Note:

.inh denotes inherited values, while those who doesn't have it are synthetized.

.s denotes subtree, ***.table_entry*** denotes entry in symbol table, ***.lexval*** denotes lexical value, ***.type*** denotes type, ***.name*** denotes the node name or node label, ***.leftchild***, ***.centerchild*** and ***.rightchild*** denotes left, center, and right childs respectively, and finally ***.op*** denotes operator (i.e. +, -, *, etc).

Maketree and **Makeleaf** semantic actions are explained in the syntax tree implementation file.

Syntax Directed Definition

program --> declaration program { program.s = maketree(“program”, declaration.s, program.s) }
| *epsilon* { program.s = *epsilon* }

declaration --> **void** { id_const.inh.type = void } **id_const** fun-dec-tail { declaration.s = maketree(“declaration”, id-const.s, fun-dec-tail.s) } { declaration.type = void; }
| nonvoid-specifier { id_const.inh.type = nonvoid-specifier.type } **id_const** { dec-tail.inh.type = nonvoid-specifier.type } dec-tail { declaration.s = maketree(“declaration”, id-const.s, dec-tail.s) }
{ declaration.type = nonvoid-specifier.type; }

nonvoid-specifier --> **int** { nonvoid-specifier.type = int }
| **bool** { nonvoid-specifier.type = bool }

id_const -> **ID** { id_const.s = makeleaf(“id”, id.table_entry) }
{ id.type = id_const.inh.type }

dec-tail --> var-dec-tail { dec-tail.s = var-dec-tail.s } { var-dec-tail.inh.type = dec-tail.type }
| fun-dec-tail { dec-tail.s = fun-dec-tail.s }

var-dec-tail --> **[**add-exp**]** { var-dec-tail'.inh.type = var-dec-tail.type } var-dec-tail' **;** { var-dec-tail.s = maketree(“array”, add-exp.s, var-dec-tail'.s) }

| { var-dec-tail'.inh.type = var-dec-tail.type } var-dec-tail' **;** { var-dec-tail.s = var-dec-tail'.s }

var-dec-tail' --> **,** { var-name.inh.type = var-dec-tail'.type } var-name { var-dec-tail2'.inh.type = var-dec-tail'1.type } var-dec-tail2' { var-dec-tail'.s = maketree("multivar", var-name.s, var-dec-tail'.s) }

| *epsilon* { var-dec-tail.s = *epsilon* }

var-name --> { id_const.inh.type = var-name.type } **id_const** var-name' { var-name.s = maketree("var-name", id-const.s, var-name'.s) }

var-name' --> **[**add-exp**]** { var-name'.s = maketree("array", add_exp.s) }

| *epsilon* { var-name.s = *epsilon* }

fun-dec-tail --> **(**params**)** compound-stmt { fun-dec-tail.s = maketree("fun-dec-tail", params.s, compound-stmt.s) }

params --> param {params'.inh.type = param.type } params' { params.s = maketree("params", param.s, params'.s)}

| **void** { params.s = makeleaf("void") } { params.type = void }

params' --> , param params' { params'.s = maketree("multiparam", param.s, params'.s) }

| *epsilon* { params'.s = *epsilon* }

param --> **ref** nonvoid-specifier { id_const.inh.type = ref + nonvoid-specifier.type } **id_const** param' {param.type = id_const.type } { param.s = maketree(ref + nonvoid-specifier.type, id-const.s, param'.s) }

| nonvoid-specifier { id_const.inh.type = nonvoid-specifier.type } **id_const** param' {param.type = id_const.type } { param.s = maketree(nonvoid-specifier.type, id-const.s, param'.s) }

param' --> **[]** { param'.s = makeleaf("array") }

| *epsilon* { param's = *epsilon* }

statement --> id-stmt { statement.s = id-stmt.s }

```
                | compound-stmt                              { statement.s = compound-stmt.s }
                | if-stmt                                    { statement.s = if-stmt.s }
                | loop-stmt                                  { statement.s = loop-stmt.s }
                | exit-stmt                                  { statement.s = exit-stmt.s }
                | continue-stmt                              { statement.s = continue-stmt.s }
                | return-stmt                                { statement.s = return-stmt.s }
                | null-stmt                                  { statement.s = null-stmt.s }

id-stmt  --> id_const id-stmt-tail                           { id-stmt.s = maketree( id-stmt-tail.name, id-const.s, id-stmt-tail.leftchild,  id-stmt-tail.centerchild) }
id-stmt-tail --> assign-stmt-tail                            { id-stmt-tail.s = assign-stmt-tail }
                 | call-stmt-tail                            { id-stmt-tail.s = call-stmt-tail }
assign-stmt-tail --> [add-exp] := expression ;               { assign-stmt-tail.s = maketree( “array_assign”, add-exp.s, expression.s ) }
                     | := expression ;                       { assign-stmt-tail.s = maketree( “assign”, expression.s ) }
call-stmt-tail --> call-tail ;                               { call-stmt-tail.s = call-tail.s }
call-tail --> ( call-tail' )                                 { call-tail.s = call-tail'.s }
call-tail' --> arguments                                     { call-tail'.s = arguments.s }
               | epsilon                                     { call-tail'.s =maketree( “no_arguments” ); }
arguments --> expression arguments'                          { arguments.s = maketree(“routine_call”, expression.s, arguments'.s) }
arguments' --> , expression arguments'                       { arguments'.s = maketree(“arguments'”, expression.s, arguments'.s) }
                 | epsilon                                   { arguments'.s = epsilon }
compound-stmt --> { compound-stmt' compound-stmt'' }         { compound-stmt.s = maketree(“compound-stmt”, commpound-stmt'.s, compound-stmt''.s) }
compound-stmt' --> nonvoid-specifier { id_const.inh.type = nonvoid-specifier.type } id_const { var-dec-tail.inh.type = id_const.type } var-dec-tail compound-stmt-'
                                                                    { compound-stmt'.s = maketree(nonvoid-specifier.type,  id-const.s, var-dectail.s, compound-stmt'.s) }
                    | epsilon                                { compound-stmt'.s = epsilon }
```

