Bart's blog

A blog about software development.

WEDNESDAY, MARCH 02, 2011

1. *TL* specification

This is part 1/9 of ***Creating your own programming language with ANTLR****.*

The language we're going to interpret is called *Tiny Language* (*TL* henceforth). It will be able to do a bit more than the most basic mathematical expression parsers, that usually only support the most common arithmetic operators like `+`, `-`, `*` etc., and perhaps variable assignments.

Besides being able to parse and evaluate boolean- and numerical expression, in *TL* you will also be able to define functions, recursively call these functions, use control statements like `for` and `while` and it will support a list data type, to name just a few features.

Below is a list of supported features/operations of *TL*. This list is no strict definition, but more an example of the syntax. A proper **BNF** grammar (or something closely like it) will be created at a later stage in the tutorial.

Basic types

TL supports 4 data types: number, string, boolean and a list (and there's `null` too).

view plain print ?

```
// Number
1
3.14159265
0.1

// Boolean
true
false

// String
'foo bar baz'
'a " b'
"c ' d"

// List
[]
[1, 2, 3, 'str']

// Null
null
```

Additive expressions

Addition and subtraction:

view plain print ?

```
3 + 8
4 - 78
[3,4,5] - 4
```

Multiplicative expressions

Multiplication, division and modulus:

view plain print ?

```
9 * 3
5 / 2
5 % 3
"ab" * 3
```

Power expressions

Raising to the power:

view plain print ?

ARCHIVE

▶ 2012 (5)
▼ 2011 (11)
▶ May (1)
▼ March (10)
9. Room for improvement
8. Interpreting and evaluating TL II
7. Interpreting and evaluating TL I
6. Creating a tree grammar
5. Building an AST
4. Syntactic analysis of TL
3. Lexical analysis of TL
2. Introduction to ANTLR
1. TL specification
Creating your own programming language with ANTLR....

```
2^10
4^2^3
```

Equality expressions

Equals and not-equals:

view plain print ?

```
3 == 1+2
1 != 2
"ab" * 3 == "ababab"
[3,4,5] - 4 == [3,5]
```

Relational expressions

Less than, greater that, less than or equals and greater than or equals:

view plain print ?

```
5 >= 5
1 < 2
'B' > 'A'
```

Unary operator

Negative minus and boolean negation:

view plain print ?

```
-3 == 1-4
true == !false
```

Built-in functions

There are four built-in functions: `println(...)`, `print(...)`, `assert(...)` and `size(...)`:

view plain print ?

```
println('print me  with  a line break');
print('print me  without  a line break');
assert(true); // if false, an error is thrown
assert(size([1,2,3]) == 3);
assert(size('a"b') == size('a\'b'));
```

Variable assignment

As you can see by the example below, *TL* is **dynamically typed**.

view plain print ?

```
lst = [10, 20, 30];
temp = true;
x = 12 + 34;
x = temp;
lst = [1,2,3,[4,5,6]];
lst[3][0] = 44;
```

Variable lookup

view plain print ?

```
lst = [[10,11,12], 20, 30, "ABC"];
temp = lst[1];
assert(temp == 20);
assert(lst[0][2] == 12);
assert(lst[3][0] == "A");
```

Function declaration and invoking

view plain print ?

```
def twice(n)
  return n+n;
end

assert(twice(5) == 10);
```

Control flow statements

There will be two types of control flow statements: `for-` and `while` statements:

view plain print ?

```
sum = 0;
```

```
for n = 1 to 5 do
  sum = sum + n;
end

assert(sum == (1+2+3+4+5));

steps = 0;
while sum > 0 do
  sum = sum - 5;
  steps = steps + 1;
end

assert(steps == 3);
```

If statement

view plain print ?

```
if 1+1 == 2 do
  assert(true);
end

if A do
  // ...
else if B do
  // ...
else do
  // ...
end
```

Variable scopes

`if`-, `for`- and `while`-statements all have their own scope, and if a variable is not defined in their own scope, they will look at their parent scope for it (and so on).

The following examples will clarify this:

view plain print ?

```
a = 2;

if true do
  b = a + a;
  assert(b == 4);
end

println(b); // ERROR: `b` is local in the `for` statement!
```

However, functions have their own private scope which has no access to a scope above it. This is illustrated in the following example:

view plain print ?

```
a = 1;

def twicePlusOne(n)
  return (n * 2) + a; // ERROR: `a` is not in the scope of the function `twicePlusOne`

end

def test(a)
  assert(a == 1);
  a = 100; // `a` is now a local variable
  assert(a == 100);
end

test(a);

assert(a == 1); // `a` is still 1
```

Okay, that's a quick overview of the language we're going to create an interpreter for using ANTLR. If you're still enthusiastic about it, continue reading **part 2: Introduction to ANTLR**.

Posted by Bart Kiers at **2:27 PM**
Labels: **antlr**, **dsl**, **java**, **parsing**

No comments:

Post a Comment

Newer Post **Home** **Older Post**

Subscribe to: **Post Comments (Atom)**