LaxScript(Relaxation Script)

SER 502 - Emerging Languages and Programming Paradigms
Project Group 37

Github Link-
https://github.com/shsavani93/SER502-Spring2022-Team37

Project Demonstration YouTube link-
https://www.youtube.com/watch?v=RRZ9qwqs89g

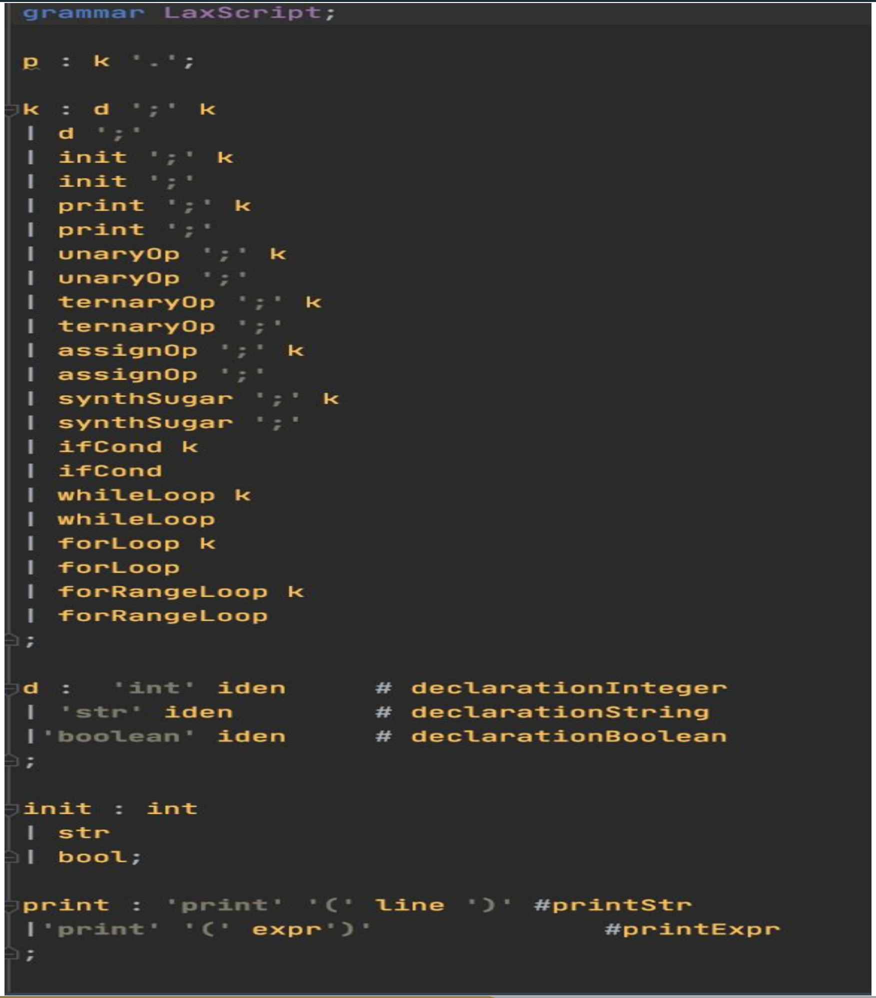
Team Members

- Vashishtha Nareshbhai Patel
- Dennis Polly Pynadath
- Saurabh Rane
- Sneha Savani
- Keshav Sethi

