**CPS 450**

**C Lite Lexical Analyzer**

**Due by Thursday, April 27, 2017 at 11:59pm via BlackBoard**

**One submission per group**

This group project consists of teams of 4-5 students. In this project, you must work as a team to develop a lexical analyzer to generate the tokens and lexemes for the C lite grammar described in ECFG (extended context-free grammar or BNF). The grammar is described in this document. Your lexical analyzer must provide the tokens and lexemes as output with an example given in this document. Your team may use flex and yacc or may use any other flex or yacc-like took (there are several tools out for Java, Python, etc) or do it all in a programing language of choice not using flex-like tools. All the groups will present their solutions to the rest of the class Friday, April 28, and must provide a short powerpoint or slide show presentation (5-6 minutes max each group) describing the solution taken to generate the tokens and lexemes. Your team must decide how the work is done and all of your names must be on the submission. You must submit all the source code you wrote (not what was generated from the tool if you used a tool like flex and yacc) as well as the powerpoint or other slide show format I can read of your presentation. Your group must decide on your team meetings/dynamics/tasks and it is expected that all of you will participate; part of the grade will involve an intergroup ranking of your performance in the team. Also, in your C Lite lexical analyzer, you must allow for comments; as you know comments are all ignored by the complier so are not part of the formal grammar but the code example shows the comments.

**ECFG grammar for C Lite**

|  |  |  |
| --- | --- | --- |
| *Program* | → | int main () { *Declarations* *Statements* } |
| *Declarations* | → | { *Declaration* } |
| *Declaration* | → | *Type* *Identifier* [ [ *Integer* ] ] { , *Identifier* [ [ *Integer* ] ] } ; |
| *Type* | → | int | bool | float | char |
| *Statements* | → | { *Statement* } |
| *Statement* | → | ; | *Block* | *Assignment* | *IfStatement* | *WhileStatement* |
| *Block* | → | { *Statements* } |
| *Assignment* | → | *Identifier* [ [ *Expression* ] ] = *Expression* |
| *IfStatement* | → | if ( *Expression* ) *Statment* [ else *Statement* ] |
| *WhileStatement* | → | while ( *Expression* ) *Statement* |
| *Expression* | → | *Conjunction* { || *Conjuction* } |
| *Conjunction* | → | *Equality* { && *Equality* } |
| *Equality* | → | *Relation* [ *EquOp* *Relation* ] |
| *EquOp* | → | == | != |
| *Relation* | → | *Addition* [ *RelOp* *Addition* ] |
| *RelOp* | → | < | <= | > | >= |
| *Addition* | → | *Term* { *AddOp* *Term* } |
| *AddOp* | → | + | - |
| *Term* | → | *Factor* { *MulOp* *Factor* } |
| *MulOp* | → | \* | / | % |
| *Factor* | → | [ *UnaryOp* ] *Primary* |
| *UnaryOp* | → | - | ! |
| *Primary* | → | *Identifier* [ [ *Expression* ] ] | *Literal*  | ( *Expression* ) | *Type* ( *Expression* ) |
| *Identifier* | → | *Letter* { *Letter* | *Digit* } |
| *Letter* | → | a | b | … | z | A | B | … | Z |
| *Digit* | → | 0 | 1 | … | 9 |
| *Literal* | → | *Integer* | *Boolean* | *Float* | *Char* |
| *Integer* | → | *Digit* { *Digit* } |
| *Boolean* | → | true | false |
| *Float* | → | *Integer* . *Integer* |
| *Char* | → | ' *ASCIIChar* ' |

**Sample code**

// /\* not a multiline comment \*/

/\*

\* Comment // not an inline comment

\* 1

\*/

int main()

{

int int0 = 6;

int int1 = 42;

float flt0 = 5.3;

if ( int0 <= int1)

{

// inline comment 2

flt0 = flt0 + int1; // inline comment 3 == not comparison

}

}

/\*\*\*\*\*\*\* COMMENT4 \*\*\*\*\*\*\*\*/

**Sample output**

Inline-comment-// /\* not a multiline comment \*/

Open-multiline-comment

Close-multiline-comment

Keyword-int

Identifier-main

Open-paren

Close-paren

Open-bracket

Keyword-int

Identifier-int0

Assignment

Integer-6

Semicolon

Keyword-int

Identifier-int1

Assignment

Integer-42

Semicolon

Keyword-float

Identifier-flt0

Assignment

Float-5.3

Semicolon

Keyword-if

Open-paren

Identifier-int0

Comparison-<=

Identifier-int1

Close-paren

Open-bracket

Inline-comment-// inline comment 2

Identifier-flt0

Assignment

Identifier-flt0

Operator-+

Identifier-int1

Semicolon

Inline-comment-// inline comment 3 == not comparison

Close-bracket

Close-bracket

Open-multiline-comment

Close-multiline-comment