Richard Robinson

Parser Project

Due: April 4, 2017

Parser Project

Description:

This project pertains to the modification of a **parser**. The parser contains grammar rules to organize tokens into appropriate groupings called an abstract syntax trees.

The **abstract syntax tree** is handled through an abstract base class that expands into the terminals and non-terminals that will be accepted by the language. The overarching class has an array list representing child nodes that branch from the current node, a static counter for the number of nodes within the entire tree, a unique identifier, a label (for labeling on the printed tree), and another AST for decoration after constraining.

The base class is then extended into the various components within the language. These derived components as mentioned above can be broken down into terminals and non-terminals. Non-terminals are factors within a language necessary for the language, but cannot be outputted: these define behaviors/production rules or are descriptors. Examples for non-terminals would be types and logical operators. Terminals are items that can be output from the language.

The terminals contain a symbol for the token, an accept method to accept visitors (explained below), and a method to return the associated symbol. An example for integer terminals would be '4' or '6'.

The non-terminals contain an accept method for visitors, but they lack a symbol as they are non-terminals and as such it makes no sense for a non-terminal to be assigned a symbol.

The **visitor** implementation is done through a class named ASTVisitor. This is abstract at its base level. It can access nodes from the AST group of classes through their accept methods. It can navigate a tree through the following logic:

visit tree(within visitor) -> accept (within AST) -> visit child -> accept…

This may repeat until a node is reached without children. The order of visiting for this is postorder; the leftmost descendant is visited first, working towards the rightmost, and finally to the parent node.

Access to components within the nodes can be granted through use of the accept method, which will be useful for constraining and ultimately compilation. This is also how we may make a visual representation of our AST.

The **compiler** class is what runs the program. It creates and executes a parser instance with the passed source file. It then creates an offset visitor and draw offset visitor which are accepted by the tree and draws the AST. Note that this compiler class lacks constraining and other factors typically found within a compiler as the decoration of ASTs and generation of byte code are not yet implemented.

Environments Used:

JDK 1.8

NetBeans 8.2

Created on a Windows 10 OS

Files Modified:

ast package

- FloatTypeTree.java
- VoidTypeTree.java

compiler package

- Compiler.java

lexer package

- SourceReader.java

+setup package

++Setup.java

parser package

-Parser.java

Visitor package

-OffsetVisitor.java

-DrawOffsetVisitor.java

Running Instructions:

javac lexer/setup/TokenSetup.java

java lexer/setup/TokenSetup

javac compiler/Compiler.java

java compiler/Compiler simple.x

*simple.x can be replaced with relevant X code file

Assumptions:

The user will provide an X file of a valid syntax to pass as a parameter.

Scope of Work:

I completed everything below and it works as intended.

1) Added Token: ";" SEMI

2) Modified parser to accept new grammers:

- TYPE -> "float"

-TYPE -> "void"

-S -> "return" ';'

-S -> NAME'(' (E list',')?')'

-E -> SE'>'SE

3) Removed debug statements

4) Created a two extensions of visitor named DrawOffsetVisitor and an OffsetVisitor in order to draw a visual representation of the generated AST.

Implementation:

SourceReader:

Made and applied a toString() method, removed debug line.

Parser:

Added the following to the grammar by adding the component to the indicated section in parser

-TYPE -> "float" : rType() Added a check for FLOAT

-TYPE -> "void" : rType() Added a check for VOID

-S -> "return" ';' : rStatement() Added if a SEMI is found after a return statement it returns

-S -> NAME'(' (E list',')?')' : rStatement() Added expressions to the appropriate tree with a comma as a separator until a right parenthesis was reached.

-E -> SE'>'SE : relationalOps had GREAT added to it

AST:

-Added FloatTypeTree and VoidTypeTree

Both are terminals and thusly contain only an accept method for the visitor alongside a blank constructor.

ASTVisitor:

-Added OffsetVisitor

-Added DrawOffsetVisitor

-All visitors had to have their visit functions updated to utilize the FloatTypeTree and VoidTypeTree

OffsetVisitor calculates offsets for each node, tracks the number of nodes present, the current and largest depth, and the next available offset at a particular level.