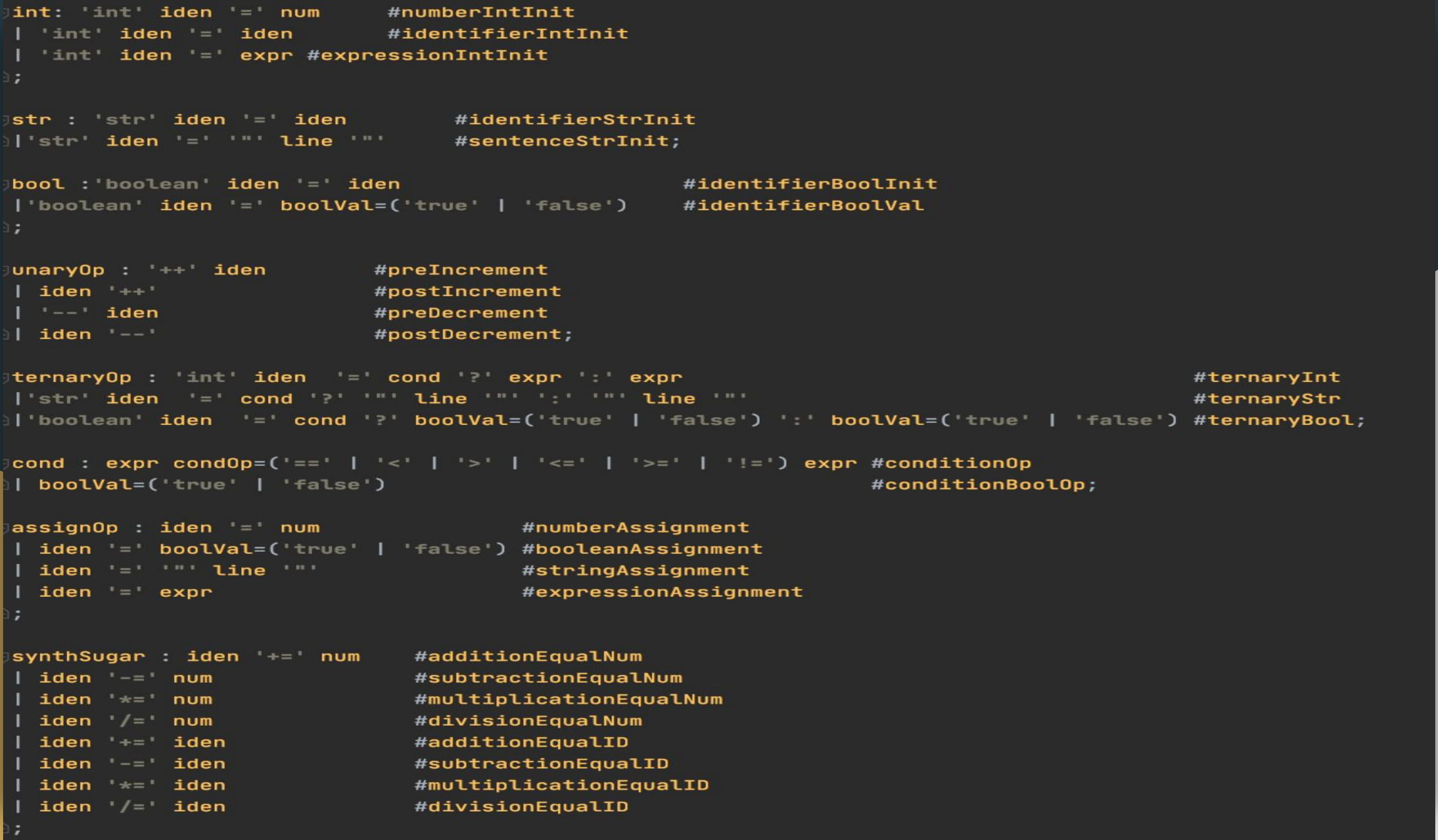
LaxScript Features

- LaxScript supports three data type: **integers**, **strings** and **boolean**.
- LaxScript **Identifier** contains English alphabets and can extend from a single to multiple alphabets for a single identifier. Used for defining the variable name.
- LaxScript supports **ternary** conditional statements.
- The basic structure of an **if-then-else** condition is supported by LaxScript.
- LaxScript provides structure of **for loop** and **while loop**.
- LaxScript provides support for **range** in the **for loop** condition.
- LaxScript provides support for **print** statements.

