Grammar

Construct of Interest: Do While & If Else in C

Example:

```
#include<stdio.h>

int main()
{
    int i = 0;
    do
    {
      if (i == 7)
        i = i + 2;
      else
        i = i + 1;
    }while(i < 10);
    return 0;
}
```

Possible Tokens Created:

#include
< , >
stdio.h, stdlib.h, ...
main
return
<------------------ Specific to construct --------------->
{ , }
(,)
;
int/float/.. etc
Identifier [a-zA-Z]\w*
Integer literal [0-9]+
- , + , * , /
~
!
&& , ||
== , != ,< , <= , > , >=
=
do
while
break
continue

Grammar:

```
program -> declarationList
declarationList -> declarationList declaration | declaration
declaration -> varDeclaration | funDeclaration | recDeclaration
recDeclaration -> record ID { localDeclarations } ;
varDeclaration -> scopedTypeSpecifier varDeclList ;
                | varDeclaration, varDeclaration ;
scopedVarDeclaration -> scopedTypeSpecifier varDeclList;
                      | varDeclaration, varDeclaration ;
varDeclList -> varDeclList, varDeclInitialize | varDeclInitialize
varDeclInitialize -> varDeclId | varDeclId: simpleExpression
varDeclId -> ID | ID [ NUMCONST ]
scopedTypeSpecifier -> static typeSpecifier | typeSpecifier
typeSpecifier -> returnTypeSpecifier | RECTYPE
returnTypeSpecifier -> int | bool | char | void | double
                     | unsigned short int | unsigned long int
                     | unsigned long long int | signed short int
                     | signed long int | signed long long int
funDeclaration -> scopedTypeSpecifier ID ( params ) { statement }
                | ID ( params ) { statement }
params -> paramTypeList | lambda
paramTypeList -> typeSpecifier paramIdList
paramIdList -> paramIdList, paramId | paramId
paramId -> ID | ID []

statement -> expressionStmt
           | selectionStmt
           | compoundStmt
           | iterationStmt
           | breakStmt
           | continueStmt
           | returnStmt
           | statement;statement
           | declaration
           | goto <ID> ;
           | { statement }
           | ;
           | lambda

expressionStmt -> expression ; | ;
compoundStmt -> { localDeclarations statementList }
localDeclarations -> localDeclarations scopedVarDeclaration | lambda
statementList -> statementList statement | lambda
selectionStmt -> if ( condition ) statement
```

```
              | if ( condition ) statement else statement
iterationStmt -> do { statement ; } while( condition )
breakStmt -> break ;
continueStmt -> continue ;
returnStmt -> return ; | return expression ;

condition -> expression | expression logop expression
expression -> mutable = expression
            | mutable += expression
            | mutable -= expression
            | mutable *= expression
            | mutable /= expression
            | mutable ++
            | mutable --
            | simpleExpression
logop -> || | &&
simpleExpression -> simpleExpression OR andExpression | andExpression
andExpression -> andExpression and unaryRelExpression
               | unaryRelExpression
unaryRelExpression -> NOT unaryRelExpression | relExpression
relExpression -> sumExpression relop sumExpression | sumExpression
relop -> > | >= | < | <= | == | !=
sumExpression -> sumExpression sumop term | term
sumop -> + | -
term -> term mulop unaryExpression | unaryExpression
mulop -> * | /
unaryExpression -> unaryop unaryExpression | factor
unaryop -> -- | ++ | - | *
factor -> immutable | mutable
mutable -> ID | mutable [ expression ] | mutable.ID
immutable -> ( expression ) | constant | call
call -> ID ( args )
args -> argList | lambda
argList -> argList, expression | expression
constant -> NUMCONST | CHARCONST | true | false
```