Week 2- AUT Assignment

Group 3

Table of contents

Problem description

Assignment 2: assignments

- Add variables to your grammar, to store the results of the expressions (or to use it inside an expression)
- Add a `print` command (to print the values of expressions (in particular: variables))

Implementation

For this week we have been tasked to add the possibility to store the results of the expressions by adding variables, and to print the results of the variables.

Below the grammar of our language can be found. In order to make it more proper, we have also added the possibility to declare strings and booleans, besides integers.

In our implementation, since we have multiple variable types, we decided to create 3 dictionaries, each dictionary representing a certain type, so that the compiler is not misled.

Our idea by using the dictionary is that we can think a variable is an identifier with a key (in this case the "key" is its value), which needs to be stored or overwritten while executing different statements.

In the "Output" section, we have provided some test cases with the terminal output and the parse tree.

```
      grammar MyGrammar;

// rules
myStart :  statement + EOF;

statement:   expression     #otherExpr
         |   variable_declaration # assign
         |   Print op=(INT| BOOLEAN |ID |STRING)  # printVar
         |   Print mathExpression  # printExpr
         ;

variable_declaration : int_variable_assignment
                     | bool_variable_assignment
                     | string_variable_assignment
                     ;
```

```
string_variable_assignment : StringType ID  # stringDeclaration
                           | StringType ID Equals STRING # stringAssign
                           | ID Equals STRING  # stringAssignValue
                           ;

bool_variable_assignment : BoolType ID  # boolDeclaration
                         | BoolType ID Equals BOOLEAN  # boolAssign
                         | ID Equals BOOLEAN  # boolAssignValue
                         ;

int_variable_assignment :  IntType ID  # intDeclaration
                         | IntType ID Equals mathExpression  # intAssign
                         | ID Equals mathExpression  # intAssignValue
                         ;

expression:   mathExpression #MathExp
          |   BOOLEAN       #ValueBoolean
          |   STRING     #ValueString
          ;

mathExpression:  mathExpression op=MUL mathExpression #  Mul
             |   mathExpression op=DIV mathExpression #  Div
             |   mathExpression op=ADD mathExpression #  Add
             |   mathExpression op=SUB mathExpression #  Sub
             |   mathExpression op=POW mathExpression #  Pow
             |   mathExpression op=FACT  #  Fact
             |   PARANL mathExpression PARANR  # parens
             |   INT #ValueNumber
             |   ID    #ValueVariable
            ;


// tokens
MUL:    '*';
DIV:    '/';
ADD:    '+';
SUB:    '-';
POW:    '^';
FACT:   '!';
PARANL: '(';
PARANR: ')';
```

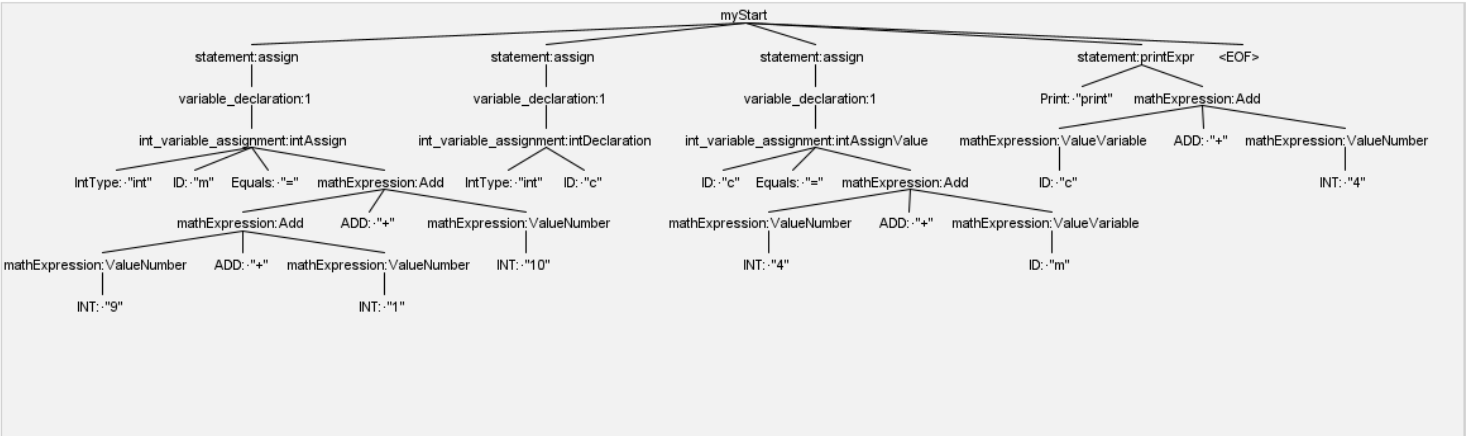
```
Equals:      '=';

IntType: 'int';
BoolType: 'bool';
StringType: 'string';

DOT: '.';
COMMA: ',';
SEMICOLON: ';';
StringParen: '"';

Print: 'print';

INT     : SUB?[0-9]+(DOT[0-9]+)? ;
BOOLEAN: 'True'|'False';
ID: [_A-Za-z][A-Za-z_!0-9.]* ;
STRING : '"' ID '"';

COMMENT : '//' .+? ('\n'|EOF) -> skip;
WS  : [ \t\r\n]+ -> skip;
```

