**Smart compiler**

**Compiler Construction**

Course Supervisor: Ms. Maryam Feroze

**B16101116-Muhammad Talib Waseem**

**B16101043-Hafiz Alqama Bin Shuja**

**Introduction:**

Smart Compiler is a simple compiler that will be made on java script as backend. The compiler Language contain all the main concept of today’s famous programming languages.

**Feature:**

1. Int, float, char, string, bool, const (immutable), let, var
2. Array List (LinkedList), Stack, Queue, 1D-Array
3. Function, Higher Order Function (JS feature)
4. Logical Operation (if-else)
5. No terminator (optional)
6. No garbage, unassigned variable can’t be used.
7. No public, private all attribute will be private and function are public.
8. Classes, Object Oriented Programming
9. Operator Overloading, function overloading
10. Inheritance (single inheritance)
11. Only Public Inheritance
12. Protected can be used to access base class property to child
13. Spread Operator ( … )
14. Switch case
15. All 3 loop with range loop
16. Ternary Operator ( ? : )
17. Static variable, function

**Word Table:**

|  |  |
| --- | --- |
| **Words** | **Class** |
| int | Datatype |
| float | Datatype |
| string | Datatype |
| short | Datatype |
| long | Datatype |
| double | Datatype |
| const | Datatype |
| var | Datatype |
| while | while |
| do | Do |
| for | for |
| if | If |
| else | else |
| switch | switch |
| case | case |
| break | break |
| continue | continue |
| true | true |
| false | false |
| class | Class |
| protected | protected |
| static | static |
| arrayList | arrayList |
| Stack | Stack |
| Queue | Queue |
| function | function |
| return | return |

**Punctuators:**

1. , use to separate parameter, object attribute
2. ; statement terminator
3. : for inheritance
4. { } for object structure
5. ( ) for function
6. [ ] for 1D,2D array
7. . to access attribute of object

**Operators:**

AOP 🡺 = += -= \*= /= %=

ADDOP 🡺 + -

MOP 🡺 \*

DOP 🡺 / %

ROP 🡺 < > <= >= != ==

LAND 🡺 &&

LOR 🡺 ||

BAND 🡺 &

BOR 🡺 |

**Identifiers:**

∑ = { alphabet, digit, \_ }

alphabet = [ A - Z ], [ a – z ]

digit = [ 0 – 9 ]

