编译原理 实验报告

评 语（4 号楷体）	成 绩	
教 师：邓岳 年 月 日		

教学班级：5 班

学生学号：20049200342

学生姓名：唐祥

实验日期：2022.12

一、实验目的

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）会用正规式设计简单语言的词法；

（2）会用产生式设计简单语言的语法；

（3）会用递归下降子程序编写语言的解释器；

（4）了解并使用递归下降算法编写解释器；

（5）通过实验了解完整的 C++分离式编译链接过程。

二、实验环境

软件：装有 GNU 的 C++编译器的 Windows，IDE 为 VSCode

开发语言：C++

硬件环境：处理器 AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz

机带 RAM 16.0 GB (15.4 GB 可用)

三、实验内容

3.1 设计思路

整个项目分为三个模块，词法分析器 scanner，语法分析器 parser，语义分析器 semantic。其中，语法分析部分是语法制导翻译的基础，语法分析器是函数绘图语言解释器的核心，因此语法分析器的构造是整个解释器构造的关键，也是最难的一部分。词法分析和语义分析都是为其服务的。

项目还包括用于测试和绘图的主程序 main，且最终的主程序是调用语法分析器对外的接口来实现绘图。

每个部分都是一个模块，可独立运行。但要实现最终的绘图效果，需要用分离式编译将每个模块编译链接起来形成可执行文件。具体解释为：g++是 c++的编译器，如果 cpp 文件是在一个文件夹中，那可以进入这个文件夹直接 g++ 加上源代码的名字就可以编译，但是项目是在不同文件夹下的，不同 cpp 文件又存在包含其他模块的头文件，此时如果想分离式编译就要先把各个文件夹下的 cpp 文件先编译为可重定位目标文件，再将这些可重定位目标文件链接为一个可执行文件，每次这样操作的话是非常繁琐的，所以就用 makefile 将编译链接命令写在一个文件中，再使用 mingw32-make 执行 makefile 文件。

所以，最终的项目结构如下：

```
C:.
|  Makefile
|
├─lexer
|      scanner.cpp
|      scanner.h
|
├─parser
|      parser.cpp
|      parser.h
|
└─semantic
        main.cpp
        semantic.cpp
        semantic.h
```

Makefile 文件内容为：

```
semantic:lexer/scanner.o parser/parser.o semantic/semantic.o semantic/main.o
    g++ -o semantic scanner.o main.o parser.o semantic.o -lgdi32
lexer/scanner.o:lexer/scanner.cpp
    g++ -c lexer/scanner.cpp
parser/parser.o:parser/parser.cpp
    g++ -c parser/parser.cpp
semantic/semantic.o:semantic/semantic.cpp
    g++ -c semantic/semantic.cpp
semantic/main.o:semantic/main.cpp
    g++ -c semantic/main.cpp
```