Output

For some reason, the Parse Tree would not get displayed accordingly by using the “grun” command, therefore we used a plugin found in the JetBrain marketplace to fix this issue.

1.
```
int m = 9 + 1 + 10
int c
c = 4 + m
print c + 4
```

Parse Tree:

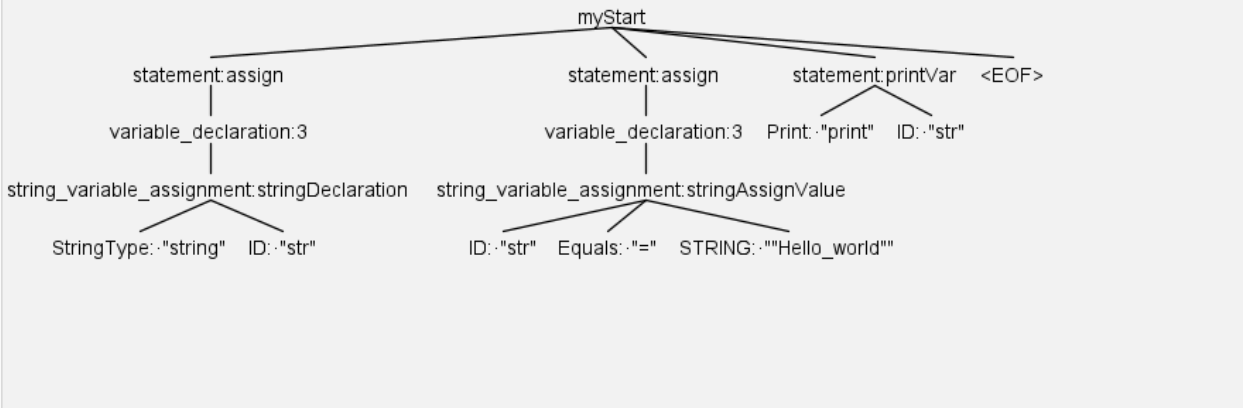


Terminal :

```
enterMyStart()
terminal-node: 'int'
terminal-node: 'm'
terminal-node: '='
terminal-node: '9'
terminal-node: '+'
terminal-node: '1'
terminal-node: 'print'
terminal-node: 'c'
Added id to letterstack: c meaning adding 24 to numberstack
terminal-node: '+'
terminal-node: '4'
Added 24 with 4
printed c+4 = 28
terminal-node: '<EOF>'
exitMyStart()
```

2.
```
string str
str = "Hello_world"
print str
```

Parse Tree:

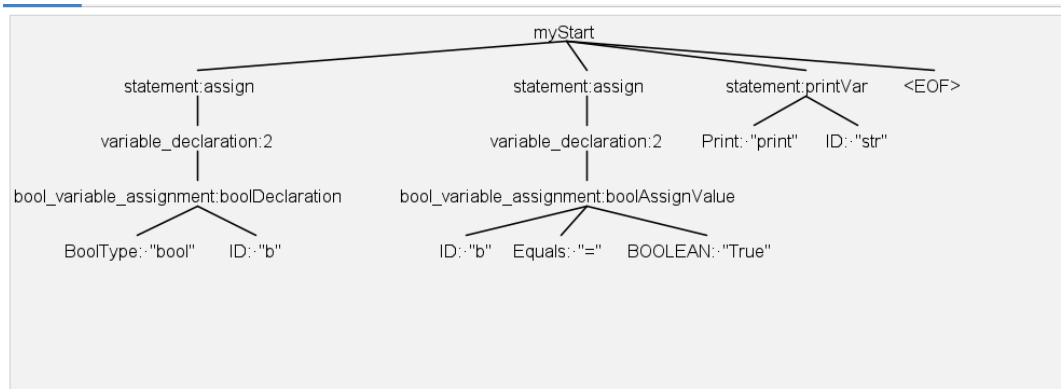


Terminal:

```
enterMyStart()
terminal-node: 'string'
terminal-node: 'str'
memory put: str =
terminal-node: 'str'
terminal-node: '='
terminal-node: '"Hello_world"'
memory put: str = Hello_world
terminal-node: 'print'
terminal-node: 'str'
print str = Hello_world
terminal-node: '<EOF>'
exitMyStart()
```

3.
```
bool b
b = True
print b
```

Parse Tree:

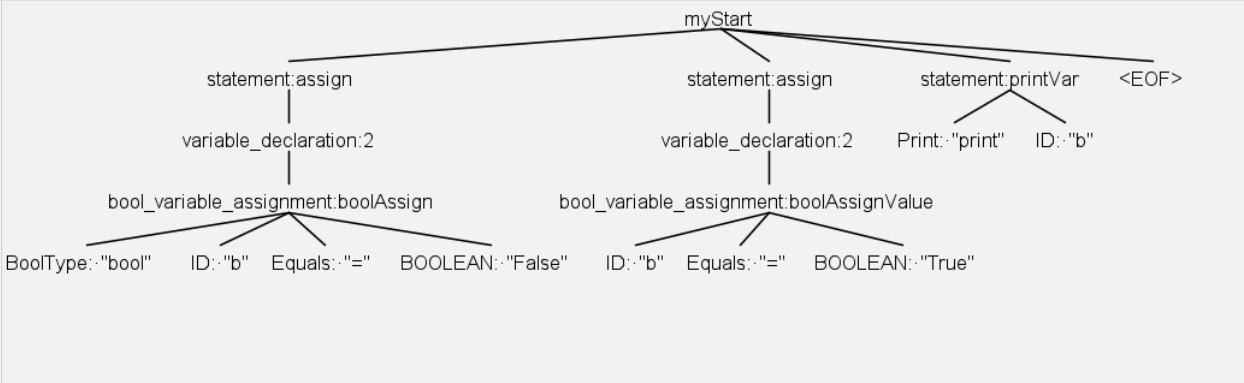


Terminal:

```
enterMyStart()
terminal-node: 'bool'
terminal-node: 'b'
memory put: b = false
terminal-node: 'b'
terminal-node: '='
terminal-node: 'True'
memory put: b = true
terminal-node: 'print'
terminal-node: 'b'
print b = true
terminal-node: '<EOF>'
exitMyStart()
```

4.
```
bool b = False
b = True
print b
```

Parse Tree:

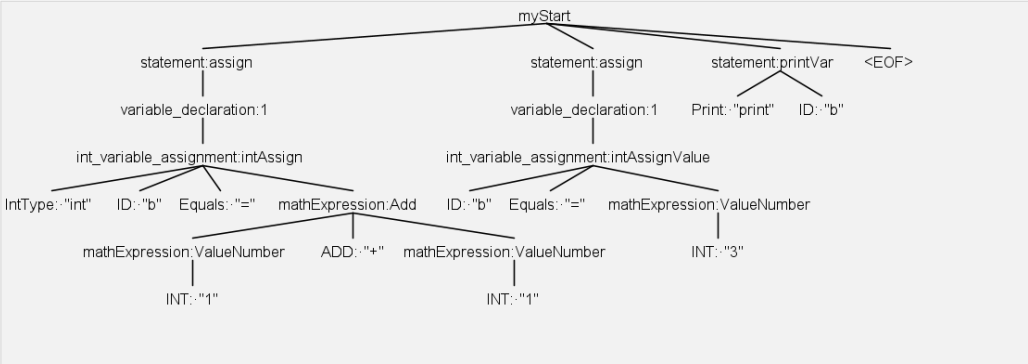


Terminal:

```
enterMyStart()
terminal-node: 'bool'
terminal-node: 'b'
terminal-node: '='
terminal-node: 'False'
memory put: b = false
terminal-node: 'b'
terminal-node: '='
terminal-node: 'True'
memory put: b = true
terminal-node: 'print'
terminal-node: 'b'
print b = true
terminal-node: '<EOF>'
exitMyStart()
```

5.
```
int b = 1 + 1
b = 3
print b
```

Parse Tree:

