**Requirements**

calculator ::= expr\* ’\0’

expr ::= exprAdd ( ‘<’ | ‘>’ ) exprAdd | exprAdd

exprAdd ::= exprAdd ( ’+’ | ’-’ ) exprPostfix | exprPostfix

exprPostfix ::= exprPostfix ‘[‘ expr ‘]’ | exprPostfix ‘(‘ ( expr ( ‘,’ expr )\* )? ‘)’ | factor

factor ::= INT | ID | ’(’ expr ’)’

INT ::= 0b[01]+ | 0x[0-9a-fA-F]+ | [0-9]+

ID ::= [a-z-A-Z\_][a-z-A-Z0-9\_]\*

SPATIU ::= [ \r\n\t] | [{] [^}]\* [}]

**Finally, the grammar will be like this:**

calculator ::= expr\* ’\0’

expr ::= **exprAdd ( ‘<’ | ‘>’ ) exprAdd** | exprAdd

exprAdd::=exprPostfix exprAdd1

exprAdd1::=(‘+’|’-’)exprPostfix exprAdd1

| eps

exprPostfix::=factor exprPostfixPrim

exprPostfixPrim::= ’[’expr’]’exprPostfixPrim

|’(’(expr(‘,’expr)\*)?’)’ exprPostfixPrim

| eps

factor ::= INT | ID | ’(’ expr ’)’

**example of expr function**

bool expr()

{

If consume(exprAdd)

If consume(‘<’)

If consume(exprAdd) return true;

Else if consume(‘>’)

If consume(exprAdd) return true;

If consume(exprAdd) return true;

return false;

}

We need the following functions:

Consume

expr

axprAdd

exprPostfix

exprAddPrim - this is a recurring function

exprPostfixPrim - this is a recurring function

factor

We also need to define the implicit atoms:

ADD::=+

SUB::=-

LESS::=<

GREATER::=>

LBRACKET::=[

RBRACKET::=]

LPAR::=(

RPAR::=)

COMMA::=,