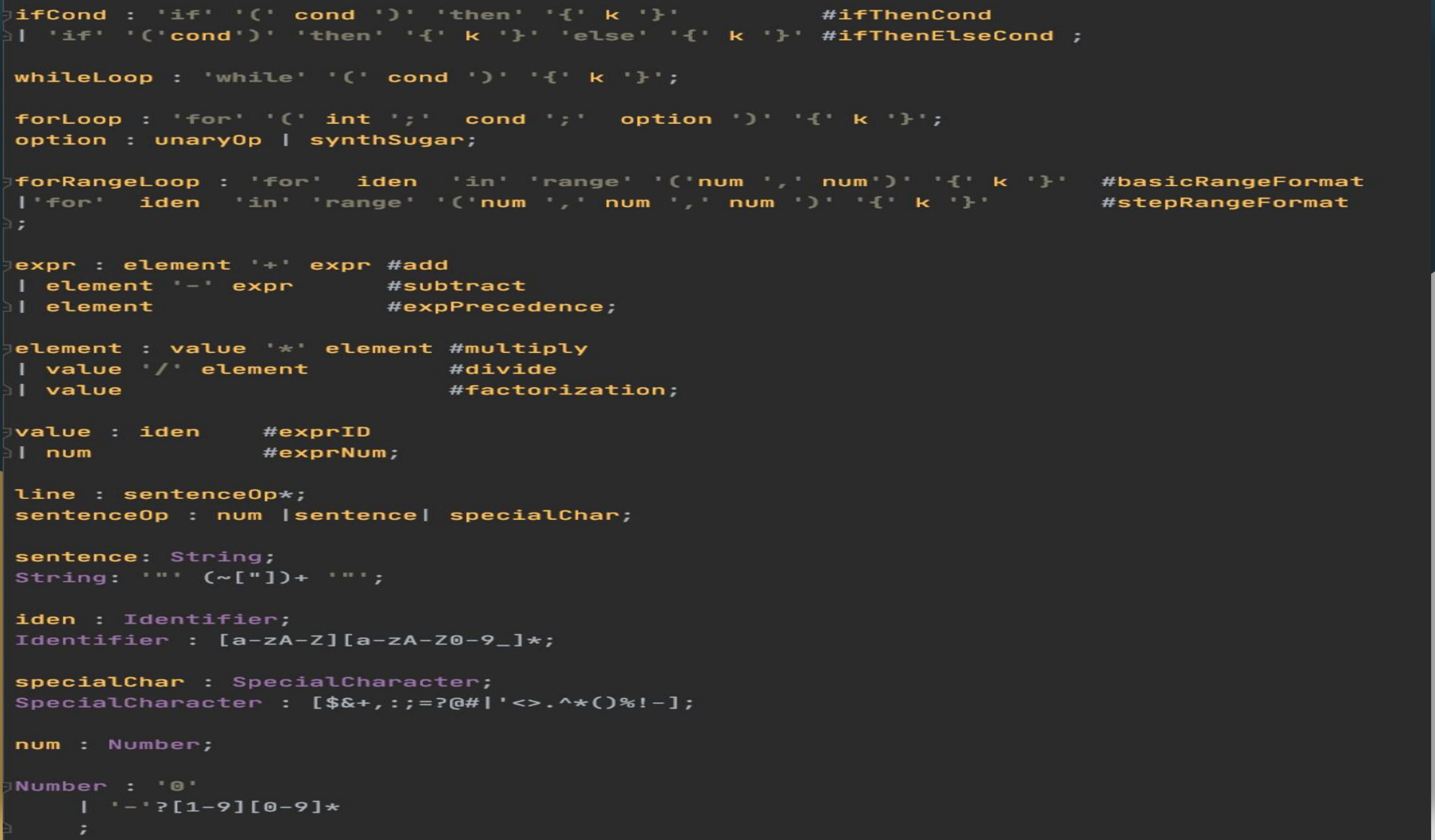
LaxScript Grammar



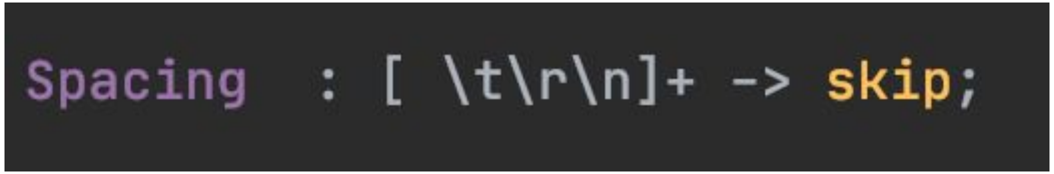
```
grammar LaxScript;

p : k '.';

k : d ';' k
 | d ';'
 | init ';' k
 | init ';'
 | print ';' k
 | print ';'
 | unaryOp ';' k
 | unaryOp ';'
 | ternaryOp ';' k
 | ternaryOp ';'
 | assignOp ';' k
 | assignOp ';'
 | synthSugar ';' k
 | synthSugar ';'
 | ifCond k
 | ifCond
 | whileLoop k
 | whileLoop
 | forLoop k
 | forLoop
 | forRangeLoop k
 | forRangeLoop
;

d :  'int' iden       # declarationInteger
 | 'str' iden         # declarationString
 |'boolean' iden      # declarationBoolean
;

init : int
 | str
 | bool;

print : 'print' '(' line ')' #printStr
 |'print' '(' expr')'           #printExpr
;
```



```
int: 'int' iden '=' num     #numberIntInit
| 'int' iden '=' iden       #identifierIntInit
| 'int' iden '=' expr #expressionIntInit
;

str : 'str' iden '=' iden        #identifierStrInit
|'str' iden '=' '"' line '"'     #sentenceStrInit;

bool :'boolean' iden '=' iden                     #identifierBoolInit
|'boolean' iden '=' boolVal=('true' | 'false')    #identifierBoolVal
;

unaryOp : '++' iden        #preIncrement
| iden '++'                #postIncrement
| '--' iden                #preDecrement
| iden '--'                #postDecrement;

ternaryOp : 'int' iden  '=' cond '?' expr ':' expr                                      #ternaryInt
|'str' iden  '=' cond '?' '"' line '"' ':' '"' line '"'                                 #ternaryStr
|'boolean' iden  '=' cond '?' boolVal=('true' | 'false') ':' boolVal=('true' | 'false') #ternaryBool;

cond : expr condOp=('==' | '<' | '>' | '<=' | '>=' | '!=') expr #conditionOp
| boolVal=('true' | 'false')                                    #conditionBoolOp;

