Yan Peng
921759056
September 26, 2021

Assignment 3 Documentation

GitHub Repository

Link for assignment 3:

https://github.com/sfsu-csc-413-fall-2021/assignment-3-parser-yuqiao1205

Project Introduction and Overview

The aim of this project is to understand the function of parser in the compilation process. Parsing can be defined as a process of analyzing a text which contains a sequence of tokens, to determine its grammatical structure with respect to a given grammar.

This project takes the tokens generated at the lexical analysis phase and transforms them into a data structure -- abstract syntax tree.

Summary of technical work: To continue the work in the compiler codebase and extend the compiler to handle the new tokens and production rules, I updated the AST, compiler, parser, and visitor classes. Also, I created two visitors OffsetVisitor and DrawOffsetVisitor to output the tree diagram where the nodes are correctly aligned. Meanwhile, I added few new production rules for list, if-then-else, doloop, etc. After adding the previously mentioned classes and methods, I can run the compiler.Compiler with some of the sample files in /sample_files/ to see if the correct output is displayed.

Scope of Work

Task				Completed
Test the implementation with production rules that test all possible cases. The following cases:				✓

if E then case	program { int i int j
	if (i>=10) then {
	y = 5
	}else{
	y = 6
	}
		valid numberlit datelit case	program { int i int j
			number n
			n = 2.36333
			}
				✓

doloop until case	program{
	doloop{
	j = write(i)
	} until(x > 5)
	}
		for in List else case	program {
			for i in [1,3,5] {
			y = 8
			} else {
			y = 9
			}
			}
				✓

example case (simple. x)	program { int i int j
	i = i + j + 7
	j = write(i)
	}
		valid datelit case	program {
			date n
			n = 1~2~12
			m = 1~2~1980
			k = 12~30~00
				✓

The following new production rules added:				✓
if E then Block				✓
doloop Block until E				✓
for Name in LIST BLOCK else BLOCK				✓
Created rList() for LIST rules				✓
The following new tokens added: doloop, until				✓
Created following classes in package ast for distinct trees:				✓
DateTree/DateTypeTree/DoloopTree/NumberTree/NumberTypeTree/ForTree/ListTree				✓
added Tokens.Greater, Tokens.GreaterEqual to EnumSet<Tokens> relationalOps				✓
Added isNextTok(Tokens.DateType) || isNextTok(Tokens.Number)) in startingDecl() method				✓

Task	Completed
Updated startingStatement() adding isNextTok(Tokens.Else) || isNextTok(Tokens.Doloop) || isNextTok(Tokens.For)	✓
Updated rType() adding Token.Number and Token.DateType	✓
Rewrote production of statement for if E then BLOCK without else in rStatement()	✓
Created production rule for for NAME IN LIST BLOCK ELSE BLOCK in rStatement()	✓
Created production rule for doloop BLOCK until E	✓
Updated rFactor() adding Tokens.NumberLit and Tokens.DateLit	✓
Created Class OffsetVisitor.java in package visitor to calculate the offset position	✓
Created Class DrawOffsetVisitor.java in package visitor to display a tree	✓
Copy changed code from Assignment 2 in this directory	✓
Removed all debug statements not required includes the listing of tokens from the assigment 2 output	✓

