

PROJECT – Parts 4 & 5: Parser and StaticSemantics



Youssef Yousfi

Hajar El Boutahiri

Hiba Aqqaoui

CSC 3315 Languages and Compilers

Professor Violetta Cavalli-Sforza

PROJECT – Parts 4 & 5: Parser and StaticSemantics

PROJECT – Parts 4 & 5: Parser and StaticSemantics

Documentation:

To complete this phase, we worked for more than 65 hours, including weekends, and whole nights.

The teammates are:

- Youssef Yousfi
- Hajar El Boutahiri
- Hiba Aqqaoui

How to run the code:

In order to run the code: you need to have first all files in one folder:

In other words:

- Start by unzipping the folder submitted on Canvas:
- Copy the address of the folder from your computer
- Open cmd to run the code
- Write cd + the address folder
- Copy the Command Python: Parser_StaticSemantics.py

```
Microsoft Windows [version 10.0.19045.2251]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\DELL>cd C:\Users\DELL\Desktop\Deliverable_4_5

C:\Users\DELL\Desktop\Deliverable_4_5>Python Parser_StaticSemantics.py
```

Input and output files used by the parser:

As input files, the parser needs “file.txt” where the code is written. And the output file is “file2.txt”. The output file will mainly display the output of the Lexer phase. Whereas, all the output of the parser phase is visible in the cmd.

Description of interaction between parser and Lexer:

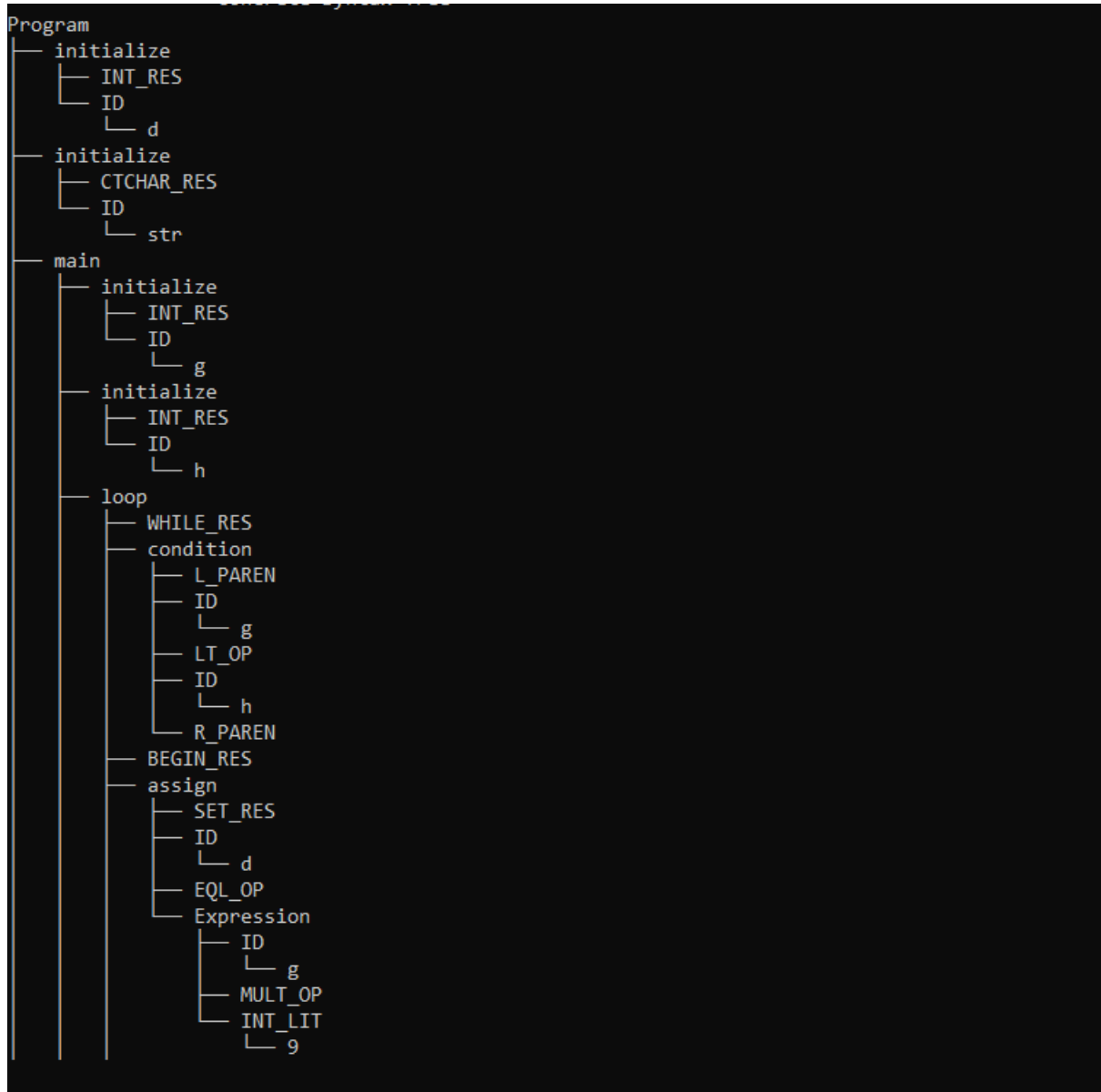
During the parser phase, the parser does not call anymore the lexer. In fact, the lexer sends a stream of tokens to the parser. And during parsing, the parser access just the stream of tokens. We are also passing to the parser, an array of line counters for error display reasons.

Description of parser output format (tree) with examples:

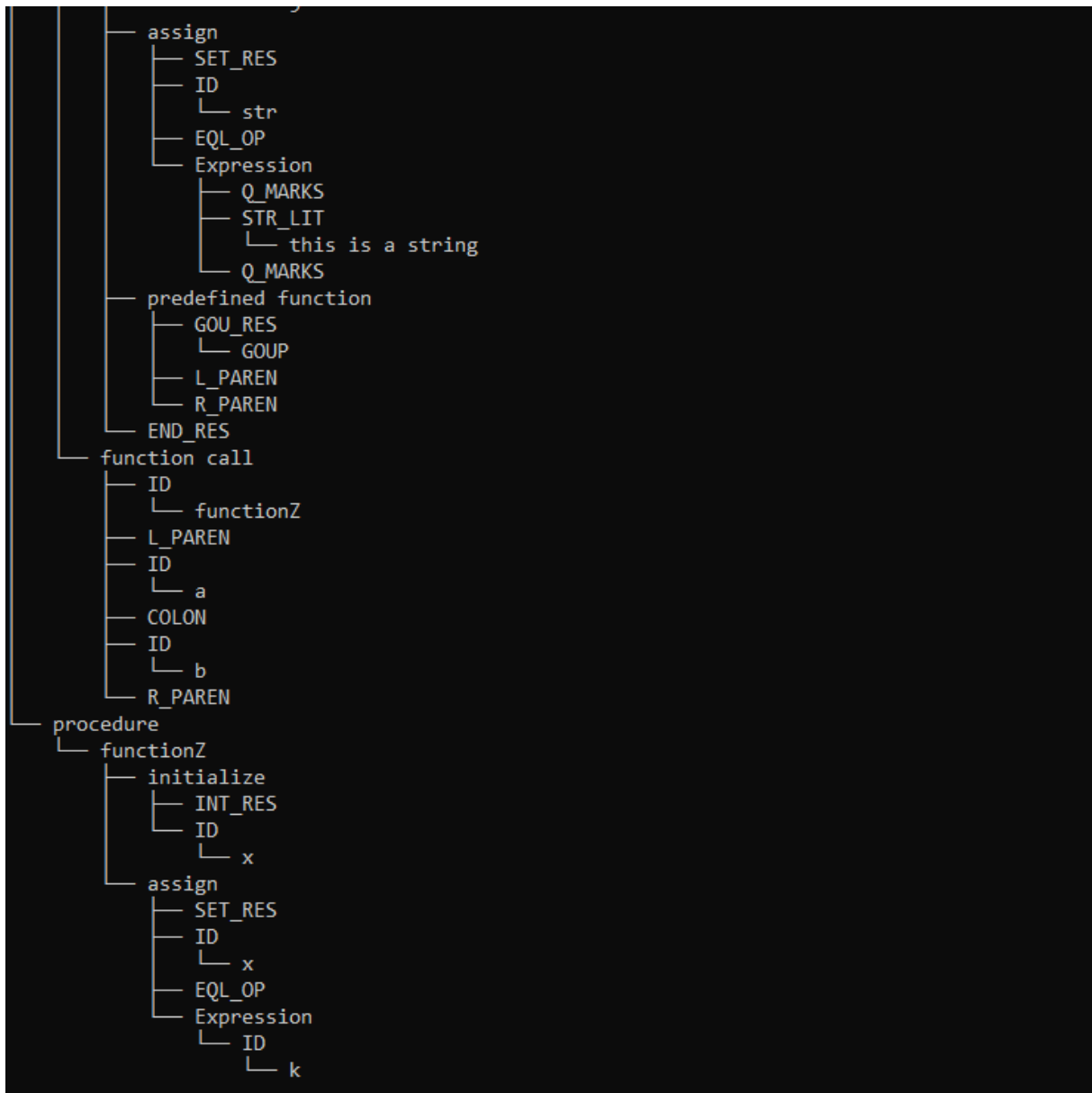
PROJECT – Parts 4 & 5: Parser and StaticSemantics