assignOp : iden '=' num               #numberAssignment
| iden '=' boolVal=('true' | 'false') #booleanAssignment
| iden '=' '"' line '"'               #stringAssignment
| iden '=' expr                       #expressionAssignment
;

synthSugar : iden '+=' num    #additionEqualNum
| iden '-=' num               #subtractionEqualNum
| iden '*=' num               #multiplicationEqualNum
| iden '/=' num               #divisionEqualNum
| iden '+=' iden              #additionEqualID
| iden '-=' iden              #subtractionEqualID
| iden '*=' iden              #multiplicationEqualID
| iden '/=' iden              #divisionEqualID
;
```



```
ifCond : 'if' '(' cond ')' 'then' '{' k '}'         #ifThenCond
| 'if' '('cond')' 'then' '{' k '}' 'else' '{' k '}' #ifThenElseCond ;

whileLoop : 'while' '(' cond ')' '{' k '}';

forLoop : 'for' '(' int ';'  cond ';'  option ')' '{' k '}';
option : unaryOp | synthSugar;

forRangeLoop : 'for'  iden  'in' 'range' '('num ',' num')' '{' k '}'  #basicRangeFormat
|'for'  iden  'in' 'range' '('num ',' num ',' num ')' '{' k '}'       #stepRangeFormat
;

expr : element '+' expr #add
| element '-' expr      #subtract
| element               #expPrecedence;

element : value '*' element #multiply
| value '/' element         #divide
| value                     #factorization;

value : iden    #exprID
| num           #exprNum;

line : sentenceOp*;
sentenceOp : num |sentence| specialChar;

sentence: String;
String: '"' (~["])+ '"';

iden : Identifier;
Identifier : [a-zA-Z][a-zA-Z0-9_]*;

specialChar : SpecialCharacter;
SpecialCharacter : [$&+,:;=?@#|'<>.^*()%!-];

num : Number;

Number : '0'
    | '-'?[1-9][0-9]*
    ;
```

