编译原理 实验报告

评 语（4 号楷体）	成 绩	
教 师：邓岳 年 月 日		

学院班级：1513012

学生学号：15130120196

学生姓名：陈金宇

实验日期：2017.12

一.实验目的

(1) 通过做上机题加深对编译器构造原理和方法的理解,巩固所学知识。

(1)会用正规式和产生式设计简单语言的语法;

(2)会用递归下降子程式序编写编译器和解释器;

(3)会写上机报告。

(2) 题目简述

1. 实现简单函数绘图语句

循环绘图　　比例设置

角度旋转　　坐标平移

注释

2.屏幕窗口的坐标系

左上角为原点

x方向从左向右增长

y方向从上到下增长

二.实验环境

Win10, CodeBlocks

三.实验内容

1.词法分析

①首先须定义好要用到的符号,包括保留字,参数,分隔符,运算符,函数,常数,空记号,颜色,出错记号。定义为枚举类型方便后面检索。

```
enum Token_Type{ ORIGIN,SCALE,ROT,IS,TO,STEP,DRAW,FOR,FROM,T,SEMICO,
L_BRACKET,R_BRACKET,COMMA,PLUS,MINUS,MUL,DIV,POWER,FUNC,CONST_ID,
NONTOKEN,ERRTOKEN,UNKOWNID,COLOR,RED,BLACK};
```

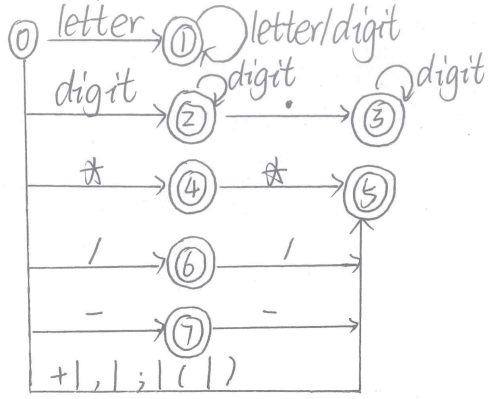
②每个符号包含4个属性:为类型,原始字符串,值及或函数值,如果没有则设为0。

```
struct Token{
    Token_Type type;
    char*  lexme;
    double  value;
    double  (*FuncPtr)(double);}
```

③定义一个符号表,把要用到的符号先存进去,比如sin,for等,这样当检测到为字符串时,则可在里面找,定义出对应的类型和值。

④在词法中用fopen和fclosfclose打开关闭文件。用getc和ungetc来获取和回退字符,取时用toupper统一返回大写字符处理。

⑤主体部用采用DFA扫描



匹配字符串：当遇到一个字符时，则进入，再遇到字符或数字时，则把其加进缓冲区并前进，直到非字符数字。最后把得到的字符串遍历字符表匹配，返回对应的属性。

匹配数字：当遇到数字，则不断加入缓冲区(字符形式)，若遇到小数点，则推进并继续匹配。最后用atof转为double类型并返回。

匹配"*"：若下一字符还是"*"，则为POWER，否则，为MUL，回退字符。

匹配"/"：若下一字符还是"/"，则为注释，不断吃掉直至遇到换行符和文件结尾；否则，为DIV，回退字符。

匹配"-"：若下一字符还是"-"，则为注释，同上；否则为MINUS，回退字符。

匹配"+"、","、";"、"("、")"：得到对应类型返回，否则为错误字符。

词法分析返回的是自定义Token类型。

2.语法分析

①此部分的核心是DFA

Program → {Program Statement SEMICO}

Statement → OriginStatement | ScaleStatement | RotStatement | ForStatement | ColorStatement

OriginStatement → ORIGIN IS L_BRACKET Expression COMMA Expression R_BRACKET

ScaleStatement → SCALE IS L_BRACKET Expression COMMA Expression R_BRACKET

ROTStatement → ROT IS Expression

ForStatement → FOR T FROM Expression TO Expression STEP Expression DRAW L_BRACKET Expression COMMA Expression R_BRACKET

ColorStatement → COLOR IS (BLACK|RED)

Expression → Term {PLUS|MINUS Term}

Term → Factor {PLUS|MINUS Factor} | Component

Component → Atom {POWER Atom}

Atom → CONST_ID|T | FUNC L_BRACKET Expression R_BRACKET | L_BRACKET Expression R_BRACKET

②语法分析采用向下递归方法，主要目的是匹配句子和建语法树，第一为语义分析服务，第二为计算服务。而从Expression开始涉及表达式，因此开始建树并返回结点。

数据结构为：

```
struct ExprNode{
    Token_Type OpCode;//记号种类
    union{
        struct{
            ExprNode *Left,*Right;
        }CaseOperator;//二元运算
        struct{
            ExprNode *Child;
            FuncPtr MathFuncPtr;
        }CaseFunc;//函数调用
        double CaseConst;//常数
        double *CaseParmPtr;//参数T
    }Content
};
```

③建树时注意带可变参数，并分清左右孩子和值。程序退出时注意释放存储空间。

3.语义分析

①画点时用到的函数为SetPixel(hDC,x,y,draw_color);

x,y为待计算的坐标值，draw_color为语法分析中获得的颜色。

②x,y应为树上实际参数的计算值，可以调用GetExpNode获得，再对其做进一步"加工"，包括乘以放大缩小值scale，角度变换，再加上原始开始值origin，即得最新的一组坐标值，传到SetPixel即可画出点。

③GetExpNode采用递归分析，根据不同的符号进行计算。

4 main函数

包括window程序主函数、初始化窗口函数、窗口处理函数。

四心得体会

1.问题：①遇到了多重定义的问题，后来发现是三个头文件相互引用，导致一些定义重复声明了，后通过#ifdef，#define解决。

②在开始时没从全局考虑，一些命名不太规范，如for里面的start,end,step，或建树时分成多个函数，没有规范使用可变参数。

⑤对窗口，画图不太熟悉，对全局变量的使用有些混乱。

2.优点：对基本画图语句的识别及实现，各个部分分工明确，联系紧密。对错误有一定的处理机制。

不足：对绘图语言的设计不够多简洁和多样性，对错误处理机制不够多强大。

3.总结：通过本次实验我学会了解释器的编写，如何将DFA转化为代码去识别，如何调用词法分析匹配语法及错误分析，对语法树的建立及求值，调用语法分析去实现语义函数，一环紧扣一环，既从全局考虑又具体到每个细节。这个对我以后做如自然语言分析、语句特征识别等都有很大帮助。