Assignment 2
Sarvansh Prasher

1. **<u>Prolog Code:</u>**

```
% @author : Sarvansh Prasher
% @version 1.0
% @created on 03-19-2020

% Solution 1 for eliminating left recursion

program --> block,[.].

block --> [begin],declaration,[;],command,[end].

declaration --> [const],identifier,[=],number,[;],declaration|
    [var],identifier,[;],declaration|
    [const],identifier,[=],number|
    [var],identifier.

command --> identifier,[:=],arithmeticExpression,[;],command|
[if],booleanExpression,[then],command,[else],command,[endif],[;],command|
[while],booleanExpression,[do],command,[endwhile],[;],command|
block,[;],command| identifier,[:=],arithmeticExpression|
[if],booleanExpression,[then],command,[else],command,[endif]|
[while],booleanExpression,[do],command,[endwhile]|block.

booleanExpression -->[true]|[false]|arithmeticExpression,[=],arithmeticExpression|
    [not],booleanExpression.

arithmeticExpression --> additionOperation,[-],arithmeticExpression|additionOperation.

additionOperation --> multiplicationOperation,[+],arithmeticExpression|multiplicationOperation.

multiplicationOperation --> divisionOperation,[*],arithmeticExpression|divisionOperation.

divisionOperation --> generalOperation,[/],arithmeticExpression|generalOperation.

generalOperation --> identifier|number.

identifier --> [x]|[y]|[z]|[u]|[v].

number --> [0]|[1]|[2]|[3]|[4]|[5]|[6]|[7]|[8]|[9].
```

Sample Run:

```
?- program([begin, const, x, =, 8, ;, var, y, ;, var, z, ;, z, :=, 0, ;, if, x, =, y, +, 2, then, z , := , 5, else, z, :=, 3, endif, ;,
while, not, x, =, z, do, z, :=, z, +, 2, endwhile, end, .],[]).

True

?- program([begin, const, x, =, 8, ;, var, y, ;, var, z, ;, z, :=, 0, ;, if, x, =, y, +, 2, then, z , := , 5, else, z, :=, 3, endif, ;, if,
x, =, y, +, 2, then, z , := , 5, else, z, :=6 , endif,end, .],[]).

True
```

2. **Prolog Code:**

```
:- use_rendering(svgtree).

program(p(Z)) --> block(Z),[.].

block(b(Z,Z1)) --> [begin],declaration(Z),[;],command(Z1),[end].

declaration(d(Z,Z1,Z2)) --> [const],identifier(Z),[=],number(Z1),[;],declaration(Z2).
declaration(d(Z,Z1)) --> [var],identifier(Z),[;],declaration(Z1).
declaration(d(Z,Z1)) --> [const],identifier(Z),[=],number(Z1).
declaration(d(Z)) --> [var],identifier(Z).

command(c(Z,Z1,Z2)) --> identifier(Z),[:=],arithmeticExpression(Z1),[;],command(Z2).
command(c(Z,Z1,Z2,Z3)) --> [if],booleanExpression(Z),[then],command(Z1),[else],command(Z2),[endif],[;],command(Z3).
command(c(Z,Z1,Z2)) --> [while],booleanExpression(Z),[do],command(Z1),[endwhile],[;],command(Z2).
command(c(Z,Z1)) --> block(Z),[;],command(Z1).
command(c(Z,Z1)) --> identifier(Z),[:=],arithmeticExpression(Z1).
command(c(Z,Z1,Z2)) --> [if],booleanExpression(Z),[then],command(Z1),[else],command(Z2),[endif].
command(c(Z,Z1,Z2)) --> [while],booleanExpression(Z),[do],command(Z1),[endwhile]|block(Z2).

booleanExpression(bo(true)) --> [true].
booleanExpression(bo(false)) --> [false].
booleanExpression(bo(Z,Z1)) --> arithmeticExpression(Z),[=],arithmeticExpression(Z1).
booleanExpression(bo(Z)) --> [not],booleanExpression(Z).

arithmeticExpression(ae(Z,Z1)) --> additionOperation(Z),[-],arithmeticExpression(Z1).
arithmeticExpression(ae(Z)) --> additionOperation(Z).

additionOperation(ao(Z,Z1)) --> multiplicationOperation(Z),[+],arithmeticExpression(Z1).
additionOperation(ao(Z)) --> multiplicationOperation(Z).

multiplicationOperation(mo(Z,Z1)) --> divisionOperation(Z),[*],arithmeticExpression(Z1).
multiplicationOperation(mo(Z)) --> divisionOperation(Z).
```