1. ID start with either \_ or alphabet
2. First char can be followed by any char.

RE = (\_+ alphabet (^ + \_ + alphabet + digit

**Constants:**

1. Integer Constant:

∑ = { digit, +, - }

RE = ( + | - | ^)(digit

1. Float Constants**:**

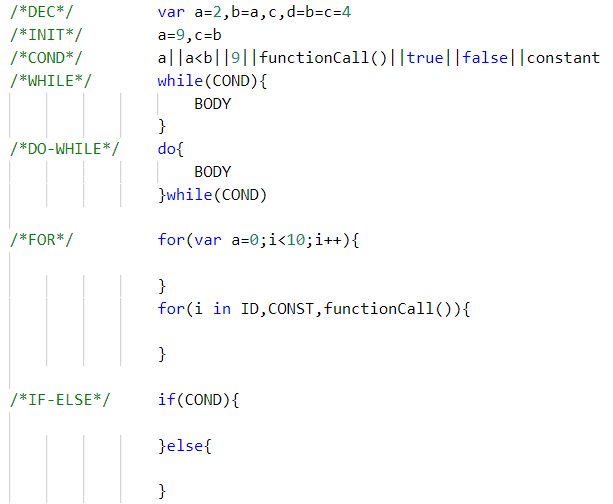
∑ = { +, -, . , digit }

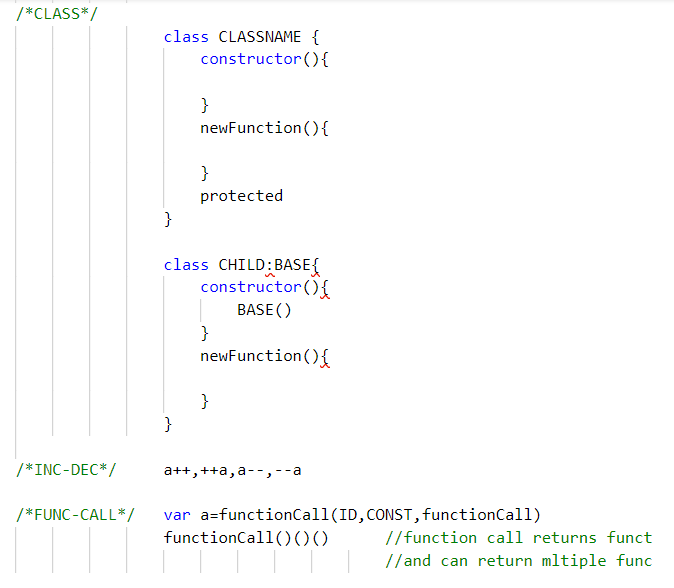
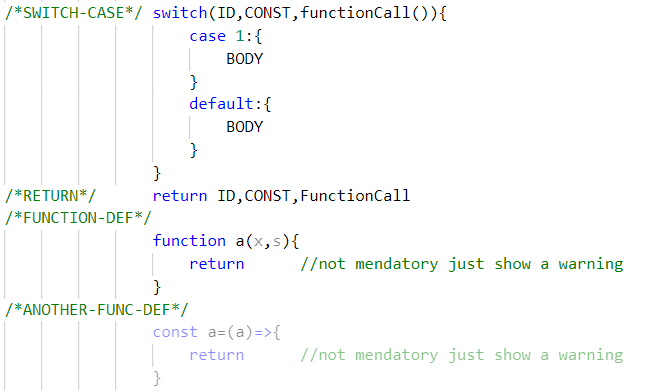
RE = ( + | - | +| ^ )(digit)\* . (digit

1. String Constant:

RE = “ [ (C+A) + (\(A+B)) ] ”

**Basic Syntax:**

****

****

**CFG’s**

1. <START> 🡪<DEFS>
2. <DEFS> 🡪 <MST> <DEFS> | <CLASS> <DEFS> | <FUNCTION-DEC><DEFS>
3. <MST> 🡪 € |<SST> <MST>
4. <SST ***>*** 🡪 <SST1>
   1. <SST1 >🡪 <DEC> | <WHILE> | <FOR > | <DO-WHILE> | <CONST-DT> | <INC-DEC-PRE> | <IF-ELSE> |<SWITCH-CASE> | <RETURN> |<GTSWID>
5. <WHILE> 🡪 while (<EXP>) <BODY>
6. <FOR> 🡪 for (<FOR-PARAM>) <BODY>
   1. <FOR-PARAM> 🡪 ID <FOR-PARAM-2> | <DEC>;<C2>;<C3> |; <C2>;<C3>
   2. <FOR-PARAM-2> 🡪 ; <C2>;<C3> | <INIT-VALUE>;<C2>;<C3> | <RANGE-FOR>
   3. <RANGE-FOR> 🡪 in <FOR\_R>
   4. <FOR\_R> 🡪 ID <N\_INIT\_VALUE>
   5. <C2> 🡪 <EXP> | €
   6. <C3> 🡪 ID | <GTSWID>
7. <DO-WHILE> 🡪 do <BODY> while (<EXP>)
8. <BODY> 🡪 ; | <SST> | {<MST>}
9. <IF-ELSE> 🡪 if (<COND>) <BODY> <ELSE>
   1. <ELSE>🡪 € | else <BODY>
10. <CONST-DT> 🡪 const ID <CONST-DT1> <NEXT-CONST-DT>
    1. <CONST-DT1> 🡪 AOR <CONST-DT2>
    2. <CONST-DT2> 🡪 new ID ( <CALLING-PARAM> ) | <INIT-VALUE-2> | ID <INIT-VALUE>
    3. <NEXT-CONST-DT> 🡪 <DEC\_PARAMS> | €
11. <GTSWID> 🡪 ID <INIT\_VALUE> <NEXT\_GTSWID>
    1. <NEXT\_GTSWID> 🡪 inc\_dec | ( <CALLING\_PARAM> ) | AOR <GTSDEC>
    2. <GTSDEC> 🡪 <DEC2> | <OBJECT> | <ARRAY>
12. <FUNCTION-DEC>🡪 function <FUNC-DEF-1>
    1. <FUNC-DEF-1>🡪ID (<DEC-PARAMS>) {<MST>}
13. <CLASS> 🡪 class ID <CLASS-STRUCT>
    1. <CLASS-STRUCT> 🡪 <CLASS-BODY> | : ID <CLASSBODY>
    2. <CLASSBODY>🡪 { <CLASS-MST> }
    3. <CLASS-MST> 🡪 € |

<CLASS\_CONSTRUCOR> <CLASS-ST> <CLASS-MST>

* + 1. <CLASS\_CONSTRUCOR> 🡪 € | constructor (<DEC\_PARAMS>) {<CONSTRUCTOR\_BODY>}
    2. <CONSTRUCTOR\_BODY> 🡪 € | <C\_B\_I> <CONSTRUCTOR\_BODY>
    3. <C\_B\_I> 🡪 <THIS\_VAR>|<SST1>
    4. <THIS\_VAR > 🡪 this. ID AOR <DEC2>
  1. <CLASS-ST> 🡪 <CLASS-FUNC> | <DEC> | <PROTECTED>
  2. <CLASS-FUNC> 🡪 ID <FUNC-DEF>
  3. <FUNC-DEF> 🡪 AOR ( <DEC-PARAMS> ) => { <MST> } |