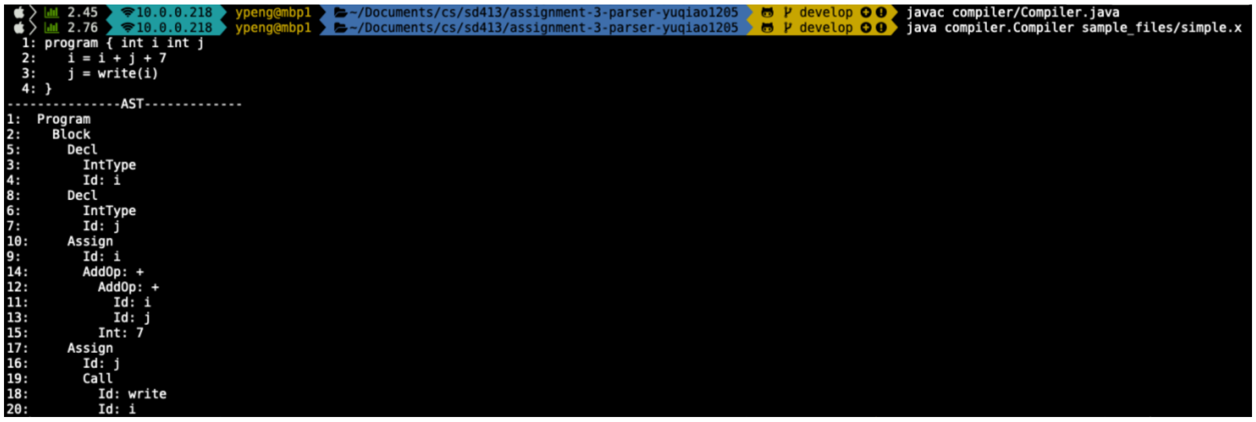
Execution and Development Environment

I developed from this code base using IntelliJ IDEA Community v. 2021.1.3. This Java application was compiled using Java JDK version 11.0.11 which is the most up to date version of the JDK.

Compilation Result

Using the instructions provided in the assignment one specification:

```
> javac compiler/Compiler.java
> java compiler.Compiler sample_files/simple.x
```



```
 2.45  10.0.0.218  ypeng@mbp1  ~/Documents/cs/sd413/assignment-3-parser-yuqiao1205  develop  javac compiler/Compiler.java
 2.76  10.0.0.218  ypeng@mbp1  ~/Documents/cs/sd413/assignment-3-parser-yuqiao1205  develop  java compiler.Compiler sample_files/simple.x
  1: program { int i int j
  2:    i = i + j + 7
  3:    j = write(i)
  4: }
---------------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
```

No error messages or warnings were displayed, and the application ran as expected

Assumptions

I assumed some tokens may be malformed and implemented some error handling for these invalid tokens.

Implementation

Compiler

The Compiler class contains the main() method of our application. We read and compile a single source file. I created drawOffsetDiagram() method using the OffsetVisitor and DrawOffsetVisitor to output the correct tree where the nodes are correctly aligned. I noted a few cases of duplicated code. Once I had the general algorithm working, I refactored this code into the displayImage() method.

ASTVisitor and the Visitor Design Pattern

Implementing this project required me to understand and implement the Visitor pattern as described in class. The Visitor pattern defines a new operation without introducing modifications to an existing object structure.
The AST and ASTVisitor classes provide the abstract base class for our implementation of this pattern, the AST specifying an abstract method accept (), which accepts an instance of a ASTVisitor subclass to each element of the structure. The interface of ASTVisitor would be implemented in derived classes to provide the varying behavior for each of the node types in the AST. All concrete classes created are under the ast packages.

OffsetVisitor

First, the OffsetVistor class extends ASTVisitor class to accept each element for varying behavior. I created LinkedHashMap mapping the Node to the correct offset. I created a small data class Position to hold the x (offset) and y (depth) information. Then, I implemented calculateOffset() method by traversing tree to get each kid's position and used the algorithm to set the offset for root or leaf. I implemented the adjustOffset() method to recursively adjust the offset of the node by an additional offset and set next available offset.

DrawOffsetVisitor

In order to draw the correct tree with the nodes correctly aligned, I pass an OffsetVisitor, which contains a mapping of nodes to positions, to the DrawOffsetVisitor. I created getMaxOffset() by traversing the AST to find the maximum number of nodes in a row and hence calculate the width of image size. Also, I implemented the draw() method to draw nodes in the correct positions along with lines to form the display of the tree.

