.
```

```
Combo meals is 4 by x array with
    chicken salad, jello, coke, cheesecake,
    burger, fries, coke, milkshake,
    quesadilla, apple, water, cherry pie,
    famine, war, pestilance, death.
```

Single dimensional arrays can be declared with “are”, and multidimensional arrays need to specify their size with the “by” keyword. Multidimensional arrays need the width portion identified, but the height can be left unspecified with a never initialized variable. Since no variables are declared with the string names, quotes can be left off; they are optional in this aspect.

2. Lexical Structure

2.1 Programs

IOE consists of one or more source files represented in Unicode characters. IOE compiles by:

1. Transforming a file of a particular character repertoire and encoding scheme into a sequence of Unicode characters.
2. Translates the Unicode input stream into tokens as part of the Lexical analysis.
3. Translates the stream of tokens into executable code as part of Syntactic analysis.

2.2 Grammars

This specification presents the syntax of the IOE where it differs from other languages.

2.2.1 Lexical Grammar Differences

<Assignment Operator> -> is | is now | are

<Mathematical Operator> -> plus | times | divided by | minus
-> + | * | / | -

<Keyword> -> <language defined> <variable defined>

<End of statement> -> , | ; | .

<End of block> -> .

2.2.2 Syntactic Grammar Differences

Differences in IOE's grammar can be seen in its literal variable structure:

<Literal variable> -> <Accessibility> <Variable data> <End>

<Accessibility> -> public | private | protected

<Variable data> -> <Variable name> is <Value>
-> <Variable name> are <Data>
-> <Variable name> is ℕ by ℕ array with <Data>

<Variable name> -> A…Z | a...z

<Data> -> <Value> <Data>
-> <Value>

<Value> -> ℝ
-> A…Z | a...z

<Literal variable> is the non-terminal representing primitive and reference type variables. Functions and classes have similar structures to variables.

<Class>	->	<Accessibility> <Class name> is <Class data>
<Class name>	->	A…Z | a...z
<Class data>	->	<Function> | <Function> <Class data>
	->	<Literal variable> | <Literal variable> <Class data>
	->	<Literal variable> <Mathematical Operator> <Literal variable> <Class data>
	->	<Loop> | <Loop> <Class data>
	->	<Conditional> | <Conditional> <Class data>
<Function>	->	<Accessibility> <Function name> is <Function data>
<Function name>	->	A…Z | a...z
<Function data>	->	<Literal variable> | <Literal variable> <Function data>
	->	<Literal variable> <Mathematical Operator> <Literal variable> <Function data>
	->	<Loop> | <Loop> <Function data>
	->	<Conditional> | <Function> <Class data>
<Loop>	->	Loop until <Literal variable> is <Conditional>
<Conditional>	->	if <Literal variable> is <Conditional operator>
	->	and <Conditional>
	->	or <Conditional>

2.3 Lexical Analysis

2.3.1 Comments

Comments in IOE need to be linked to a function, class, or variable. This works because function, class, and variable names become keywords during compile. You can also use the "this" keyword to attach the comment to the parent container. IOE builds documentation based upon comments, and can be sorted by container and keyword. An example can be seen below.

```
1  Describe double as a function that returns the input x 2.
2  function double with parameter x is
3      return x + x.
4
5  function to determine if positive with parameter x is
6      Describe this as a test for negatives;
7      if x is less than zero return false;
8      Describe this as test for positive;
9      if x is not less than zero return true.
```

2.4 Tokens

There are several kinds of tokens: identifiers, keywords, literal, operators, and punctuators. White space and comments are not tokens, though they act as separators where needed.

tokens:
- identifier
- keyword
- binary-number-literal
- character-literal
- boolean-literal
- operator-or-punctuator

2.4.1 Keyword Differences

A keyword is an identifier-like sequence of characters that is reserved, and cannot be used as an identifier except when prefaced by the @ character.

New Keywords:

Describe	quote	is	now	are	with	as	a
Loop	by	plus	minus	divided	times	and	parameters

Removed Keywords:

Do	for	//	try/catch/finallyswitch
Namespace/import		void	

3. Types

IOE has two main types, Primitive and Reference.

3.1 Value Types

Type	Description
Binary	Represents numerical numbers such as short, int, or long. It uses a similar fashion to COBOL's PICTURE clause, but it is implicit.
Character	Represents a single Unicode character.
Boolean	True or Flase

3.2 Reference Types

IOE has few reference types as IOE does not support boxing/unboxing.

Type	Description
Arrays	A collection of elements.

4. Examples

4.1 Even and Odd filter

```
function check if even with parameter x is:
    if x mod 2 is 0,
        return true;
    otherwise return false.

y is 2.
check if even with y.
```

4.2 Fibonacci sequence

```
function fibonacci with parameter old num and new num is:
    temp is new num;
    new num is now new num plus old num;
    old num is now temp;
    Display new num;
    return new num.

x is 0.
y is 1.
Display x.
Display y.
Loop until y is 144,
    y is fibonacci with x and y.
```

4.3 Making Change

```
function change with parameter x and y is:
    int z is x/y;
    return z.

money is 32.00
ten dollar bill is change with money and 10.
money is now money minus ten dollar bill times ten.
five dollar bill is change with money and 5.
money is now money minus five dolar bill times five.
single dollar bill is change with money and 1.
money is now money minus single dolar bills.
quarter is change with money and .25.
money is now money minus quarter times .25.
dimes is change with money and .10.
money is now money minus dimes times .10.
nickel is change with money and .05.
money is now money minus nickel times .05.
penny is change with money and .01.
money is now money minus penny times .01.
```

4.4 Prime Number checker

```
function prime with parameter x is:
    int y is 0;
    returnValue is true;
    Loop until y is equal to x,
        if x mod y is 0,
            returnValue is false;
        y is now y plus 1.
    return returnValue.

z is prime with 32.
```

4.5 Caesar Cipher

```
Function encrypt with parameters x and shift is:
    For each element in x,
        element is now char((ord of element plus shift) mod 26);
    Display x;
    return x.

Function decrypt with parameters x and shift is:
    shift is now 26 minus shift;
    s is encrypt with x and shift.

Function solve with parameter x is:
    u is 0;
    Loop until u is 26,
        t is encrypt with x and u.

x is "Hello World".
shift is 7.

x is now encrypt with x and 7.
x is now decrypt with x and 7.
solve with x.
```

5. Conclusion

In conclusion, I hope that IOE brings a unique language to the table. This is not a clone of COBOL with its same intent, purpose, and syntax; or any other language for that matter. It was designed with one purpose: To be readable by resembling spoken word patterns. With this simple goal in mind, IOE hoped to increase write-ability, increase readability, and act as an introductory language that enforced common practices while sacrificing flexibility for simplicity. In my review of the language thus far, there are some quips I'd like to fix with future versions. Currently, the statement terminals tend to be slightly annoying. Nesting with commas, semi-colons, and sometimes periods add confusion to the program. A new programmer may wonder which one should be used or may get hung up on syntax errors. In future versions I hope to make nested statements not need any fragment terminals, and that the compiler would know which block it belonged to based on the nesting structure like Python; only the end of the block would have a period. This would allow new programmers to understand how to write their code while enforcing readability while alleviating some of the stress in selecting the proper way to end a code phrase. A feature that went into the design of the language was that programs could have been shared or programmed through voice, but the nesting issues seem to get in the way. I believe the Python way would be a step in the right direction, with an optional indent keyword to be used in vocal situations.