(<DEC-PARAMS>){<MST>}

* 1. <PROTECTED>🡪 protected: <PRO-NEXT>
  2. <PRO-NEXT> 🡪 <CLASS-ST> | { <PRO-BODY>}
  3. <PRO-BODY> 🡪 <CLASS-MST>

1. <DEC\_PARAMS > 🡪 DT ID <DEC1><NEXT\_PARAM> |

ID <CONST\_DEC\_PARAM> <NEXT\_PARAM> |

€

* 1. <CONST\_DEC\_PARAM> 🡪 € | AOR <CONST\_DT1>
  2. <NEXT\_PARAM> 🡪 , <NEXT\_DEC\_PARAM> | €
  3. <NEXT\_DEC\_PARAM> 🡪 DT ID <DEC1><NEXT\_PARAM> |

ID <CONST\_DEC\_PARAM> <NEXT\_PARAM>

1. <Dec> 🡪 DT ID <DEC1> <NEXTDEC>
   1. <DEC1> 🡪 AOR <DEC2> | €
   2. <DEC2> 🡪 ID <DEC3> | <E><EXP1>
   3. <DEC3> 🡪 <MERGED\_INIT> <EXP1> | <EXP1> | €
   4. <MERGED\_INIT> 🡪 <MOV> | [<EXP>] <M\_N\_ARR>
   5. <M\_N\_ARR> 🡪 <MOV> | [<EXP>] <M\_N\_ARR2> | €
   6. <M\_N\_ARR2> 🡪 <MOV> | €
   7. <MOV> 🡪 .ID<MERGED1> | (<CALLING\_PARAMS>) <MERGED>
   8. <MERGED1> 🡪 <EXP1> | <MERGED\_INIT> <DEC1>
   9. <MERGED> 🡪 <EXP1> | <MERGED\_INIT>
2. <EXP> 🡪 <VAL> <EXP1>
   1. <EXP1> 🡪 <Q\_DASH> <R\_DASH> <S\_DASH><T\_DASH> <E\_DASH>
   2. <E\_DASH> 🡪 ||<T><E\_DASH> | €
   3. <T> 🡪 <S> <T\_DASH>
   4. <T\_DASH> 🡪 && <S><T\_DASH> | €
   5. <S> 🡪 <R> <S\_DASH>
   6. <S\_DASH> 🡪 ROP <R> <S\_DASH> | €
   7. <R> 🡪 <Q> <R\_DASH>
   8. <R\_DASH> 🡪 PM <Q> <R\_DASH> | €
   9. <Q> 🡪 <VAL> <Q\_DASH>
   10. <Q\_DASH>🡪 MDM <VAL> <Q\_DASH> | €
   11. <VAL> 🡪 <F> | <E>
   12. <F> 🡪 ID <NEW\_ASGN>
   13. <E> 🡪 (<EXP >) |! <VAL > | -- <F > | ++ <F > | <CONST > |this . ID <NEW\_ASGN>
   14. <NEW\_ASGN > 🡪 ++ | -- | <N\_INIT\_VALUE>
3. <N\_INIT\_VALUE>🡪€ | <OTHER\_N\_VALUE> | [<EXP>] <N\_ARR\_N>
   1. <N\_ARR\_N>🡪 € | <OTHER\_N\_VALUE> | [<EXP>] <N\_ARR\_2N>
   2. <N\_ARR\_2N> 🡪 € | <OTHER\_N\_VALUE>
   3. <OTHER\_N\_VALUE> 🡪 .ID <N\_INIT\_VALUE> | (<CALLING\_PARAM>) <N\_INIT\_VALUE>
4. <INIT\_VALUE> 🡪 € | <OTHER\_VALUE> | [ <EXP> ] <N\_ARR>
   1. <OTHER\_VALUE> 🡪 . ID <INIT\_VALUE> | ( <CALLING\_PARAM>) <AFT\_VALUE>
   2. <AFT\_VALUE> 🡪 <OTHER\_VALUE> | [ <EXP> ] <N\_ARR>
   3. <N\_ARR> 🡪 € | <OTHER\_VALUE> | [ <EXP> ] <N\_ARR2>
   4. <N\_ARR2> 🡪 € | <OTHER\_VALUE>
5. <ARRAY> 🡪 [ <ARRAY-INNER>]
   1. <ARRAY-INNER> 🡪 € | <ARRAY-VALUES>
   2. <ARRAY-VALUES> 🡪<ARRAY-VAL><NEXT-VAL>
   3. <ARRAY-VAL> 🡪 <INIT-VALUE-2> | … ID <N\_INIT-VALUE>
   4. <NEXT-VAL>🡪 , <ARRAY-VALUES> | €
6. <SPREAD> 🡪 … ID <N-INIT-VALUE>
7. <OBJECT> 🡪 {<PROP>}
   1. <PROP>🡪 <OBJECT1> <NEXT-PROP>
   2. <OBJECT1> 🡪 ID : <EXP> | <SPREAD>
   3. <NEXT-PROP>🡪, <PROP>| €

