11712639-Project1-Report

Basic Requirement

> regular expression for INT, FLOAT, CHAR, ID and as well as their wrong format

```
decimal 0|[1-9][0-9]*
heximal 0[xX](0|[1-9a-fA-F][0-9a-fA-F]*)
WrongHex 0[xX]([0-9a-fA-F]*[e-zE-Z][0-9a-fA-F]*|0[0-9a-fA-F]*)

INT {decimal}|{heximal}
FLOAT [0-9]+\.[0-9]+
CHAR \'.\'|\'\\x0\'|\'\\x[1-9a-fA-F][0-9a-fA-F]?\'
ID [_a-zA-Z][0-9_a-zA-Z]*

WrongCHAR \'\\x[0-9a-zA-Z]*\'
WrongID [0-9]+{ID}
```

> build parse tree

create leaf nodes when lexical analysis realizes a token.

build tree when syntax analysis reduce a production.

```
//example (functions' details are in tree.c file.)
{INT} {
    yylval.node = createTerminal("INT", yytext, yylineno, 1);
    return INT;
}
VarDec: VarDec LB INT RB { $$=createNonTerminal("VarDec"); buildTree($$, 4, $1, $2,
$3, $4); }
```

> handle some lexical error and syntax error

in lexical analysis, add unknown token or wrong token such as wrong ID to match unknown or wrong lexme.

in syntax analysis, error recover in potential productions.

Bonus features

> operations: %, ++, --, +=. -=, *=, /=, %=

add tokens and define associations, then add production in `Exp`.

```
//take mod operation as example
//in lex.l file:
"%" {
    yylval.node = createTerminal("MOD", "", yylineno, 0);
    return MOD;
}
//in syntax.y file:
%token <node> MOD
%left MUL DIV MOD
Exp:Exp MOD Exp { $$=createNonTerminal("Exp"); buildTree($$, 3, $1, $2, $3); }
```

> for statement

and `FOR` token and add production in `Stmt`.

```
//in lex.l file:
"for" {
    yylval.node = createTerminal("FOR", "", yylineno, 0);
    return FOR;
}
//in syntax.y file:
%token <node> FOR
Stme: FOR LP Def Exp SEMI Exp RP Stmt { $$=createNonTerminal("Stmt"); buildTree($$, 8,
$1, $2, $3, $4, $5, $6, $7, $8); }
```

> comments(single line and multiple line)

only design the match rules in lex.l file, do not participate in syntax analysis.

Note that yylineno should increase when match multiple line comments.

```
commentSingle \/\/.*\n
commentMulMiddle .*\n
commentMul \/\*{commentMulMiddle}*.*\*\/
{commentSingle} {
    printf("commentSingle\n");
    yylineno++;
}
{commentMul} {
    char *ptr = yytext;
    int i = 0;
    while (i < yyleng){
        if(ptr[i] == '\n'){
            yylineno++;
        }
        i++;
    }
    printf("commentMul\n");
}
```

> file include

add `INCLUDE` token and add related production.

```
//in lex.l file:
INCLUDE #include\ (<.*\..*>|\".*\..*\")
```

```
{INCLUDE} {
    yylval.node = createTerminal("INCLUDE", "", yylineno, 0);
    return INCLUDE;
}
//in syntax.y file:
%token <node> INCLUDE
Program:
      IncludeList ExtDefList {root=createNonTerminal("Program"); buildTree(root, 2,
$1, $2);}
    ;
IncludeList:
      INCLUDE IncludeList { $$=createNonTerminal("IncludeList"); buildTree($$, 2, $1,
$2); }
    | { $$=createNonTerminal("IncludeList"); buildTree($$, 0); }
    ;
```