# 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.