1. <INIT-VALUE-2> 🡪 <EXP> | <ARRAY> | <OBJECT>
2. <CALLING-PARAMS> 🡪 <CP-VALUE>
   1. <CP-VALUE> 🡪 € | <CP-VALUE2>
   2. <CP-VALUE2> 🡪 <INIT-VALUE-2> <NEXT-CPVALUE>
   3. <NEXT-CPVALUE> 🡪 € |, <CP-VALUE2>
3. <SWITCH-CASE> 🡪 switch (<N-INIT-VALUE>) <SW-BODY>
   1. <SW-BODY> 🡪 {<CASES> <DEFAULT>}
   2. <CASES>🡪 case <CASE-VALUE>: <BODY>
   3. <CASE-VALUE> 🡪 <INNER-CASE-VAL>
   4. <INNER-CASE-VAL> 🡪 <EXP>
   5. <DEFAULT> 🡪 default : <BODY>
4. <INC-DEC-PRE> 🡪 <inc-dec-op> ID <N-INIT-VALUE>

**Attributed Grammar:**

1. <START> 🡪 ***{ S = CreateScope() }*** <DEFS***(S)***>***{ DeleteScope() }***
2. <DEFS***(S)***> 🡪 <MST***(S)***> <DEFS***(S)***> |

<CLASS> <DEFS***(S)***> |

<FUNCTION-DEC><DEFS***(S)***> | $

1. <MST ***(S, &RT)***> 🡪 € |

<SST ***(S, &RT)***><MST ***(S, &RT)***>

1. <SST ***(S, &RT)>*** 🡪 <SST1 ***(S, &RT)*** >

* 1. <SST1 ***(S, &RT)*** >🡪 <DEC***(S)***> | <WHILE ***(S)*** > | <FOR ***(S)*** > | <DO-WHILE ***(S)*** > | <CONST-DT ***(S)*** > | <INC-DEC-PRE ***(S)*** > | <IF-ELSE ***(S)*** > |<SWITCH-CASE***(S)*** > | <RETURN***(S, &RT)***>

|<GTSWID ***(S)*** >

1. <WHILE***(S)***> 🡪 while (<EXP***(&T,S)***> ***{if(!Compatibility(&T,bool,”cond”)) {Type Mismatch}}***) ***{S=CreateScope()}***

<BODY***(S)***>

1. <FOR ***(S)***> 🡪 for ( ***{S=CreateScope()}*** <FOR-PARAM ***(S)***>) <BODY>
   1. <FOR-PARAM ***(S)***> 🡪 ID ***{N = ID.VP}*** <FOR-PARAM-2 ***(N, S)***> | <DEC ***(S)***>;<C2 ***(S)*** >;<C3 ***(S)*** > |; <C2 ***(S)*** >;<C3 ***(S)*** >
   2. <FOR-PARAM-2 ***(N, S)***> 🡪 ***{if(!LookupST(N, S)){ Undeclared Variable }}*** ; <C2 ***(S)*** >;<C3 ***(S)*** > | <INIT-VALUE ***(N, S ,T)*** >;<C2 ***(S)*** >;<C3 ***(S)*** > | <RANGE-FOR ***(N, S)***>
   3. <RANGE-FOR ***(N, S)***> 🡪 in <FOR\_R ***(S, &T)***> ***{!insertST(N,T,S){Redeclaration }}***
   4. <FOR\_R ***(S, T)***> 🡪 ID ***{N = ID.VP}*** <N\_INIT\_VALUE ***(N, T, S, ref, CN)***>

* 1. <C2 ***(S)***> 🡪 <EXP ***(&T, S)***> ***{if(!Compatibility(T, bool, =){ Mismatch Type}}***| €
  2. <C3 ***(S)***> 🡪 <GTSWID ***(S)***> | €