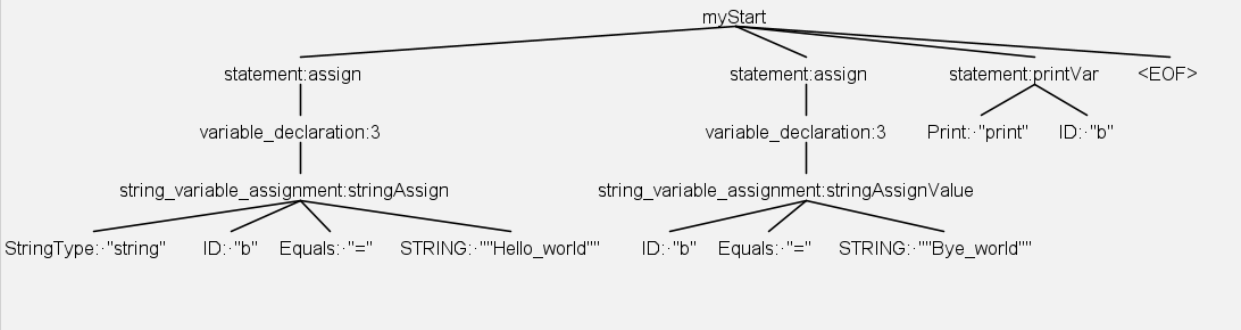


Terminal:

```
enterMyStart()
terminal-node: 'int'
terminal-node: 'b'
terminal-node: '='
terminal-node: '1'
terminal-node: '+'
terminal-node: '1'
Added 1 with 1
memory put: b = 2
terminal-node: 'b'
terminal-node: '='
terminal-node: '3'
memory put: b = 3
terminal-node: 'print'
terminal-node: 'b'
print b = 3
terminal-node: '<EOF>'
exitMyStart()
```

6.
```
string b = "Hello_world"
b = "Bye_world"
print b
```

Parse Tree:

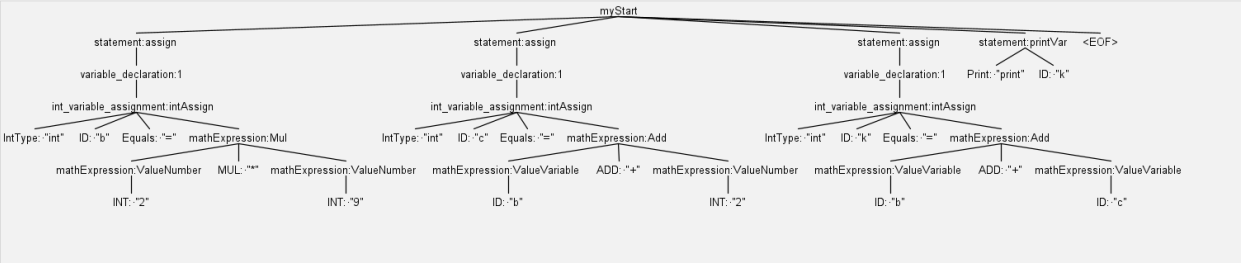


Terminal:

```
enterMyStart()
terminal-node: 'string'
terminal-node: 'b'
terminal-node: '='
terminal-node: '"Hello_world"'
memory put: b = Hello_world
terminal-node: 'b'
terminal-node: '='
terminal-node: '"Bye_world"'
memory put: b = Bye_world
terminal-node: 'print'
terminal-node: 'b'
print b = Bye_world
terminal-node: '<EOF>'
exitMyStart()
```

7.
```
int b = 2 * 9
int c = b + 2
int k = b + c
print k
```

Parse Tree:



Terminal:

```
enterMyStart()
terminal-node: 'int'
terminal-node: 'b'
terminal-node: '='
terminal-node: '2'
memory put: b = 18
terminal-node: 'int'
terminal-node: 'c'
terminal-node: '='
terminal-node: 'b'
Added id to letterstack: b meaning adding 18 to numberstack
terminal-node: '+'
terminal-node: '2'
Added 18 with 2
memory put: c = 20
terminal-node: 'int'
terminal-node: 'k'
terminal-node: '='
terminal-node: 'b'
Added id to letterstack: b meaning adding 18 to numberstack
terminal-node: '+'
terminal-node: 'c'
Added id to letterstack: c meaning adding 20 to numberstack
Added 18 with 20
memory put: k = 38
terminal-node: 'print'
terminal-node: 'k'
print k = 38
```