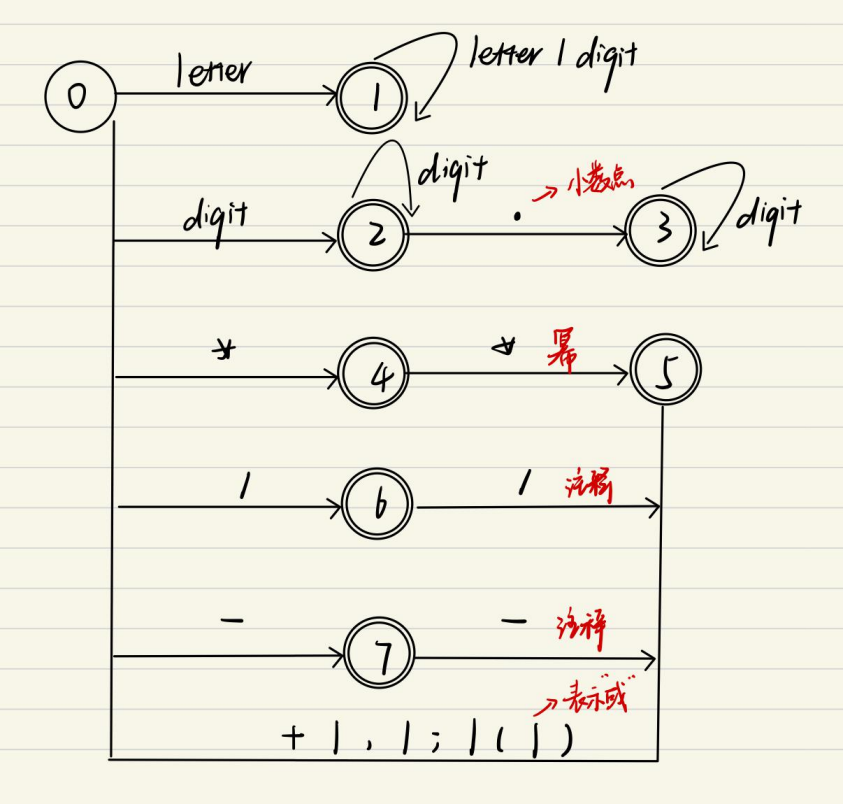
3.2 词法分析

词法分析器具体思路：

首先设定 Token 类，然后依据正规式和记号的类型设置查找表。词法分析中，最为重要的就是 DFA 的实现。编写 gettoken 函数一个记号一个记号的读。首先写一个循环对特殊情况进行处理，用来过滤掉源程序中的空格、TAB、回车等分隔符，文件结束返回空记号，遇到一个字符串尾部为\n 就说明要换行了，是使用 getchar 函数一个字符一个字符的读取文件，getchar 函数的实现使用 getc 来读取文件并将字符转换为大写，并加入到缓冲区数组 TokenBuffer，如果遇到空格就说明该记号已经读完了，并将 Token 类的属性指向它，后期也不用为属性 lexme 赋值，省去了后期对 Token 的一小部分赋值的操作。接着判断记号类型，

如果是字母打头，就可以采用硬编码的方式，将缓冲区数组送入到JudgeKeyToken对ID、字符常量以及保留字进行识别并返回(其他字符遇到@，#，￥，%，等等就返回ERROTOKEN)。数字只用判断下一个字符是否是数字或者小数点即可。最后判断是否是分隔符或者运算符。gettoken一直识别到eof位置。对于注释，如果是-或者/就判断下一个字符，如果也是的话那就是注释符，整行都不用考虑，这时候就有两种情况，一是这一行不是最后一行，另一种是这一行是最后一行，所以用一个while循环把整行读掉。

简化的DFA图示为：



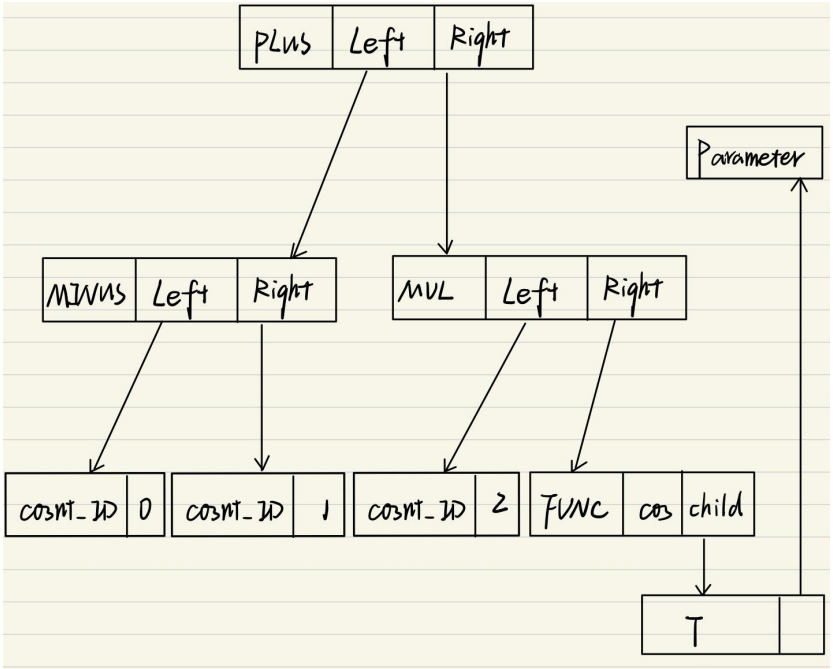
3.3 语法分析

因为最终构造的是递归下降的语法分析器，要求文法是LL(1)的，所以需要对二义文法进行改造，先改写为非二义的文法，再消除左递归和公共左因子最后转换为产生式与递归子程序有相应关系的形式。

语法分析器的具体思路，从外面整体来看的话，就是打开词法分析器，递归下降分析，关闭词法分析器。递归下降分析中，分为两大类。第一类：四个绘图语句，for，scale，origin，rot。不需要构造语法树，非常的简单。第二类：表达式。需要构造语法树。