The parser outputs two trees: The first tree is a concrete syntax tree which contains all the nodes that describe the way our grammar is built, and the second tree is an abstract syntax tree which is a simplified version of the concrete syntax tree where we do not have the nodes that are not necessary for code generation.

Output 1: Concrete Syntax Tree

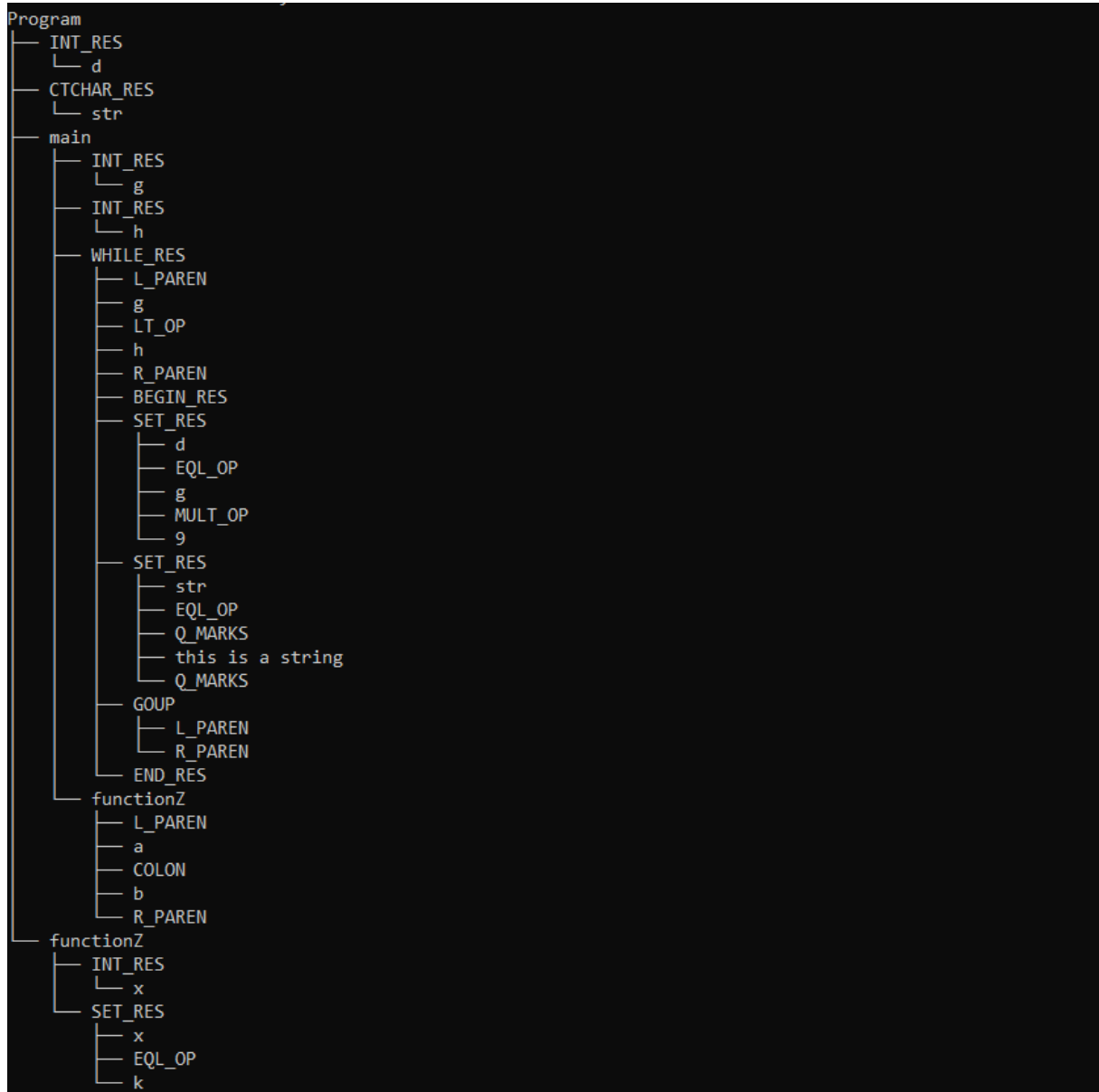


PROJECT – Parts 4 & 5: Parser and StaticSemantics



Output 2: Abstract Syntax Tree

PROJECT – Parts 4 & 5: Parser and StaticSemantics



Parser:

Grammar coverage of acceptor:

The acceptor covers the whole grammar and basically after the acceptance of each part of the code, the acceptor displays a message. At the end of the

PROJECT – Parts 4 & 5: Parser and StaticSemantics

execution, if everything abides the grammar rules. The acceptor displays a message otherwise the code stops executing, and an error message is displayed.

Concrete Syntax Tree construction:

A concrete syntax tree named 'CST' is made from the `Tree()` function in the `treelib` module. The `treelib` module allows us to create nodes and add them to the CST by using the statement '`CST.create_node(tag,id,parent)`' :

In this statement, `tag` refers to the text that will be printed as the node's name when the concrete syntax tree is printed, `id` is the unique identifier of that node in that tree (`treelib` generates a unique identifier if this field is left empty), `parent` is the parent node of the node we currently creating (if left empty it will make the node the root of the tree, if one does not exist already)

Concrete Syntax Tree printing:

The Concrete Syntax tree is printed through the statement:

`CST.show(key=False)`; we used `key=false`: for the nodes in the tree not to be sorted alphabetically and instead using a first in order.

Error handling and reporting:

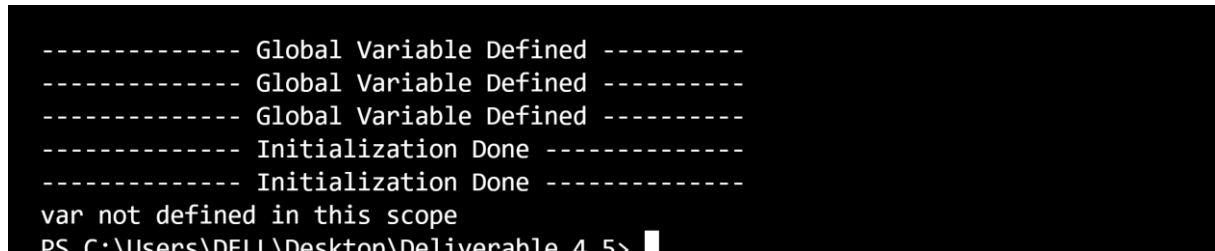
This phase handles all possible errors from incomplete code, missing lexemes, non-defined variables or/ and functions, wrong arguments passage, wrong type assignments...

