**Lab1实验报告**

**201240069 曹语桐 201240060 林彦葶**

**一、如何编译运行程序**

在Code 目录下执行：

make clean

make

生成可执行文件parser。

假设测试文件为./test.c，运行./parser test.c就可以得到输出结果。

PS：我们设置了简洁打印模式与详细打印模式，在main.c中令sim=0则输出详细词法信息，令bisonsim=0则输出详细语法信息，令errorsim=0则输出详细错误恢复信息，将这些变量置为1则输出信息是实验要求的标准输出。

**二、实现的主要功能**

1.

首先，我们实现了讲义上所要求的全部必做和选做内容：查出C--语言测试文件中的词法错误和语法错误并输出报错信息，如果没有词法和语法错误，输出程序的语法树信息。其中：

lexical.l负责实现词法分析功能，出现未定义符号时报错，输出错误类型Error Type A及对应行号。出现单行注释//时，利用flex的库函数input()将该行所有符号丢弃。出现/\*时，将丢弃从/\*开始到识别到的第一个\*/之内的所有字符。没有错误时，返回词法单元给syntax.y。

syntax.y 负责实现语法分析功能，出现语法错误时，输出Error Type B及对应语法单元的第一行行号；没有错误时，添加每个终结符和非终结符信息到语法树上，最终输出语法树信息。

tree.h和tree.c负责实现语法树，输出没有语法和词法错误时程序的语法树结构，其中tree.h 定义NODE结构体信息，声明添加节点和打印语法树的函数。tree.c定义添加终结符到语法树的函数、添加非终结符到语法树的函数以及打印语法树的函数。

2.

在此基础上，设计错误恢复方案，使语法错误发生时，程序能够丢弃尽可能少的词法单元，尽快同步成功。

具体的错误恢复方案是：

a.主要在语句级别上进行错误恢复，即在ExtDef，Stmt，Def上进行错误恢复。

ExtDef-> error SEMI

*//出错但结尾SEMI还在*

| error ExtDef

*//SEMI缺失，把这条ExtDef全部弹出栈（此时此条ExtDef的错误已被输出），然后开始分析下一条ExtDef，因为ExtDef里嵌套着error错误恢复，所以下一条ExtDef的错误可以被正确的分析出来*

| error CompSt

| Specifier error SEMI

| Specifier error CompSt

*//以上情况是ExtDef的结构还在，中间一些模块出现错误*

Stmt->error SEMI

| IF LP error RP Stmt %prec LOWER\_THAN\_ELSE

| IF LP error RP Stmt ELSE Stmt

| WHILE LP error RP Stmt

*//以上的错误恢复产生式是我们假设了一些常见的错误，即语句的结构还在，但其中出现了一些小错误，这种情况下，SEMI没有缺失。*

| Exp error

| RETURN Exp error

| error Stmt

*//以上的错误恢复产生式是针对SEMI缺失的情况，可以使得下一条Stmt的开头的文法符号不被吞掉，当Stmt能正确识别到Exp时，后面错误部分被丢掉，填入error，下一条Stmt仍然可以正确分析；当Stmt能正确识别到RETURN Exp时同理；当Stmt无法识别出Exp或RETURN Exp时，把这条Stmt全部弹出栈（此时此条Stmt的错误已被输出），然后开始分析下一条Stmt，因为Stmt里嵌套着error错误恢复，所以下一条Stmt的错误可以被正确的分析出来*

Def->Specifier error SEMI

*//以上的错误恢复产生式针对DecList发生错误的情况，SEMI还在*

|Specifier DecList error

|error Def

*//以上两种错误恢复产生式针对SEMI缺失的情况，思路与Stmt的思路一致，当能正确识别到Specifier DecList时，把后面的错误部分弹出栈，放入error；当无法识别到Specifier DecList时把这条Def全部弹出栈（此时此条Def的错误已被输出），然后开始分析下一条Def，因为Def里嵌套着error错误恢复，所以下一条Def的错误可以被正确的分析出来*

b.Exp级别上的错误恢复

Exp-> LP error RP

| LP Exp error

| MINUS error %prec NEG

| NOT error

| ID LP error RP