The offsets are contained within a hash table with both key and data fields being integers. The hashing table is keyed through the node identification given within the AST classes, these start with 1 and is incremented for each new node created. The number of nodes becomes important here as it is how this table will be walked through. The maximum depth is required for scaling of the future output.

The offsets are calculated as follows:

-Number of nodes is incremented by 1.

-If the current depth is less than the maximum depth then it becomes the maximum depth.

-If a node has no children: set offset to next available for that level, increment by 2 and return

-Increment the current depth by 1

-Visit children in post order

-Decrement depth by 1

-Compare the current node's offset to that of the average of its rightmost and leftmost children.

-If the current offset is greater than the average the children will be recursively shifted by an amount equal to the parents offset subtracted by the average. The next offset available at the current depth + 1 is incremented by 2.

-If the current offset is less than the average the parent node's offset will be set to the average and the next offset available at the current depth is incremented by 2.

-The current depth is incremented and an offset is assigned to that particular node, the next available offset is then incremented by 1

There are methods to return the number of nodes, hash table of offsets, and the maximum depth of the tree.

Each AST class has a visit method implemented for them to calculate the offsets as the tree is being traversed. These methods will be called through the AST classes if they have children in the manner mentioned previously.

The **DrawOffsetVisitor** takes a hash map with offsets for nodes keyed by node IDs, the number of nodes, and the maximum depth of the modes in order to draw a visual representation of the AST. The class initializes its frame's width through calculating the maximum offset found through a method that grabs the offsets from the hash map. Its height is calculated from the maximum depth of the nodes. The horizontal spacing for each particular node is found though the offset of that node multiplied by the base horizontal step amount. The vertical spacing is found through the depth of nodes multiplied by the base vertical step amount. The nodes are drawn through Java's fillOval method and connected through lines drawn from the parent (starting x and y) and children (ending x and y) through Java's drawLine method.

Compiler:

No longer uses CountVisitor or DrawVisitor and instead uses OffsetVisitor and DrawOffsetVisitor

Discussion:

I initially made my own unique method for creating offsets, which was different than the one provided, but had some spacing issues when generating offsets for subtrees with children that required recursive shifting. My method assigned offsets and incremented next available while traversing down the tree instead of working upwards. My final project uses a slightly modified method to the one discussed in class. Mine uses parent nodes when recursively shifting children instead of the next available offset.

This project was very enjoyable, but also extremely time consuming for me. This was the first time that I had to implement double dispatch and tree navigation within a project and I had to read and trace through everything within the AST and ASTVisitor classes to be able to apply them effectively. I also had to trace the Parser class but that took significantly less time than the AST and ASTVisitor.

Honestly, I wish we had time to see this entire compilation process to the end instead of stopping here at the parser, but I know that is not possible with the more exciting game project.

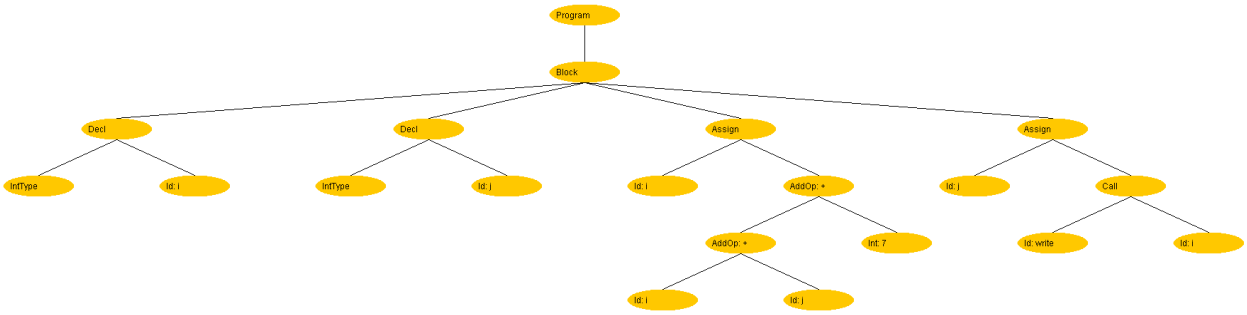
The most difficulty found within this project was correctly calculating offsets into an aesthetically-pleasant manner. The drawing implementation was not difficult for me, however making meaningful and consistent offsets was frustrating.