Static Semantics:

Check that identifiers defined before used:

PROJECT – Parts 4 & 5: Parser and StaticSemantics

In this part, we have ensured that all our identifiers have been defined before: we have showed an example of error messages in case of non-initial definition of the used identifier (see Test Suite 8). And here is a screenshot:

A screenshot of a terminal window with a black background and white text. The text shows several status messages: 'Global Variable Defined' repeated three times, followed by 'Initialization Done' repeated twice. Then, an error message 'var not defined in this scope' is displayed. At the bottom, a command prompt shows the path 'PS C:\Users\DELL\Desktop\Deliverable 4_5>' followed by a cursor.

```
----- Global Variable Defined -----  
----- Global Variable Defined -----  
----- Global Variable Defined -----  
----- Initialization Done -----  
----- Initialization Done -----  
var not defined in this scope  
PS C:\Users\DELL\Desktop\Deliverable 4_5>
```

Scope indicator added in Symbol Table:

Scope indicators (local or global) have been added to the Symbol Table. Local scope if it has been defined inside a function and global variable if it has been defined at the beginning of the program.

Function call checks (actuals match formals):

We have added a function called “call_function_check()”, to check if all called functions are defined on the code or not. Please see the screenshot below:

```

21     }
22     void delete(bool d){
23     | | int r;
24     }
25
26

```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

>

```

----- Assignment Done -----
----- While Loop -----
----- Return Statement -----
----- Main Defined -----
----- Initialization Done -----
----- Function Defined -----

***** Acceptor Message: Correct! *****

Error: Function modify is Not Defined
PS C:\Users\DELL\Desktop\Deliverable 4 5>

```

An abstract syntax tree named ‘AST’ is also made from the `Tree()` function in the `treelib` module. The `treelib` module allows us to create nodes and add them to the AST by using the statement ‘`AST.create_node(tag,id,parent)`’ :

In the construction of the AST, the nodes that are not immediately useful in generating code get skipped.

PROJECT – Parts 4 & 5: Parser and StaticSemantics

Abstract Syntax Tree printing:

The Abstract Syntax tree is printed through the statement:

`AST.show(key=False)`; we used `key=false`: for the nodes in the tree not to be sorted alphabetically and instead using a first in order.

Testing:

Test suite coverage:

We have added 10 test suites: The first one is the correct version to test all the functions and hence our whole grammar. For the nine lefts, each one of them has a specific error (from the parser functionalities to the static semantic ones).

- Test Suite 1: Correct Version to be able to test our entire grammar
- Test Suite 2: Missing Semi Colon
- Test Suite 3: Didn't close the main curly braces
- Test Suite 4: Missing SET keyword
- Test Suite 5: Missing While reserved word in the loop
- Test Suite 6: Not passing an argument to the function
- Test Suite 7: Calling a non-defined function
- Test Suite 8: Using a non-defined variable
- Test Suite 9: Wrong type assignment
- Test Suite 10: Passing arguments more than needed

⇒ All these test suites can be found on the shared folder in a folder called “Test Suites”.

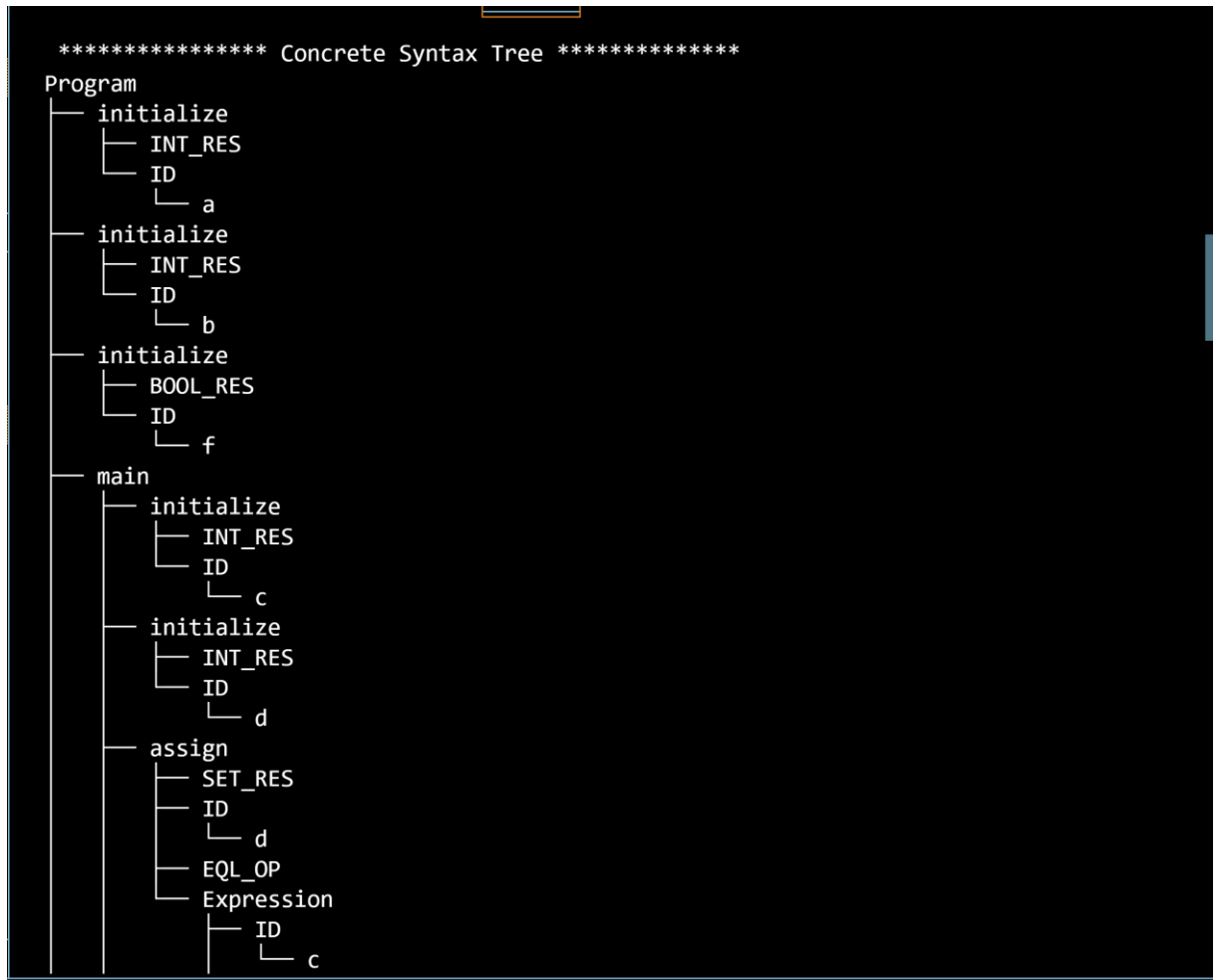
To be able to test them, please copy the code on the file named “file.txt” and follow the previous guidelines to run the code.

