Ruby Parser 2: Build AST tree

Homework Summary

For this part of the Ruby Interpreter Collection of Assignments we will be modifying our existing Parser so that instead of printing parser output, it will build an **AST** (abstract syntax tree) that our interpreter can use to "run" programs written in the Tiny Language.

Details

I have given you all the files necessary to complete the assignment. You will need to modify **Parser.rb** so that it builds the AST while it is parsing a file.

I have included an **AST.rb** file that defines an AST node object that can be used to build this tree. It also has a method that can print the tree in list format (convenient since our interpreter will be written in a list-based language!).

You should NOT modify AST.rb unless you are attempting the extra credit (see Extra Credit section below).

I have done some of the work for you in Parser, so that you can use that as an example of how you should proceed. At the end of this document are screenshots showing what correct output should look like given some input files that I will also provide to you for debugging.

Note: We are building an **AST** and **NOT** a parse tree. What is the difference anyway!? The difference is that a

Parse Tree

parse tree would look much like the examples we went through in class when we compared parse trees to derivations. They would INCLUDE every rule in our class as root nodes or sub nodes in the tree. However, an

AST (Abstract Syntax Tree)

AST **ONLY** includes the actual lexemes that are needed to be interpreted. If you take a look at the compiler series article here: https://ruslanspivak.com/lsbasi-part1/ , they illustrate that in one of the series articles. Here is the specific article in the series that introduces AST's: https://ruslanspivak.com/lsbasi-part7/ .

For our assignment, we will always start with a program node as the root of the whole tree. That is the only "not concrete" thing that should be part of our tree.

Extra Credit (25 pts) – must score 100 to qualify for any extra credit points. Otherwise, you only get 20 pts per test case passed.

Part 1 (5 pts)

Because of the way our grammar is defined (and because we want to produce a tree in list format) our tree **WILL** be nested correctly **BUT** our operands will be in reverse order. Since it is something that is consistent, it is an easy problem to handle in our interpreter (we already know it will behave this way, so just read in the operands in reverse order).

However, it is possible to produce the tree in list form with the operands in correct order. In order to do this, you will need to modify the **AST.rb** class. One way to do this is to:

1) add a method in **AST.rb** that allows you to swap the first child of a node
2) instead of calling **addChild** everywhere in **Parser.rb**, there are at least 2 instances where you will need to call your new custom function instead (maybe called **addAsFirstChild**).

Part 2 (10 pts)

This potential issue is a bit harder to fix (and describe) and requires that you (1) have a really good understanding of exactly how this AST tree works and (2) how our interpreter is going to process our tree (specifically that any math

operator (except for equal) requires at least 2 operands). So, (- 3 4) is ok, but (- 4) is not. Scheme can actually handle this, but our interpreter won't be able to. See examples below.

To get a really good understanding of how our AST tree is being built, I would recommend studying the **toStringList()** method in AST.rb to understand how its going through the tree in order to print things. You could then create a second **toStringList()** in AST.rb that is a modification of the first one. You could attempt to create this second one so that instead of printing the Token AND the Lexeme, it ONLY prints the lexeme. This will actually end up giving you something you can copy and paste into scheme to verify if it's computing the answer correctly.

Example of the problem:

1 – 2 – 3 – 4 is -8.

You can type that into a calculator and get the correct result. However, you can't type that into scheme (the way it is written) and get a correct result. Since scheme is a list-based language, it expects that the first item in the list be the operator.

If you don't attempt the first part of the extra credit (5 pts), this is what your parser will produce. You can get this string exactly, by simply removing the Token names (or by writing a second **toStringList()** method that will print lexemes only). In this example, we are not reversing first operand (we can handle this in Interpreter). You can paste the partial tree below into scheme and it will produce a result, however, our interpreter will not be able to process it because there is an operator with just a single operand in this list; the inner **(- 4)**.

(- 2 (- 3 **(- 4)**) 1) ← scheme will give you -6 as a result, which is still incorrect, even though it runs

If you do attempt the first part of the extra credit (5 pts), this is what your parser will produce. Note that we STILL have that inner list **(- 4)** where the minus operator has only a single operand, the 4. Although you could type this expression into scheme and it would give you a correct result, our interpreter will still crash because of the the inner list **(- 4)**.

(- 1 2 (- 3 **(- 4)**))

Below is what you get 10 points for!

In addition to swapping the first child of some nodes (part 1) you will also occasionally need to "reverse" an entire sub-tree. You can think about this as if you were "flipping" a sub-tree over, where the bottom (the leaf/leaves) become the new root and the top (the root) becomes the new leaf/leaves. Doing this would produce a tree that scheme could use to correctly calculate a result AND our interpreter will ALSO be able to use the tree to produce the correct output. Continuing the example above, this method would produce the following list that would be correct for our interpreter.