1. <BODY ***(S, &RT)***> 🡪 ; |

<SST***(S,RT)***> ***{DeleteScope()}*** |

{<MST***(S,RT)***> } ***{DeleteScope()}***

1. <DO-WHILE> 🡪 do ***{S = CreateScope() }***

<BODY***(S)***> while ( <EXP***(&T)***> ***{if(!Compatibility(&T,bool,”cond”)) {Type Mismatch}}*** ) ***{ DeleteScope() }***

1. <IF-ELSE ***(S)***> 🡪 if ( <EXP***(&T,S)***> ***{if(!Compatibility(&T,bool,”cond”)) {Type Mismatch}}***) ***{S=CreateScope()}*** <BODY***(S, null)***> <ELSE***(S)***>
   1. <ELSE ***(S)***>🡪 € | else ***{S=CreateScope()}*** <BODY***(S, null)***>
2. <GTSWID ***(S)***> 🡪 ID ***{N = ID.VP}*** <INIT\_VALUE ***(N, S)***> <NEXT\_GTSWID>
   1. <NEXT\_GTSWID> 🡪 inc\_dec | ( <CALLING\_PARAM> ) | AOR <GTSDEC>
   2. <GTSDEC> 🡪 <DEC2> | <OBJECT> | <ARRAY>
3. <INIT\_VALUE> 🡪 € | <OTHER\_VALUE> | [ <EXP> ] <N\_ARR>
   1. <OTHER\_VALUE> 🡪 . ID <INIT\_VALUE> | ( <CALLING\_PARAM>) <AFT\_VALUE>
   2. <AFT\_VALUE> 🡪 <OTHER\_VALUE> | [ <EXP> ] <N\_ARR>
   3. <N\_ARR> 🡪 € | <OTHER\_VALUE> | [ <EXP> ] <N\_ARR2>
   4. <N\_ARR2> 🡪 € | <OTHER\_VALUE>
4. <FUNCTION-DEC>🡪 function <FUNC-DEF-1>
   1. <FUNC-DEF-1>🡪ID ***{N = ID. VP}***

(***{S = CreateScope ()}***

<DEC-PARAMS ***(&PL,S)***>)

{<MST ***(S, &RT)***>} ***{if (! insertCT (N, PL->RT, null, null) {Redeclaration Error} DeleteScope ()}***

1. <CLASS> 🡪 class ID ***{N = ID. VP}*** <CLASS-STRUCT***(N)***>
   1. <CLASS-STRUCT ***(N)***> 🡪 <CLASS-BODY ***(N, null)***> |

: ID ***{PN = ID.VP if(LookupCT(PN)){ PL = ID.VP }}*** <CLASSBODY ***(N, PL)***>

* 1. <CLASSBODY ***(N, PL)***>🡪***{ref = createRef() if(!insertCT(N, ”class”, PL , & ref) Redeclaration Error }***

{<CLASS-MST ***(&ref)***>}

* 1. <CLASS-MST ***(&ref)***> 🡪 € |

<CLASS-CONSTRUCOR ***(&ref)***> <CLASS-ST***(&ref)***> <CLASS-MST ***(&ref)***>

* + 1. <CLASS-CONSTRUCOR ***(&ref)***> 🡪 € | constructor ***{S = CreateScope()}***  (<DEC-PARAMS **(&PL,S)**>) ***{if(!insertCDT(N,PL->void,*** ***public, null, ref)) {Redeclaring Constructor}*** {<CONSTRUCTOR-BODY ***(S, ref)***>} ***{DeleteScope()}***
    2. <CONSTRUCTOR-BODY ***(S, ref)***> 🡪 € |

<C-B-I ***(S, ref)***>

<CONSTRUCTOR-BODY ***(S, ref)***>

* + 1. <C-B-I ***(S, ref)***> 🡪 <THIS-VAR ***(S, ref)*** >|<SST1 ***(S, &RT, ref)***>
    2. <THIS\_VAR ***(S, ref)***> 🡪 this. ID ***{N = ID. VP T=lookupCDT(N, ref)}***

AOR <DEC2 ***(TR, S, ref)***>

***{TA­ =Compatibility (T, TR***, ***AOR)***

***if(!TA ){Type Mismatch }***

* 1. <CLASS-ST***(&ref)***> 🡪 <CLASS-FUNC ***(&ref)*** > |

<DEC ***(null, &ref)*** > |

<CONST\_DT***(null, &ref)***> |

<PROTECTED ***(&ref)*** >

* 1. <CLASS-FUNC ***(&ref)***> 🡪 ID ***{N = ID.VP }*** <FUNC-DEF***(&ref, N)***>

* 1. <FUNC-DEF ***(&ref, N)***> 🡪 AOR (***{ S = CreateScope() }***

<DEC-PARAMS***(&PL,S)***> ) => { <MST ***(S, &RT)*** > } ***{if(!insertCDT(N, PL->RT, public, null, ref ) { Redeclaration Error }}*** |