PROJECT – Parts 4 & 5: Parser and StaticSemantics

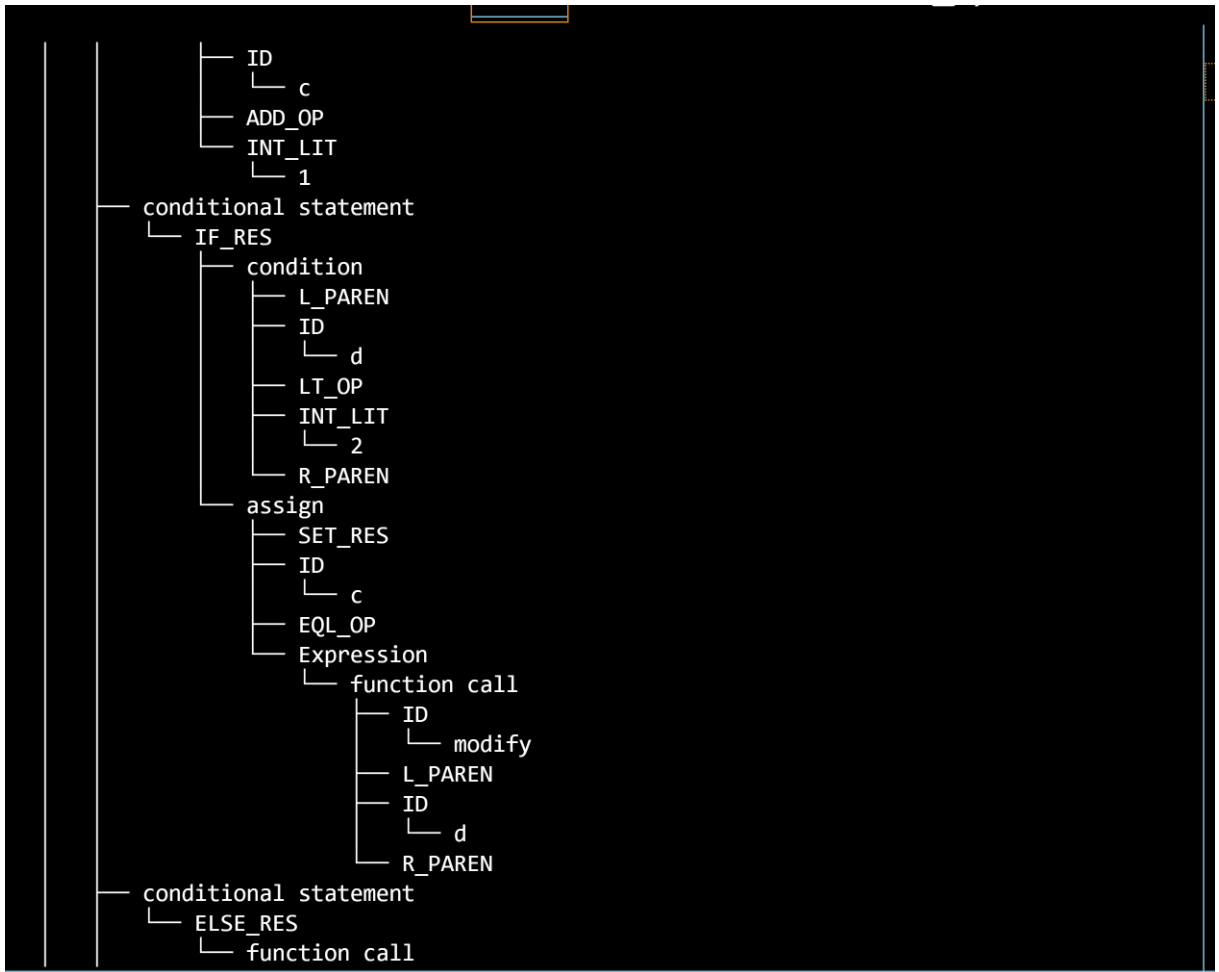
Test suite clarity (input, CST, AST):

Test Suite 1: Correct Version to be able to test our entire grammar

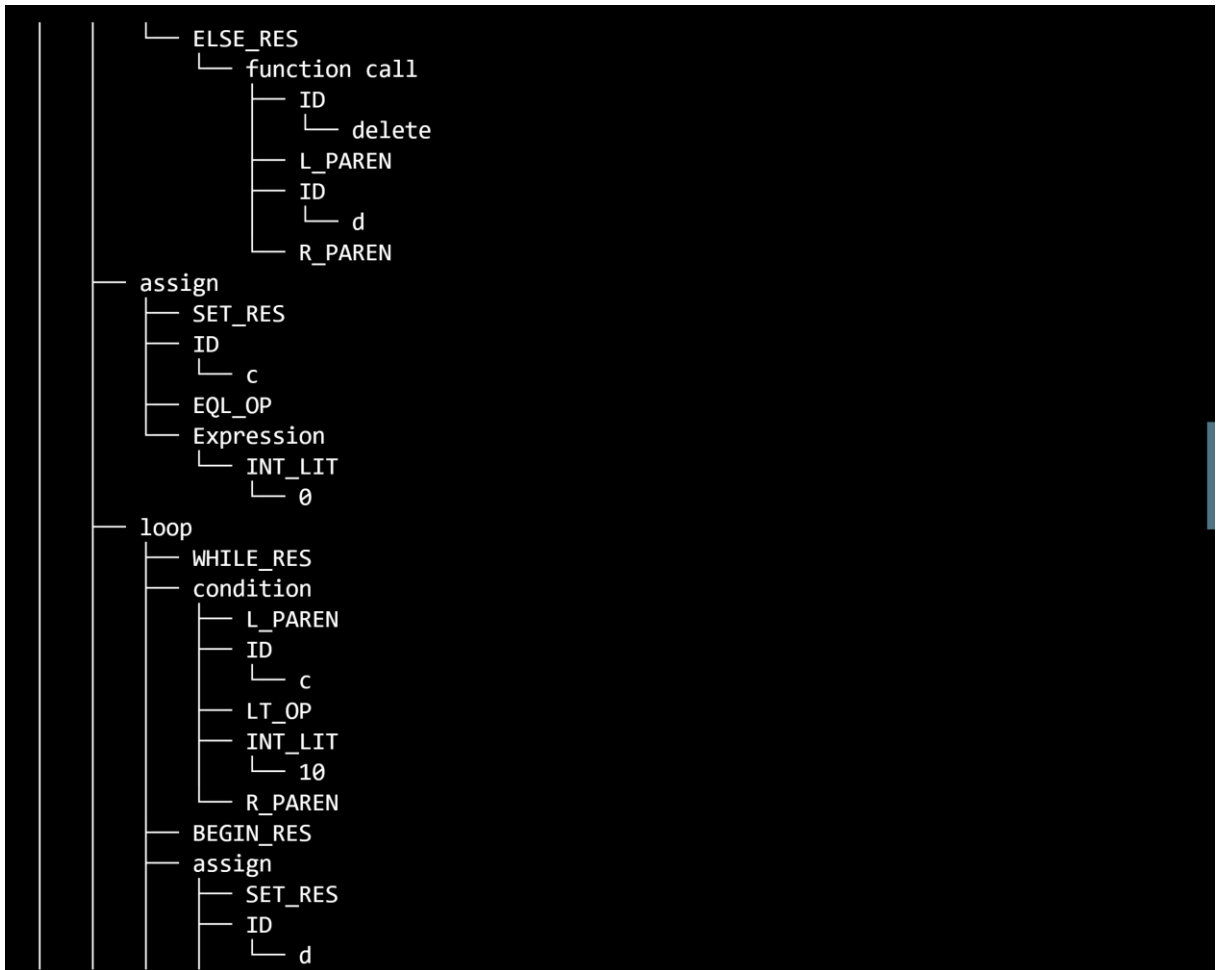
- Input: see Test Suite 1 program
- CST:



