AST SEMANTIC RULES

CS F363 – Compiler Construction | Group 24

Nachiket Kotalwar (2020A7PS0024P)
Parth Patel (2020A7PS0026P)
Partha Sarathi Purkayastha (2020A7PS0043P)
Labeeb Ahsan (2020A7PS0045P)
Anish Atul Kulkarni (2020A7PS0975P)

```
Program Moduledeclarations Othermodules1 Drivermodule Othermodules2 .
{
   Moduledeclarations.list_inh=createHead()
   Othermodules1.list_inh = createHead()
   Othermodules2.list_inh = createHead()


   BOTTOM_UP:
   Moduledeclarations.node_syn =
create_node("Moduledeclarations",Moduledeclarations.list_syn)
   Othermodules1.node_syn = create_node("Othermodules1", Othermodules1.list_syn)
   Othermodules2.node_syn = create_node("Othermodules2", Othermodules2.list_syn)

   Program.node_syn = create_node("Program", Moduledeclarations.node_syn,
Othermodules1.node_syn,Drivermodule.node_syn, Othermodules2.node_syn)
   free(Moduledeclarations)
   free(Othermodules1)
   free(Drivermodule)
   free(Othermodules2)
}

Moduledeclarations Moduledeclaration Moduledeclarations1 .
{
   TOP_DOWN:

Moduledeclarations.list_inh=insert_at_end(Moduledeclarations.list_inh,Moduledeclaration.
node_syn)
   Moduledeclarations1.list_inh=Moduledeclarations.list_inh

   BOTTOM_UP:
   Moduledeclarations.list_syn = Moduledeclarations1.list_syn
```

```
   //Moduledeclarations.list_syn = insert_at_head(Moduledeclarations1.list_syn,
Moduledeclaration.node_syn)
   free(Moduledeclaration)
   free(Moduledeclarations1)
}

Moduledeclarations .
{
   BOTTOM_UP:
   Moduledeclarations.list_syn = Moduledeclarations.node_inh
}


Moduledeclaration declare module id semicol .
{
   BOTTOM_UP:
   Moduledeclaration.node_syn = id
   free(declare)
   free(module)
   free(semicol)
}

Othermodules Module Othermodules1 .
{
   TOP_DOWN:
   Othermodules.list_inh = insert_at_end(Othermodules.list_inh, Module.node_syn)
   Othermodules1.list_inh = Othermodules.list_inh
   BOTTOM_UP:
   Othermodules.list_syn = Othermodules1.list_syn

   free(Module)
   free(Othermodules1)
}

Othermodules .
{
   Othermodules.list_syn = Othermodules.list_inh
}

Drivermodule driverdef driver program driverenddef Moduledef .
{
   BOTTOM_UP:
   Drivermodule.node_syn = create_node("DRIVER", Moduledef.node_syn)
   free(driverdef)
   free(driver)
   free(program)
   free(driverenddef)
```

```
   free(Moduledef)
}

Module def module id enddef takes input sqbo Input_plist sqbc semicol Ret Moduledef .
{

   BOTTOM_UP:
   Module.node_syn = create_node("Module",id,create_node("PARAMETERS_LIST",
Input_plist.list_syn),Ret.node_syn,Moduledef.node_syn)
   free(def)
   free(module)
   free(enddef)
   free(takes)
   free(input)
   free(sqbo)
   free(Input_plist)
   free(sqbc)
   free(semicol)
   free(Ret)
   free(Moduledef)
}

Ret returns sqbo Output_plist sqbc semicol .
{
   Output_plist.list_inh=createHead()

   BOTTOM_UP:
   Ret.node_syn = create_node("PARAMETERS_LIST", Output_plist.list_syn)
   free(returns)
   free(sqbo)
   free(Output_plist)
   free(sqbc)
   free(semicol)
}

Ret .
{
   BOTTOM_UP:
   Ret.node_syn = create_node("PARAMETERS_LIST", NULL)
}

Input_plist id colon Datatype A' .
