Author: Nghia Duc Hong

Student Number: 292119

Email: duc.hong@tuni.fi

Phase 3 summary:

1. Syntax tree is a tree representation of syntactic structure of the source code. These node of the syntax tree are built on the syntax analysis phase. The syntax tree is abstract in the way that it does not contain every tokens, but rather the context-based constraints.
2. In the PLY tool, grammar rules are defined by a Python function where docstring to that function contains the appropriate context-free grammar specification. Then with each grammar rule, the AST node struct is introduced to link the node of the tree together, creating a hierarchical tree while each node represents a meaningful construct occurring from the source code.
3. A scalar (variable) definition is built when the grammar are recognized from keyword "scalar", followed by an identifier and an optional initializer ("= scalar_expr"). A scalar expression is also a node of AST. Then the scalar definition node is built with nodetype is "scalar_definition", its value is the name of the scalar (identifier) and a child node for initializing the scalar. The uninitialized scalar will be initialized as 0.0 .

A loop AST node will be recognized as a node with type "for". It contains list of 2 children: range_list and statement_list. The range_list will be used for looping, while the statement_list will be the body of the for loop.

A function call will be recogized as an AST node whose type is "function_call". The node only captures two children which are the function identifier and the arguments for the function itself.

4. A) I want to preserve consistency, so I did not use attribute child_ in the code. There is some cases that the children_ attribute of the node should be None as there is no need to store any additional information about the note. For example, the node 'decimal' has its value is a tuple with a mathematical sign and a decimal value. Then there is no extra information to be used so the children_ attribute should not contain anything. The nodes I implemented "scalar", "sheet", "range" also has the values are the scalar identifier, sheet identifier and range identifier, so there is also no need to have the children_ attribute. I also have created a node type compare (which means the mathematical compare operator). This type of node also does not need to have child node.

B) I have simplified the scalar_expr to be an atom if in the scalar_expr only contains one element (one child). The way I can do this is that I only recognized the terms, simple expression, scalar expression when they have at least two children. In that way, the tree will be automatically reduced to the simplest form but still useful and recognizable with recursive grammar. For example, a factor is recognized as a term, but if the term only contains this factor, I need this term still be a factor node.

5. I did all the implementation for the tree (functions and subroutines) and it worked with the tests quite well. The tree has printed all the necessary information from the source code which I think can be useful for the phase 4.
6. The assignment is quite challenging to me. If the implement only remembering all the node from phase 2, of course it is easy to make. But I am eager to do the phase 4 to improve my score, so it is challenging when I have to think about how to reduce the tree to the simplier form, but still conserve the consistency to be executable in phase 4.
7. Extra : I did implement the conversion of the coordinate, for example, A4 would be recogized as (column 0, row 3)