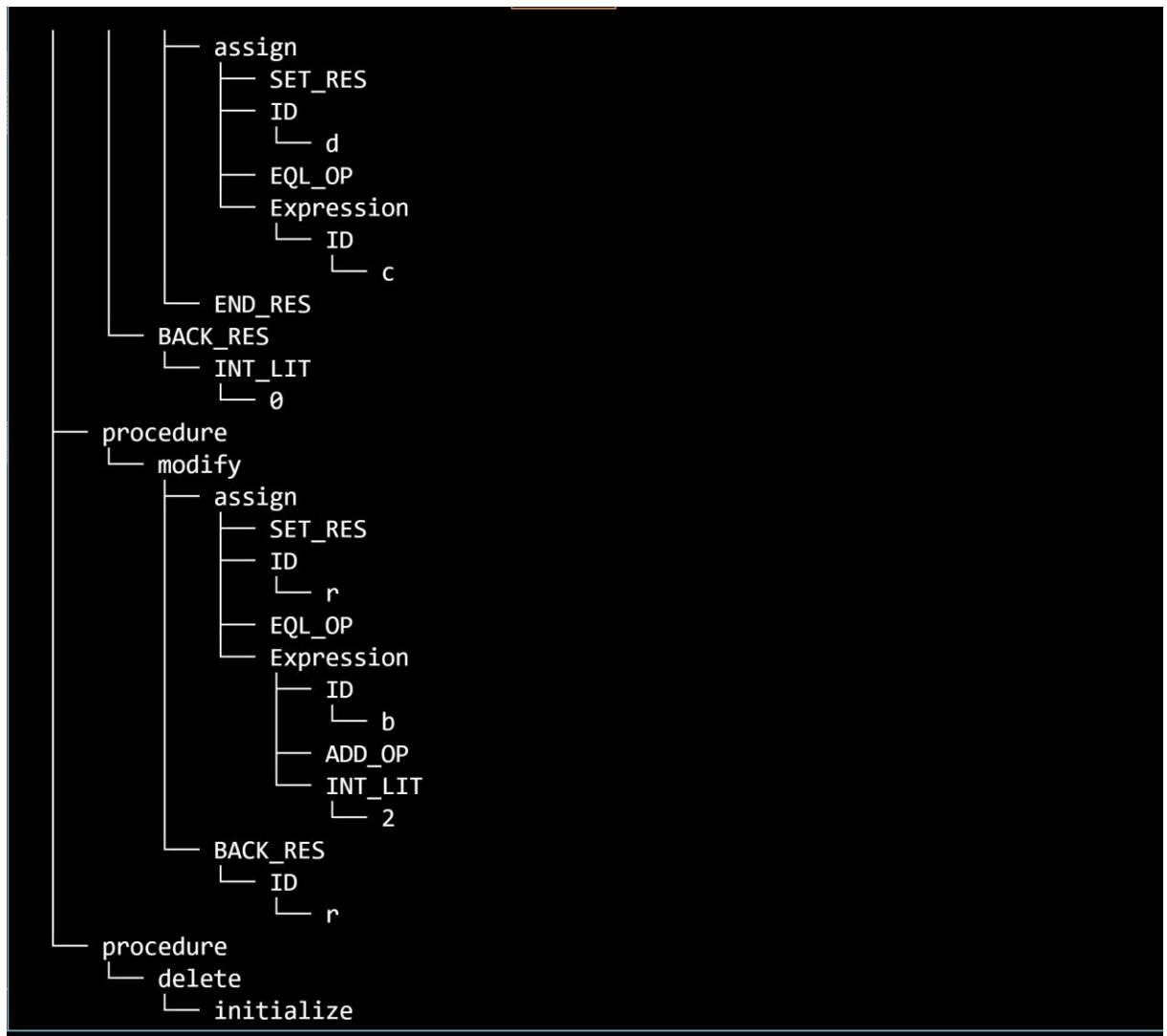
PROJECT – Parts 4 & 5: Parser and StaticSemantics



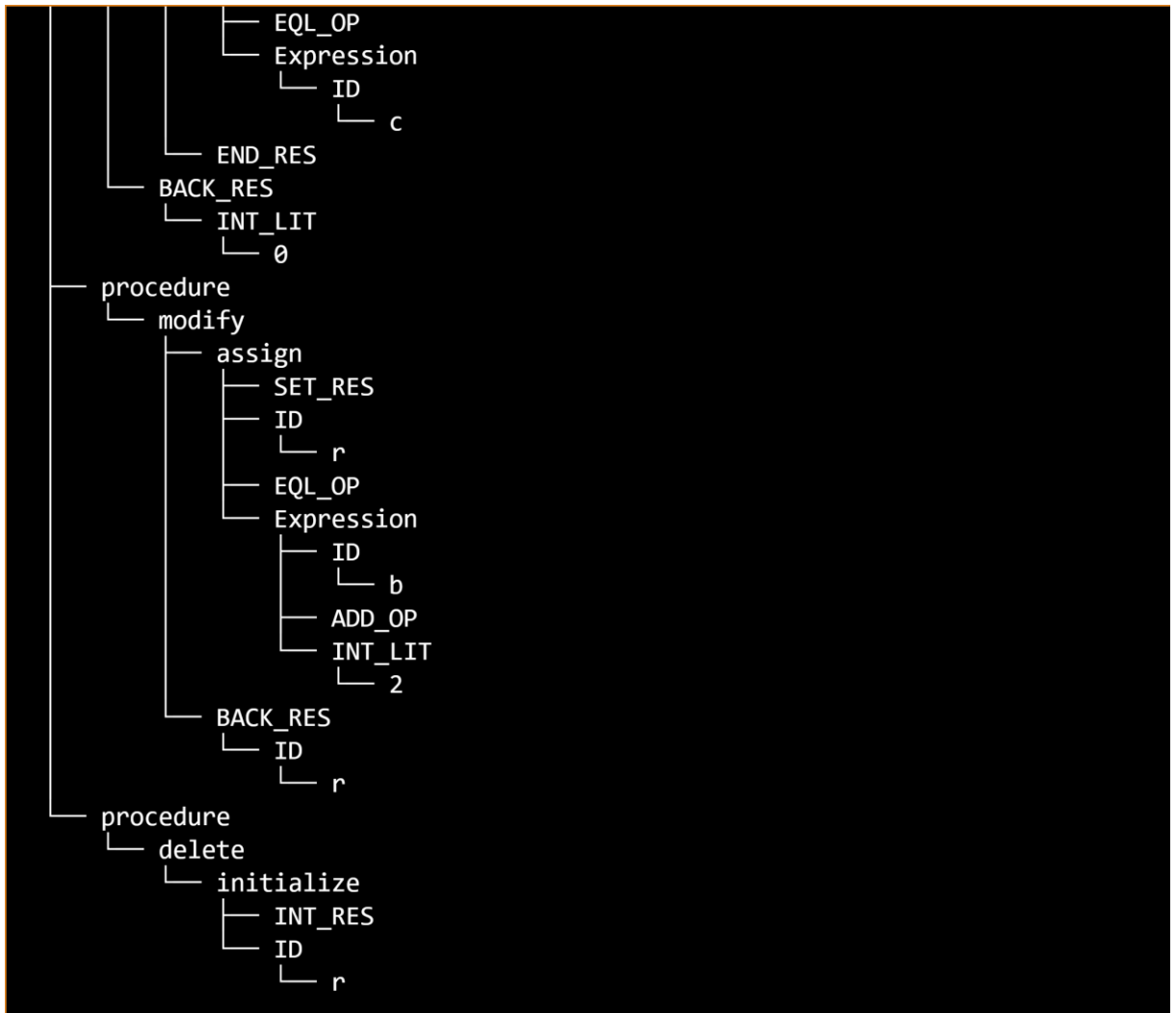
PROJECT – Parts 4 & 5: Parser and StaticSemantics