{
   //BOTTOM_UP:
   //Input_plist.list_syn = A'.list_syn
   //TOP_DOWN:
```

```
   //Input_plist.list_syn = insert_at_end(A'.list_syn,id,create_node("Datatype-
id",id,Datatype.node_syn))
   //Input_plist.list_inh = newHead()
   //A'.list_inh = Input_plist.list_inh

TOP_DOWN:
   A'.list_inh=createHead()
   A'.list_inh=insert_at_end(A'.list_inh,create_node("Datatype-Id",id,Datatype.node_syn))

BOTTOM_UP:
   Input_plist.list_syn=A'.list_syn

   free(colon)
   free(Datatype)
   free(A')

}

A' comma id colon Datatype A'1 .
{
   TOP_DOWN:
   A'.list_inh = insert_at_end(A'.list_inh,create_node("Datatype-id",id,Datatype.node_syn))
   A'1.list_inh = A'.list_inh
   BOTTOM_UP:
   A'.list_syn = A'1.list_syn

   free(comma)
   free(colon)
   free(Datatype)
   free(A'1)
}

A' .
{
   BOTTOM_UP:
   A'.list_syn = A'.list_inh
}

Output_plist id colon Datatype B' .
{
   BOTTOM_UP:
   Output_plist.list_syn = B'.list_syn

   TOP_DOWN:
   B'.list_inh=createHead()
   B'.list_inh=insert_at_end(B'.list_inh,create_node("Datatype-Id",id,Datatype.node_syn))
```

```
   free(colon)
   free(Datatype)
   free(B')
}

B' comma id colon Datatype B'1 .
{
   TOP_DOWN:
   B'.list_inh = insert_at_end(B'.list_inh,create_node("Datatype-id",id,Datatype.node_syn))
   B'1.list_inh = B'.list_inh
   BOTTOM_UP:
   B'.list_syn = B'1.list_syn

   free(comma)
   free(colon)
   free(Datatype)
   free(B'1)
}

B' .
{
   BOTTOM_UP:
   B'.list_syn = B'.list_inh
}

Datatype integer .
{
   BOTTOM_UP:
   Datatype.node_syn = integer
}

Datatype real .
{
   BOTTOM_UP:
   Datatype.node_syn = real
}

Datatype boolean .
{
   BOTTOM_UP:
   Datatype.node_syn = boolean
}

Datatype array sqbo Rangenew sqbc of Type .
{
   BOTTOM_UP:
```

```
   Datatype.node_syn = create_node("RANGE
DATATYPE",Rangenew.node_syn,Type.node_syn)

   free(array)
   free(sqbo)
   free(Rangenew)
   free(sqbc)
   free(of)
   free(Type)
}

Rangenew Indarray1 rangeop Indarray2 .
{
   BOTTOM_UP:
   Rangenew.node_syn = create_node("ARRAY RANGE", Indarray1.node_syn,
Indarray2.node_syn)
   free(Indarray1)
   free(rangeop)
   free(Indarray2)
}

Indarray Sign Indexcoef .
{
   BOTTOM_UP:
   Indarray.node_syn = create_node("BOUND",Sign.node_syn,Indexcoef.node_syn)
   free(Sign)
   free(Indexcoef)
}

Indarray Indexcoef .
{
   BOTTOM_UP:
   Indarray.node_syn = create_node("BOUND",NULL,Indexcoef.node_syn)
   free(Indexcoef)
}

Indexcoef num .
{
   BOTTOM_UP:
   Indexcoef.node_syn = num
}

Indexcoef id .
{
   BOTTOM_UP:
   Indexcoef.node_syn = id
}
```

```
Sign plus .
{
   BOTTOM_UP:
   Sign.node_syn = plus
}

Sign minus .
{
   BOTTOM_UP:
   Sign.node_syn = minus
}

Type integer .
{
   BOTTOM_UP:
   Type.node_syn = integer
}

Type real .