(- (- (- 1 2) 3) 4) ← I'll go over this in class

Ruby Parser 2: Build AST tree
Screenshots

No extra credit

input1.tiny (20 pts)

```
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ cat input1.tiny
rubyrdog = 1
x = (2 + 3)/ 4
y = dog / (x + 3)
print x + y  rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ ruby main.rb
There were 0 parse errors found.
 ( program program ( equalop = id rubyrdog int 1 ) ( equalop = id x ( divop / int 4 ( addop + in
t 3 int 2 ) ) ) ( equalop = id y ( divop / ( addop + int 3 id x ) id dog ) ) ( print print ( add
op + id y id x ) ) )
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$
```

input2.tiny (20 pts)

```
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ cat input2.tiny
x = 3
y = 5
z = (x + y) / 3
print z
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ ruby main.rb
There were 0 parse errors found.
 ( program program ( equalop = id x int 3 ) ( equalop = id y int 5 ) ( equalop = id z ( divop /
int 3 ( addop + id y id x ) ) ) ( print print id z ) )
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$
```

input3.tiny (20 pts)

```
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ cat input3.tiny
x = (2 + 3)* dog
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ ruby main.rb
There were 0 parse errors found.
 ( program program ( equalop = id x ( multop * id dog ( addop + int 3 int 2 ) ) ) )
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$
```

input4.tiny (this one has syntax errors) (20 pts)

```
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ cat input4.tiny
x + y = 3
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ ruby main.rb
Expected equalop found +
There were 1 parse errors found.
 ( program program assignment assignment ( equalop = id y int 3 ) )
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$
```

input5.tiny (this one has syntax errors) (20 pts)

```
rsardinas@DESKTOP-1080ti:~/ruby_programs/ruby_parser2/key$ cat input5.tiny
print x + y;
rsardinas@DESKTOP-1080ti:~/ruby_programs/ruby_parser2/key$ ruby main5.rb
Expected id found ;
There were 1 parse errors found.
 ( program program ( print print ( addop + id y id x ) ) assignment assignment )
```

Ruby Parser 2: Build AST tree

Extra credit (5 pts only) – MUST score 100 to get the +5 points, otherwise you simply get 20 pts for each test passed. This is for correcting the order of the operands in the tree. (Non-Boolean Grammar)

input1.tiny (20 pts)

```
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ cat input1.tiny
rubyrdog = 1
x = (2 + 3)/ 4
y = dog / (x + 3)
print x + y  rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ ruby main.rb
There were 0 parse errors found.
 ( program program ( equalop = id rubyrdog int 1 ) ( equalop = id x ( divop / ( addop + int 2 in
t 3 ) int 4 ) ) ( equalop = id y ( divop / id dog ( addop + id x int 3 ) ) ) ( print print ( add
op + id x id y ) ) )
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$
```

input2.tiny (20 pts)

```
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ cat input2.tiny
x = 3
y = 5
z = (x + y) / 3
print z
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ ruby main.rb
There were 0 parse errors found.
 ( program program ( equalop = id x int 3 ) ( equalop = id y int 5 ) ( equalop = id z ( divop /
( addop + id x id y ) int 3 ) ) ( print print id z ) )
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$
```

input3.tiny (20 pts)

```
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ cat input3.tiny
x = (2 + 3)* dog
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ ruby main.rb
There were 0 parse errors found.
 ( program program ( equalop = id x ( multop * ( addop + int 2 int 3 ) id dog ) ) )
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$
```

input4.tiny (this one has syntax errors) (20 pts)

```
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ cat input4.tiny
x + y = 3
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ ruby main.rb
Expected equalop found +
There were 1 parse errors found.
 ( program program assignment assignment ( equalop = id y int 3 ) )
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$
```

input5.tiny (this one has syntax errors) (20 pts)

Ruby Parser 2: Build AST tree

```
rsardinas@DESKTOP-1080ti:~/ruby_programs/ruby_parser2/key$ cat input5.tiny
print x + y;
rsardinas@DESKTOP-1080ti:~/ruby_programs/ruby_parser2/key$ ruby main5.rb
Expected id found ;
There were 1 parse errors found.
 ( program program ( print print ( addop + id x id y ) ) assignment assignment )
```

Extra credit (10 pts) – MUST score 100 AND get 5 pts extra credit to qualify for these points. Each test worth 5 pts. This is for resolving the issue where you end up with a single operand as a leaf to a subtree (Non-Boolean Grammar).

input6.tiny

```
rubydog = 1 - 2 - 3 - 5 - 6 - 7 - 8 / 2 / 2 / 9 * 2
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ ruby main.rb
There were 0 parse errors found.
 ( program program ( equalop = id rubydog ( minusop - ( minusop - ( minusop - ( minusop -
( minusop - ( minusop - int 1 int 2 ) int 3 ) int 5 ) int 6 ) int 7 ) ( multop * ( divop /
 ( divop / ( divop / int 8 int 2 ) int 2 ) int 9 ) int 2 ) ) ) )
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$
```

input7.tiny

```
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ cat input7.tiny
print 1 - 2 - 3 - 4
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$ ruby main.rb
There were 0 parse errors found.
 ( program program ( print print ( minusop - ( minusop - ( minusop - int 1 int 2 ) int 3 ) int 4 ) ) )
rsardinas@DESKTOP-1080ti:~/ruby_programs/parser2$
```

Extra credit (10 pts) – MUST score 100 AND get 5 pts extra credit to qualify for these points. Each test worth 5 pts. These are for implementing the Boolean version of the grammar.

input8.tiny

```
rsardinas@scarstrix22:~/ruby_stuff/parser2$ ruby main8.rb
There were 0 parse errors found.
 ( program program ( equalop = id x int 10 ) ( whileop while (
 gt > id x int 0 ) ( thenop then ( print print id x ) ( equalo
p = id x ( minusop - id x int 1 ) ) endop end ) ) )
```

input9.tiny

```
rsardinas@scarstrix22:~/ruby_stuff/parser2$ ruby main9.rb
There were 0 parse errors found.
 ( program program ( equalop = id x int 10 ) ( ifop if ( gt >
id x int 0 ) ( thenop then ( print print id x ) ( equalop = id
 x ( minusop - id x int 1 ) ) ( print print id x ) endop end )
 ) )
```