PROJECT – Parts 4 & 5: Parser and StaticSemantics

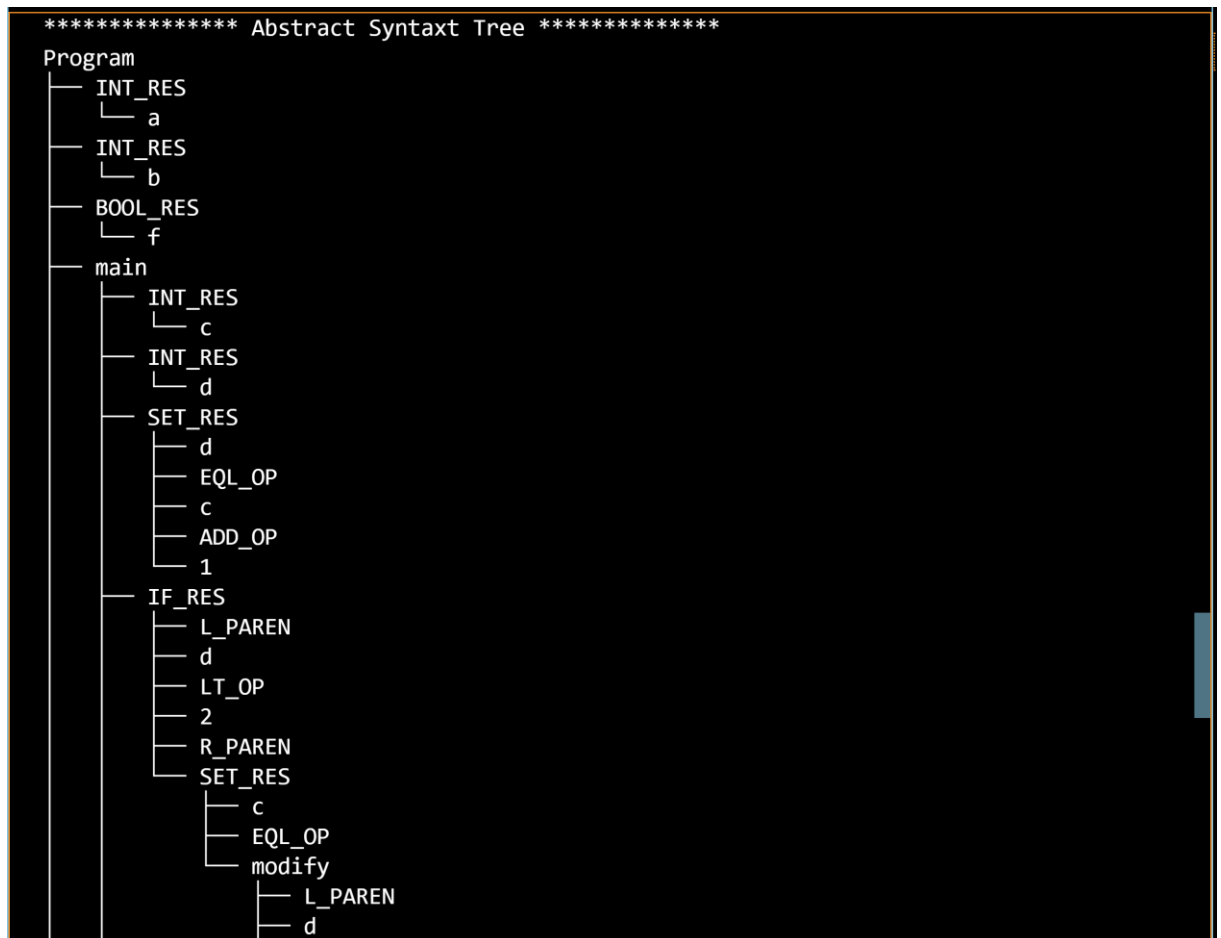


PROJECT – Parts 4 & 5: Parser and StaticSemantics

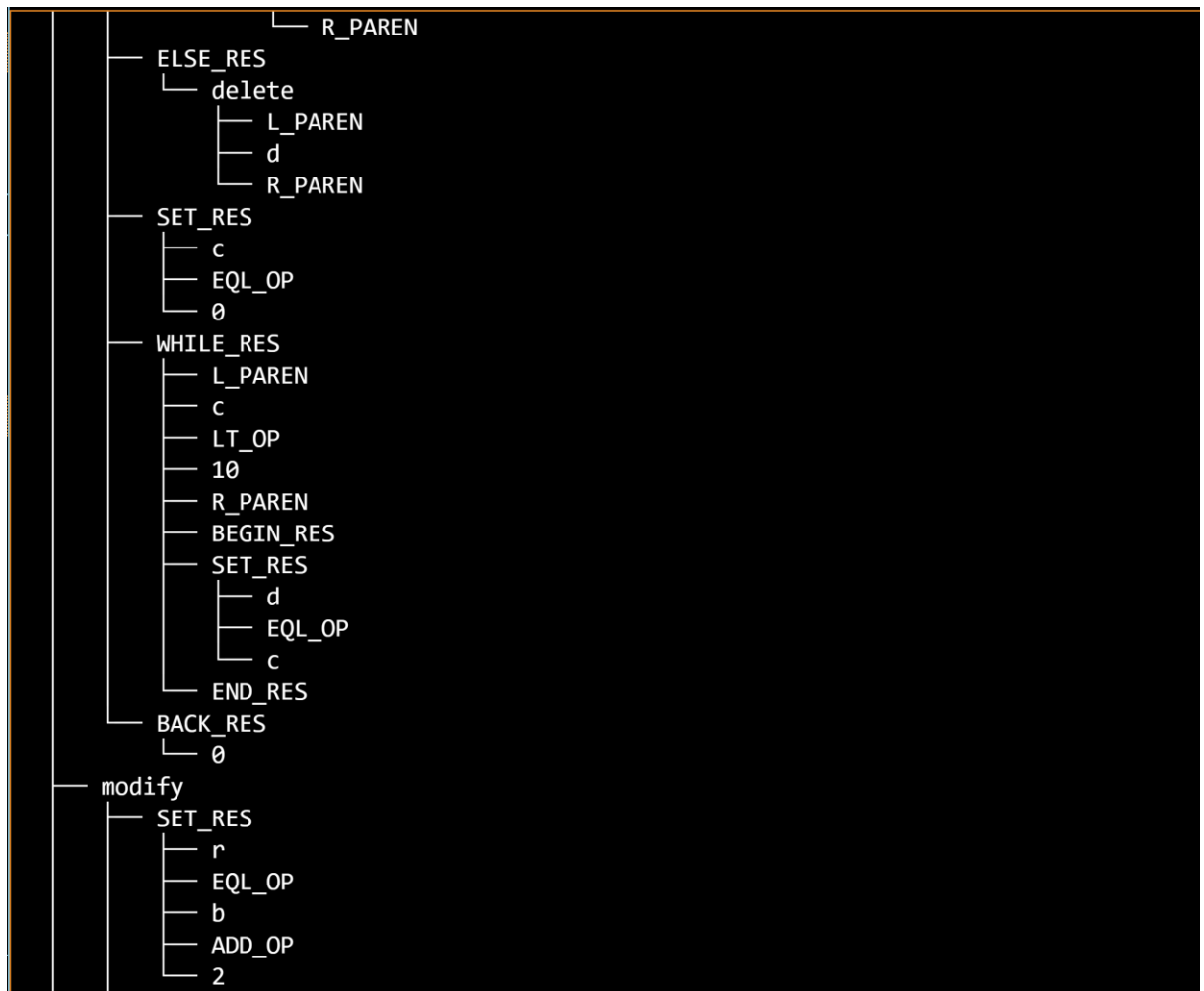


- AST:

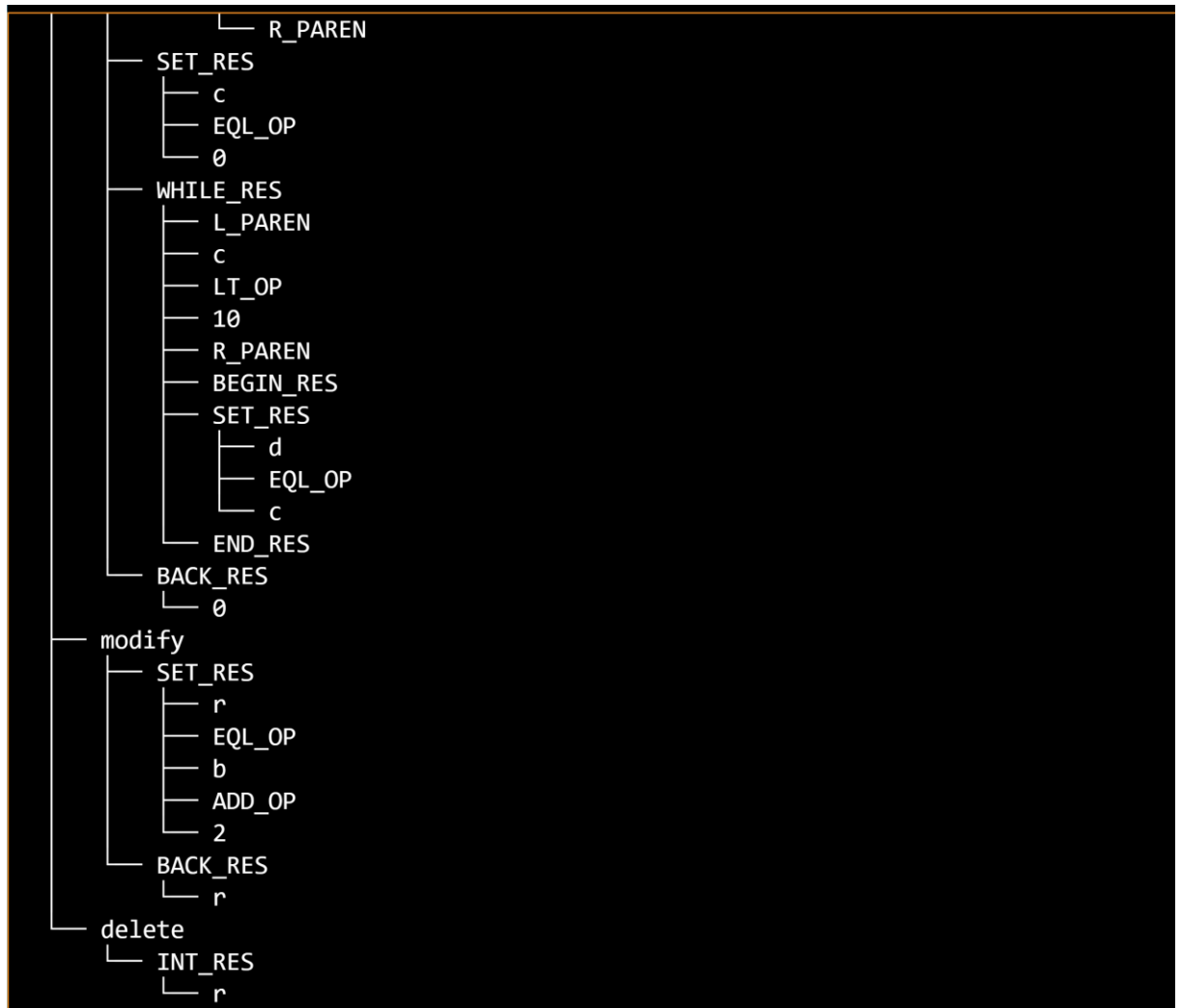
PROJECT – Parts 4 & 5: Parser and StaticSemantics



PROJECT – Parts 4 & 5: Parser and StaticSemantics



PROJECT – Parts 4 & 5: Parser and StaticSemantics



- Output:

PROJECT – Parts 4 & 5: Parser and StaticSemantics

```
----- Global Variable Defined -----
----- Global Variable Defined -----
----- Global Variable Defined -----
----- Initialization Done -----
----- Initialization Done -----
----- Assignment Done -----
----- Function Called -----
----- Assignment Done -----
----- If Statement -----
----- Function Called -----
----- Else Statement -----
----- Assignment Done -----
----- Assignment Done -----
----- While Loop -----
----- Return Statement -----
----- Main Defined -----
----- Assignment Done -----
----- Return Statement -----
----- Function Defined -----
----- Initialization Done -----
----- Function Defined -----
```

```
***** Acceptor Message: Correct! *****
```

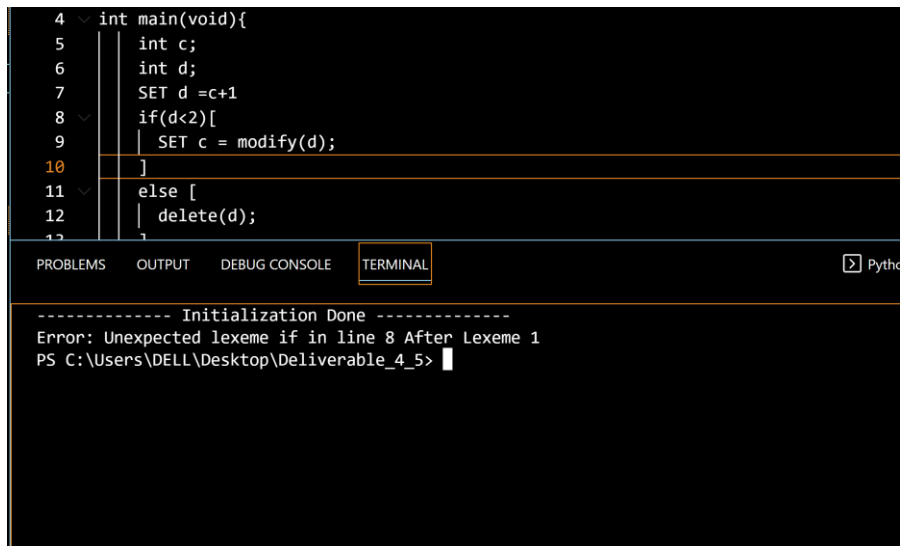
```
***** Concrete Syntax Tree *****
```

```
Program
├── initialize
│   ├── INT_RES
│   │   └── ID
│   │       └── a
```

Test Suite 2: Missing Semi Colon

- Input: see Test Suite 2 program
- Output:

PROJECT – Parts 4 & 5: Parser and StaticSemantics



```
4  int main(void){
5      int c;
6      int d;
7      SET d =c+1
8      if(d<2)[
9          SET c = modify(d);
10     ]
11     else [
12         delete(d);
13     ]
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Python

----- Initialization Done -----
Error: Unexpected lexeme if in line 8 After Lexeme 1
PS C:\Users\DELL\Desktop\Deliverable_4_5>

Test Suite 3: Didn't close the main curly braces

- Input: see Test Suite 3 program
- Output:

PROJECT – Parts 4 & 5: Parser and StaticSemantics

```
file.txt
11  else
12      | delete(d);
13      ]
14      SET c= 0;
15      WHILE(c<10)
16      BEGIN
17      | | SET d=c;
18      END;
19      back 0;
20
21  int modify(int r) {
22      | | SET r= b+2;
23      | | back r;
24      }
25  void delete(bool d){
26      | | int r;
27      }
28
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL**

```
----- While Loop -----
----- Return Statement -----
Error: Expected lexeme after modify in line 21
PS C:\Users\DELL\Desktop\Deliverable_4_5> |
```

⇒ The program considered int modify as an initialization and hence is looking for a semi colon

Test Suite 4: Missing SET keyword

- Input: see Test Suite 4 program
- Output:

PROJECT – Parts 4 & 5: Parser and StaticSemantics

```
1  int a;
2  int b;
3  bool f;
4  ✓ int main(void){
5      |   int c;
6      |   int d;
7      |   SET d =c+1;
8  ✓   |   if(d<2)[
9      |   |   c = modify(d);
10    |   ]

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL**

```
----- Global Variable Defined -----
----- Global Variable Defined -----
----- Global Variable Defined -----
----- Initialization Done -----
----- Initialization Done -----
----- Assignment Done -----
Error: Unexpected lexeme = in line 9 After Lexeme c
PS C:\Users\DELL\Desktop\Deliverable_4_5> 
```

Test Suite 5: Missing While statement

- Input: see Test Suite 5 program
- Output:

PROJECT – Parts 4 & 5: Parser and StaticSemantics

```
9      | SET c = modify(d);
10     |
11     else[
12     | delete(d);
13     ]
14     SET c= 0;
15
16     BEGIN
17     | SET d=c;
18     END;
19     back 0;
20 }
21 int modify(int r) {
22 | SET r= b+2;
23 | back r;
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL**

```
----- If Statement -----
----- Function Called -----
----- Else Statement -----
----- Assignment Done -----
Error: Unexpected lexeme BEGIN in line 16 After Lexeme ;
PS C:\Users\DELL\Desktop\Deliverable_4_5> 
```

Test Suite 6: Not passing an argument to the function

- Input: see Test Suite 6 program
- Output:

```
5 int c;
6 int d;
7 SET d =c+1;
8 if(d<2){
9 | SET c = modify();
10 |
11 | else[
12 | delete(d);
13 |
14 SET c= 0;
15 WHILE(c<10)
16 BEGIN
17 | SET d=c;
18 END;
19 back 0;
```