{
   BOTTOM_UP:
   Type.node_syn = real
}

Type boolean .
{
   BOTTOM_UP:
   Type.node_syn = boolean
}

Moduledef start Statements end .
{
   BOTTOM_UP:
   Moduledef.node_syn = create_node("Statements", Statements.list_syn)
   free(start)
   free(Statements)
   free(end)
}

Statements Statement Statements1 .
{
   TOP_DOWN:
   Statements.list_inh  = insert_at_end(Statements.list_inh,Statement.node_syn);
   Statements1.list_inh = Statements.list_inh
   BOTTOM_UP:
   Statements.list_syn = Statements1.list_syn
```

```
   free(Statement)
   free(Statements1)
}

Statements .
{
   Statements.list_syn = Statements.list_inh;
}

Statement Iostmt .
{
   BOTTOM_UP:
   Statement.node_syn = Iostmt.node_syn
   free(Iostmt)
}

Statement Simplestmt .
{
   BOTTOM_UP:
   Statement.node_syn = Simplestmt.node_syn
   free(Simplestmt)
}

Statement Declarestmt .
{
   BOTTOM_UP:
   Statement.node_syn = Declarestmt.node_syn
   free(Declarestmt)
}

Statement Conditionalstmt .
{
   BOTTOM_UP:
   Statement.node_syn = Conditionalstmt.node_syn
   free(Conditionalstmt)
}

Statement Iterativestmt .
{
   BOTTOM_UP:
   Statement.node_syn = Iostmt.node_syn
   free(Iterativestmt)
}

Iostmt get_value bo id bc semicol .
{
```

```
   BOTTOM_UP:
   Iostmt.node_syn = create_node("GET_VALUE",id);
   free(get_value)
   free(bo)
   free(bc)
   free(semicol)
}

Iostmt print bo Printvar bc semicol .
{
   BOTTOM_UP:
   Iostmt.node_syn = create_node("PRINT",Printvar.node_syn);
   free(print)
   free(bo)
   free(Printvar)
   free(bc)
   free(semicol)
}

Varnew Printvar .
{
   BOTTOM_UP:
   Varnew.node_syn = create_node("Parameter variable",NULL,Printvar.node_syn)
   free(Printvar)
}

Varnew Sign Printvar .
{
   BOTTOM_UP:
   Varnew.node_syn = create_node("Parameter variable",Sign.node_syn,Printvar.node_syn)
   free(Sign)
   free(Printvar)
}

Printvar Bool .
{
   BOTTOM_UP:
   Printvar.node_syn = Bool.node_syn
   free(Bool)
}

Printvar num .
{
   BOTTOM_UP:
   Printvar.node_syn = num
}
```

```
Printvar rnum .
{
   BOTTOM_UP:
   Printvar.node_syn = rnum
}

Printvar id J' .
{
   BOTTOM_UP:
   Printvar.node_syn = create_node("ARRAY ELEM PRINT",id,J'.node_syn)
   free(J')
}

J' sqbo Indarray sqbc .
{
   BOTTOM_UP:
   J'.node_syn = Indarray.node_syn
   free(sqbo)
   free(Indarray)
   free(sqbc)
}

J' .
{
   BOTTOM_UP:
   J'.node_syn = NULL
}

Simplestmt Assignmentstmt .
{
   BOTTOM_UP:
   Simplestmt.node_syn = Assignmentstmt.node_syn
   free(Assignmentstmt)
}

Simplestmt Modulereusestmt .
{
   BOTTOM_UP:
   Simplestmt.node_syn = Modulereusestmt.node_syn
   free(Modulereusestmt)
}

Assignmentstmt id Whichstmt .
{
   BOTTOM_UP:
   Assignmentstmt.node_syn = Whichstmt.node_syn
   TOP_DOWN:
```

```
   Whichstmt.node_inh = id
   free(Whichstmt)
}

Whichstmt Lvalueidstmt .