I am proud to say that I learned from the project and had an overwhelmingly positive experience, however I wish that I had time to start on it before coming back from Spring break. This project frustrated me, albeit in a positive fashion that helped build upon my skills as a programmer.

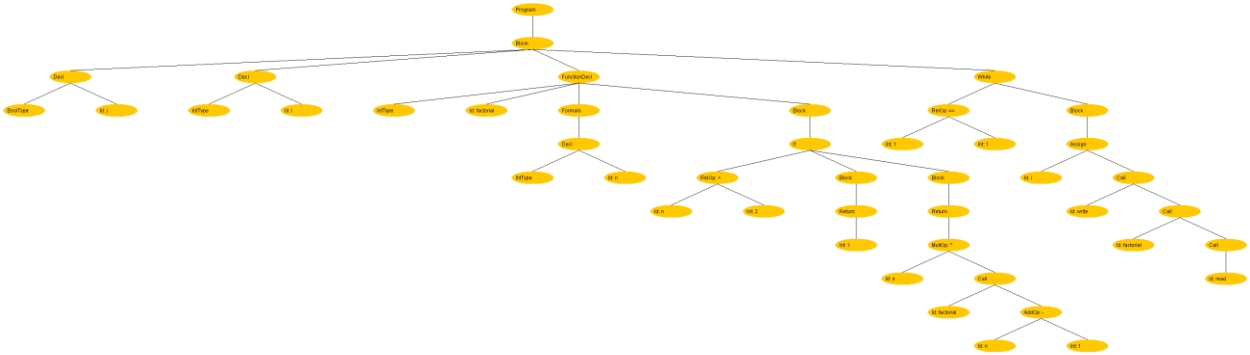
GITHUB: https://github.com/SFSU-CSC-413/assignment-3-Renaird

