Semantic Rules

<program> <moduleDeclarations> <otherModules> <driverModule> <otherModules> {

traversal: post order

<program>.syn_list = <moduleDeclarations>.syn_list

insertatend(<program>.syn_list, <otherModules>1.syn_list)

insertatend(<program>.syn_list, <driverModule>.syn_addr)

insertatend(<program>.syn_list, <otherModules>2.syn_list)

free (moduleDeclarations)

free (otherModules1)

free (driverModule)

free (otherModules2)

}

<moduleDeclarations> <moduleDeclaration> <moduleDeclarations> {

traversal: post order

<moduleDeclarations>1.syn_list = insertatbeginning(<moduleDeclarations>2.syn_list, <moduleDeclaration>.addr)

free (moduleDeclaration)

free (moduleDeclarations2)

}

<moduleDeclarations> EPS {

traversal: post order

<moduleDeclarations>.syn_list = NULL

free (EPS)

}

<moduleDeclaration> DECLARE MODULE ID SEMICOL {

traversal: post order

<moduleDeclaration>.addr = ID.addr

free (DECLARE)

free (MODULE)

free (SEMICOL)

}

<otherModules> <module> <otherModules> {

traversal: post order

<otherModules>1.syn_list = insertatbeginning(<otherModules>2.syn_list, <module>.addr)

free (module)

free (otherModules2)

}

<otherModules> EPS {

traversal: post order

<otherModules>.syn_list = NULL

free(EPS)

}

<driverModule> DRIVERDEF DRIVER PROGRAM DRIVERENDDEF <moduleDef> {

traversal: post order

<driverModule>.addr = makeNode(DRIVERMODULE, moduleDef.addr)

free(DRIVERDEF)

free(DRIVER)

free(PROGRAM)

free(DRIVERENDDEF)

free(moduleDef)

}

<module> DEF MODULE ID ENDDEF TAKES INPUT SQBO <input_plist> SQBC SEMICOL <ret> <moduleDef> {

traversal: post order

<module>.addr = makeNode(MODULE, ID.addr, <input_plist>.addr, <ret>.addr, <moduleDef>.addr)

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 {
        traversal: post order
   <ret>.addr = <output_plist>.addr
        free(RETURNS)
        free(SQBO)
        free(output_plist)
        free(SQBC)
        free(SEMICOL)
}
<ret> EPS {
        traversal: post order
   <ret>.addr = NULL
        free(EPS)
}
<input_plist> ID COLON <dataType> <_input_plist> {
        traversal: post order
   newnode = makeNode(INPUT_PLIST, ID.addr, <dataType>.addr)
   <input_plist>.syn_list = insertatbeginning(<_input_plist>.syn_list, newnode)
        free(COLON)
        free(dataType)
        free(_input_plist)
}
<_input_plist> COMMA ID COLON <dataType> <_input_plist> {
   newnode = makeNode(INPUT_PLIST, ID.addr, <dataType>.addr)
   <input_plist>1.syn_list = insertatbeginning(<input_plist>2.syn_list, newnode)
```

```
        free(COMMA)
        free(COLON)
        free(dataType)
        free(_input_plist2)
}
<_input_plist> EPS {
        traversal: post order
   <_input_plist>.syn_list = NULL
        free(EPS)
}
<output_plist> ID COLON <_type> <_output_plist> {
        traversal: post order
   newnode = makeNode(OUTPUT_PLIST, ID.addr, <_type>.addr)
   <output_plist>.syn_list = insertatbeginning(<_output_plist>.syn_list, newnode)
        free(COLON)
        free(dataType)
        free(_output_plist)
}
<_output_plist> COMMA ID COLON <_type> <_output_plist> {
        traversal: post order
   newnode = makeNode(OUTPUT_PLIST, ID.addr, <_type>.addr)
   <output_plist>1.syn_list = insertatbeginning(<output_plist>2.syn_list, newnode)
        free(COMMA)
        free(COLON)
        free(dataType)
        free(_output_plist2)
}
<_output_plist> EPS {
        traversal: post order
   <_output_plist>.syn_list = NULL
        free(EPS)
```