{
   TOP_DOWN:
   Lvalueidstmt.node_inh = Whichstmt.node_inh

   BOTTOM_UP:
   Whichstmt.node_syn = Lvalueidstmt.node_syn
   free(Lvalueidstmt)
}

Whichstmt Lvaluearrstmt .
{
   TOP_DOWN:
   Lvaluearrstmt.node_inh = Whichstmt.node_inh

   BOTTOM_UP:
   Whichstmt.node_syn = Lvaluearrstmt.node_syn
   free(Lvaluearrstmt)
}

Lvalueidstmt assignop Expression semicol .
{
   TOP_DOWN:
   Lvalueidstmt.node_syn = create_node("ASSIGN", Lvalueidstmt.node_inh,
Expression.node_syn)
   free(assignop)
   free(Expression)
   free(semicol)
}

Lvaluearrstmt sqbo Elemindex sqbc assignop Expression semicol .
{
   TOP_DOWN:
   Lvaluearrstmt.node_syn = create_node("ASSIGN",create_node("ARRAY
ELEM",Lvaluearrstmt.node_inh,Elemindex.node_syn),Expression.node_syn)
   free(sqbo)
   free(Elemindex)
   free(sqbc)
   free(assignop)
   free(Expression)
   free(semicol)
}
```

```
Elemindex Sign G' .
{
   BOTTOM_UP:
   Elemindex.node_syn = create_node("ARRAY INDEX",Sign.node_syn,G'.node_syn)
   free(Sign)
   free(G')
}

Elemindex Aexpr .
{

   Elemindex.node_syn = Aexpr.node_syn
   free(Aexpr)
}

G' Indexcoef .
{
   G'.node_syn = Indexcoef.node_syn
   free(Indexcoef)
}

G' bo Aexpr bc .
{
   G'.node_syn = Aexpr.node_syn
   free(bo)
   free(Aexpr)
   free(bc)
}

Modulereusestmt Optional use module id with parameters Actparalist semicol .
{
   Modulereusestmt.node_syn = create_node("REUSE
STMT",id,Optional.node_syn,create_node("ACCPARA LIST",Actparalist.list_syn))
   free(Optional)
   free(use)
   free(module)
   free(with)
   free(parameters)
   free(Idlist)
   free(semicol)
}

Optional sqbo Idlist sqbc assignop .
{
   Optional.node_syn = Idlist.node_syn
   free(sqbo)
   free(Idlist)
```

```
   free(sqbc)
   free(assignop)
}

Optional .
{
   Optional.node_syn = NULL
}

Idlist id C' .
{
   C'.list_inh = createHead()
   C'.list_inh = insert_at_end(C'.list_inh,id)

   BOTTOM_UP:
   Idlist.node_syn = C'.list_syn

   free(C')
}

C' comma id C'1 .
{
   TOP_DOWN:
   C'.list_inh = insert_at_end(C'.list_inh,id)
   C'1.list_inh = C'.list_inh
   BOTTOM_UP:
   C'.list_syn = C'1.list_syn

   free(comma)
   free(C'1)
}

C' .
{
   BOTTOM_UP:
   C'.list_syn = C'.list_inh
}

Actparalist Varnew N11 .
{
   N11.list_inh=createHead()
   N11.list_inh=insert_at_end(N11.list_inh,Varnew.node_syn)

   Actparalist.node_syn=N11.node_syn

   //Actparalist.h = insert_at_head(N11.list_syn,Varnew.node_syn)
   free(Varnew)
```

```
   free(N11)
}

N11 comma Varnew N11a .
{
   TOP_DOWN:
   N11.list_inh = insert_at_end(N11.list_inh, Varnew.node_syn)
   N11a.list_inh = N11.list_inh
   BOTTOM_UP:
   N11.list_syn = N11a.list_syn
   free(comma)
   free(Varnew)
   free(N11)
}

N11 .
{
   N11.list_syn = N11.list_inh
}

Expression Aorbexpr .