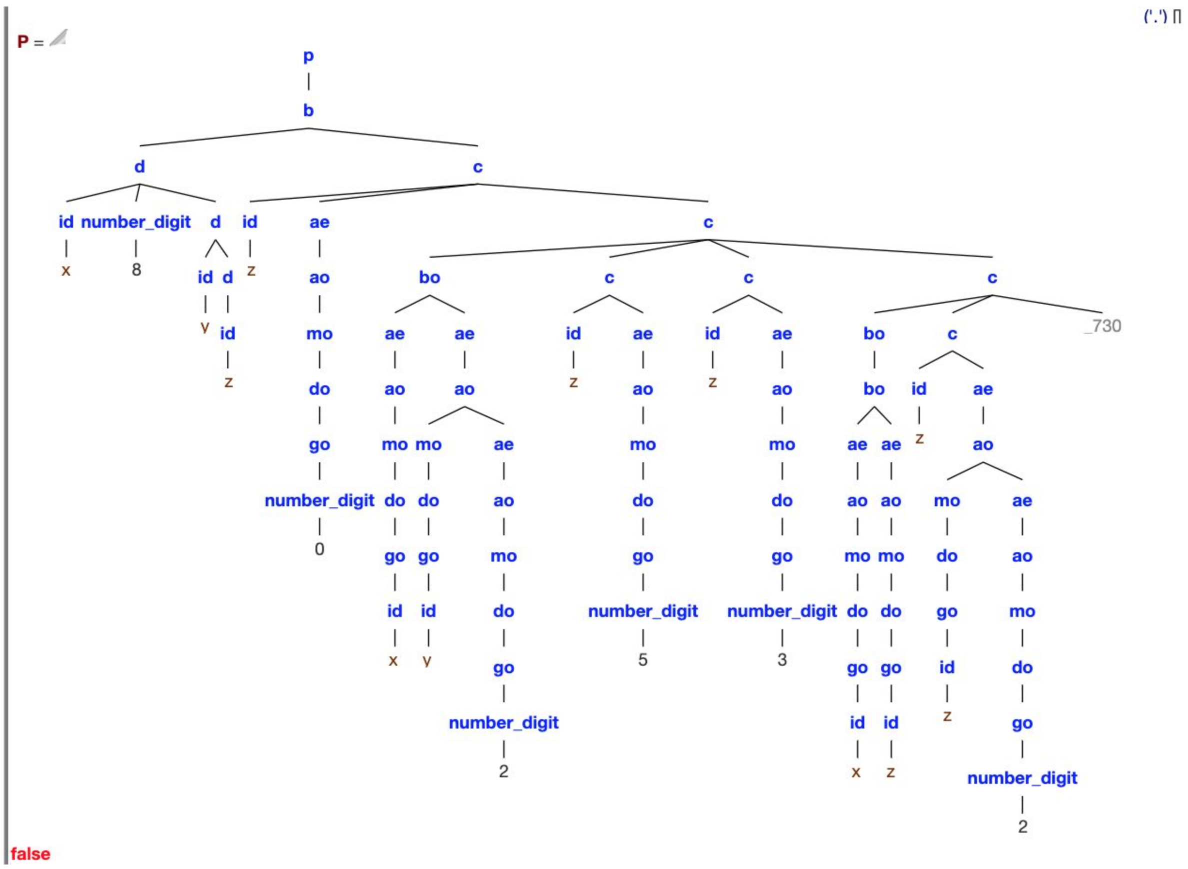
```
divisionOperation(do(Z,Z1)) --> generalOperation(Z),[/],arithmeticExpression(Z1).
divisionOperation(do(Z)) --> generalOperation(Z).

generalOperation(go(Z)) --> identifier(Z).
generalOperation(go(Z)) --> number(Z).

identifier(id(x)) --> [x].
identifier(id(y)) --> [y].
identifier(id(z)) --> [z].
identifier(id(u)) --> [u].
identifier(id(v)) --> [v].

number(number_digit(0)) --> [0].
number(number_digit(1)) --> [1].
number(number_digit(2)) --> [2].
number(number_digit(3)) --> [3].
number(number_digit(4)) --> [4].
number(number_digit(5)) --> [5].
number(number_digit(6)) --> [6].
number(number_digit(7)) --> [7].
number(number_digit(8)) --> [8].
number(number_digit(9)) --> [9].
```

