lab1 实验报告

本实验已实现正确的词法和语法分析功能

一、编译方式

> make parser

> ./test.sh

通过makefile提供的方式可以对本实验项目进行编译，此外我们写了一个简单的脚本用于对 `../Test/` 中的所有单元测试进行一键测试

二、实现过程

1、flex部分

- 书写用于词法分析的正则表达式(几个比较重要的正则表达式和书写过程中的注意事项)
  - 书写正则表达式的顺序应使得flex能进行某些**优先匹配**，比如“==”和“=”，应当将前者写在前面，否则就会将“==”识别为两个“=”。类似的情况还出现在ID和部分保留符之中。
  - 使用以下的正则表达式对**多行注释**进行匹配，可以满足,当有嵌套时总是识别最近的 `*/` ：
    > "/*" ([^*] | * + [^*/]) ** +"/"
  - 对于八进制，十六进制，科学计数法，也要书写正则表达式，并进行相应的进制转换，交给语法分析的属性值应是十进制和浮点数。下面是科学计数法的正则表达式：
    > ({digit}+.{digit}+|.{digit}+|{digit}+.)(e|E)?[-+]?{digit}+

    相应的进制转换代码可以在代码中查看
- 创建终结符的叶子节点，用于语法分析中建立语法树
  关键的创建节点语句:
  `yylval.node=create_node("FLOAT",FLOAT,yylineno);` （创建名为FLOAT的树节点）
  flex通过对全局变量yylval进行赋值，从而实现将词法单元的属性值交给bison继续使用

2、bison部分

- 抄写产生式，声明符号类型和优先级都只需按照pdf的讲解进行，不再赘述。
- 语法树相关
  - `yylval的声明: %union{ Tnode* node;} \\指针类型`
  - 当发生归约时应创建新的节点，并进行插入操作
    `$$=create_node("ExtDefList",0,@$.first_line);Ninsert($$,2,$1,$2);`
  - 多叉树数据结构的实现

```
struct treenode{
        char name[16];   //语法单元名称
        int line;        //所在行
        int type;        //词法单元类型
        struct treenode *firstchild;  //子结点
        struct treenode *nextbro;     //兄弟节点
        union{
                int i_val;
                float f_val;
                char *s_val;
};

Tnode *create_node(char *,int ,int );
void insert_node(Tnode *p,Tnode* newn); //将newn节点插入p节点的子结点中
void Ninsert(Tnode* p,int n,...); //不定长参数，用于连续插入多个节点
void dfs(Tnode* s,int dep); //遍历并打印语法树
```

- ***错误恢复***

  这一部分是比较麻烦且繁琐的，因为很难实现一个万能的错误恢复机制。在这一部分中我们还使用的 `yyerrok;` 进行立即错误恢复（如果没有比较特殊的连续错误也可以不使用）

  - **一个无法尽善尽美的例子**

    ```
    //global 以下三行代码都是在全局定义中
    1.int i
    2.int j
    3.int k=1;
    ```

    对于上述的三行代码我们应当报三个错误：

    ```
    错误1 error B at 2 : missing ';'
    错误2 error B at 3 : missing ';'
    错误3 error B at 3 : unexpected '='   //全局中只能定义不能赋值
    ```

    - 错误恢复产生式假设1：

      ```
      ExtDef:
          | Specifier error SEMI
          | error SEMI
      ```

      则只会报 `错误1` 一个错误，因为只有当找到一个SEMI，才能真正完成再同步，所以2、3行的词法单元都直接被丢弃

    - 假设2

```
ExtDef:
    | Specifier error SEMI{yyerrok;}
    | error SEMI          {yyerrok;}
    | Specifier error     {yyerrok;}
```

此时 `错误1，2，3` 都能找出来,**但在第三行的等号会重复两次报错**，因为当读入 `int k=` 时发生错误，且直接按 `Specifier error` 进行归约，所以此时不需要丢弃任何的词法单元。于是‘=’再一次被读入, `=1;` 报错，并按 `error SEMI` 归约。

对于上面的这种同一个符号报两次错的情况，可以通过修改 `yyerror` 函数解决（增加一个行标志位，用于一行只输出一个错误）

强如gcc也只会报一个错：