Position

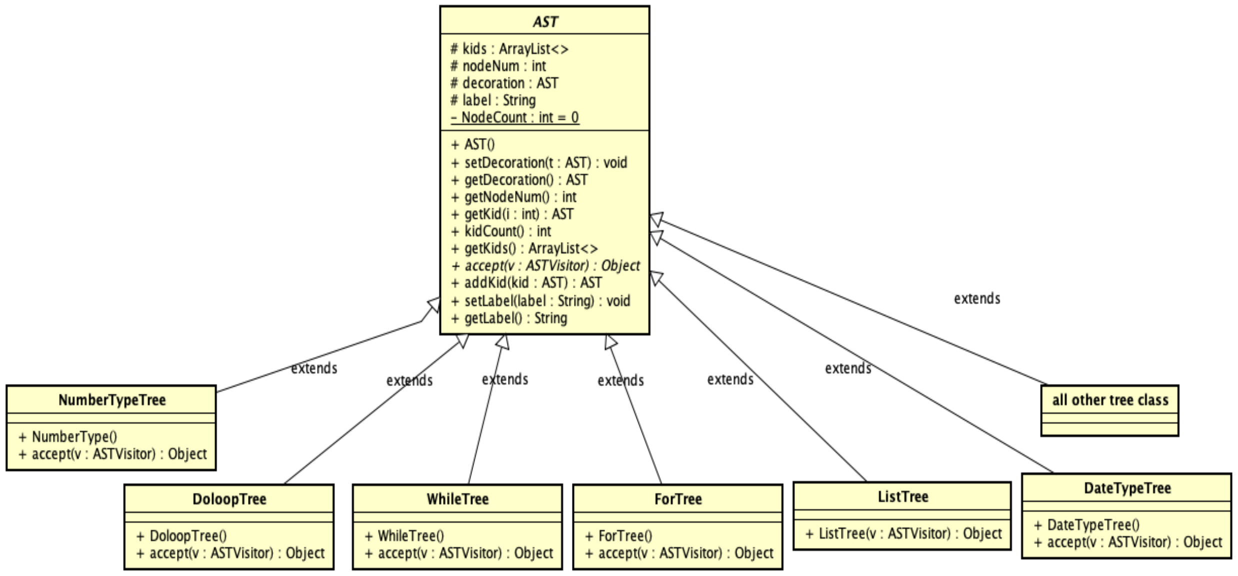
I decided to create the Position class to support OffsetVisitor and DrawOffset. So I created addOffset() method for helping to adjust offset position and getOffset() for getting the correct position , and getDepth() for DrawOffset to find the y axis position.