( ***{ S = CreateScope() }*** <DEC-PARAMS ***(&PL,S)*** >)

{<MST ***(S, &RT)*** >} ***{if(!insertCDT(N, PL->RT, public, null, ref ) { Redeclaration Error }}***

* 1. <PROTECTED ***(&ref)*** >🡪 protected: <PRO-NEXT ***(&ref)***>
  2. <PRO-NEXT ***(&ref)***> 🡪 <CLASS-ST ***(&ref)***> | {<PRO-BODY ***(&ref)***>}
  3. <PRO-BODY ***(&ref)***> 🡪 <CLASS-MST ***(&ref)***>

1. <CONST-DT ***(S, ref)***> 🡪 const ID ***{N = ID.VP***  <CONST-DT1***(&TL,S)***>***if(! ref !insertST(N,S){Redeclaration Error} else if(!insertCDT(N,T,public,null,ref))}*** <NEXT-CONST-DT>
   1. <CONST-DT1 ***(&TL, S)***> 🡪 AOR <CONST-DT2***(&TL, S)***>
   2. <CONST-DT2 ***(&TL, S)***> 🡪 new ID***{ N = ID.VP ref=LookupCT(N) if(!ref){Class Not defined }}*** ( <CALLING-PARAM***(PL)***> ) ***{if(LookupCDT(constructor,PL)) Constructor Not Defined}***|

<INIT-VALUE-2> |

ID <INIT-VALUE>

1. <DEC\_PARAMS ***(&PL, S)***> 🡪 DT ***{T=DT.VP}***

ID ***{N = ID. VP}***

<DEC1 ***(T, S)***> ***{ if (!insertST(N ,T ,S){Redeclaration***

***Error} else {PL = T}}*** <NEXT\_PARAM ***(&PL, S)*** > |

ID ***{N = ID. VP, T = ’const’}***

<CONST\_DEC\_PARAM ***(T, S)*** > ***{if (!insertST (N ,T ,S){ Redeclaration Error} else {PL = T}}***

<NEXT\_PARAM ***(&PL, S)***> |

€ ***{PL=null}***

* 1. <CONST\_DEC\_PARAM ***(T, S)***> 🡪 € | AOR <CONST\_DT1>
  2. <NEXT\_PARAM ***(&PL, S)***> 🡪 , <NEXT\_DEC\_PARAM ***(&PL, S)***> | €
  3. <NEXT\_DEC\_PARAM ***(&PL, S)***> 🡪 DT ***{T=DT.VP}***

ID ***{N = ID.VP}***

<DEC1 ***(T, S)***> ***{ if (!insertST(N ,T ,S){Redeclaration***

***Error} else {PL += T}}***<NEXT\_PARAM ***(&PL, S)*** > |

ID ***{N = ID. VP, T = ’const’}***

<CONST\_DEC\_PARAM ***(T, S)***> ***{if (!insertST(N ,T, S){ Redeclaration Error} else {PL += T}}***

<NEXT\_PARAM ***(&PL, S)***>

1. <DEC ***(S, ref)***> 🡪 *DT* ***{T = DT. VP}***

ID ***{N = ID. VP}***

***<***DEC1***(T, S, ref)***> ***{if (! insertST (N, T, S)) {Redeclaration}}***

<NEXTDEC ***(T, S)***>

* 1. <DEC1***(T, S, ref)***> 🡪 AOR

<DEC2***(&T****R****, S, ref)****>* ***{TA­ =Compatibility (T, TR***, ***AOR) if(!TA ){Type Mismatch }***| €

* 1. <DEC2***(&TR, S, ref)***> 🡪 ID ***{N = ID. VP, CN = null}***

<DEC3***(N, T, S, CN,ref)***> |

<EXP***(&T, S , ref)***>

* 1. <DEC3***(N, T, S, CN)***> 🡪 <MERGED\_INIT ***(N, &TL, S, CN)***>

<EXP1***(TL, &T, S)***> |

***{if(CN!==null) { TL = ST.lookupCDT(N,CN)}***

***Else T= LookupST (N, S) if (!TL){ Undeclared Variable}}*** <EXP1***(TL, &T, S, null)***> |

€ ***{if(CN!==null) { T=ST.lookupCDT(N,CN)}***

***Else T= LookupST (N, S) if (!TL){ Undeclared Variable } }***

* 1. <MERGED\_INIT ***(&T, N, CN, S)*** > 🡪 <MOV ***(&T, N, CN, S)*** > |

[<EXP***(T1)***> ***{IF (! Compatibility (T1, INT) {Type Mismatch}***]

