编译原理实验一报告

在实验一中，我们需要借助flex和bison完成一个简易的C语言词法分析器和语法分析器。

词法分析器的实现

flex是GNU下的一个词法分析器构建工具，这个部分难度并不高，只需要为每个词法单元写出其对应的正则表达式即可。 但这个部分也有一些注意点。

- 和C文法附录的顺序相反，ID应该排在保留字之后，这样可以确保保留字能够优先匹配。同理，注释的匹配也要先于DIV。
- 为了能正确检测出A类错误，当不存在规则能够匹配时，需要报告错误，这可以通过在规则末尾添加一个通配符的规则来实现。
- C语言的空白符并没有意义，为了防止报错，flex规则中要添加对空白符的匹配。

定义完了这些规则后，为了和bison配合，我们需要在匹配到每个规则后返回对应的TOKEN，这些TOKEN稍后将在bison中被定义， 当然例如注释的规则就无需返回值， bison会等待下一个有效的返回值。为了构建AST，我们对yylval进行初始化为树的一个节点， 在后续的bison中进行连接。

语法分析器的实现

bison是GNU下的一个语法分析器构建工具，语法分析相较词法分析难得多。所幸文法附录中已经给出了所有的模式，我们只需要把它 原封不动地搬到bison中即可实现语法分析器的基本功能。

AST的构建

对于一个规则，例如 $A:B\quad C$ 来说，A是子树的根，BC是A的子节点。由于A是规约出来的新节点，需要对其进行初始化， 然后将B和C通过brother指针连接起来，作为A的子节点即可。注意点主要有三：

- 空推导无需在AST中显示，遇到空推导时可把根节点置为NULL，方便构建时判断。
- $$,$1等并不是C语言语法，bison构建时会对它们进行替换，但如果在宏或其他地方中使用了它们会导致语法错误，当然我们在实现的 时候为了避免这一问题选择取了地址。
- 行号是应该使用栈底词素的行号，同时要加入"%option yylineno"。

错误恢复

错误恢复是本次实验中最难的部分，bison在灵活性上的欠缺让它变得困难。这个部分没有唯一答案，不过我们总是希望错误能够 遇到SEMI,RP,RC等符号时恢复。 我们的做法是将错误尽量放在相对高层，大约在一条语句的等级进行恢复。

同时，需要尽量避免规约/规约冲突与偏移/规约冲突，这两者往往是因为error规则定义过于模糊，例如对于规则

$$Def:Specifier\quad DecList\quad SEMI$$

来说，$Def:Specifier\quad error\quad SEMI$效果较为理想，而$Def:error\quad SEMI$规则定义较模糊，产生了很多规约/规约冲突。经过对一些样例的调试，我们选择在Def、Stmt、CompSt、FunDec、VarDec、ExtDef处进行错误恢复，虽然 不是很完美，但是已经通过了所有测试。

如果要我设计一门语言的语法分析器，我肯定不会使用bison😂

如何运行

已经按照oj的要求打包好了，使用make便可编译运行。