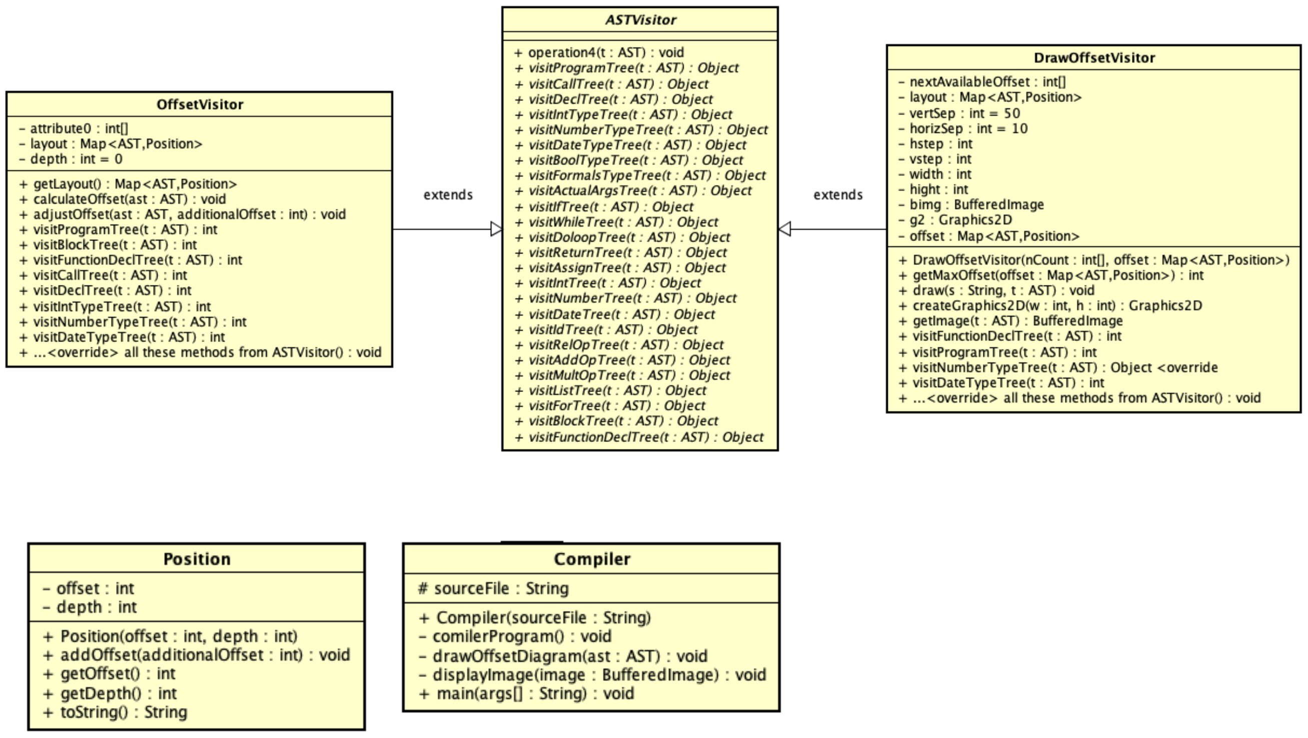
Code Organization

To organize the code as much as possible, in the visitor package, I created OffsetVisitor class.
Within OffsetVisitor I used a recursive method to update the child nodes with adjusted offset.

Class Diagram

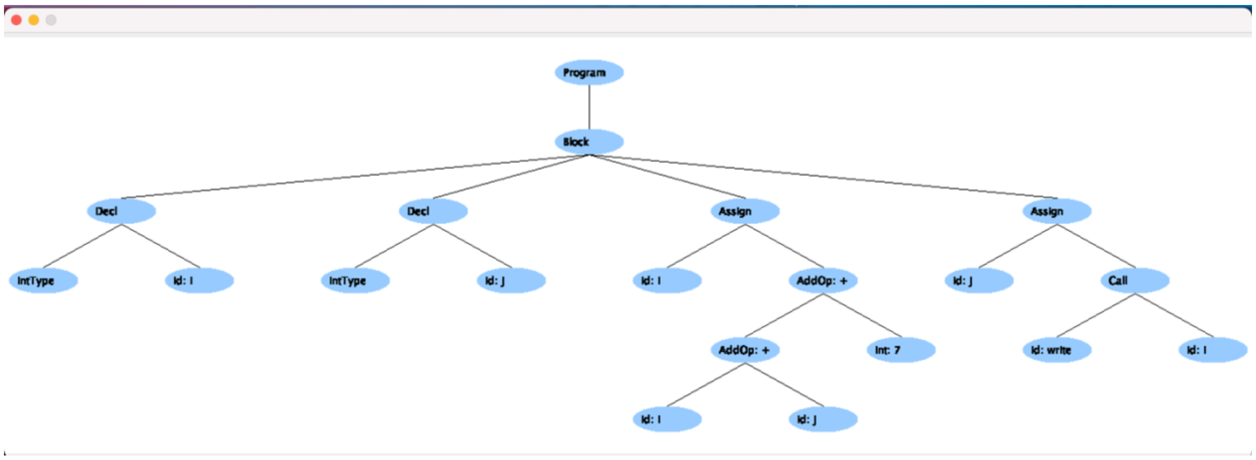
The following class diagrams show the details of all the classes in this project, including the inheritance hierarchy