compound-stmt" --> statement compound-stmt"'	{ compound-stmt".s = maketree(“compound-stmt" ", statement.s, compound-stmt"'.s) }
compound-stmt"' --> statement compound-stmt"'	{ compound-stmt"'.s = maketree(“compound-stmt"' ", statement.s compound-stmt"'.s) }
| *epsilon*	{ compound-stmt"'.s = *epsilon* }
if-stmt --> **if (** expression **)** statement if-stmt'	{ if-stmt.s = maketree(“if-stmt”, expression.s, statement.s, if-stmt'.s) }
if-stmt' --> **else** statement	{ if-stmt'.s =statement.s }
| *epsilon*	{ if-stmt'.s = *epsilon* }
loop-stmt --> **loop** statement loop-stmt' **end ;**	{ loop-stmt.s = maketree(“loop-stmt”, statement.s, loop-stmt'.s) }
loop-stmt' -->statement loop-stmt'	{ loop-stmt'.s = maketree(“loop-stmt'”, statement.s, loop-stmt'.s) }
| *epsilon*	{ loop-stmt'.s = *epsilon* }
exit-stmt --> **exit ;**	{ exit-stmt.s = makeleaf(“exit”); }
continue-stmt --> **continue ;**	{ continue-stmt.s = makeleaf(“continue”); }
return-stmt --> **return** return-stmt' ;	{ return-stmt.s = maketree(“return”, return-stmt') }
return-stmt' --> expression	{ return-stmt'.s = expression.s }
| *epsilon*	{ return -stmt'.s = *epsilon* }
null-stmt --> **;**	{ null-stmt.s = null }
expression --> add-expr expression'	<u>case expression' did not derive epsilon:</u>
	{ expression.s = maketree(expression'.name, add-exp.s, expression'.leftchild) }
	<u>case expression' derived epsilon:</u>
	{ expression.s = add-exp.s }
expression' --> relop add-exp	{ expression'.s = maketree(relop.op, add-exp) }
| *epsilon*	{ expression'.s = *epsilon* }
add-exp --> uminus term add-exp'	<u>case add-exp' did not derive epsilon:</u> (fix the uminus!)

	{ add-exp.s = maketree(addop.name+“main”, term.s, add-exp'.leftchild, add-exp'.centerchild);
	case add-exp' did not derive epsilon:
	{ add-exp.s = term }
| term add-exp'	case add-exp' did not derive epsilon:
	{ add-exp.s = maketree(addop.name+“main”, term.s, add-exp'.leftchild, add-exp'.centerchild);
	case add-exp' did not derive epsilon:
	{ add-exp.s = term }
add-exp' --> addop term add-exp'	{ add-exp'.s = maketree(addop.op, term.s, add-exp') }
| *epsilon*	{ add-exp'.s = *epsilon* }
term --> factor term'	case term' did not derive epsilon:
	{ term.s = maketree(multop.name +”main”, factor.s, term'.leftchild, term'.centerchild) }
	case term' derived epsilon:
	{ term.s = factor.s }
term' --> multop factor term'	{ term'.s = maketree(multop.op, factor, term') }
| *epsilon*	{ term'.s = *epsilon* }
factor --> nid-factor	{ factor.s = nid-factor.s }
| id-factor	{ factor.s = id.factor.s }
nid-factor --> **not** factor	{ nid-factor.s = maketree(“not”, factor) }
| **(** expression **)**	{ nid-factor.s = expression.s }
| **num**	{ nid-factor.s = makeleaf(“num”, num.table_entry) }
	{ num.type = int }
| **blit**	{ nid.factor.s = makeleaf(“blit”, blit.table_entry) }
	{ blit.type = bool }
id-factor --> **id_const** id-tail	*if id-tail did not derived epsilon:*

	{ id-factor.s = maketree(array_or_call, id-const.s, id-tail-s) }
	if id-tail did derived epsilon
	{ id.factor.s = id-const.s }
id-tail --> var-tail	{ id-tail.s = var-tail.s }
| call-tail	{ id-tail.s = call-tail.s }
var-tail --> **[**add-exp**]**	{ var-tail-s = add-exp.s }
| *epsilon*	{ var-tail.s = *epsilon* }
relop --> **<=**	{ relop.op = “lteq” }
| **<**	{ relop.op = “gt” }
| **>**	{ relop.op = “lt” }
| **>=**	{ relop.op = “gteq” }
| **=**	{ relop.op = “eq” }
| **/=**	{ relop.op = “neq” }
addop --> **+**	{ addop.op = “plus” }
| **-**	{ addop.op = “minus” }
| **or**	{ addop.op = “or” }
| **orelse**	{ addop.op = “orelse” }
multop --> *****	{ addop.op = “mult” }
| /	{ addop.op = “div” }
| **mod**	{ addop.op = “mod” }
| **and**	{ addop.op = “and” }
| **andthen**	{ addop.op = “andthen” }
uminus --> **-**	{ uminus.s = makeleaf(“uminus”) } (fix later!)