```
}
<dataType> INTEGER {
        traversal: post order
   <dataType>.addr = INTEGER.addr
}
<dataType> REAL {
        traversal: post order
   <dataType>.addr = REAL.addr
}
<dataType> BOOLEAN {
        traversal: post order
   <dataType>.addr = BOOLEAN.addr
}
<dataType> ARRAY SQBO <range_arrays> SQBC OF <_type> {
        traversal: post order
   <dataType>.addr = makeNode(ARRAY,<range_arrays>.addr,<_type>.addr)
        free(ARRAY)
        free(SQBO)
        free(range_arrays)
        free(SQBC)
        free(OF)
        free(<_type>)
}
<range_arrays> <index_arr> RANGEOP <index_arr> {
        traversal: post order
   <range_arrays>.addr = makeNode(RANGE,<index_arr>1.addr,<index_arr>2.addr)
        free(index_arr1)
        free(RANGEOP)
        free(index_arr2)
}
<_type> INTEGER {
```

```
        traversal: post order
    <_type>.addr = INTEGER.addr
}
<_type> REAL {
        traversal: post order
    <_type>.addr = REAL.addr
}
<_type> BOOLEAN {
        traversal: post order
    <_type>.addr = BOOLEAN.addr
}
<moduleDef> START <statements> END {
        traversal: post order
    <moduleDef>.addr = <statements>.addr
        free(START)
        free(statements)
        free(end)
}
<statements> <statement> <statements> {
        traversal: post order
    <statements>1.syn_list = insertatbeginning(<statements>2.syn_list, <statement>.addr)
        free(statement)
        free(statements2)
}
<statements> EPS {
        traversal: post order
    <statements>.syn_list = NULL
        free(EPS)
}
<statement> <ioStmt> {
        traversal: post order
```

```
    <statement>.syn_addr = <ioStmt>.syn_addr
            free(ioStmt)
}
<statement> <simpleStmt> {
            traversal: post order
    <statement>.syn_addr = <simpleStmt>.syn_addr
            free(simpleStmt)
}
<statement> <declareStmt> {
            traversal: post order
    <statement>.syn_addr = <declareStmt>.syn_addr
            free(declareStmt)
}
<statement> <conditionalStmt> {
            traversal: post order
    <statement>.syn_addr = <conditionalStmt>.syn_addr
            free(conditionalStmt)
}
<statement> <iterativeStmt> {
            traversal: post order
    <statement>.syn_addr = <iterativeStmt>.syn_addr
            free(iterativeStmt)
}
<ioStmt> GET_VALUE BO ID BC SEMICOL {
            <post> : <ioStmt>.addr = ID.addr
            free(GET_VALUE) free(BO) free(BC) free(SEMICOL)
}
<ioStmt> PRINT BO <var_print> BC SEMICOL {
            <post> : <ioStmt>.addr = <var_print>.addr
            free(PRINT) free(BO) free(BC) free(SEMICOL)
}
```

<var_print> ID <P1> {

<var_print>.addr = makeNode(ARR,ID.addr,P1.addr)

free(P1)

}

<var_print> NUM {

<post> : <post> <var_print>.addr = NUM.addr

}

<var_print> RNUM {

<post> : <var_print>.addr = RNUM.addr

}

<var_print> <boolConstt> {

<post> : <var_print>.addr = <boolConstt>.addr

free(boolConstt)

}

<boolConstt> TRUE {

<post> : <boolConstt>.addr = TRUE.addr

}

<boolConstt> FALSE {

<post> : <boolConstt>.addr = FALSE.addr

}

<P1> SQBO <index_arr> SQBC {

<post> : <P1>.addr = <index_arr>.addr

free(index_arr)

}

<P1> EPS {

<P1>.addr = NULL

free(EPS)

}

<simpleStmt> <assignmentStmt> {

<post> : <simpleStmt>.addr = <assignmentStmt>.syn_addr

free(assignmentStmt)

}

<simpleStmt> <moduleReuseStmt> {

<post> : <simpleStmt>.addr = <moduleReuseStmt>.syn_addr

free(moduleReuseStmt)

}

<assignmentStmt> ID <whichStmt> {

<post> : <assignmentStmt>.addr = ID.addr

<pre> : <whichStmt>.inh_addr = <assignmentStmt>.addr

<post> : <assignmentStmt>.syn_addr = <whichStmt>.syn_addr

free(whichStmt)

}

<whichStmt> <lvalueIDStmt> {

<pre> : <lvalueIDStmt>.inh_addr = <whichStmt>.inh_addr

<post> : <whichStmt>.syn_addr = <lvalueIDStmt>.addr

free(lvalueIDStmt)

}

<whichStmt> <lvalueARRStmt> {

<pre> : <lvalueARRStmt>.inh_addr = <whichStmt>.inh_addr

<post> : <whichStmt>.syn_addr = <lvalueARRStmt>.addr

free(lvalueARRStmt)

}

<lvalueIDStmt> ASSIGNOP <expression> SEMICOL {

<lvalueIDStmt>.addr = makeNode(ASSIGNOP,<lvalueIDStmt>.inh_addr,<expression>.addr,NULL)

free(ASSIGNOP)

free(SEMICOL)

}

<lvalueARRStmt> SQBO <element_index_with_expressions> SQBC ASSIGNOP <expression> SEMICOL {

<lvalueIDStmt>.addr = makeNode(ASSIGNOP,<lvalueIDStmt>.inh_addr,<element_index_with_expressions>.addr,<expression>.addr,NULL)

