编译原理第一次实验

实验报告

171860691 王辛有 171860005 宋斯涵

一、 实现功能

1） **识别词法错误（错误类型 A）**：出现 C—词法中未定义的字符以及任何不符合 C—此法单元定义的字符。
2） **识别语法错误（错误类型 B）。**
3） **识别“//”和“/*…*/”形式的注释。**

二、 重要的数据结构、函数和变量说明

语法树节点的数据结构

```
typedef struct TREE
{
    char name[32];//表示终结符或非终结符的名称
    char text[32];//若是终结符，则保存词素；若是非终结符则是空。
    int childnum;//该节点的子节点的数量
    struct TREE *child[20];//存储子节点的数组
    int line;//第一个子节点所在的行号（若是终结符则就是它本身的行号）
    int flag;//终结符则 flag 置 0，非终结符置 1.
}Tree;
```

创建语法树节点的函数

```
Tree* create(char *name,char *s,int flag);
```

（第一个参数是节点的名称，第二个参数是节点的词素内容（仅对终结符有效），第三个参数是判断该节点是否为终结符，若是则置 1，反之置 0）

添加子节点的函数

```
void add(int chilenum,Tree* node,...);
```

（第一个参数是子节点的个数，第二个参数是将要插入子节点的节点，后面的可变参数是即将要插入的子节点）

打印语法树的函数

```
void Treeprint(Tree* root,int count);
```

（第一个参数是根节点，第二个参数是用来在输出时记录缩进的空格数的）

全局变量

int mistake=0;//用于统计程序中错误的数目。

三、 实现方法

1） 识别词法错误

首先我们先写出了相应的正则表达式，对于所有需要匹配的模式都进行了编写。然后对于剩下的无法识别的、未定义的字符，统一采用“.”进行匹配。如果进入了“.”模式，就意味着出现了词法错误，也就是错误类型 A，此时要产生相应的输出

2） 识别语法错误和错误恢复

附录 A 中的所有产生式就是 C--语言的文法定义，每当词法分析程序从 yylex()得到了一个词法单元，如果当前状态并没有针对这个词法单元的动作，此时就发生了语法错误，为了能够检查出文件中的所有错误，需要利用保留字 error 来进行错误恢复。本质上，带有 error 的产生式就是用来匹配不正确的词法单元流，并输出相应的错误信息，实现再同步，使得后面的语法分析能够正常进行。

在实现中，我们的想法是尽量把 error 放在终结符的前面，特别是”,”，”{“，”,}”，”(“，”)”的前面。但是考虑到这样并不能覆盖全部的语法错误，我们也会将 error 放到一些词法单元的前面，例如，放到终结符 ELSE 的前面：

```
| IF error ELSE Stmt {
    mistake++;
    printf("Error Type B at Line %d: syntax error.\n", @2.first_line);
}
```

3） 识别“//”和“/*…*/”形式的注释

我们对这两种注释的分析都在 lexical.l 中完成。

- 对于“//”型的注释：当 flex 识别到连续的两个’/’字符时，便进入一段代码。在这段代码中，我们一直从输入流中读取字符，直到遇到换行符停止。这些字符仅仅被读入，并无其他动作，也没有向语法分析中 return。故达到了注释的效果。
- 对于“/*…*/”型的注释：处理方法与“//”型注释类似。当 flex 程序连续读入’/’和’*’字符时，便进入一段代码。在这段代码中会一直从输入流读入字符，直到遇到连续的两个字符’*’和’/’为止。需要注意的一点时，如果一直读到文件结束符都没有读入到*/，那么就会产生语法错误：缺少与/*相匹配的*/。

4） 语法树的生成与输出

在词法分析中，对于识别到的每个词法单元，我们都将创建节点。例如：

```
0|[1-9]{digit}*  {yylval.type_tree=create("INT",yytext,0);return INT;}//INT
```

（注：yylval 的类型已被定义为联合体，type_tree 类型实际为 struct TREE*）

在语法分析中，当发生归约动作时，以 A->B C D 这个产生式为例。先创建产生式左端的节点，即 A 节点。由于在该归约动作发生时，B，C，D 节点已经完成了创建，所以接下来就调用 add 函数，将 B，

C，D 三个节点依次插入到 A 节点下，作为 A 节点的子节点。
若选择了产生空串的表达式，如 A->ε，那么在创建 A 节点时会将 A 节点设为 NULL。从而在打印语法树时，A 节点不会被显式打印出来。
最后如果整个过程中没有错误产生，那么会调用 Treeprint 函数先序遍历语法树。在遍历的过程中，会先检查当前节点是否为一个词法单元，这会通过节点的 flag 成员来进行检查。（若为词法单元则 flag 为 0，反之则为 1）如果是一个词法单元，则不用输出行号，反之需要输出行号。对于词法单元节点，还要检查该节点的名称。如果是 INT，FLOAT，TYPE 节点则还需要输出该词法单元的词素。

四、 编译指令

1. bison -d -v syntax.y
2. flex lexical.l
3. gcc main.c syntax.tab.c tree.c -lfl -ly -o parser