编译原理第一次实验报告

161220179 周科， 161220115 汤佳铭

实验任务

使用词法分析工具GNU Flex和语法分析工具GNU Bison编写一个程序对使用C--语言书写的源代码进行词法分析和语法分析

实验思路

Flex词法分析部分

词法分析的主要任务是将输入文件中的字符流组织成词法单元流

大部分词法单元按照网站上提供的C--文法讲义即可定义,剩下几个特殊的需要自己写

```
SPACE [ \t\r]* //对空语法单元抓取，但分析动作为空
INT [0-9][0-9]*//整数语法单元
FLOAT ([0-9]+\.[0-9]+)|((([0-9]*\.[0-9]+)|([0-9]+\.))[Ee][+-]?[0-9]+) //浮点数语法单元（包含对选做部分指数型浮点数进行处理
LINE \n //对换行符进行抓取以使用yylineno
```

抓取之后执行语法树构造节点操作，具体见语法树部分

Bison构造规则部分

主要任务是读入词法单元流、判断输入 程序是否匹配程序设计语言的语法规范，并在匹配规范的情况下构建起输入程序的静态结构。

Bison框架构造

> *暂时不考虑具体语义动作*

Token直接根据Flex部分可以编写

Type和Rule部分可根据C--文法讲义部分编写

我们还需要对二义性和冲突进行处理

通过%left和%right对终结符号的结合性进行定义

还有解决悬空else问题

```
/*combine*/
%right ASSIGNOP
%left OR
%left AND
%left RELOP
%left PLUS MINUS
%left STAR DIV
%right NOT
%left LP RP LB RB DOT

%precedence ELSE
```

Bison具体语义动作（构造语法树）

此时需要给token和type指定具体的数据结构

```
%union{
struct node* o;
double d;
}
```

语法多叉树数据结构:

```
struct node {
   char *nodename;//节点名称
   int lineno;//节点所在行号

   int typeno;//节点类型
   // id为1，type为2，int为3，float为4，其他终结符为5, 空的语法单元为0;
   char *special_val;
   struct node* cld;//子节点
   struct node* bro;//兄弟节点
};
```

语法树的构建与打印

语法树构建函数主要有两个函数，功能分别为叶子节点（终结符）的构建与非叶子节点的构建。

叶子节点构建函数leafcreate，遇到打印时需要特殊输出的终结符（如int等），会给其typeno赋对应值，并存下其yytext（char*类型）。把这个写进节点构造函数里，可以使得我们的lexical.l中终结符的语义动作非常简洁，只需要调用一次函数，传回名字和行数即可。

非叶子节点构建函数是用来处理产生式的。其参数有一个num，这个num表明该产生式右侧共有几个语法单元。在c—中，由于产生式右侧最多有7个语法单元，故而设置了全局变量：7个指向树节点的数组指针。数这么做的好处在于避免了该函数调用非固定个数的参数。我们重新定义了yyvalue，使其适应我们的数据结构，在写语义动作的时候，只需使用1、2等将产生式右侧的语法单元挨个存入数组，随后调用该函数生成节点即可。这样做下来，我们的syntax.y的语义动作也变得异常清晰简洁。

语法树的打印由于需要开头缩进，其缩进空格数是跟节点高度线性相关的。故而引入了height参数，递归子节点时，树高+1，递归兄弟节点则高度不变。

Bison错误恢复

重写yyerror函数，打印符合实验要求的内容。

在语法规则中指定error符号的位置

```
| error SEMI {nodelist[0] = $1;nodelist[1] =
$2;$$=nodecreate("error",2);yyclearin;yyerrok;}
  | error RC {nodelist[0] = $1;nodelist[1] =
$2;$$=nodecreate("error",2);yyclearin;yyerrok;}
  ;
```

经过多次测试，在stmt和def处添加效果最好，在exp出添加会出现重复报错现象。

实验中遇到的问题

yylineno始终固定为1(Flex没有对\n进行单独抓取)

如何判断error符号的放置。

打印语法错误的时候不知道怎么分析missing character