{
   Expression.node_syn = Aorbexpr.node_syn
   free(Aorbexpr)
}

Expression Unaryexpr .
{
   Expression.node_syn = Unaryexpr.node_syn
   free(Unaryexpr)
}

Unaryexpr Op3 Unaryop .
{
   Unaryop.node_inh = Op3.node_syn
   Unaryexpr.node_syn = Unaryop.node_syn
   free(Op3)
   free(Unaryop)

}

Unaryop bo Arithmeticexpr bc .
{
   Unaryop.node_syn = create_node("UNARY
OP",Unaryop.node_inh,Arithmeticexpr.node_syn)
   free(bo)
   free(Arithmeticexpr)
```

```
   free(bc)
}

Unaryop Var_idnum .
{
   Unaryop.node_syn = create_node("UNARY OP",Unaryop.node_inh,Var_idnum.node_syn)
   free(Var_idnum)
}

Var_idnum num .
{
   BOTTOM_UP:
   Var_idnum.node_syn = num
}

Var_idnum rnum .
{
   BOTTOM_UP:
   Var_idnum.node_syn = rnum
}

Var_idnum id .
{
   BOTTOM_UP:
   Var_idnum.node_syn = id
}

Op3 plus .
{
   BOTTOM_UP:
   Op3.node_syn = plus
}

Op3 minus .
{
   BOTTOM_UP:
   Op3.node_syn = minus
}

Aorbexpr Genterm H' .
{
   Aorbexpr.node_syn = H'.node_syn
   H'.node_inh = Genterm.node_syn
   free(Genterm)
   free(H')
}
```

```
H' Logicalop Genterm H'1 .
{
   H'.node_inh = create_node("LOGICAL
OP",H'.node_inh,Logicalop.node_syn,Genterm.node_syn)
   H'1.list_inh=H'.list_inh

   H'.node_syn = H'1.node_syn

   free(Logicalop)
   free(Genterm)
   free(H'1)

}

H' .
{
   H'.node_syn = H'.node_inh

}

Genterm Arithmeticexpr I' .
{

   Genterm.node_syn = I'.node_syn
   I'.node_inh = Arithmeticexpr.node_syn

}

I' Relationalop Arithmeticexpr .
{
   I'.node_syn = create_node("RELATIONAL
OP",I'.node_inh,Relationalop.node_syn,Arithmeticexpr.node_syn)
   free(Relationalop)
   free(Arithmeticexpr)
}

I' .
{
   I'.node_syn = I'.node_inh
}

Arithmeticexpr Term D' .
{
   Arithmeticexpr.node_syn = D'.node_syn
   D'.node_inh = Term.node_syn

   free(Term)
```

```
   free(D')
}

D' Op1 Term D'1 .
{
   TOP_DOWN:
   D'.node_inh = create_node("Op1",D'.node_inh,Term.node_syn)
   D'1.node_inh = D'.node_inh

   BOTTOM_UP:
   D'.node_syn = D'1.node_syn

   free(Op1)
   free(Term)
   free(D'1)
}

D' .
{
   BOTTOM_UP:
   D'.node_syn = D'.node_inh
}

Op1 plus .
{
   BOTTOM_UP:
   Op1.node_syn = plus
}

Op1 minus .
{
   BOTTOM_UP:
   Op1.node_syn = minus
}

Term Factor E' .
{
   E'.node_inh = Factor.node_syn

   Tern.node_syn = E'.node_syn

   free(Factor)
   free(E')
}

E' Op2 Factor E'1 .
{
```

```
   TOP_DOWN:
   E'.node_inh = create_node("Op2",E'.node_inh,Factor.node_syn)
   E'1.node_inh = E'.node_inh

   BOTTOM_UP:
   E'.node_syn = E'1.node_syn

   free(Op2)
   free(Factor)
   free(E'1)
}

E' .
