Program = (typespecifier ID (VarPart | FuncPart))\*

Funcpart = “(“ params-list ? “)” compound-stmt

Varpart = “[“ INTLITERAL “]” )? ( “=” initializer )? (“,” init\_decl)\* “;”

Variable-def = typespecifier init-decl-list “;”

init-decl-list = init-decl (“,” init-decl)\*

init\_decl = declarator ( “=” initializer)?

compound-stmt = “{“ variable-def\* stmt\* “}”

stmt = compound-stmt | if-stmt | while-stmt | for-stmt | return expr? “;” | ID “=” expr “;” | ID “[“ expr “]” “=” expr “;” | ID arglist “;”

if-stmt = if “(“ expr “)” stmt (else stmt) ?

while-stmt = while “(“ expr “)” stmt

for-stmt = for “(“ asgnexpr? “;” expr? “;” asgnexpr? “)” stmt

expr = and-expr (“||” and-expr)\*

and-expr = relational-expr ( “&&” relational-expr )\*

relational-expr = add-expr((“==”|”!=”|”<” | “<=” | “>” | “>=”) add-expr)?

add-expr = mult-expr ((“+”|”-“)mult-expr)\*

mult-expr = unary-expr ((“\*”|”/”)unary-expr)\*

unary-expr = primary-expr | (“+”|”-“|”!”) unary-expr

primary-expr = ID (arglist? | “[“ expr “]”) | “(“ expr “)” | INTLITERAL | BOOLLITERAL | FLOATLITERAL | STRINGLITERAL

asgnexpr = ID “=” expr

”

declarator = ID ( “[ INTLITERAL ”]” ) ?

initializer = expr | “(“ expr ( “,” expr ) \* “)”

typespecifier = void | int | bool | float

params-list = typespecifier declarator (“,” typespecifier declarator)\*

argList = “(“ (expr ( “,” expr )\*)? “)”

program, function-def, variable-def

params,params-list,parameter-decl

argList, args, arg