如何构建语法树？首先确定数据结构：表达式树结点的类型分为四个。用匿名联合使其共用一块内存空间，这样就不用设计四个结点。二元运算的结点（加减乘除，乘方）：类型，左右孩子。函数调用的结点：类型，一个孩子（指向参数），函数指针，说明自己是什么函数。参数 T：类型，左值。常量的结点：类型，常量的值。因为我们考虑了运算符的优先级和结合性，所以在语法分析中会率先识别出优先级最低的运算符即 expression 这个非终结符对应的符号，将此作为根节点。其次是内部节点，也就是运算符节点是有两个孩子：加 PLUS、减 MINUS、乘 MUL、除 DIV、幂运算 POWER（有两个特例就是一元加和一元减，比如+5，这个+的左孩子就是 NULL，右孩子是 5，就将+5 转换成 5 了；还有-5，这个-的左孩子就是 0，右孩子就是 5，就将-5 转换成 0-5 了。有一个孩子的内部节点也就是函数节点：函数 FUNC。叶节点：常数 CONST_ID、参数 T。用一个例子来说明：-1 + 2 * cos(T)，是先识别出+号作为根节点，然后他有左右孩子，先看左孩子是-号，有左右孩子，左孩子是 0 右孩子是-1，再看+号的右孩子是 2 * cos(T)，识别出乘号，因为它是位于优先级次低的 term，左孩子是常量 2，右孩子是函数，其有一个指向函数名的指针，还有一个孩子为参数 T。

图示为：



3.4 语义分析及主程序

对于语义分析，除了编写绘制点坐标的函数，最重要的是计算语法分析中表

达式的值：参数是表达式的根，后续遍历语法树，根据不同的节点类型计算当前根节点（表达式）的值。

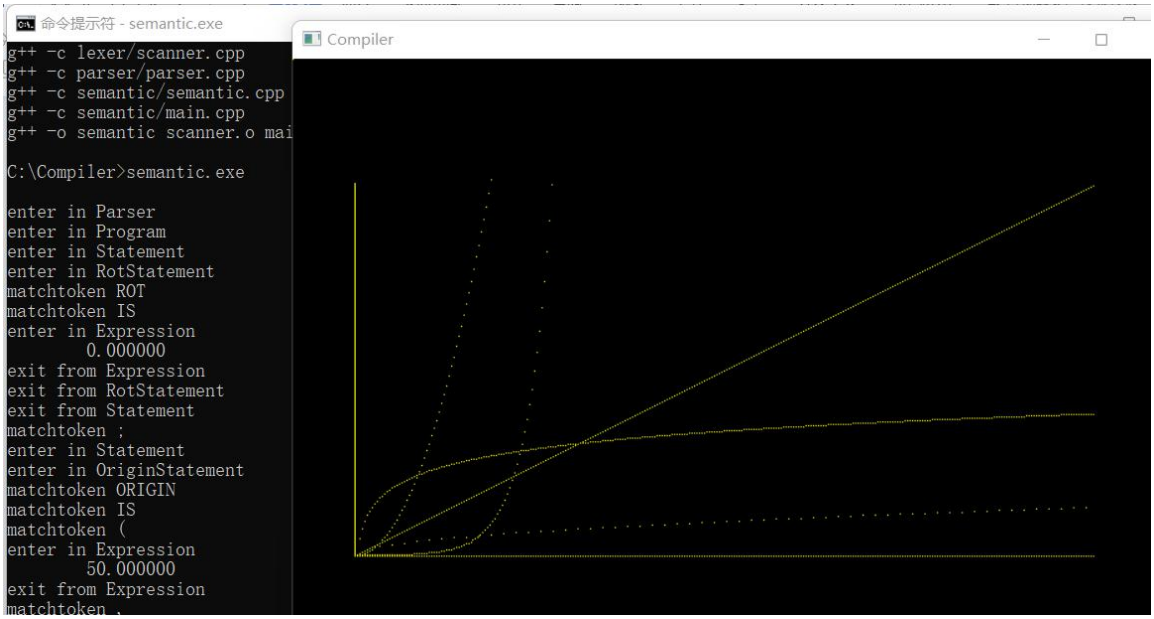
至此，解释器已经完成。由于最后要求输出图像且是在 Windows 操作系统下，故采用的方法是在 vc++环境下调用 win32api 来实现窗口化作图。对于主程序，WinMain 函数是提供给用户的 Windows 应用程序入口点，函数流程：注册窗口；创建窗口；显示窗口；调用绘图语言解释器读取文件并解释；消息循环，不断从线程的消息队列中取出消息并进行响应；消息处理：Windows 程序是事件驱动的，对于一个窗口，它的大部分例行维护是由系统维护的。每个窗口都有一个消息处理函数。在消息处理函数中，对传入的消息进行处理。客户写一个消息处理函数，在窗口建立前，将消息处理函数与窗口关联。这样，每当有消息产生时，就会去调用这个消息处理函数。通常情况下，客户都不会处理全部的消息，而是只处理自己感兴趣的消息，其他的，则送回到系统的缺省消息处理函数中去。