{
   BOTTOM_UP:
   E'.node_syn = E'.node_inh
}

Op2 mul .
{
   BOTTOM_UP:
   Op2.node_syn = mul
}

Op2 div .
{
   BOTTOM_UP:
   Op2.node_syn = div
}

Factor bo Aorbexpr bc .
{
   BOTTOM_UP:
   Factor.node_syn = Aorbexpr.node_syn

   free(bo)
   free(Aorbexpr)
   free(bc)
}

Factor Bool .
{
   BOTTOM_UP:
   Factor.node_syn = Bool.node_syn
   free(Bool)
}
```

```
Factor num .
{
   BOTTOM_UP:
   Factor.node_syn = num
}

Factor rnum .
{
   BOTTOM_UP:
   Factor.node_syn = rnum
}

Factor id K' .
{
   K'.node_inh = id
   free(K')
}

K' sqbo Elemindex sqbc .
{
   K'.node_syn = create_node("ARRAY ELEM",K'.node_inh,Elemindex.node_syn)
   free(sqbo)
   free(Elemindex)
   free(sqbc)
}

K' .
{
   K'.node_syn = K'.node_inh
}

Aexpr Aterm N4 .
{
   N4.node_inh = Aterm.node_syn
   Aexpr.node_syn = N4.node_syn

   free(Aterm)
   free(N4)
}

N4 Op1 Aterm N41 .
{
   N4.node_inh = create_node("ARITH OP1",N4.node_inh,Op1.node_syn,Aterm.node_syn)
   N41.node_inh = N4.node_inh

   N4.node_syn = N41.node_syn
```

```
   free(Op1)
   free(Aterm)
   free(N41)
}

N4 .
{
   N4.node_syn = N4.node_inh
}

Aterm Afactor N5 .
{
   N5.node_inh = Afactor.node_syn
   Aterm.node_syn = N5.node_syn

   free(Afactor)
   free(N5)
}

N5 Op2 Afactor N51 .
{
   N51.node_inh = create_node("ARITH OP2",N5.node_inh,Op2.node_syn,Afactor.node_syn)
   N5.node_syn = N51.node_syn
   free(Op2)
   free(Afactor)
   free(N51)
}

N5 .
{
   N5.node_syn = N5.node_inh
}

Afactor id .
{
  Afactor.node_syn = id
}

Afactor num .
{
   Afactor.node_syn = num
}

Afactor Bool .
{
   Afactor.node_syn = Bool.node_syn
   free(Bool)
```

```
}

Afactor bo Aexpr bc .
{
   Afactor.node_syn = Aexpr.node_syn
   free(bo)
   free(Aexpr)
   free(bc)
}

Bool true .
{
   BOTTOM_UP:
   Bool.node_syn = true
}

Bool false .
{
   BOTTOM_UP:
   Bool.node_syn = false
}

Logicalop AND .
{
   BOTTOM_UP:
   LOGICALOP.node_syn = and
}

Logicalop OR .
{
   BOTTOM_UP:
   LOGICALOP.node_syn = or
}

Relationalop lt .
{
   BOTTOM_UP:
   RELATIONALOP.node_syn = lt
}

Relationalop gt .
{
   BOTTOM_UP:
   RELATIONALOP.node_syn = gt
}

Relationalop le .
```

```
{
   BOTTOM_UP:
   RELATIONALOP.node_syn = le
}
Relationalop ge .
{
   BOTTOM_UP:
   RELATIONALOP.node_syn = ge
}

Relationalop eq .
{
   BOTTOM_UP:
   RELATIONALOP.node_syn = eq
}

Relationalop ne .
{
   BOTTOM_UP:
   RELATIONALOP.node_syn = ne
}

Declarestmt declare Actparalist colon Datatype semicol .
{
   BOTTOM_UP:
   DECALRESTMT.node_syn =
create_node("DECLARE",Datatype.node_syn,create_node("ACCPARA
LIST",Actparalist.list_inh))
   free(declare)
   free(Actparalist)
   free(colon)
   free(Datatype)
   free(semicol)
}

Conditionalstmt switch bo id bc start Casestmt Default end .
