Programming Assignment 4

Leo Kivikunnas 525925, Jaakko koskela 526050

March 2020

1 Augmented LL(1) Grammar

```
Program -> Declaration-list #END
Declaration-list -> Declaration Declaration-list  EPSILON
Declaration -> Declaration-initial Declaration-prime
Declaration-initial -> Type-specifier #PID ID
Declaration-prime -> #FUNCTION Fun-declaration-prime  #VARIABLE Var-declaration-prime
Var-declaration-prime -> ;  #ARRAY [ NUM ] ;
Fun-declaration-prime -> #BEGINSCOPE  #START_PARAM_COUNTER Params #STOP_PARAM_COUNTER  Compound-stmt #ENDSCOPE
Type-specifier -> int  void
Params -> int #PID ID Param-prime Param-list  void Param-list-void-abtar
Param-list-void-abtar -> #PID ID Param-prime Param-list  EPSILON
Param-list -> , Param Param-list  EPSILON
Param -> Declaration-initial Param-prime
Param-prime -> #ARRAY #ARRAY_PARAM [ ]  #PARAM EPSILON
Compound-stmt -> { Declaration-list Statement-list }
Statement-list -> Statement Statement-list  EPSILON
Statement -> Expression-stmt  Compound-stmt  Selection-stmt  Iteration-stmt  Return-stmt  Switch-stmt
Expression-stmt -> #START_TYPE_CHECK Expression #TYPE_CHECK ;  #CONTINUE continue ;  #BREAK break ;  #TYPE_CHECK
Selection-stmt -> if  #START_TYPE_CHECK Expression #TYPE_CHECK_IN_BRACKETS  Statement else Statement
Iteration-stmt -> while #ENTER_WHILE  #START_TYPE_CHECK Expression #TYPE_CHECK_IN_BRACKETS  Statement #EXIT_WHILE
Return-stmt -> return Return-stmt-prime
Return-stmt-prime -> ;  #START_TYPE_CHECK Expression #TYPE_CHECK ;
Switch-stmt -> switch #ENTER_SWITCH_CASE  #START_TYPE_CHECK Expression #TYPE_CHECK_IN_BRACKETS  { Case-stmts Defa
Case-stmts -> Case-stmt Case-stmts  EPSILON
Case-stmt -> case NUM : Statement-list
Default-stmt -> default : Statement-list  EPSILON
Expression -> Simple-expression-zegond  #USE_PID #ADD_TO_TYPE_CHECK ID B
B -> #NOT_FUNCTION_CALL = Expression  #NOT_FUNCTION_CALL [ #INDEXING #START_TYPE_CHECK Expression #TYPE_CHECK ] H
H -> = Expression  G D C
Simple-expression-zegond -> Additive-expression-zegond C
Simple-expression-prime -> Additive-expression-prime C
C -> Relop Additive-expression  EPSILON
Relop -> <  ==
Additive-expression -> Term D
Additive-expression-prime -> Term-prime D
Additive-expression-zegond -> Term-zegond D
D -> Addop Term D  EPSILON
Addop -> +  -
Term -> Factor G
Term-prime -> Factor-prime G
```

```
Term-zegond -> Factor-zegond G
G -> Factor ->  #START_TYPE_CHECK Expression #TYPE_CHECK_IN_BRACKETS   #USE_PID #ADD_TO_TYPE_CHECK ID Var-call-pr
Var-call-prime -> #FUNCTION_CALL  #START_ARGUMENT_COUNTER Args #STOP_ARGUMENT_COUNTER   #NOT_FUNCTION_CALL Var-pr
Var-prime -> [ #INDEXING #START_TYPE_CHECK Expression #TYPE_CHECK ]  EPSILON
Factor-prime -> #FUNCTION_CALL  #START_ARGUMENT_COUNTER Args #STOP_ARGUMENT_COUNTER   #NOT_FUNCTION_CALL EPSILON
Factor-zegond ->  #START_TYPE_CHECK Expression #TYPE_CHECK_IN_BRACKETS   #ADD_TO_TYPE_CHECK NUM
Args -> Arg-list  EPSILON
Arg-list -> #ARGUMENT #START_TYPE_CHECK Expression #TYPE_CHECK Arg-list-prime
Arg-list-prime -> , #ARGUMENT #START_TYPE_CHECK Expression #TYPE_CHECK Arg-list-prime  EPSILON
```

---

2 Error Detection

Our program can detect all 8 error types listed in the assignment.