```
        free(SQBO) free(SQBC)
        free(ASSIGNOP)
        free(SEMICOL)
}
<index_arr> <sign> <new_index> {
        <index_arr>.addr = makeNode(ARR_INDEX,<sign>.addr,<new_index>.addr)
        free(sign)
        free(new_index)
}
<new_index> NUM {
        <post> : <new_index>.addr = NUM.addr
}
<new_index> ID {
        <post> : <new_index>.addr = ID.addr
}
<sign> PLUS {
        <post> : <sign>.addr = PLUS.addr
}
<sign> MINUS {
        <post> : <sign>.addr = MINUS.addr
}
<sign> EPS {
        <sign>.addr = NULL
        free(EPS)
}
<moduleReuseStmt> <optional> USE MODULE ID WITH PARAMETERS <actual_para_list> SEMICOL {
        <pre> : <actual_para_list>.inh_addr = ID.addr
        <post> : <moduleReuseStmt>.syn_addr = <actual_para_list>.addr
        <pre> : <optional>.inh_addr = <moduleReuseStmt>.syn_addr
        <post> : <moduleReuseStmt>.syn_addr = <optional>.addr
        free(USE,MODULE,WITH,PARAMETERS,SEMICOL)
```

```
}
<actual_para_list> <K> <N_12> {
        //newNode1 pointing to K
        <post> : <actual_para_list>.syn_list = insertAtFront(<N_12>.syn_list,<K>.addr)
        <actual_para_list>.addr =
makeNode(MODULE,<actual_para_list>.inh_addr,<actual_para_list>.syn_list)
        free(K)
        free(N_12)
}
<actual_para_list> <sign> <K> <N_12> {
        //newNode1 pointing to sign and K
        <actual_para_list>.syn_list = insertAtFront(<N_12>.syn_list,<sign>.addr,<K>.addr)
        <actual_para_list>.addr =
makeNode(MODULE,<actual_para_list>.inh_addr,<actual_para_list>.syn_list)
        free(K)
        free(N_12)
}
<N_12> COMMA <sign> <K> <N_12> {
        //if sign not eps, newNode1 points to sign and K else only K
        <post> : <N_12>.syn_list = insertAtFront(<N_12>1.syn_list,<sign>.addr,<K>.addr)
        free(COMMA)
        free(K)
        free(N_12)
}
<N_12> EPS {
        <N_12>.syn_list = NULL
        free(EPS)
}