Resulting ASTs: new tokens

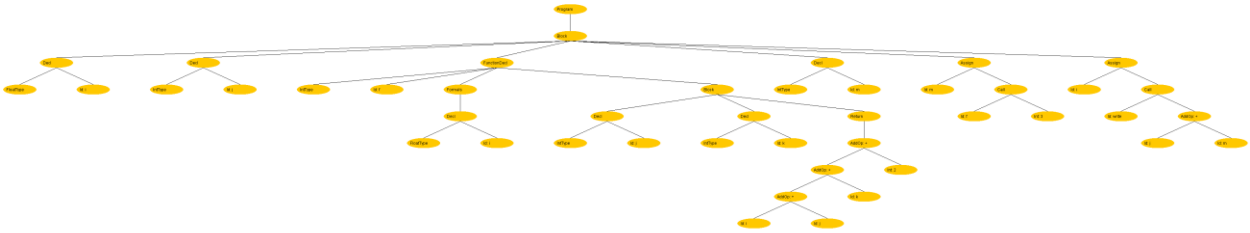
Simple.x



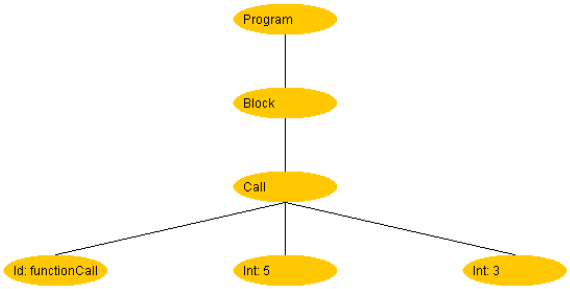
Factorial.x



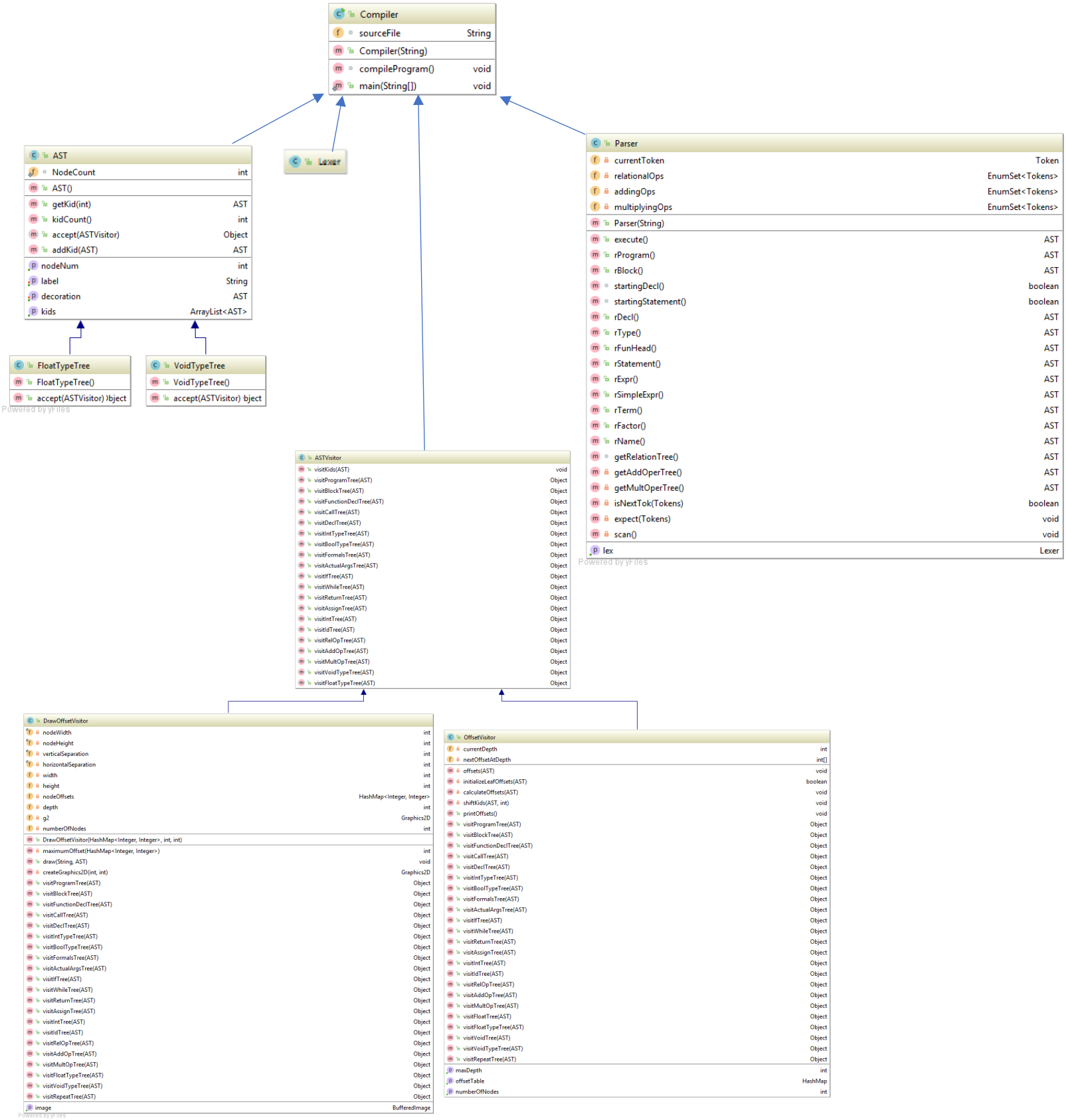
Scopes.x



Showing a function call without assignment



UML: Showing details on classes we changed or created


Compiler
sourceFile String
Compiler(String)
compileProgram() void
main(String[]) void
AST
NodeCount int
AST()
getKid(int) AST
kidCount() int
accept(ASTVisitor) Object
addKid(AST) AST
nodeNum int
label String
decoration AST
kids ArrayList<AST>
Lexer
Parser
currentToken Token
relationalOps EnumSet<Tokens>
addingOps EnumSet<Tokens>
multiplyingOps EnumSet<Tokens>
Parser(String)
execute() AST
rProgram() AST
rBlock() AST
startingDecl() boolean
startingStatement() boolean
rDecl() AST
rType() AST
rFunHead() AST
rStatement() AST
rExpr() AST
rSimpleExpr() AST
rTerm() AST
rFactor() AST
rName() AST
getRelationTree() AST
getAddOperTree() AST
getMultOperTree() AST
isNextTok(Tokens) boolean
expect(Tokens) void
scan() void
lex Lexer
FloatTypeTree
FloatTypeTree()
accept(ASTVisitor) bject
VoidTypeTree
VoidTypeTree()
accept(ASTVisitor) bject
ASTVisitor
visitKids(AST) void
visitProgramTree(AST) Object
visitBlockTree(AST) Object
visitFunctionDeclTree(AST) Object
visitCallTree(AST) Object
visitDeclTree(AST) Object
visitIntTypeTree(AST) Object
visitBoolTypeTree(AST) Object
visitFormalsTree(AST) Object
visitActualArgsTree(AST) Object
visitIfTree(AST) Object
visitWhileTree(AST) Object
visitReturnTree(AST) Object
visitAssignTree(AST) Object
visitIntTree(AST) Object
visitIdTree(AST) Object
visitRelOpTree(AST) Object
visitAddOpTree(AST) Object
visitMultOpTree(AST) Object
visitVoidTypeTree(AST) Object
visitFloatTypeTree(AST) Object
DrawOffsetVisitor
nodeWidth int
nodeHeight int
verticalSeparation int