<M\_N\_ARR ***(&T, N, S)***>

* 1. <M\_N\_ARR ***(&T, N, S)***> 🡪 <MOV ***(&T, N, S)***> |

[<EXP>***(T1)***> ***{IF (! Compatibility (T1, INT) {Type Mismatch}***]

<M\_N\_ARR2***(&T, N, S)*** > |

€ ***{T=LookupST (N, S) if (!T){ Undeclared***

***Variable }***

* 1. <M\_N\_ARR2 ***(&T, N, S)***> 🡪 <MOV ***(&T, N, S)***> |

€ ***{T=LookupST (N, S) if (! T) {Undeclared Variable}***

* 1. <MOV ***(&T, N, CN, S)***> 🡪 . ***{if(CN==null){CN = LookupST (N, S)}***

***else{ A = LookupCDT(N, CN) if(A==DT){T=A} else {CN=A} }***

***//check if for Undeclared***

ID ***{N= ID. VP}***

<MERGED1 ***(&T, N, CN, S)***> |

(<CALLING\_PARAMS***(&PL,scope)***>)***{CN=LookupFT(N,PL,CN)}***

<MERGED ***(&T, null, CN, S)***>

* 1. <MERGED1 ***(&T, N, CN, S)***> 🡪 ***{if ( CN != null ) T1 = LookupCDT (N, CN)}***

<EXP1***(&T, T1)***> |

<MERGED\_INIT ***(&T, N, CN, S)***>

<DEC1>

* 1. <MERGED ***(&T, N, CN, S)***> 🡪 ***{if ( CN != null ) T1 = LookupCDT (N, CN)}***

<EXP1***(&T, T1)***> |

<MERGED\_INIT ***(&T, N, CN, S)***>

1. <EXP ***(&T, S, ref)***> 🡪 <VAL ***(&TL, S, ref)***> <EXP1 ***(&TL, &T, S, ref)***>
   1. <EXP1 ***(&TL, &T, S, ref)***> 🡪 € ***{T = TL}*** |

<Q\_DASH ***(T­L ,&T ,S, ref)***>

<R\_DASH ***(T­L ,&T ,S, ref)***>

<S\_DASH ***(T­L ,&T ,S, ref)***>

<T\_DASH ***(T­L ,&T ,S, ref)***>

<E\_DASH ***(T­L ,&T ,S, ref)***>

* 1. <E\_DASH ***(T­L ,&T, S, ref)***> 🡪 || ***{O = ||.VP}***

<T ***(&TR , S, ref)***> ***{TA = Compatibility(TL, TR , O)}***

<E\_DASH ***(T­A ,&T ,S, ref)***> |

€ ***{T = TL}***

* 1. <T ***(&T, S, ref)***> 🡪 <S ***(&TL , S, ref)***> <T\_DASH ***(T­L ,&T ,S, ref)***>
  2. <T\_DASH ***(T­L ,&T ,S, ref)***> 🡪 && ***{O = &&.VP}***

<S ***(&TR, S, ref)***> ***{TA = Compatibility(TL, TR , O)}***

<T\_DASH ***(T­A ,&T ,S, ref)***> | € ***{T = TL}***

* 1. <S ***(&T, S, ref)***> 🡪 <R ***(&TL,S, ref)***> <S\_DASH ***(T­L ,&T ,S, ref)***>
  2. <S\_DASH ***(T­L ,&T ,S, ref)***> 🡪 ROP ***{O = ROP.VP}***

<R ***(&TR, S, ref)***> ***{TA = Compatibility(TL, TR , O)}*** <S\_DASH ***(T­A ,&T ,S, ref)***> | € ***{T = TL}***

* 1. <R ***(&T, S, ref)***> 🡪 <Q ***(&TL , S, ref)***> <R\_DASH ***(T­L ,&T ,S, ref)***>
  2. <R\_DASH ***(T­L ,&T ,S, ref)***> 🡪 PM ***{O = PM.VP}***

<Q ***(&TR , S, ref)***> ***{TA = Compatibility(TL, TR , O)}***   
<R\_DASH ***(T­A ,&T ,S, ref)***> | € ***{T = TL}***

* 1. <Q ***(&T, S, ref)***> 🡪 <VAL ***(&TL , S, ref)***> <Q\_DASH ***(T­L ,&T ,S, ref)***>
  2. <Q\_DASH ***(T­L ,&T ,S, ref)***>🡪 MDM ***{O = MDM.VP}***