<K> NUM {
        <post> : <K>.addr = NUM.addr
}
```

```
<K> RNUM {
        <post> : <K>.addr = RNUM.addr
}
<K> <boolConstt> {
        <post> : <K>.addr = <boolConstt>.addr
        free(boolConstt)
}
<K> ID <N_11> {
        <post> : <K>.addr = makeNode(arr,ID.addr,<N_11>.addr)
        free(N_11)
}
<optional> SQBO <idList> SQBC ASSIGNOP {
        <optional>.addr = makeNode(OPTIONAL1,<idList>.addr, <optional>.inh_addr)
        free(SQBO,SQBC,ASSIGNOP)
}
<optional> EPS {
        <optional>.addr = makeNode(OPTIONAL2,<optional>.inh_addr)
        free(EPS)
}
<idList> ID <N3> {
        <post> : <idList>.addr = insertAtFront(<N3>.syn_list,ID.addr)
        free(N3)
}
<N3> COMMA ID <N3> {
        <post> : <N3>.syn_list = insertAtFront(<N3>1.syn_list,ID.addr)
        free(COMMA)
        free(<N3>1)
}
<N3> EPS {
        <N3>.syn_list = NULL;
        free(EPS)
```

```
}
<expression> <arithmeticOrBooleanExpr> {
        traversal : bottom-up
        <expression>.addr = <arithmeticOrBooleanExpr>.syn_addr;
        free(<arithmeticOrBooleanExpr>);
}
<expression> <U> {
        traversal : bottom-up
        <expression>.addr = <U>.syn_addr;
        free(<U>);
}
<U> <unary_op> <new_NT> {
        /*
        U.addr = unary_op.addr;
        unary_op.addr->child = new_NT->addr;
        */
        traversal : top-down
        <new_NT>.addr = makeNode('<unary_op>',NULL,<U>.addr);
        traversal : bottom-up
        <U>.syn_addr = <new_NT>.syn_addr;
        free(<new_NT>);
}
<new_NT> BO <arithmeticExpr> BC {
        traversal : bottom-up
        <new_NT>.syn_addr = <arithmeticExpr>.syn_addr;
        free(BO);
        free(BC);
        free(<arithmeticExpr>);
}
<new_NT> <var_id_num> {
        traversal : bottom-up
```

```
        <new_NT>.addr = <var_id_num>.syn_addr
        free(<var_id_num>);
}
<unary_op> PLUS {
        traversal : bottom-up
        <unary_op>.addr = PLUS.addr;
}
<unary_op> MINUS{
        traversal : bottom-up
        <unary_op>.addr = MINUS.addr
}
<var_id_num> ID {
        traversal : bottom-up
        <var_id_num>.addr = ID.addr;
}
<var_id_num> NUM {
        traversal : bottom-up
        <var_id_num>.addr = NUM.addr;
}
<var_id_num> RNUM {
        traversal : bottom-up
        <var_id_num>.addr = RNUM.addr;
}
<arithmeticOrBooleanExpr> <AnyTerm> <N7> {
        traversal : top-down
        <N7>.inh_addr = <AnyTerm>.addr;
        traversal : bottom-up
        <arithmeticOrBooleanExpr>.syn_addr = <N7>.syn_addr;
        free(<N7>);
        free(<AnyTerm>);
}
```

<N7> <logicalOp> <AnyTerm> <N7>' {

traversal : top-down

<N7>'.inh_addr = makeNode(logicalOp,NULL,<N7>.inh_addr);

<N7>'.inh_addr->child->next=<AnyTerm>.addr;

traversal : bottom-up

<N7>.syn_addr = <N7>'.syn_addr;

free(<N7>);

free(<AnyTerm>);

free(<logicalOp>);

}

<N7> EPS {

traversal : bottom-up

<N7>.syn_addr = <N7>.inh_addr;

}

<AnyTerm> <arithmeticExpr> <N8> {

traversal : top-down

<N8>.inh_addr = <arithmeticExpr>.addr;

traversal : bottom-up

<AnyTerm>.syn_addr = <N8>.syn_addr;

free(<N8>);

free(<arithmeticExpr>);

}

<AnyTerm> <boolConstt> {

traversal : bottom-up

<AnyTerm>.syn_addr = <boolConstt>.syn_addr;

free(<boolConstt>);

}

<N8> <relationalOp> <arithmeticExpr>{

traversal : top-down

<arithmeticExpr>.inh_addr = <relationalOp>.addr;

traversal : bottom-up