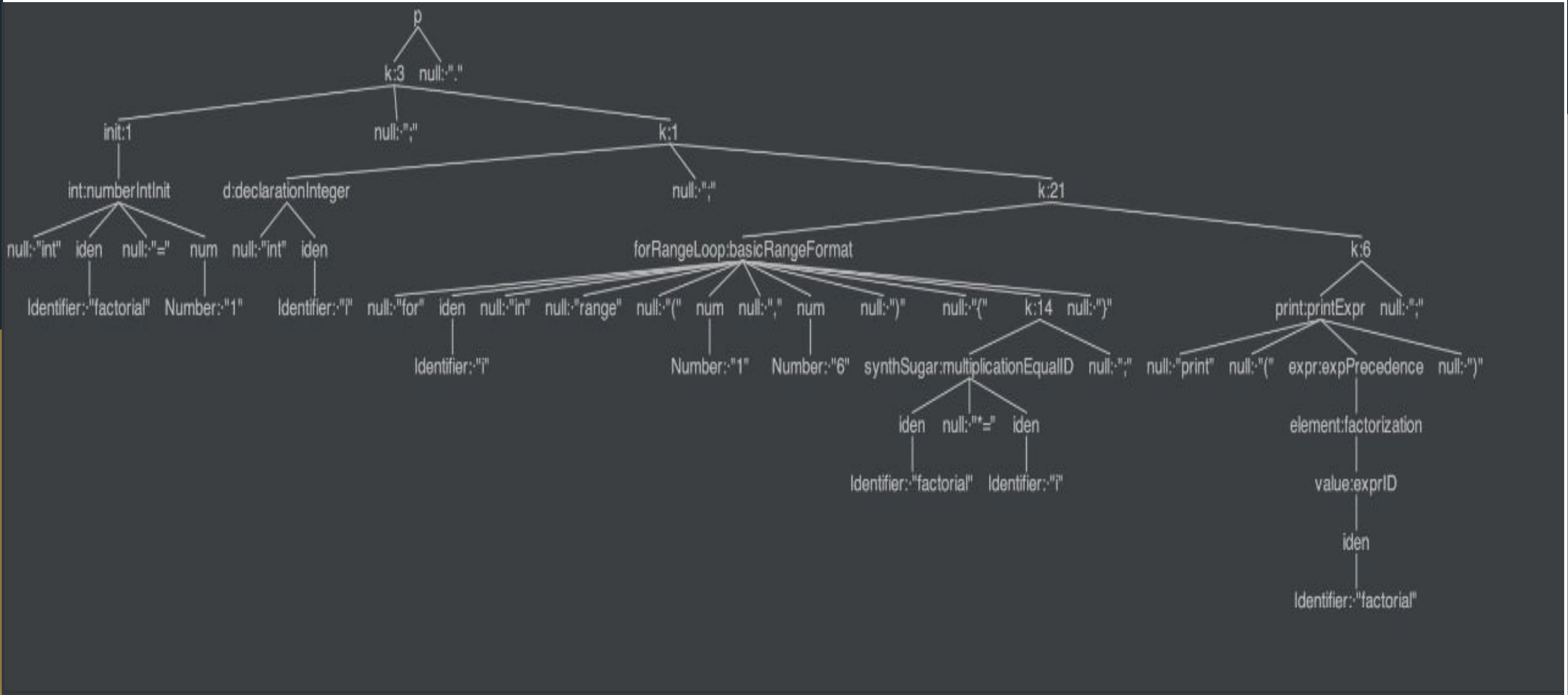


```
Spacing  : [ \t\r\n]+ -> skip;
```

Parse Tree

```
int factorial = 1;
int i;
for i in range(1,6){
      factorial *= i;
}
print(factorial);
.
```

