Assignment – Stage 3.

The AST pattern for the for command was added to the FunEnconder file and highlighted with comments such as //EXTENSION FOR in order to identify where the changes were made.

Besides it, a code template was devised and put in the beginning of the FunEncoder file. The code template for the following for loop is as follows:

```
for n = 1 to 5:
        write(n)
.
```

```
4: LOADC   1          - [code to declare the control variable "n = 1"]
7: LOADC   5          - [code to evaluate "n to 5 (n < 5 + 1)"]
10: LOADL  2          - [code to evaluate "n to 5 (n < 5 + 1)"]
13: CMPLT             - [code to evaluate "n to 5 (n < 5 + 1)"]
14: JUMPT  33         - [code to jump out of the for loop]
17: LOADL  2          - [code to execute "write(n)"]
20: CALL   32767      - [code to execute "write(n)"]
23: LOADL  2          - [code to increment "n" by 1]
26: INC               - [code to increment "n" by 1]
27: STOREL 2          - [code to increment "n" by 1]
30: JUMP   7          - [code to jump to the for command's expressions evaluation]
33: RETURN 0
```

As in this phase only source codes that does not give errors are analysed the test was done only for the rightTypeAndScopeFor.fun attached file, and its result is described below:

rightTypeAndScopeFor.fun:
In this file a simple for was created, going from 1 to 5. No syntactic errors given. No contextual errors given. The code behaves as expected and gives the following object code and output respectively:

```
Code generation ...
Object code:
     0: CALL    4
     3: HALT
     4: LOADC   1
     7: LOADC   5
    10: LOADL   2
    13: CMPLT
    14: JUMPT   33
    17: LOADL   2
    20: CALL    32767
    23: LOADL   2
    26: INC
    27: STOREL  2
    30: JUMP    7
    33: RETURN  0

Interpretation ...
1
2
3
4
5
```