```
        <N8>.syn_addr = <arithmeticExpr>.syn_addr;
        free(<arithmeticExpr>);
        free(<relationalOp>);
}
<N8> EPS {
        traversal : bottom-up
        <N8>.syn_addr = <N8>.inh_addr;
}
<arithmeticExpr> <term> <N4> {
        traversal : top-down
        <N4>.inh_addr = <term>.addr;
        traversal : bottom-up
        <arithmeticExpr>.syn_addr = <N4>.syn_addr;
        free(<term>);
        free(<N4>);
}
<N4> <op1> <term> <N4>' {
        traversal : top-down
        <N4>'.inh_addr = makeNode(op1,NULL,<N4>.inh_addr);
        <N4>'.inh_addr->child->next=<term>.addr;
        traversal : bottom-up
        <N4>.syn_addr = <N4>'.syn_addr;
        free(<op1>);
        free(<term>);
        free(<N4>');
}
<N4> EPS {
        traversal : bottom-up
        <N4>.syn_addr = <N4>.inh_addr;
}
<term> <factor> <N5> {
```

```
        traversal : top-down
        <N5>.inh_addr = <factor>.addr;
        traversal : bottom-up
        <term>.syn_addr = <N5>.syn_addr;
        free(<factor>);
        free(<N5>);
}
<N5> <op2> <factor> <N5>' {
        traversal : top-down
        <N5>'.inh_addr = makeNode(logicalOp,NULL,<N5>.inh_addr);
        <N5>'.inh_addr->child->next=<factor>.addr;
        traversal : bottom-up
        <N5>.syn_addr = <N5>'.addr;
        free(<op2>);
        free(<factor>);
        free(<N5>');
}
<N5> EPS {
        traversal : bottom-up
        <N75>.syn_addr = <N5>.inh_addr;
}
<factor> BO <arithmeticOrBooleanExpr> BC {
        traversal : bottom-up
        <factor>.addr = <arithmeticOrBooleanExpr>.syn_addr;
        free(BO);
        free(BC);
        free(<arithmeticOrBooleanExpr>);
}
<factor> NUM {
        traversal : bottom-up
        <factor>.addr = NUM.addr;
```

```
}
<factor> RNUM {
        traversal : bottom-up
        <factor>.addr = RNUM.addr;
}
<factor> <boolConstt> {
        traversal : bottom-up
        <factor>.addr = <boolConstt>.addr;
        free(<boolConstt>);
}
<factor> ID <N_11> {
        traversal : top-down
        <N_11>.inh_addr = ID.addr;
        traversal : bottom-up
        <factor>.syn_addr = <N_11>.syn_addr;
        free(<N_11>);
}
<N_11> SQBO <element_index_with_expressions> SQBC {
        traversal : bottom-up
        <N_11>.addr = <element_index_with_expressions>.syn_addr;
        free(SQBO);
        free(SQBC);
        free(<element_index_with_expressions>);
}
<N_11> EPS {
        traversal : bottom-up
        <N_11>.syn_addr = <N_11>.inh_addr;
}
<element_index_with_expressions> <sign> <N_10> {
        traversal : top-down
        <N_10>.inh_addr = <sign>.addr;
```

