Captain report Abdelhadi Marjane

How to run the parser and abstract parser

Keep the file token open even if it doesn’t have any value on it,

You need to check if you have installed the AnyTree library in your local machine

Then run the parser code first or run the abstract first   
either way they are connected when you run one it will run the other’s

Assuming that the test file is already in the machine the parser will search for it

It will update the file token   
and the parser will generate the tree at the terminal and also generate a txt file called concrete tree file to view the entire tree and see it in a much intuitive way

For the abstract you don’t need to give it anything it will call the parser and parser will call the lexer   
the abstract parser will generate the abstract tree in the terminal and generate the txt file called   
abstract\_tree

You can also change what are the terminals that you want to reduce from the tree you can include and add the nodes that you want to keep so the current tree can be further reduced

I didn’t know what to keep or not to keep but the tree can be further reduced by changing the update\_tree list

Added to the parser the ability to keep the values of the tokens so that I can insert it in the tree

Also added more exceptions for syntax

Challenges/Issues

Started at Monday 15 worked on the parser logic day and night for 4 days fixing the logic of the parser code there were a lot of logical issues withing the code , then when the logic was working spent some time testing and debugging then I started connecting the nodes using the AnyTree library

added functionality so that the user will run the code from the parser and it will call the lexer code to create the tokens

tried to fix the lexer to remove the end of line issue but it run into issues where the token file will skip a line if it did not encounter the end of line

so I left the code the same as before fixed some parts from the original lexer that was submitted in part 2  
there is still an issue where the file token needs to be there for it to work

for part 5 I got stuck in it as I didn’t how to approach the vim module, definitely If I was in a team it would have been much better and I could think it through with other teammates   
I left the part 5 I did not do it   
I can say that the parser was the biggest challenge I had to rewrite most of the code from scratch and fix the nodes to be at their right place in the tree   
as for the abstract tree it was also challenging to do but I did do it by only leaving the children of the terminals and back tracking the nodes to their parent while leaving only the terminals   
for statatic semantics I did not do it as we don’t have global variables

And couldn’t find a way to keep track if variables were declared before

Grammar

Updated the grammar I simplified it a lot more completed the parser now it should parse the tokens

Made some changes to the arrays   
made changes to the arithmetic operations to accept more operations

Added non terminals Zorto and functionRE  
removed the usage of ag.i(4,4) removed the dot operation and simplified it to only ag(4,4) normal function calls

Created the statement with endline and statement without end of line to simplfy the grammar

Removed statements non terminal and added statement that has all of the possible operations that can happen in the code