Tyler Hsieh
920216320

Assignment 3 Documentation

Github Repository

https://github.com/sfsu-csc-413-fall-2022-roberts/assignment-3---parser-Tyler9648

Project Introduction and Overview

The parser project required me to add more tokens to the parser, remove debug statements, and properly align the generated image of the programs AST.

Scope of Work

Task	Completed
Add more tokens:	Yes
Timestamp Type	Yes
Timestamp Literal	Yes
Utf16StringToken Type	Yes
Utf16StringToken Literal	Yes
GreaterThanEqual >=	Yes
GreaterThan >	Yes
Switch	Yes
If (without else)	Yes
Case Block	Yes
Case Statement	Yes
Default Statement	Yes
Case List	Yes
Remove all debug statements	Yes
OffsetVisitor	Yes
DrawOffsetVisitor	Yes

Execution and Development Environment

I used Intellij IDE to code and compile everything for my program on MacOS.

Compilation Result

```
sfs-wifi-dhcp-10-143-142-105:assignment-3---parser-Tyler9648-main Tyler$
javac compiler/Compiler.java
sfs-wifi-dhcp-10-143-142-105:assignment-3---parser-Tyler9648-main Tyler$ java
compiler.Compiler simple.x
1 program { int i int j
2    i = i + j + 7
3    j = write(i)
4  switch( i ){
5
6  case [1]# :
7  j = i
8
9  case [2, 3]# :
10  i = 3
11
12  default # :
13  i = 6
14
15  }
16
17 }
---------------AST-------------
1:  Program
2:    Block
5:      Decl
3:        IntType
4:        Id: i
8:      Decl
6:        IntType
7:        Id: j
10:     Assign
9:        Id: i
14:       AddOp: +
12:         AddOp: +
11:           Id: i
13:           Id: j
15:         Int: 7
17:     Assign
16:       Id: j
19:       Call
18:         Id: write
20:         Id: i
21:     Switch
22:       Id: i
```

```
23:       SwitchBlock
24:         Case
25:           Int: 1
27:           Assign
26:             Id: j
28:             Id: i
29:         Case
30:           Int: 2
31:           Int: 3
33:           Assign
32:             Id: i
34:             Int: 3
35:         Default
37:           Assign
36:             Id: i
38:             Int: 6
```

Also outputted a simple.x file. Program runs as expected with no errors.

Assumptions

I had to assume that my previous assignment 2 lexer fully works with the parser prior to making any changes. Based on this, I replaced all of the lexer files in assignment 3 with the working lexer I made in assignment 2.

I also had to assume that DrawOffsetVisitor would completely replace DrawVisitor so I removed DrawVisitor and used the source code of DrawVisitor as a base for DrawOffsetVisitor.

Another assumption I made was that the case list for the switch statement were just a list of integers that represented the case number. Because of this, case list can only be numbers, and a case must have at least one corresponding case number.

The most important assumption I had to make was that the AST class code and the tree visitor algorithm provided by the professor fully works. Based on this assumption, I used code for CountVisitor as the base for OffsetVisitor. For example, the count method in CountVisitor is modified to also properly offset the AST objects in OffsetVisitor, and the name of the method changed to offset.

Implementation

AST trees
I added a new tree class for some of the new tokens added. This includes: TimestampType, TimestampLit, Utf16StringType, Utf16StringLit, Switch, Default (switch case), Switch case, and Switch block.

Relational Operator Tokens
GreaterThanEqual and GreaterThan were the only two relational operator tokens added, and only needed to be added in EnumSet relationalOps in the parser.

If without else
The If without else was rather simple to add. In the part of the rStatement method where it checks for If tokens, I had it check if there was an else token after the if statement block. If there was, it'd expect an else token then also add the else block. Otherwise, if there was no else token next, it returns an if AST tree without the else block.

Switch
The switch was the most complex and difficult statement to add due to the many components of it such as the switch itself, the identifier, the switch block, the case statements and default statements which are inside the switch block, and the actual statements and case numbers (case only) inside the case and default statements.

Switch
When the switch token is the next token, a switch tree class is created. Under the switch tree is the identifier and switchBlock.

Switch Block and Case Block
I created a caseBlock method that creates a switchBlock tree and adds caseBlocks to it as kids. In my code, it should be noted that the terms caseBlock and case are the same and refer to each case as a block. The case statement method I made is called in order to add the case statements and numbers for each caseBlock.

Contents of Case Block
The caseStatement method returns a case tree with a statement and case number(s). After all the normal case statements are added, the caseBlock method then checks if there's a default token for the default case, and adds a default case accordingly.

It should be noted that I also added a startingCase method that works similarly to startingDecl and startingStatement, that checks if the next token is a Case token or not.

OffsetVisitor

OffsetVisitor works similarly to CountVisitor in the sense that it is used for the layout of the outputted AST image. Because of this, the base code from OffsetVisitor is from CountVisitor.

The offset method in OffsetVisitor recursively sets the offset for all AST objects and their kids. If the mean offset of the kids is greater than the current offset, then current offset is replaced by the mean offset. If the mean offset of the kids is less than the current offset, then the adjust method is called to recursively increment the offset of the kids so the mean of the kids is the same as the parents offset.

The offset value of each AST object is stored inside a private HashMap in OffsetVisitor.

DrawOffsetVisitor

My base code for DrawOffsetVisitor is from DrawVisitor, as for this assignment, DrawOffsetVisitor essentially completely replaces DrawVisitor. Two more parameters are added onto DrawVisitors draw method, which are taken in the AST HashMap for offsets and the maxOffset from OffsetVisitor. Only parts from DrawVisitor that remain untouched are vertical attributes such as height and vertical separation.

