编译原理第一次实验报告

161240005 陈勇虎

1. 完成的功能:
   (1) 能够查出 C 源代码中的词法错误
   (2) 能够查出 C-源代码中的语法错误
   (3) 没有词法和语法错误的情况下，打印语法树
   (4) 词法分析中为了测试需求，目前可以支持：
      1) 二进制,八进制,十进制,十六进制整数
      2) 浮点数，包括指数型浮点数
   (5) 语法分析中，使用了 error verbose 报错，和一些简单错误的"人工提示"，例如下图中，error verbose 会指出错误的地方，此外还会另外输出可能的原因,由于添加了对嵌套注释的检查，所以还会报出嵌套循环的错误.

```
[#13#cyh loves wmy!@~/Compiler/Compiler/Code]$ make && make test
flex -o ./lex.yy.c ./lexical.l
bison -o ./syntax.tab.c -d -v ./syntax.y
gcc -c ./syntax.tab.c -o ./syntax.tab.o
gcc -o parser ./syntax.tab.o ./grammertree.o -lfl -ly
./parser ../Test/test12.cmm
Error Type B at line 4: nested comment error!
Error type B at Line 4:syntax error, unexpected ID
```

   (6) 为了方便看出语法分析树的结构等，除了完成讲义的基本内容外，还改变了不同结点的颜色，部分打印情况如下，颜色设定情况可以参考 Code/grammertree.h 文件

```
------------------Dec (5)
--------------------VarDec (5)
----------------------ID: x
----------------SEMI
-----------StmtList (6)
-------------Stmt (6)
----------------Exp (6)
------------------Exp (6)
--------------------Exp (6)
----------------------ID: y
--------------------DOT
--------------------ID: image
------------------ASSIGNOP
------------------Exp (6)
--------------------FLOAT: 3.500000
```

2. 实现方法:
   (1) 借助 flex，并自定义正则表达式完成词法分析
   (2) 借助 bison，并根据附录中的提示完成基本的语法分析
   (3) 进行了简单的错误处理

(4) 词法分析中建立结点，语法分析建立分析结点，并维护结点间的关系，最后打印语法树

(5) 信息的位置，报错等使用了系统提供的方法，一些错误消息和提示信息由自定义设计。

3. 数据结构

实验需要建立一个多叉树，因此在每一个结点需要保留结点的信息和与别的结点的直接关系，故建立一下结构体已维护，依次为:该结点的行号(如果有的话)，结点名，结点的值(数值型的结点)，结点的类型(数值，逻辑符等)，结点颜色(为了方便打印调试)，结点的父结点，结点的子结点(MAX_CHILD_NUM 为自定义的量，暂时为 10，因为子结点并不会很多)。

```
typedef struct Node{
  int lineno;
  char name[32];
  char value[32];
  int child_num;
  Node_Type type;
  Node_Color color;
  struct Node *parent;
  struct Node *child[MAX_CHILD_NUM];
}TreeNode;
```

多叉树的建立和打印方法见 Code/gammertree.c 文件。

4. 编译运行方法

待分析文本均位于 Test 文件夹下面，讲义中所有样例，包括选做样例也在其中，若需要分析名为 testaa.cmm 的文件，将 Code/Makefile 文件内容 test 伪目标内容修改为的 "./parser ../Test/testaa.cmm"，并在 Code 目录下 make && make test 即可，若分析的文件没有语法和词法错误，将会打印语法分析树.

5. 实验总结

借助 flex 和 bison 可以很快完成对一段代码的分析，并通过自定义的数据结构完成对代码分析树的建立。