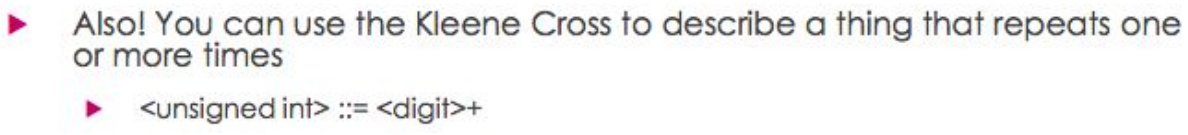
Report for PL assignment

EBNF

When I wrote the EBNF I followed the syntax from wikipedia, available on this link: https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form. There is only one exception from the article and that is the repetition in number and ident. There I used this from the lecture:



If my answer is not accepted an alternative way to do it is this:
number =
(‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’),
{(‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)};
Other than that all it follows the given link.

YACC & LEX

Even though this part is by far what took the most time I don’t feel like there is anything special I should point out. The Lex part is pretty straight forward. I’ve token given names with underscores and upper case letters to separate them from non-tokens.

In the Yacc code I was unsure as to how I would solve a potential reduce/reduce conflict. However, it never turned up and so I went with the mentality of don’t fix it unless it’s broken. For that reason my choice to create separate rules with empties in them for the optional parts remained unchanged. I’m also using several of yacc’s functions as becomes apparent in the code. That’s obvious though as I wouldn’t be able to make it without some of them.

I also made a make file. This ends up making the executable etc.

Testing

Working cases:

For the working cases I decided to write up a file that covered all fields. It is named “testworking” and is included in the folder. If you pipe the file by writing “./QUENYARGOL < testworking” you’ll get a successful test. At that point each rule has been tested. However, every path for each rule is not being tested. For instance I’m not showing that it works for every possible solution inside of “declaration_unit”. It seemed pointless, and so by showing one I feel like it becomes obvious

that the other options would work the same way. By showing that each rule in fact finishes I consider the test to follow the instructions of the assignment.

When a rule is finished the name of the rule is printed. This is to make it easier to see what rule you've completed and make it easier to see where it had crashed.

I also want to point out that some of the rules are redundant. This doesn't change anything apart from making them impossible to show crashing. For instance you can't crash inside of "implementation part" as the only thing it does is pass it on to statement. The crashing will always occur in statement as statement recognises that the syntax is wrong.

Failing cases:

When making fail cases I figured I could do it pretty fast by copying the syntax that goes through the entire program (testworking). What I then did was make 31 files that each have one altercation to the syntax making a rule fail in each. I had a checklist to check that I broke every rule. As it was not specified how you wanted the rules to break I just added a bunch of meaningless stuff to the part of the syntax that I wanted to break.

To run it I made a script in bash that just runs the program and pipes all the files in the folder "failcases". This script is called "shellsklipt.sh". The information that you get will probably be pretty messy to read, but it breaks every one of the rules except for the impossible ones (The ones that just pass it on to other rules). I added added an echo "------..." line to make it easier to separate different attempt to run the syntax test. The printed lines can be used to see what was broken.

There were a few things to keep in mind when breaking the syntax. For instance that you don't break "procedure_call" by writing "123CALL hello". This would break whatever was expecting the "procedure call". You could however break it by writing "CALL 123a+++xD" as it would not receive what it expected. You would also not break if if you wrote "CALL hello I love potatoes" as that would break whatever was after the "procedure_call" as the "procedure_call" already received the ident it was expecting and so was completed. Other than that, breaking the rules was pretty easy.