编译原理 **Lab 1** 实验报告

董杨静 171860558

实现的功能

所有的必做、选做内容，包括：

生成语法树

我定义了树结构

```
typedef struct __Tree{
  int stype /*词法单元类型*/, show /*若匹配到空串，则不显示*/;
  struct __Tree *ch[MAXCH] /*子节点*/;
  int lineno; /*词法单元所在行号*/
  union{ /*属性值*/
    unsigned int int_val;
    float float_val;
  };
}Tree;
```

并且每一个 Token 和语法单元的属性值都是一个指向树节点的指针。
然后用 flex 和 bison 两个工具生成语法书并打印即可。

检测词法错误

词法分析器识别以下两种词法错误

- 未识别的字符，如 `~`, `"`, `#` 等。
- 未结束的多行注释，比如以 `/* AAAA` 结尾的输入文件。

词法分析器不检查其他词法错误，如非法的浮点数(臭名昭著的 `8.e`)，或者 `0xGG` 等字符串，而是把它们尽可能地拆分成合法的Token，并交给语法分析器。

检测语法错误 **&** 错误恢复

我按照讲义要求，实现了错误恢复。但是因为存在 前面的错误会导致后面其他错误、稀奇古怪的错误等等情况，所以可能不会打印所有错误。

测试

为了保证分析器的正确性，我做了大量的测试。测试方法是生成各种各样的输入文件，并将输出结果和其他同学比对。（我们没有直接交换源文件）

生成随机输入

除了手动构造的测试输入之外，通过使用 python 的 nltk 库，我还生成了符合 C-- 词法和语法的随机程序。python 脚本大致如下，参考自 https://stackoverflow.com/questions/603687/how-do-i-generate-sentences-from-a-formal-grammar/603749#3292027

```
from nltk import CFG, ChartParser
from random import choice

def produce(grammar, symbol):
    words = []
    productions = grammar.productions(lhs = symbol)
    production = choice(productions)
    for sym in production.rhs():
        if isinstance(sym, str):
            words.append(sym)
        else:
            words.extend(produce(grammar, sym))
    return words

grammar = CFG.fromstring('''
Program -> ExtDef ExtDef ExtDef ExtDefList
ExtDefList -> ExtDef ExtDefList |
ExtDef -> Specifier ExtDecList SEMI | Specifier SEMI | Specifier FunDec CompSt | Specifi
ExtDecList -> VarDec | VarDec COMMA ExtDecList
Specifier -> TYPE | TYPE | StructSpecifier
StructSpecifier -> STRUCT OptTag LC DefList RC | STRUCT Tag
OptTag -> ID |
Tag -> ID
VarDec -> ID | ID | VarDec LB INT RB
FunDec -> ID LP VarList RP | ID LP RP
VarList -> ParamDec COMMA VarList | ParamDec
ParamDec -> Specifier VarDec
CompSt -> LC DefList Stmt StmtList RC
```

```
StmtList -> Stmt StmtList | Stmt StmtList | Stmt StmtList | Stmt StmtList |
Stmt -> Exp SEMI | Exp SEMI | Exp SEMI | Exp SEMI | Exp SEMI | CompSt | RETURN Exp SEMI
DefList -> Def DefList |
Def -> Specifier DecList SEMI
DecList -> Dec | Dec COMMA DecList
Dec -> VarDec | VarDec ASSIGNOP Exp
Exp -> Exp2 | Exp2 | Exp2 | Exp2 | ID | INT | FLOAT
Exp2 -> Exp ASSIGNOP Exp | Exp AND Exp | Exp OR Exp | Exp RELOP Exp | Exp PLUS Exp | Exp
Args -> Exp COMMA Args | Exp
INT -> '123'
FLOAT -> '1.23'
ID -> 'id'
SEMI -> ';'
COMMA -> ','
ASSIGNOP -> '='
RELOP -> '>' | '<' | '>=' | '<=' | '==' | '!='
PLUS -> '+'
MINUS -> '-'
STAR -> '*'
DIV -> '/'
AND -> '&&'
OR -> '||'
DOT -> '.'
NOT -> '!'
TYPE -> 'int' | 'float'
LP -> '('
RP -> ')'
LB -> '['
RB -> ']'
LC -> '{'
RC -> '}'
STRUCT -> 'struct'
RETURN -> 'return'
IF -> 'if'
ELSE -> 'else'
WHILE -> 'while'
''')
gr = ChartParser(grammar).grammar();
print(' '.join(produce(gr, gr.start())));
```

致谢

感谢 谢乃容、鄢振宇、张天昀 等同学分享的测试数据，并且提供了他们的分析器对测试数据的输出。