实验报告

151160055 吴宇昊 地理与海洋科学学院

程序如何被编译：

bison -d syntax.y
flex lexical.l
gcc main.c syntax.tab.c -lfl -o parser
将 makefile 里的上述语句粘贴至 linux 终端中，使用./parser example1.txt 即可执行

程序功能：

在 flex 中构造了一个结构体 sign

```
struct sign{
    int line;
    int column;
    int flag;
    int token;
    char type[32];
    char text[32];
    struct sign* root;
    struct sign* leaf;
    struct sign* brother;
};
```

line 表示行号；column 表示列号；flag 表示该 toekn 是否含有多种含义或者多种信息,如 type 有 int，float 两种类型,int 类型数据除了本身的类型信息还有它本身的数据信息等等;token 表示该表达式是否为 token;type 存储类型信息;text 存储 flag 中的特定类型的指针;root 保存根节点的指针;leaf 保存第一个子节点的信息;brother 保存临近兄弟节点的指针.

在 syntax.y 语法分析的代码中,%union{struct sign* sign;}将 yylval 的类型改变为 sign*类型,即 flex 进行词法分析得到的 token 的属性为 sign*类型,可以传递到 bison 中使用,同时在非终结符后加上<sign>使得 bison 中的非终结符属性也为 sign*类型.

struct sign* create_sign(char type_[],int line_)函数为 token 创造其对应的结构体.

借助 flex,正则表达式,结构体的创建,完成词法分析,并将 token 的位置,类型,数据信息通过 lex.yy.c 传递到了 bison 中.

bison 引用 lex.yy.c 得到 token 进行语法分析.

```
int brother(struct sign* brother1,struct sign* brother2);
int links(struct sign* root,struct sign* leaf);
int print(struct sign *root);
```

声明了三个函数

brother 用于兄弟节点之间根结点的传递以及将弟节点传给兄节点.

links 用于根节点与第一个子节点之间的传递，将根节点赋予子节点的根节点，子节点赋予根节点的子节点,同时将根节点的 token 赋值为 0,即进行了归约动作的根节点必然为非终结符.

print 用于绘制语法树.

使用附录中的文法,在每个归约动作为非终结符创造 sign 类型节点,进行 links 与 brother 动作,在空串表达式里只进行 create_sign 动作.

在语法分析的最顶层 Program 中会执行 print 函数,将 Program 的节点作为根节点传递到 print 函数中.print 函数绘制语法树的过程中,首先进行错误检测:有两个参数-flex 中的全局变量 flexerror 与 bison 中的全局变量 error,两个参数初始值为 0,分别在 flex 和 bison 的报错中才会进行数值的改变,if(!flexerror&&!error)保证了在没有出现语法错误和词法错误的情况下才会绘制语法树.根据 LALR 自底向上的分析方法,语法树的绘制遵从深度优先的原理,优先进行深度的搜索,即该节点若有子节点,优先进行跳转,其次进行兄弟节点的跳转,若一个节点没有子节点也没有兄弟节点,则进行根节点的回溯,若回溯至 root 节点则代表树遍历完毕循环结束,完成树的绘制,printf 的格式根据实验要求,在函数中声明了一个 int 型 j 变量，用于记录深度,每往下一层 j++,每回溯一次 j--,同时用于树的” “空格代表深度的绘制.

完成了基础的词法分析和语法分析,附加内容可能会陆续更新.