This file is example.lang, it contains the regular expressions emeded in my langage format for use with read\_lang.py

Semicolon = ;

EqualSign = \=

Integer = [0-9]+

Identifier = [a-zA-Z]+

opperator = \+|-|/|\\*

decimalPoint = \.

Statement = <Expression><Semicolon>

Expression = <Assign>|<Equation>

Assign = <Identifier><EqualSign><Equation>

Equation =

<Equation><opperator><Equation>

<Number>

<Identifier>

Decimal = <Integer><decimalPoint><Integer>|<decimalPoint><Integer>

Number = <Decimal>|<Integer>

#name = [a-zA-Z]+

#const = [0-9]+

#newline = \n

#open\_parenth = \(

#closing\_parenth = \)

#equality = =

#

#statement = <expr><newline>|<open\_parenth><expr><closing\_parenth>

#expr = <assign>|<equation>

#assign = <name><equality><equation>

#equation =

# <const>

# <name>

# <equation><opperator><equation>|

here are the contents of output.txt it was a bit large to screenshot so I pasted the contents here,

because I was feeling fancy I got the computer not only tokenizing the incoming data, but also generating a parse tree based on an arbitrary input.lang file. So the output concists of the parse tree detected for a given input, followed by the series of tokens that were used to construct that tree. The screen shots are also included at the bottom of the word doc, there was a bit TOO much output to include in one screenshot so its pasted here for convinence.

end of the file

Statement

x=2+3-5;

|-Expression

| x=2+3-5

|-|-Assign

| | x=2+3-5

|-|-|-Identifier

| | | x

|-|-|-EqualSign

| | |

|-|-|-Equation

| | | 2+3-5

|-|-|-|-Equation

| | | | 2

|-|-|-|-|-Number

| | | | | 2

|-|-|-|-|-|-Integer

| | | | | | 2

|-|-|-|-opperator

| | | | +

|-|-|-|-Equation

| | | | 3-5

|-|-|-|-|-Equation

| | | | | 3

|-|-|-|-|-|-Number

| | | | | | 3

|-|-|-|-|-|-|-Integer

| | | | | | | 3

|-|-|-|-|-opperator

| | | | | -

|-|-|-|-|-Equation

| | | | | 5

|-|-|-|-|-|-Number

| | | | | | 5

|-|-|-|-|-|-|-Integer

| | | | | | | 5

|-Semicolon

| ;

<Identifier><EqualSign><Integer><opperator><Integer><opperator><Integer><Semicolon>

Statement

y=2\*x;

|-Expression

| y=2\*x

|-|-Assign

| | y=2\*x

|-|-|-Identifier

| | | y

|-|-|-EqualSign

| | |

|-|-|-Equation

| | | 2\*x

|-|-|-|-Equation

| | | | 2

|-|-|-|-|-Number

| | | | | 2

|-|-|-|-|-|-Integer

| | | | | | 2

|-|-|-|-opperator

| | | | \*

|-|-|-|-Equation

| | | | x

|-|-|-|-|-Identifier

| | | | | x

|-Semicolon

| ;

<Identifier><EqualSign><Integer><opperator><Identifier><Semicolon>

Statement

3+5+5;

|-Expression

| 3+5+5

|-|-Equation

| | 3+5+5

|-|-|-Equation

| | | 3

|-|-|-|-Number

| | | | 3

|-|-|-|-|-Integer

| | | | | 3

|-|-|-opperator

| | | +

|-|-|-Equation

| | | 5+5

|-|-|-|-Equation

| | | | 5

|-|-|-|-|-Number

| | | | | 5

|-|-|-|-|-|-Integer

| | | | | | 5

|-|-|-|-opperator

| | | | +

|-|-|-|-Equation

| | | | 5

|-|-|-|-|-Number

| | | | | 5

|-|-|-|-|-|-Integer

| | | | | | 5

|-Semicolon

| ;

<Integer><opperator><Integer><opperator><Integer><Semicolon>

Statement

5\*x;

|-Expression

| 5\*x

|-|-Equation

| | 5\*x

|-|-|-Equation

| | | 5

|-|-|-|-Number

| | | | 5

|-|-|-|-|-Integer

| | | | | 5

|-|-|-opperator

| | | \*

|-|-|-Equation

| | | x

|-|-|-|-Identifier

| | | | x

|-Semicolon

| ;

<Integer><opperator><Identifier><Semicolon>

Statement

2\*x+5.4;

