编译原理lab1实验报告

姓名：张玄逸	学号：201220194
日期：2022.10.8	邮箱：1822771416@qq.com

实验内容

- 实现了词法分析和语法分析
- 错误判断，语法树输出

测试方法

代码可使用Makefile进行编译，得到可执行文件parser。

在文件根目录下输入 ./parser [文件名] 即可执行，如 ./parser Code/test1.cmm

实验思路

词法分析

首先需要补充的是INT，FLOAT，ID三个token的正则表达式，另外还需要添加换行，空格，注释符号的处理。最后使用 . 匹配所有的不合法字符，就可以实现词法分析了。

```
number [0-9]
hexnumbers [0-9a-fA-F]
letter [_a-zA-Z]
oct 0[0-7]*
dec 0|([1-9]{number}*)
hex 0[xX]{hexnumbers}+
decimal ({number}*\.{number}+)|({number}+\.{number}*)

INT {oct}|{dec}|{hex}
FLOAT {decimal}([eE][+-]?{number}+)?
ID {letter}({letter}|{number})*
```

语法分析

首先是语法树的数据结构。用Node来表示树的节点，其中叶节点就是所有的词法节点。

因为在打印语法树时，语法节点需要输出行号，所以用变量if_token记录是否是词法节点（叶节点）。

```
typedef union{
```

```
    int int_val;
    double float_val;
    char char_val[108];
}Valtype;//节点的值，分别对应int,float,id，其它类型的值(yytext)也都存在char_val中

struct node{
    int line_num;
    char name[30];//名字，即语法分析时获得的类型名称
    Valtype val;//值
    int if_token;//用于记录是否是词法节点
    struct node* child;//第一个孩子节点
    struct node* next;//下一个兄弟节点
};
typedef struct node Node;
#define YYSTYPE Node*
```

这样就建立了一棵语法树，每个内部节点都是一个语法单元，由一到多个语法或词法节点构成子树。通过链式前向星的方法，能够访问所有的子树。

在添加节点时，词法和语法节点的动作不同。词法节点需要根据传入的名字，用不同的类型来赋值；语法节点需要将其与自己的子树连接。

```
{INT} {yylval=new_leaf(yylineno,"INT");
    return INT;}//词法单元新建为新的叶子节点
...
Args : Exp COMMA Args {$$=new_node(@1.first_line,"Args",3,$1,$2,$3);}
| Exp {$$=new_node(@1.first_line,"Args",1,$1);}
;//语法单元是对应表达式元素的结合
```

一旦词法或语法分析出现错误，if_over被置为0，最后就不会输出语法树。

总结与反思

- 词法单元要保持简洁
    - 我曾经在词法分析中添加了“错误数字”的正则判定，也就是在正常的INT和FLOAT之后直接把所有的数字，小数点，正负号的组合字符串判定为不合法数字。但这样会将某些正常的单元，如"---1""e1.e1"判定为错误。
    - 但如果细化“错误数字”的规则，很可能造成缺漏。最好的办法还是按照原本的方法匹配，将进一步的筛查交给语法分析。
- 注意空节点
    - 因为语法规则里某些产生式可能是空串，也就会产生NULL节点。输出的时候要注意判定。
- 准确定位行号
    - 因为单个error可能覆盖了不止一行，所以在使用错误输出时应该定位到error开始的第一行。
- 小心对待error
    - 虽然error是一个凭感觉添加的东西，但最好做到无冲突且全覆盖，即错误的判定不重不漏。error放在两个确定的单元中间比较安全，在表达式最左或最右要注意。
    - 不确定的情况下，最好按照原本的语法规则模拟错误情况，分号SEMI是很好的同步符号。