lab1 词法分析及语法分析　　121220026 谷雨

一、实验要求

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（C语言的文法参见附录A），并打印分析结果。

二、实现功能

基本功能以及八进制数，十六进制数，科学计数法表示的浮点数的识别；注释符号//及/**/的识别。

三、编译步骤

bison －d syntax.y

flex lexical.l

gcc main.c syntax.tab.c node.c -lfl -ly -o parser

四、实现方法

将yylval的属性定义为node*，在词法分析阶段中，ID,INT,FLOAT,OCT,HEX几种类型需要对其val属性进行赋值，由于其val类型各不相同，有字符数组，整型，浮点型3种，故在结点的结构体中定义一个枚举型的val，匹配到不同词法单元时根据具体情况选择具体的类型。在语法分析阶段中完成各结点及由结点组成的树的建立，初步完成实验时，包括所有终结符号结点在内的所有的结点都是在语法分析阶段完成的，然而在测试用例过程中发现同一个产生式中若有两个ID则会出现问题，因为后一个从yylex中得到的ID的属性值会覆盖前一个，导致两个ID的属性值相同，故对程序进行了修改，将ID类型的结点的构造过程放到了词法分析阶段（其实最好的做法是将所有的终结符号结点构造都放到词法分析阶段，由于时间比较紧迫，为了避免一些可能出现的麻烦，在提交前暂不做修改）。

每个结点中都包含一个用于存储指向其其子节点的指针的数组，先序遍历

输出采用递归即可。

构建结点函数：

node* construct(char [], int, int);

添加子节点函数：

void add_child(node **, node *);

打印树函数（采用递归实现）：

void print_tree(node *, int);

附加功能的实现

八进制正则表达式：(0[0-7]+)

十六进制正则表达式：(0[Xx][1-9a-fA-F][1-9a-fA-F]*)|0x0

带科学计数法的浮点数正则表达式：([0-9]+\.[0-9]+)|([0-9]*\.[0-9]+[Ee][+-]?[0-9]+)

对于8进制与16进制数，光能将其识别为正确的词法单元还是不够的，我们还需要得到其具体的值，即将提取出的字符串转换为10进制的int型整数，不同于字符串转化为10进制数直接使用函数atoi()即可，8进制与16进制数的转换需要自己来实现，下面给出16进制的转换方式（8进制同理）。

```
int i;
int value = 0;
for(i = 2; yytext[i] != '\0'; i ++){
    value = value * 16 + (yytext[i] - '0');
}
```

value值即对应的10进制i整型值。

五、结点数据结构

```
typedef struct Node{
```

```
        char name[20];
        int lineno;
        int flag;//1 stands for INT, 2 stands for FLOAT, 3 stands for ID, 4 stands for TYPE,
//0 stands for others
        union{
            int type_int;
            float type_float;
            char type_ID[20];
                char type_Type[20];
        }val;
        int numOfChildren;
        struct Node* children[10];
}node;
```

其中name即结点的类型名，lineno为所在行号，flag用以标记输出该结点时是否需要输出其属性值（只有ID,INT,FLOAT有属性值，OCT与HEX在词法分析阶段均return INT）。

val即属性值域，由于不同类型结点的属性值域的类型是不同的，故采用枚举结构。numOfChildren， 顾名思义，就是该结点子节点的个数，而子节点个数为0作为print_tree递归终止的条件。

最后维护一个存储指向各个子节点的指针的数组，用以访问各子节点进行先序遍历。