|-Expression

| 2\*x+5.4

|-|-Equation

| | 2\*x+5.4

|-|-|-Equation

| | | 2\*x

|-|-|-|-Equation

| | | | 2

|-|-|-|-|-Number

| | | | | 2

|-|-|-|-|-|-Integer

| | | | | | 2

|-|-|-|-opperator

| | | | \*

|-|-|-|-Equation

| | | | x

|-|-|-|-|-Identifier

| | | | | x

|-|-|-opperator

| | | +

|-|-|-Equation

| | | 5.4

|-|-|-|-Number

| | | | 5.4

|-|-|-|-|-Decimal

| | | | | 5.4

|-|-|-|-|-|-Integer

| | | | | | 5

|-|-|-|-|-|-decimalPoint

| | | | | |

|-|-|-|-|-|-Integer

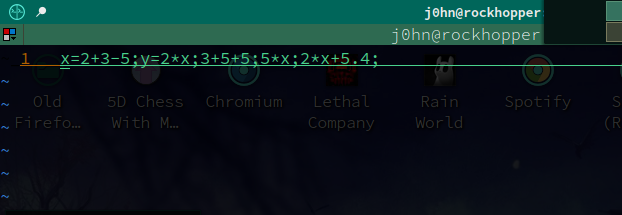
| | | | | |

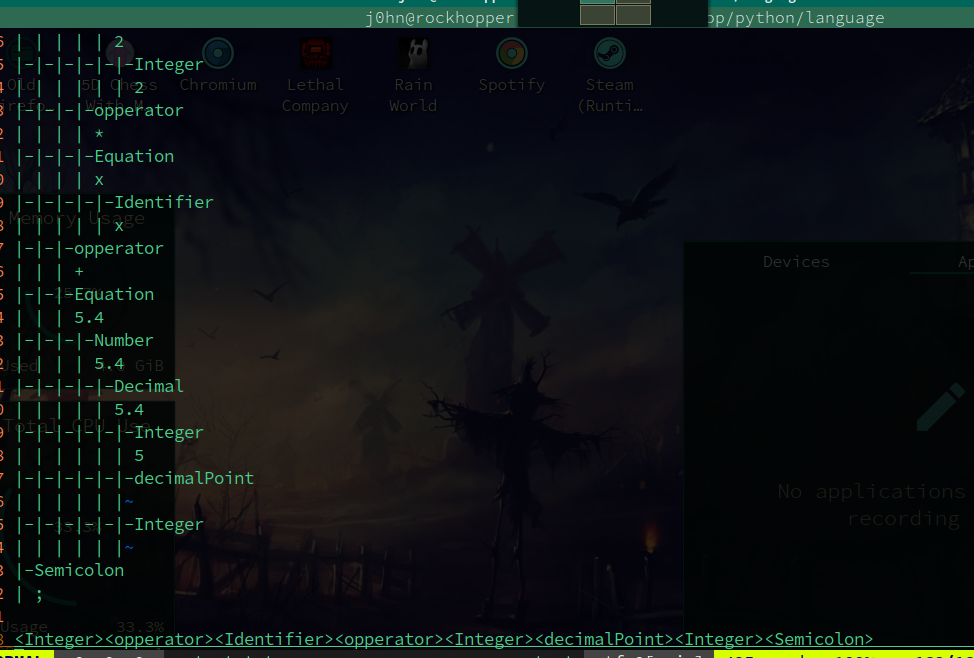
|-Semicolon

| ;

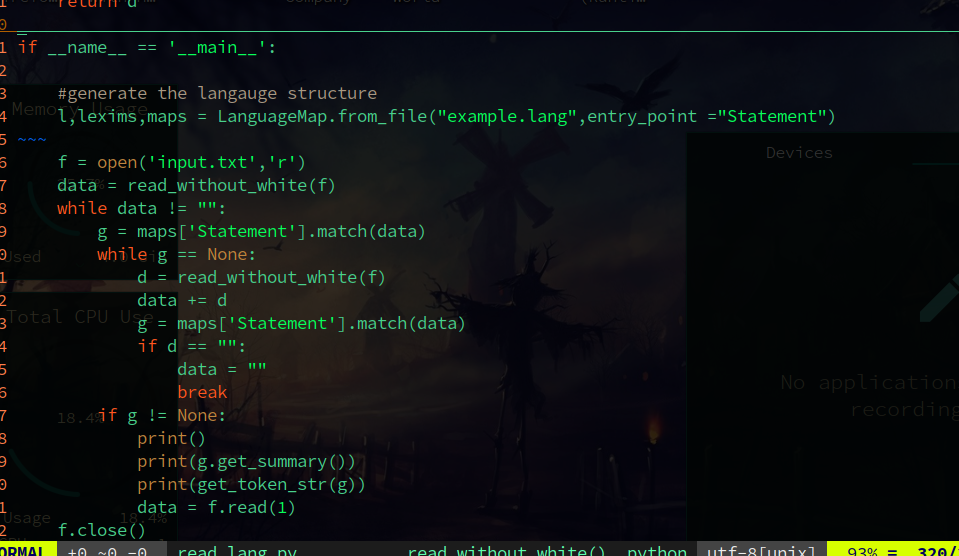
<Integer><opperator><Identifier><opperator><Integer><decimalPoint><Integer><Semicolon>

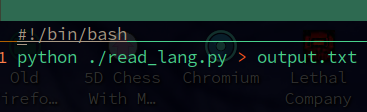
and then the initial input.txt





Source Code File Screenshots. The code is a bit of a mess, and could be significantly improved, but it gets the job done for what I am trying to do.



command used to redirect the output, stored in run.sh

Issues:

most of the code works as expected, anoyingly the program halts on a new line in the input with the read code, and there appears to be a bug when rendering out the parse trees. Occasionally some of the lexims do not display on the screen in the parse tree, they appear to parse correctly dispite this however, so for now I am assuming it is a render error as apposed to an underlying logic error. The code is also VERY noodaly and could do with a serious re-write. In particular when I do this again I think I am going to rely more on pythons match dataclass so im not copying and sending strings around. I might also try the same problem in c++ to force efficient memory usage and thinking about datastructures. I anticipate my second attempt to be much cleaner after running through significant hurtles to get this one to work.