```
        traversal : bottom-up
        <element_index_with_expressions>.syn_addr = <N_10>.syn_addr;
        free(<sign>);
        free(<N_10>);
}
<element_index_with_expressions> <arrExpr> {
        traversal : bottom-up
        <element_index_with_expressions>.addr = <arrExpr>.syn_addr;
        free(<arrExpr>);
}
<N_10> <new_index> {
        traversal : bottom-up
        <N_10>.addr = <new_index>.syn_addr;
        free(<new_index>);
}
<N_10> BO <arrExpr> BC {
        traversal : bottom-up
        <N_10>.addr = <arrExpr>.syn_addr;
        free(BO);
        free(BC);
        free(<arrExpr>);
}
<arrExpr> <arrTerm> <arr_N4> {
        traversal : top-down
        <arr_N4>.inh_addr = <arrTerm>.addr;
        traversal : bottom-up
        <arrExpr>.syn_addr = <arr_N4>.syn_addr;
        free(<arrTerm>);
        free(<arr_N4>);
}
<arr_N4> <op1> <arrTerm> <arr_N4> {
```

```
        traversal : top-down
        <arr_N4>'.inh_addr = makeNode(op1,NULL,<arr_N4>.inh_addr);
        <arr_N4>'.inh_addr->child->next=<arrTerm>.addr;
        traversal : bottom-up
        <arr_N4>.syn_addr = <arr_N4>'.syn_addr;
        free(<op1>);
        free(<arrTerm>);
        free(<arr_N4>);
}
<arr_N4> EPS {
        traversal : bottom-up
        <arr_N4>.syn = <arr_N4>.inh
}
<arrTerm> <arrFactor> <arr_N5> {
        traversal : top-down
        <arr_N5>.inh_addr = <arrFactor>.addr;
        traversal : bottom-up
        <arrTerm>.syn_addr = <arr_N5>.syn_addr;
        free(<arrFactor>);
        free(<arr_N5>);
}
<arr_N5> <op2> <arrFactor> <arr_N5>' {
        traversal : top-down
        <arr_N5>'.inh_addr = makeNode(OP2,NULL,<arr_N5>.inh_addr);
        <arr_N5>'.inh_addr->child->next=<arrFactor>.addr;
        traversal : bottom-up
        <arr_N5>.syn_addr = <arr_N5>'.syn_addr;
        free(<op2>);
        free(<arrFactor>)
        free(<arr_N5>');
}
```

```
<arr_N5> EPS {
        traversal : bottom-up
        <arr_N5>.syn = <arr_N5>.inh
}
<arrFactor> ID {
        traversal : bottom-up
        <arrFactor>.addr = ID.addr;
}
<arrFactor> BO <arrExpr> BC {
        traversal : bottom-up
        <arrFactor>.addr = <arrExpr>.addr;
        free(BO);
        free(BC);
        free(<arrExpr>);
}
<arrFactor> <boolConstt> {
        traversal : bottom-up
        <arrFactor>.addr = <boolConstt>.addr;
        free(<boolConstt>);
}
<arrFactor> NUM {
        traversal : bottom-up
        <arrFactor>.addr = NUM.addr;
}
<op1> PLUS {
        traversal : bottom-up
        <op1>.addr = PLUS.addr;
}
<op1> MINUS {
        traversal : bottom-up
        <op1>.addr = MINUS.addr;
```