OffsetVisitor
- attribute0 : int[]
- layout : Map<AST,Position>
- depth : int = 0
+ getLayout() : Map<AST,Position>
+ calculateOffset(ast : AST) : void
+ adjustOffset(ast : AST, additionalOffset : int) : void
+ visitProgramTree(t : AST) : int
+ visitBlockTree(t : AST) : int
+ visitFunctionDeclTree(t : AST) : int
+ visitCallTree(t : AST) : int
+ visitDeclTree(t : AST) : int
+ visitIntTypeTree(t : AST) : int
+ visitNumberTypeTree(t : AST) : int
+ visitDateTypeTree(t : AST) : int
+ ...<override> all these methods from ASTVisitor() : void
extends
ASTVisitor
+ operation4(t : AST) : void
+ visitProgramTree(t : AST) : Object
+ visitCallTree(t : AST) : Object
+ visitDeclTree(t : AST) : Object
+ visitIntTypeTree(t : AST) : Object
+ visitNumberTypeTree(t : AST) : Object
+ visitDateTypeTree(t : AST) : Object
+ visitBoolTypeTree(t : AST) : Object
+ visitFormalsTypeTree(t : AST) : Object
+ visitActualArgsTree(t : AST) : Object
+ visitIfTree(t : AST) : Object
+ visitWhileTree(t : AST) : Object
+ visitDoloopTree(t : AST) : Object
+ visitReturnTree(t : AST) : Object
+ visitAssignTree(t : AST) : Object
+ visitIntTree(t : AST) : Object
+ visitNumberTree(t : AST) : Object
+ visitDateTree(t : AST) : Object
+ visitIdTree(t : AST) : Object
+ visitRelOpTree(t : AST) : Object
+ visitAddOpTree(t : AST) : Object
+ visitMultOpTree(t : AST) : Object
+ visitListTree(t : AST) : Object
+ visitForTree(t : AST) : Object
+ visitBlockTree(t : AST) : Object
+ visitFunctionDeclTree(t : AST) : Object
extends
DrawOffsetVisitor
- nextAvailableOffset : int[]
- layout : Map<AST,Position>
- vertSep : int = 50
- horizSep : int = 10
- hstep : int
- vstep : int
- width : int
- hight : int
- bimg : BufferedImage
- g2 : Graphics2D
- offset : Map<AST,Position>
+ DrawOffsetVisitor(nCount : int[], offset : Map<AST,Position>)
+ getMaxOffset(offset : Map<AST,Position>) : int
+ draw(s : String, t : AST) : void
+ createGraphics2D(w : int, h : int) : Graphics2D
+ getImage(t : AST) : BufferedImage
+ visitFunctionDeclTree(t : AST) : int
+ visitProgramTree(t : AST) : int
+ visitNumberTypeTree(t : AST) : Object <override
+ visitDateTypeTree(t : AST) : int
+ ...<override> all these methods from ASTVisitor() : void
Position
- offset : int
- depth : int
+ Position(offset : int, depth : int)
+ addOffset(additionalOffset : int) : void
+ getOffset() : int
+ getDepth() : int
+ toString() : String
Compiler
# sourceFile : String
+ Compiler(sourceFile : String)
- comilerProgram() : void
- drawOffsetDiagram(ast : AST) : void
- displayImage(image : BufferedImage) : void
+ main(args[] : String) : void

Results and Conclusion

The Parser works as described and outputs the correct values, with OffsetVisitor calculating the right offsets. DrawOffsetVisitor is accurately placing all the nodes at the correct x, y locations within the image.



Challenges

It was challenging for me to implement the algorithm to place the offset in the right place. I chose to perform a post order traversal, as I visit a node, I will update the position where it will be rendered (with offset and depth). The result is saved in HashMap for each node. To diagnose the errors that I was seeing during development, I made continuous use of the debugger, placing a break point at the beginning of the calculateOffset method to step through the code and observe the changes to the offset as the nodes were processed.