Sample Run:

```
?- L = [begin, const, x, =, 8, ;, var, y, ;, var, z, ;, z, :=, 0, ;, if, x, =, y, +, 2, then, z , := , 5, else, z, :=, 3, endif, ;, while,
not, x, =, z, do, z, :=, z, +, 2, endwhile, end, .],
program(P, L, []).
```

```
P = p(b(d(id(x), number_digit(8), d(id(y), d(id(z)))), c(id(z), ae(ao(mo(do(go(number_digit(0)))))),
c(bo(ae(ao(mo(do(go(id(x)))))), ae(ao(mo(do(go(id(y))))), ae(ao(mo(do(go(number_digit(2))))))))), c(id(z),
ae(ao(mo(do(go(number_digit(5))))))), c(id(z), ae(ao(mo(do(go(number_digit(3))))))), c(bo(bo(ae(ao(mo(do(go(id(x)))))),
ae(ao(mo(do(go(id(z)))))))), c(id(z), ae(ao(mo(do(go(id(z))))), ae(ao(mo(do(go(number_digit(2))))))))), _730)))))
```

SVG tree:

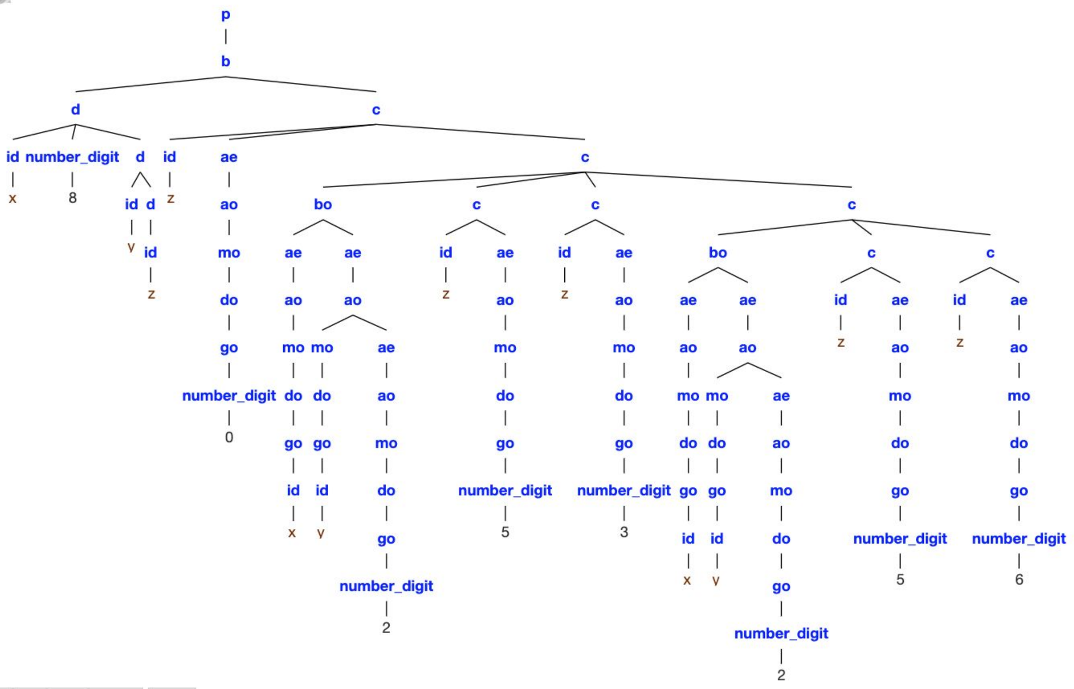


```
?- L =  [begin, const, x, =, 8, ;, var, y, ;, var, z, ;, z, :=, 0, ;, if, x, =, y, +, 2, then, z , := , 5, else, z, :=, 3, endif, ;, if, x, =,
y, +, 2, then, z , := , 5, else, z, :=,6 , endif,end, .],
program(P, L, []).
```

```
P = p(b(d(id(x), number_digit(8), d(id(y), d(id(z)))), c(id(z), ae(ao(mo(do(go(number_digit(0)))))),
c(bo(ae(ao(mo(do(go(id(x)))))), ae(ao(mo(do(go(id(y)))), ae(ao(mo(do(go(number_digit(2))))))))), c(id(z),
ae(ao(mo(do(go(number_digit(5))))))), c(id(z), ae(ao(mo(do(go(number_digit(3))))))), c(bo(ae(ao(mo(do(go(id(x)))))),
ae(ao(mo(do(go(id(y)))), ae(ao(mo(do(go(number_digit(2))))))))), c(id(z), ae(ao(mo(do(go(number_digit(5))))))), c(id(z),
ae(ao(mo(do(go(number_digit(6))))))))))))
```

<u>SVG tree:</u>

P =


p
b
d
c
id number_digit d id ae c
x 8 id d z ao bo c c c
y id mo ae ae id ae id ae bo c c
z do ao ao z ao z ao ae ae id ae id ae
go mo mo ae mo mo ao ao z ao z ao
number_digit do do ao do do mo mo ae mo mo
0 go go mo go go do do ao do do
id id do number_digit number_digit go go mo go go
x y go 5 3 id id do number_digit number_digit
number_digit x y go 5 6
2 number_digit
2