```
}
<op2> MUL {
        traversal : bottom-up
        <op1>.addr = MUL.addr;
}
<op2> DIV {
        traversal : bottom-up
        <op1>.addr = DIV.addr;
}
<logicalOp> AND {
        traversal : bottom-up
        <logicalOp>.addr = AND.addr;
}
<logicalOp> OR {
        traversal : bottom-up
        <logicalOp>.addr = OR.addr;
}
<relationalOp> LT {
        traversal : bottom-up
        <relationalOp>.addr = LT.addr;
}
<relationalOp> LE {
        traversal : bottom-up
        <relationalOp>.addr = LE.addr;
}
<relationalOp> GT {
        traversal : bottom-up
        <relationalOp>.addr = GT.addr;
}
<relationalOp> GE {
        traversal : bottom-up
```

<relationalOp>.addr = GE.addr;

}

<relationalOp> EQ {

traversal : bottom-up

<relationalOp>.addr = EQ.addr;

}

<relationalOp> NE {

traversal : bottom-up

<relationalOp>.addr = NE.addr;

}

<declareStmt> DECLARE <idList> COLON <dataType> SEMICOL{

traversal: post order

<declareStmt>.addr = makeNode(DECLARE, <idList>.addr, <dataType>.addr)

free(all RHS)

}

<conditionalStmt> SWITCH BO ID BC START <caseStmts> <_default> END {

<caseStmts>.inh_addr=ID.addr

<_default>.inh_addr = <caseStmts>.addr.end(); //traverse through all case statements to get to the end of the case list

<conditionalStmt>.addr = <caseStmts>.addr;

free(SWITCH); free(BO); free(BC); free(START); free(END);

}

<caseStmts> CASE <value> COLON <statements> BREAK SEMICOL <N9> {

<caseStmts>.addr = makeNode(SWITCH, <caseStmts>.inh_addr, makeNode(CASE,<value>.addr,<statements>.addr));

<caseStmts>.addr->child->next = <N9>.syn_addr;

free(CASE); free(COLON); free(BREAK); free(SEMICOL);

//computed going bottom up

}

<N9> CASE <value> COLON <statements> BREAK SEMICOL <N9>1 {

<N9>.syn_addr = makeNode(CASE,<value>.addr,<statements>.addr);

```
        <N9>.syn_addr->next = <N9>1.syn_addr;
        free(CASE); free(COLON); free(BREAK); free(SEMICOL);
        //computed going bottom up
}
<N9> EPS  {
        <N9>.syn_addr = NULL;
        free(EPS);
}
<value> NUM {
        <value.addr> = NUM.addr;
}
<value> TRUE {
        <value.addr> = TRUE.addr;
}
<value> FALSE {
        <value.addr> = FALSE.addr;
}
<_default> DEFAULT COLON <statements> BREAK SEMICOL  {
        <_default>.inh_addr->next=makeNode(DEFAULT,NULL,<statements>.addr);
        free(DEFAULT); free(COLON); free(BREAK); free(SEMICOL);
        //creates default node
}
<_default> EPS  {
        <_default>.inh_addr->next=NULL;
        free(EPS);
}

<iterativeStmt> FOR BO ID IN <range_for_loop> BC START <statements> END {
        <iterativeStmt>.addr = makeNode(FOR,ID.addr,<range_for_loop>.addr);
        <iterativeStmt>.addr->child->next = <statements>.addr;
        free(FOR); free(BO); free(BC); free(IN); free(START); free(END);
```

}

<iterativeStmt> WHILE BO <arithmeticOrBooleanExpr> BC START <statements> END {

<iterativeStmt>.addr = makeNode(WHILE,NULL,<arithmeticOrBooleanExpr>.addr);

<iterativeStmt>.addr->child-next = <statements>.addr;

free(WHILE); free(BO); free(BC); free(START); free(END);

}

<range_for_loop> <index_for_loop>1 RANGEOP <index_for_loop>2 {

<range_for_loop>.addr = makeNode(RANGE,NULL,<index_for_loop>1.addr);

<range_for_loop>.addr->child->next = <index_for_loop>2.addr

free(RANGEOP);

}

<index_for_loop> <sign_for_loop> <new_index_for_loop> {

<index_for_loop>.addr=makeNode(RANGENUM,<sign_for_loop>.addr,<new_index_for_loop>.addr);

}

<new_index_for_loop> NUM {

<new_index_for_loop>.addr=NUM.addr;

}

<sign_for_loop> PLUS {

<sign_for_loop>.addr=PLUS.addr;

}

<sign_for_loop> MINUS {

<sign_for_loop>.addr=MINUS.addr;

}

<sign_for_loop> EPS {

<sign_for_loop>.addr=PLUS.addr;

}