{
   BOTTOM_UP:
   Conditionalstmt.node_syn = create_node("SWITCH",id,create_node("CASES",
Casestmt.list_syn), Default.node_syn)
   free(switch)
   free(bo)
   free(bc)
   free(start)
   free(Casestmt)
   free(Default)
   free(end)
```

```
}

Casestmt case Value colon Statements break semicol F' .
{
   BOTTOM_UP:
   //Casestmt.node_syn = insert_at_head(F'.list_syn,create_node("Value-Datatype-
Statements",Value.node_syn,Datatype.node_syn,Statements.list_syn))
   Casestmt.list_syn = F'.list_syn
   TOP_DOWN:
   Casestmt.list_inh = newHead()
   Casestmt.list_inh = insert_at_end(Casestmt.list_inh,create_node("Value-Datatype-
Statements",Value.node_syn,create_node("STATEMENTS",Statements.list_syn)))
   F'.list_inh = Casestmt.list_inh

   free(case)
   free(Value)
   free(colon)
   free(Statements)
   free(break)
   free(semicol)
   free(F')
}

F' case Value colon Statements break semicol F'1 .
{
   TOP_DOWN:
   F'.list_inh = insert_at_end(F'.list_inh,create_node("Value-Datatype-
Statements",Value.node_syn,Statements.list_syn))
   F'1.list_inh = F'.list_inh
   BOTTOM_UP:
   F'.list_syn = F'1.list_syn

   free(case)
   free(Value)
   free(colon)
   free(Statements)
   free(break)
   free(semicol)
   free(F'1)
}

F' .
{
   F'.list_syn = F'.list_inh
}

Value num .
```

```
{
   BOTTOM_UP:
   Value.node_syn = num
}

Value true .
{
   BOTTOM_UP:
   Value.node_syn = true
}

Value false .
{
   BOTTOM_UP:
   Value.node_syn = false
}

Default default colon Statements break semicol .
{
   Default.node_syn = create_node("DEFAULT",Statements.list_syn)
   free(default)
   free(colon)
   free(Statements)
   free(break)
   free(semicol)
}

Default .
{
   Default.node_syn = NULL
}

Iterativestmt for bo id in Range bc start Statements end .
{
   Iterativestmt.node_syn = create_node("FOR
LOOP",id,Range.node_syn,create_node("STATEMENTS",Statements.list_syn))

   free(for)
   free(bo)
   free(in)
   free(Range)
   free(bc)
   free(start)
   free(Statements)
   free(end)
}
```

```
Range Indloop1 rangeop Indloop2 .
{
   BOTTOM_UP:
   Range.node_syn = create_node("LOOP RANGE", Indloop1.node_syn, Indloop2.node_syn)
   free(Indloop1)
   free(rangeop)
   free(Indloop2)
}

Indloop Signloop Indcoefloop .
{
   BOTTOM_UP:
   Indloop.node_syn = create_node("RANGE LIMIT",Sign.node_syn,Indcoefloop.node_syn)
   free(Signloop)
   free(Indcoefloop)
}

Indcoefloop num .
{
   Indcoefloop.node_syn = num
}

Signloop plus .
{
   BOTTOM_UP:
   Signloop.node_syn = plus
}

Signloop minus .
{
   BOTTOM_UP:
   Signloop.node_syn = minus
}

Signloop .
{
   BOTTOM_UP:
   Signloop.node_syn = NULL
}

Iterativestmt while bo Aorbexpr bc start Statements end .
{
   Iterativestmt.node_syn =
create_node("WHILELOOP",Aorbexpr.node_syn,create_node("STATEMENTS",Statements.list
_syn))

   free(while)
```

```
    free(bo)
    free(Expression)
    free(bc)
    free(start)
    free(Statements)
    free(end)
}
```