编译原理Lab1报告(选做2.2)

##一. 功能 ## **1.识别词法错误**

2.识别语法错误

3.识别指数形式的浮点数

4.构造语法树，在没有任何错误时，按照先序遍历的方式打印每一个结点的信息

二.实现思路

1.词法分析. 根据附录A中Tokens的定义构建正则表达式，其中我们组为选做2.2需要构建指数类型的浮点数。

```
digit [0-9]
integer [1-9][0-9]*|0
letter [_a-zA-Z]
id {letter}[_a-zA-Z0-9]*
nor_float [0-9]+\.[0-9]+
exp_float (([0-9]*\.[0-9]+)|([0-9]+\.[0-9]*))[eE][+-]?[0-9]+
```

2.语法分析 使用%token定义终结符，%type定义非终结符，同时定义优先级和结合性。接着逐步构建文法分析的基本框架，其中每个产生式的语义动作包括使用init函数为左边的非终结符创建新的节点，储存在$$中，然后调用insert函数，将产生式右边的符号设置为$$的孩子节点。其中Program为初始符号。

3.语法树的构建 语法树的结构如下

```
union Unit
{
    int val_int;
    float val_float;
    char val_char[100];
};
struct Node
{
union Unit unit;
int is_unit;
int line;
char name[100];
struct Node*first_child;
struct Node*sibling;
struct Node*last_child;
};
```

其中is_Unit表示该节点是不是词法单元，词法单元由一个联合体Unit定义，val_int表示词法单元是INT时的内容，val_float表示词法单元是FLOATS时的内容，val_char表示词法单元是ID和TYPE时的内容。line表示该节点所

在的行号。first_child为指向该节点第一个孩子的指针，sibling表示兄弟节点指针，last_child表示指向最后一个节点的指针。 程序使用root来储存根节点，在对Program符号进行分析时将ROOT初始化。

```
void insert(struct Node*parent,struct Node*child)
{
if(child != NULL)
{
    if(parent->first_child == NULL)
    {
       parent->first_child = child;
       parent->last_child = child;
    }
    else
    {
       parent->last_child->sibling = child;
       parent->last_child = child;
    }
}
}
```

在执行插入操作时，将child插入到parent的节点中，如果child不为空，首先判断parent有没有孩子，如果没有，child为parent的第一个孩子同时也为最后一个；如果有，child为parent目前最后一个孩子的兄弟，随后将parent的最后一个孩子设为child.

要打印节点信息时，需要根据前序遍历打印，在打印完当前节点的信息时，递归打印第一个孩子和兄弟节点的信息。在这个过程中需要维护一个变量depth，表示生成树的深度

```
    Print(start->first_child,depth+1);
    Print(start->sibling,depth);
```

三.编译方法

使用makefile编译，输入指令make