3.5 关于项目的执行

项目位于本地 C 盘 Compiler 文件夹下，控制台进入此文件夹，执行 make 指令（将 mingw32-make 重命名为 make），执行 makefile 文件，生成 semantic.exe。

```
C:\Compiler>make
g++ -c lexer/scanner.cpp
g++ -c parser/parser.cpp
g++ -c semantic/semantic.cpp
g++ -c semantic/main.cpp
g++ -o semantic scanner.o main.o parser.o semantic.o -lgdi32
```

文件传入通过主程序的硬编码实现。首先在 winmain 这个函数里将文件名传入，再调用绘图语言解释器，对外接口是 parser 函数，然后 parser 函数里面会对词法分析器进行初始化，初始化时会以只读形式打开文件（其实初始化词法分析器和关闭词法分析器就是对应着文件的打开与关闭）。直接控制台运行可执行文件，左侧为语法分析阶段对四种语句以及构建的表达式语法树的打印。右侧为实验结果 。



四、心得体会

4.1 遇到的问题

由于是自己一个人完成，所以在设计初期对于整个项目的模块规划以及代码编写等十分不规范，导致后期对项目进行整合时思维混乱。

写语法分析器的时候，非终结符的递归子程序不太会写，对于 factor、component、atom 理解并不是很深。看了书和网上的代码，因为一元加减和幂运算都是右结合的，它们都用的是递归调用自己（尾递归），不太明白为什么这样做。还有创建语法树的时候所运用的变参函数不太熟悉，看得书上的代码。

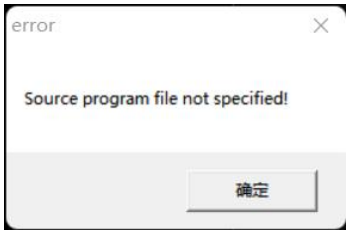
其次还有主程序那一部分，因为之前也从来没有接触过用 c 或者 c++画图之类的，所以一开始就比较蒙，不知道从哪入手。后来看到了书上采用的方法是在 vc++环境下调用 win32api 来窗口化画图，觉得还挺有意思的所以去学了下。

4.2 优点

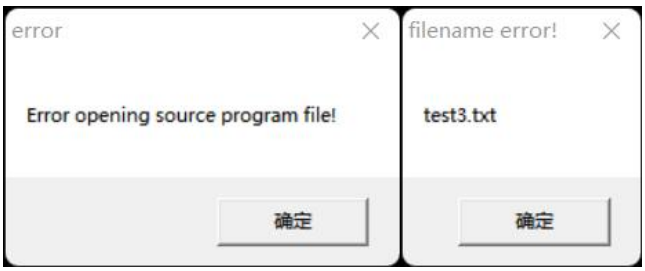
项目对报错展示进行了较为细致的处理，程序报错主要分为两个方面：

源文件传入错误：

- 如果传入文件名为空，则会报错：未指定文件名。



• 如果传入文件名错误（不存在该文件），则会报错：文件名错误。



源文件内容错误：

• 单词错误：记号类型匹配错误，由 FetchToken 函数抛出，词法分析获取的记号类型为 ERRTOKEN，即不构成正确的单词。比如将 SCALE 改成 SCALEL。

• 记号开头错误：由 Statement 函数抛出，不符合文法定义，句子开头的 Token 不正确。比如将参数 T 改成 123T。

• 表达式错误：由 Atom 函数抛出，不符合文法定义，表达式不正确。比如 Draw 的参数列表将参数 T 换为 ORIGIN。

• 保留字匹配错误，语义错误：由 MatchToken 函数抛出，不符合文法定义，语句错误（保留字、标点符等匹配失败）。比如再 SCALE IS 后面再加一个 ORIGIN。

4.3 不足

在报错展示中，如果源文件出现多处错误，解释器只能在检测出第一处错误后终止程序，从而发现不了源文件的所有错误。

整个解释器的是按照书上给的思路和顺序以及接口进行完善编写，一定程度上限制了个人思维。

每个模块可独立运行，刚开始为词法和语法分析器都写了一个测试程序，后来脑子发晕删掉了（语法分析器的测试结果保留了下来）,所以最终项目看来只有一个 main.cpp 作为测试文件。