Assignment – Stage 3.

The AST pattern for the repeat until command was added to the FunEnconder file and highlighted with comments such as //EXTENSION REPEAT UNTIL 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 repeat until loop is as follows:

int n = 1

repeat:

write(n)

n = n + 1

until(n < 5).

4: LOADC 1 - [code to execute "int n = 1"]

7: LOADL 2 - [code to execute "write(n)"]

10: CALL 32767 - [code to execute "write(n)"]

13: LOADL 2 - [code to evaluate "n + 1"]

16: LOADC 1 - [code to evaluate "n + 1"]

19: ADD - [code to evaluate "n + 1"]

20: STOREL 2 - [code to assign "n = n + 1"]

23: LOADL 2 - [code to evaluate "n < 5"]

26: LOADC 5 - [code to evaluate "n < 5"]

29: CMPLT - [code to evaluate "n < 5"]

30: JUMPF 36 - [code to jump out of the repeat until loop]

33: JUMP 7 - [code to execute the repeat until command's body]

36: RETURN 0

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

rightTypeAndScopeUntil.fun:

In this file a simple repeat until loop was created, going from 1 to 4. 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: LOADL 2

10: CALL 32767

13: LOADL 2

16: LOADC 1

19: ADD

20: STOREL 2

23: LOADL 2

26: LOADC 5

29: CMPLT

30: JUMPF 36

33: JUMP 7

36: RETURN 0

Interpretation ...

1

2

3

4