horizontalSeparation int
width int
height int
nodeOffsets HashMap<Integer, Integer>
depth int
g2 Graphics2D
numberOfNodes int
DrawOffsetVisitor(HashMap<Integer, Integer>, int, int)
maximumOffset(HashMap<Integer, Integer>) int
draw(String, AST) void
createGraphics2D(int, int) Graphics2D
visitProgramTree(AST) Object
visitBlockTree(AST) Object
visitFunctionDeclTree(AST) Object
visitCallTree(AST) Object
visitDeclTree(AST) Object
visitIntTypeTree(AST) Object
visitBoolTypeTree(AST) Object
visitFormalsTree(AST) Object
visitActualArgsTree(AST) Object
visitIfTree(AST) Object
visitWhileTree(AST) Object
visitReturnTree(AST) Object
visitAssignTree(AST) Object
visitIntTree(AST) Object
visitIdTree(AST) Object
visitRelOpTree(AST) Object
visitAddOpTree(AST) Object
visitMultOpTree(AST) Object
visitFloatTypeTree(AST) Object
visitVoidTypeTree(AST) Object
visitRepeatTree(AST) Object
image BufferedImage
OffsetVisitor
currentDepth int
nextOffsetAtDepth int[]
offsets(AST) void
initializeLeafOffsets(AST) boolean
calculateOffsets(AST) void
shiftKids(AST, int) void
printOffsets() void
visitProgramTree(AST) Object
visitBlockTree(AST) Object
visitFunctionDeclTree(AST) Object
visitCallTree(AST) Object
visitDeclTree(AST) Object
visitIntTypeTree(AST) Object
visitBoolTypeTree(AST) Object
visitFormalsTree(AST) Object
visitActualArgsTree(AST) Object
visitIfTree(AST) Object
visitWhileTree(AST) Object
visitReturnTree(AST) Object
visitAssignTree(AST) Object
visitIntTree(AST) Object
visitIdTree(AST) Object
visitRelOpTree(AST) Object
visitAddOpTree(AST) Object
visitMultOpTree(AST) Object
visitFloatTree(AST) Object
visitFloatTypeTree(AST) Object
visitVoidTree(AST) Object
visitVoidTypeTree(AST) Object
visitRepeatTree(AST) Object
maxDepth int
offsetTable HashMap
numberOfNodes int