The width of the entire image is now the maxOffset times the width of a node and the horizontal gap between nodes.

The offset of each node (used by draw method) is the corresponding offset of the object from the OffsetVisitor's HashMap, times the total horizontal space for each node (node's width and horizontal gap/separator).

The endx which is used when drawing the line between nodes is changed to the kid's offset times the hstep plus half of the nodes width.

Asides from these changes, I left the rest of the original code from DrawVisitor relatively untouched.

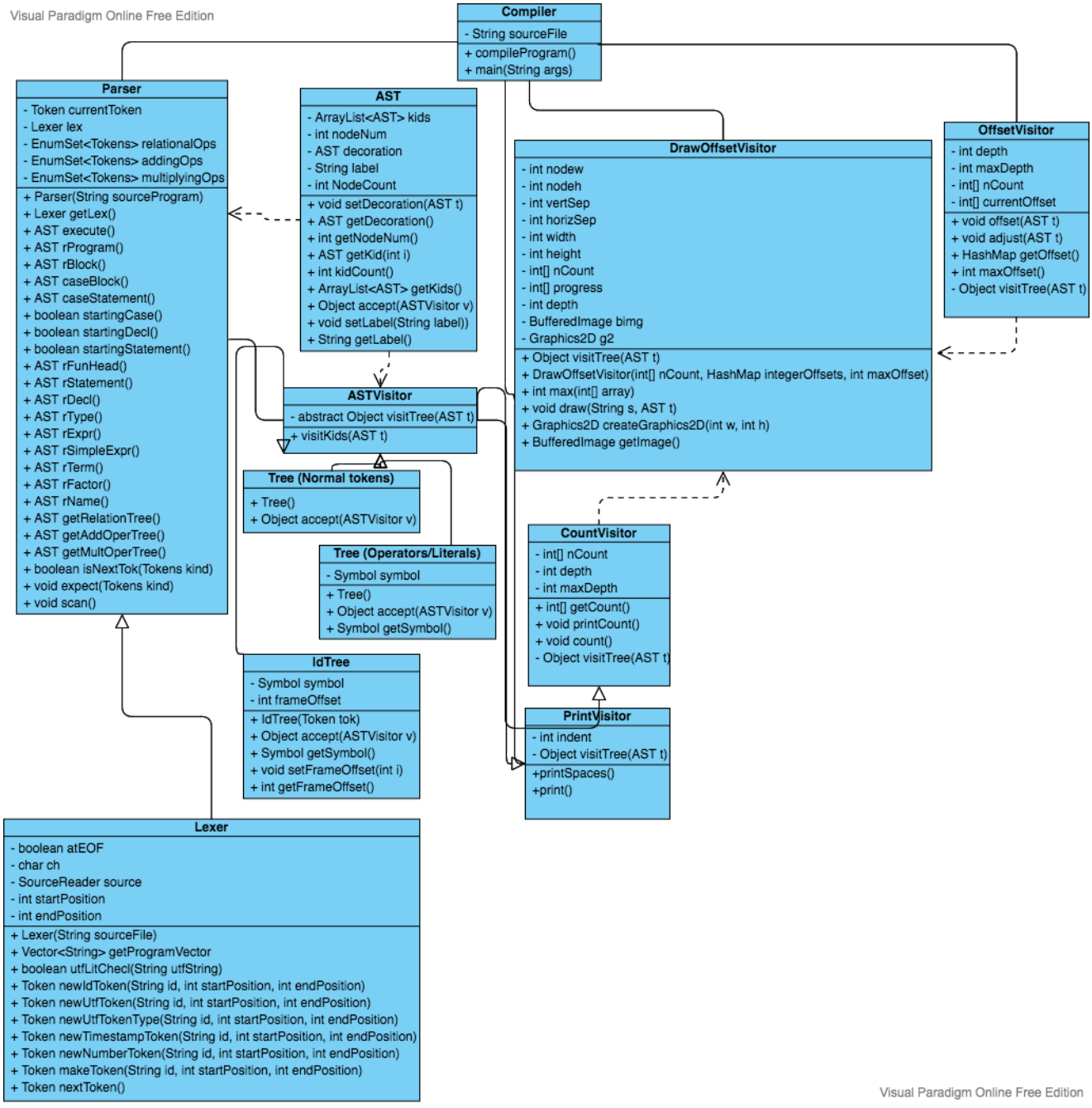
Removing debug statements

To remove the debug statements, I just removed print statements from SourceReader in Lexer, and print statements from compileProgram in the compiler.

Code Organization

I kept the original file structure given by the professor as it is already organized. AST visitors are all kept in the visitor folder, the lexer code from assignment 2 is kept under the Lexer folder, Parser kept under the parser folder, AST tree classes are kept under the AST folder, and test files are kept in the samples files folder, although I did move some of them out for convenience when testing.

Class Diagram



Results and Conclusions

This project helped me grasp the concept of programs that have many classes and objects. It also helped me improve my use of recursion in my algorithms, such when I implemented it in the offset visitor.

I successfully implemented new tokens, OffsetVisitor, and DrawOffsetVisitor. I also removed debug statements, and made the else after if optional.

Challenges

Some of the major challenges I had while coding was properly offsetting the grandchild of a parent, as only the child/parent would correctly offset. I eventually used recursive methods in OffsetVisitor to keep offsetting the children of a parent and the children of the children etc.

Implementing the switch statement was also a bit tricky for me as there were multiple blocks and statements within the switch, along with case matches. I eventually created a new method that checks for the case keyword and a caseBlock method which took inspiration from the rBlock method.

Removing the debug statement was also a bit annoying as I had to dig through the lexer's code to find and remove the print statements, which were in SourceReader.