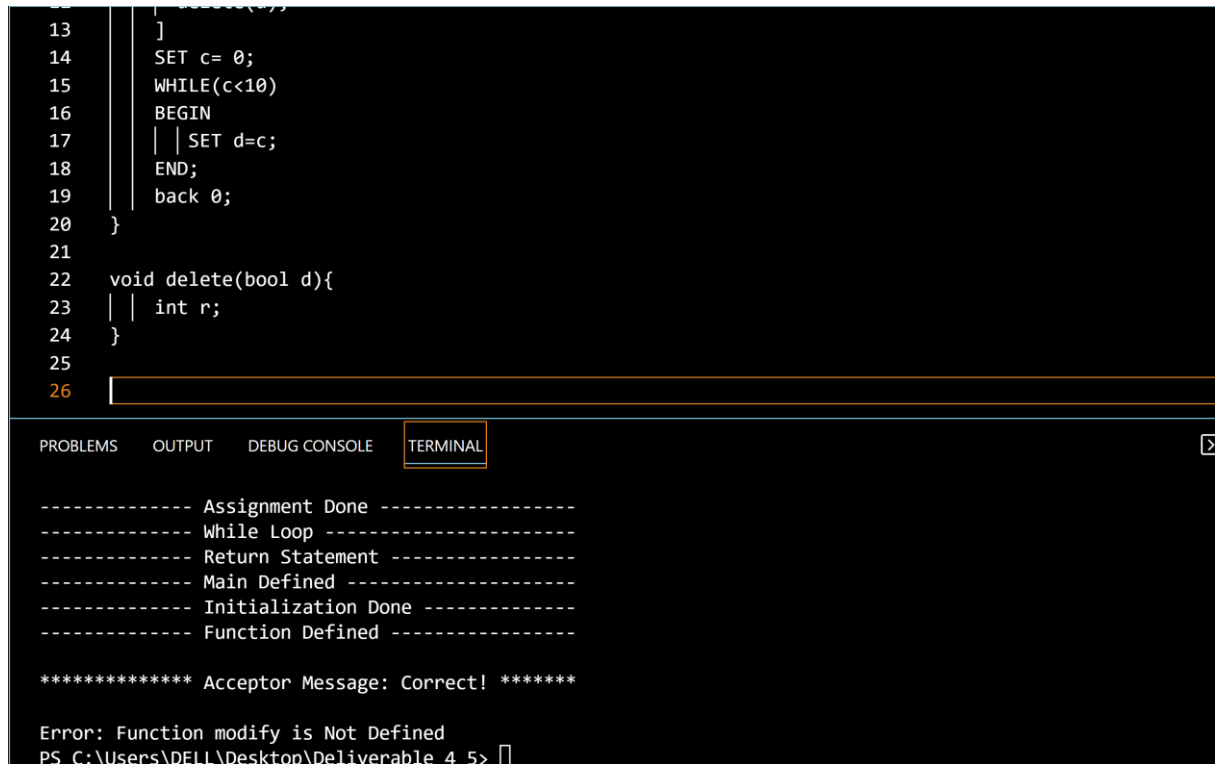
PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** Python + v [] [X]

```
----- Main Defined -----
----- Assignment Done -----
----- Return Statement -----
----- Function Defined -----
----- Initialization Done -----
----- Function Defined -----
***** Acceptor Message: Correct! *****
Error: Number of passed parameters of modify : 0 is different than the number of arguments defined 1
PS C:\Users\DELL\Desktop\Deliverable_4_5> 
```

PROJECT – Parts 4 & 5: Parser and StaticSemantics

Test Suite 7: Calling a non-defined function

- Input: see Test Suite 7 program
- Output:



```
13     ]
14     SET c= 0;
15     WHILE(c<10)
16     BEGIN
17     | | SET d=c;
18     END;
19     back 0;
20 }
21
22 void delete(bool d){
23 | | int r;
24 }
25
26
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL**

```
----- Assignment Done -----
----- While Loop -----
----- Return Statement -----
----- Main Defined -----
----- Initialization Done -----
----- Function Defined -----

***** Acceptor Message: Correct! *****

Error: Function modify is Not Defined
PS C:\Users\DELL\Desktop\Deliverable 4_5>
```

Test Suite 8: Using a non-defined variable

- Input: see Test Suite 8 program
- Output:

PROJECT – Parts 4 & 5: Parser and StaticSemantics

```
file.txt
2  int b;
3  bool f;
4  int main(void){
5      int c;
6      int d;
7      SET d =c+1;
8      if(d<2)[
9          | SET variable = modify(d);
10         ]
11     else[
12         | delete(d);
13     ]
14     SET c= 0;
15     WHILE(c<10)
16     BEGIN
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL**

```
PS C:\Users\DELL\Desktop\Deliverable_4_5> & "C:/Program Files/Python310/python.exe" "c:/Users/DELL/Des
arser (5).py"

----- Global Variable Defined -----
----- Global Variable Defined -----
----- Global Variable Defined -----
----- Initialization Done -----
----- Initialization Done -----
----- Assignment Done -----
variable not defined in this scope
PS C:\Users\DELL\Desktop\Deliverable_4_5> 
```

Test Suite 9: Wrong type assignment

- Input: see Test Suite 9 program
- Output:

PROJECT – Parts 4 & 5: Parser and StaticSemantics

```
file.txt
3 0001 r;
4 int main(void){
5     int c;
6     int d;
7     SET c = "Languages and Compilers";
8     if(d<2)[
9         SET c = modify(d);
10    ]
11    else[
12        delete(d);
13    ]
14    SET c= 0;
15    WHILE(c<10)
16    BEGIN
17        SET d=c;
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL**

```
PS C:\Users\DELL\Desktop\Deliverable_4_5> & "C:/Program Files/Python310/python.exe" "c:/User
arser (5).py"

----- Global Variable Defined -----
----- Global Variable Defined -----
----- Global Variable Defined -----
----- Initialization Done -----
----- Initialization Done -----
c is not of type const char
PS C:\Users\DELL\Desktop\Deliverable_4_5>
```

PROJECT – Parts 4 & 5: Parser and StaticSemantics

```
14  SET c= 0;
15  WHILE(c<10)
16  BEGIN
17  | SET d=c;
18  END;
19  back 0;
20  }
21  int modify(int r) {
22  | SET r= b+2;
23  | back r;|
24  }
25  int delete(bool d){
26  | SET f=d;
27  | back f;
28  }
29
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** Python +

```
----- Assignment Done -----
----- While Loop -----
----- Return Statement -----
----- Assignment Done -----
----- Return Statement -----
----- Function Defined -----
f is not of type int
```

Test Suite 10: Passing arguments more than needed

- Input: see Test Suite 10 program
- Output:

PROJECT – Parts 4 & 5: Parser and StaticSemantics

```
7      SET d = c+1;
8      if(d<2)[
9          | SET c = modify(d,f,e,g);
10     ]
11     else[
12         | delete(d);
13     ]
14     SET c= 0;
15     WHILE(c<10)
16     BEGIN
17         | SET d=c;
18     END;
19     back 0;
20 }
21 int modify(int r) {
22     ...
23 }
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** Python + v

```
----- Main Defined -----
----- Assignment Done -----
----- Return Statement -----
----- Function Defined -----
----- Initialization Done -----
----- Function Defined -----

***** Acceptor Message: Correct! *****

Error: Number of passed parameters of modify : 4 is different than the number of arguments defined 1
PS C:\Users\DELL\Desktop\Deliverable 4 5>
```

What have been changed from our previous submissions:

- Parts of our grammar have changed:
- Assignments:

// assign a value to a variable

```
<assign> ::= SET_RES ID <EQL_OP> ( ( ID || INT_LIT ) [<operator> ( ID || INT_LIT ) ] ) | (
Q_MARKS (<sentence> ) Q_MARKS | INT_LIT ) | ( <function_call> ) ) $ COLON

<operator> ::= ADD_OP | SUB_OP | DIV_OP | MULT_OP | MOD_OP
```

- Removed Logical Operators from our grammar
- Conditions:

//Put a condition; as an example $a > c$

```
<condition> ::= ( ID | INT_LIT ) <conditionaloperator> ( ID | INT_LIT )
```

PROJECT – Parts 4 & 5: Parser and StaticSemantics