COMP 442 Project: Lexical Analyzer:

The lexical specification was highlighted in the assignment document:

A screenshot of a computer program

Description automatically generated

In the above picture, the only tokens are id, integer and float. Alphanum, fraction, letter digit and nonzero help us deduce the automaton for the tokens.

Regular Expressions:

ID: ([a-z|A-Z][a-zA-Z0-9\_]\*)

Integer: (0 | ([1-0][0-9]\*)

Float: (0 | [1-9][0-9]\*)\.(0|[0-9]\*[1-9])([eE][+-]?(0|1-9[0-9]\*))

Operators: (==|<>|<=|>=|<|>|:=|+|-|\|and|or|not)

In addition to these, the operators and punctuation and reserved words were specified too:

A white background with black text

Description automatically generated

The inline comment and block comments are also considered in the specification.

Finite State Machine: The FA below highlights the major lexical specification of my analyzer:

A white board with black dots and lines

Description automatically generated

Design: The project is implemented in C++. The lexical analyzer is following handwritten scanner as it is straightforward and covers the cases for a simple language given to us. In token.h, I have specified all the tokens as enums and token.cpp plays a part to return type, lexeme, line when it is setup correctly. The errors are also specified as tokens in the tokentype. Lexer.h and Lexer.cpp list all the cases of the dfa presented above. As this is a handwritten scanner, I’m still working on some cases. However, the majority of the positive cases are implemented. The inline comments and block comments are made as tokens in this project.

Libraries/External Tools

I have’nt used any external libraries to aid the project. The libraries used were for streaming/writing and reading a file.

Assignment Submission:

/src contains all the code needed to run the program.

To run the program, go to the src directory and run:

./lexer lexpositivegrading.src

./lexer lexnegativegrading.src

Etc.