Parse Tree

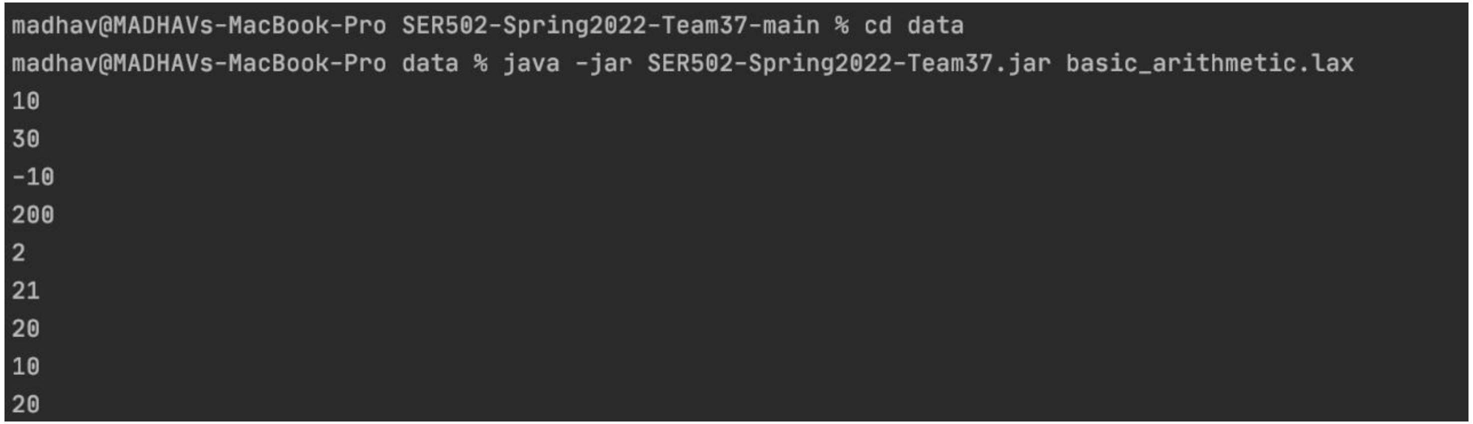

p
k:3
null:"."
init:1
null:";"
k:1
int:numberIntInit
d:declarationInteger
null:";"
k:21
null:"int"
iden
null:"="
num
null:"int"
iden
forRangeLoop:basicRangeFormat
k:6
Identifier:"factorial"
Number:"1"
Identifier:"i"
null:"for"
iden
null:"in"
null:"range"
null:"("
num
null:","
num
null:")"
null:"{"
k:14
null:"}"
print:printExpr
null:";"
Identifier:"i"
Number:"1"
Number:"6"
synthSugar:multiplicationEqualID
null:";"
null:"print"
null:"("
expr:expPrecedence
null:")"
iden
null:"*="
iden
element:factorization
Identifier:"factorial"
Identifier:"i"
value:exprID
iden
Identifier:"factorial"

Interpreter Architecture

```java
CharStream cStream = CharStreams.fromString(data);

LaxScriptLexer lsLexer = new LaxScriptLexer(cStream);
CommonTokenStream tokenStream = new CommonTokenStream(lsLexer);
LaxScriptParser lsParser = new LaxScriptParser(tokenStream);
ParseTree pTree = lsParser.p();
LaxScriptEvaluate eval = new LaxScriptEvaluate();
eval.visit(pTree);
```

Sample Code

```
int a;
a = 10;
print(a);

int b;
b = 20;

int c;

c = a+b;
print(c);

c = a -b;
print(c);

c = a * b;
print(c);

c = b/a;
print(c);

b++;
print(b);

b--;
print(b);

b/= 2;
print(b);

b*= 2;
print(b);
.
```



```
madhav@MADHAVs-MacBook-Pro SER502-Spring2022-Team37-main % cd data
madhav@MADHAVs-MacBook-Pro data % java -jar SER502-Spring2022-Team37.jar basic_arithmetic.lax
10
30
-10
200
2
21
20
10
20
```

Future Implementation

- Arrays.
- Functions
- String operations : Slicing, Multiplication, Concatenation

Thank You