<VAL ***(&TR , S, ref)***> ***{TA = Compatibility(TL, TR , O)}***

<Q\_DASH ***(T­A ,&T ,S, ref)***> | € ***{T = TL}***

* 1. <VAL ***(&T, S, ref)***> 🡪 <F ***(&T, S, ref)***> | <E ***(&T, S, ref)***>
  2. <F ***(&T, S, ref)***> 🡪 ID ***{N = ID.VP CN=null}***

<NEW\_ASGN ***(N, &T, S, ref, CN)***>

* 1. <E ***(&T, S, ref)***> 🡪 (<EXP ***(&T, S, ref)***>) |

!<VAL ***(&T, S, ref)***> |

-- <F ***(&T, S, ref)***> |

++ <F ***(&T, S, ref)***> |

<CONST ***(&T)***> |

this . ID ***{N = ID.VP CN=null}***

<NEW\_ASGN ***(N, &T, S, ref, CN)***>

* 1. <NEW\_ASGN ***(N, &T, S, ref)***> 🡪 ***{ if(ref) {TL = LookupCDT(N,ref) }***

***else {TL =­ LookupST(N)}*** inc\_dec ***T=Compatibility( T, int, ’+’ )}***|

<N\_INIT\_VALUE ***(N, &T, S, ref)*** >

1. <N\_INIT\_VALUE ***(N, &T, S, ref, CN)*** >🡪 € **(if(ref.current.used!=null){ T=lookupCDT(N, ref)**

**ref.current.used = null }** | <OTHER\_N\_VALUE ***(N, &T, S, ref, CN)*** > | [ ***if(CN) TL=LookupCDT(N,CN) else TL=LookupST(N,S)*** <EXP***(TEXP, S, null)***> ***(if(!TEXP){Type Mismatch}*** ] <N\_ARR\_N ***(Tl, T, S, ref, CN)***>

* 1. <N\_ARR\_N ***(N, &T, S, ref, CN)*** >🡪 € |

<OTHER\_N\_VALUE***(N, &T, S, ref, CN)*** > | [<EXP>] <N\_ARR\_2N>

* 1. <N\_ARR\_2N ***(N, &T, S, ref)*** > 🡪 € |

<OTHER\_N\_VALUE>

* 1. <OTHER\_N\_VALUE ***(N, &T, S, ref, CN)*** > 🡪 . ID***{N = ID.VP*** ***if(CN) CN=LookupCDT(N,CN) else CN=LookupST(N,S)*** ***}*** <N\_INIT\_VALUE ***(N, &T, S, ref, CN)*** > | (<CALLING\_PARAM>) <N\_INIT\_VALUE>

1. <ARRAY> 🡪 [ <ARRAY-INNER>]
   1. <ARRAY-INNER> 🡪 € | <ARRAY-VALUES>
   2. <ARRAY-VALUES> 🡪<ARRAY-VAL><NEXT-VAL>
   3. <ARRAY-VAL> 🡪 <INIT-VALUE-2> | … ID <N\_INIT-VALUE>
   4. <NEXT-VAL>🡪 , <ARRAY-VALUES> | €
2. <SPREAD> 🡪 … ID <N-INIT-VALUE>

1. <OBJECT> 🡪 {<PROP>}
   1. <PROP>🡪 <OBJECT1> <NEXT-PROP>
   2. <OBJECT1> 🡪 ID : <EXP> | <SPREAD>
   3. <NEXT-PROP>🡪, <PROP>| €

1. <INIT-VALUE-2> 🡪 <EXP> | <ARRAY> | <OBJECT>
2. <CALLING-PARAMS ***(&PL, S)***> 🡪 <CP-VALUE ***(&PL, S)***>
   1. <CP-VALUE ***(&PL, S)***> 🡪 € ***{PL=void}*** | <CP-VALUE2 ***(&PL, S)***>
   2. <CP-VALUE2 ***(&PL, S)***> 🡪 <INIT-VALUE-2> <NEXT-CPVALUE>
   3. <NEXT-CPVALUE> 🡪 € |, <CP-VALUE2>
3. <SWITCH-CASE> 🡪 switch (<N-INIT-VALUE>) <SW-BODY>
   1. <SW-BODY> 🡪 {<CASES> <DEFAULT>}
   2. <CASES>🡪 case <CASE-VALUE>: ***{S=CreateScope()}*** <BODY>
   3. <CASE-VALUE> 🡪 <INNER-CASE-VAL>
   4. <INNER-CASE-VAL> 🡪 <EXP>
   5. <DEFAULT> 🡪 default : ***{S=CreateScope()}*** <BODY>
4. <INC-DEC-PRE> 🡪 inc\_dec ID <N-INIT-VALUE>