编译原理语法分析实验报告

选课序号： 19 专业班级： 计算机科学与技术2019-3班

姓名：\_胡聿鑫\_学号：\_ 2220192813\_\_\_\_\_\_日期： 2021年11月16日

# 实验目的

通过设计、编制、调试一个典型的语法分析程序，实现对词法分析程序所提供的单词序列进行语法检查和结构分析，检查语法错误，进一步掌握常用的语法分析方法。

# 实验内容及要求

输入单词流文件，输出语法树。采用自顶向下分析方法中的递归下降法或LL(1)方法实现语法分析程序。要求如下：

（1）能发现语法错误，并将错误信息输出到屏幕上，自定义错误处理模式；

（2）建立抽象语法树并输出抽象语法树。

# 实验相关理论知识

语法分析是编译过程的核心部分，它的主要功能是按照程序语言的语法规则，从由词法分析输出的源程序符号串中识别出各类语法成分，同时进行语法检查，为语义分析和代码生成作准备。语法分析程序在分析过程中检查符号串是否为该程序的句子，若是则输出该句子的语法分析树，否则就表示源程序存在语法错误，并报告错误的性质与位置。

对每个非终极符按其产生式结构构造相应语法分析子程序，其中终极符产生匹配命令，而非终极符则产生过程调用命令。因为文法递归相应子程序也递归，所以称这种方法为递归子程序下降法或递归下降法。递归下降分析法的原理是利用函数之间的递归调用来模拟语法树自上而下的构建过程。从根节点出发，自顶向下为输入串中寻找一个最左匹配序列，建立一棵语法树。在不含左递归和每个非终结符的所有候选终结首字符集都两两不相交条件下，我们就可能构造出一个不带回溯的自顶向下的分析程序，这个分析程序是由一组递归过程（或函数）组成的，每个过程（或函数）对应文法的而一个非终结符。**给定语法**如下：

<表达式>::=[+|-]<项>{<加法运算符> <项>}  
 <项> ::=<因子>{<乘法运算符> <因子>}  
 <因子>::= <标识符>|<无符号整数>|(<表达式>)  
 <加法运算符>::= +|-  
 <乘法运算符>::= \*|/

需要注意的是：自顶向下的分析算法通过在最左推导中描述出各个步骤来分析记号串输入。预测分析程序试图利用一个或多个先行记号来预测出输入串中的下一个构造。实际上，待编译的文本首先通过词法分析器，分解得到Token序列，再对Token进行语法分析，得到分析树。因此，语法分析算法部分的处理对象是Token结构体，而不是简单的字符序列。

**计算FIRST集的方法：**

 如果 X 是终结符号，那么FIRST(X)={X}。

 如果 X 是非终结符号，且 X -> Y1Y2Y3…Yk 是产生式：

如果a在FIRST(Yi)中，且 ε 在FIRST(Y1)，FIRST(Y2)，…，FIRST(Yi-1)中，那么a也在FIRST(X)中；

如果ε 在FIRST(Y1)，FIRST(Y2)，…，FIRST(Yk)中，那么ε在FIRST(X)中。

 如果X是非终结符号，且有X->ε，那么ε在FIRST(X)中。

**计算FOLLOW集的方法：**

 将右端结束标记 $ 放到 FOLLOW(S) 中

 按照下面两个规则**不断迭代**，直到所有的FOLLOW集合都不再增长为止：

1. 如果存在产生式A -> αBβ ，那么 FIRST(β)中所有非ε的符号都在FOLLOW(B)中；
2. 如果存在产生式A -> αB，或者A -> αBβ 且FIRST(β)包含ε，那么FOLLOW(A)中的所有符号都加入到FOLLOW(B)中。

语法分析过程的依赖关系如下图所示：



算法的输出为抽象语法分析树，在打印树时，以缩进数代表结点的深度。只有在语法分析成功后才会输出语法树，存在语法错误则编译停止。

# 实验过程、运行结果

首先给出实验的总步骤如下：

(1) 语言的语法描述，确定语法规则，即文法的设计；

(2) 把文法形式符号中所隐含的信息内容挖掘出来并用分析表的形式表示出来；

(3) 语法分析的数据输入和输出形式的确定；

(4) 语法分析程序各个模块的设计与调试。

本实验基于词法分析程序进行，因此调试时首先需要对TEST程序进行词法分析，只有在词法分析通过后才会开始执行语法分析程序。在语法分析程序中，为避免代码过于冗长，首先将词法分析程序作为头文件放在程序中，顺利通过词法分析后执行语法分析程序，成功则输出抽象语法树，否则进行错误处理，其流程如下图所示：



在各个函数中定义变量es，每个函数结束后返回es的值，再通过switch函数写明各错误出现的原因，以es的值对应case后的编号，便可以让语法分析程序在遇见错误时停止，并显示错误原因。要想确定调用的函数，则可以调用statement函数，将token和各关键字、常量、标识符等逐一进行比较。

针对输入的单词流，逐个字符进行读入，在statement函数中作出相应的比较判断，以便让程序知道何时调用何种函数，其流程如下图所示：



针对FIRST和FOLLOW集的计算，有如下定义：

**计算FIRST集：**

  FIRST(<表达式>)={ +， -， (， <标识符>， <无符号整数> }

  FIRST(<因子>)={ <标识符>， <无符号整数>， ( }

  FIRST(<项>)={ <标识符>， <无符号整数>， ( }

  FIRST(<加法运算符>)={ +， - }

  FIRST(<乘法运算符>)={ \*， / }

**计算FOLLOW集：**

  FOLLOW(<表达式>)={ ) }

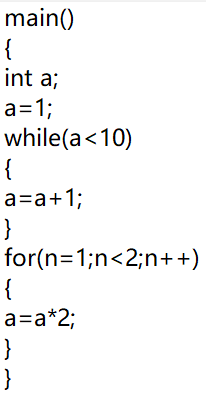
  FOLLOW (<项>)={ +，- }

  FOLLOW (<因子>)={ \*，/ }

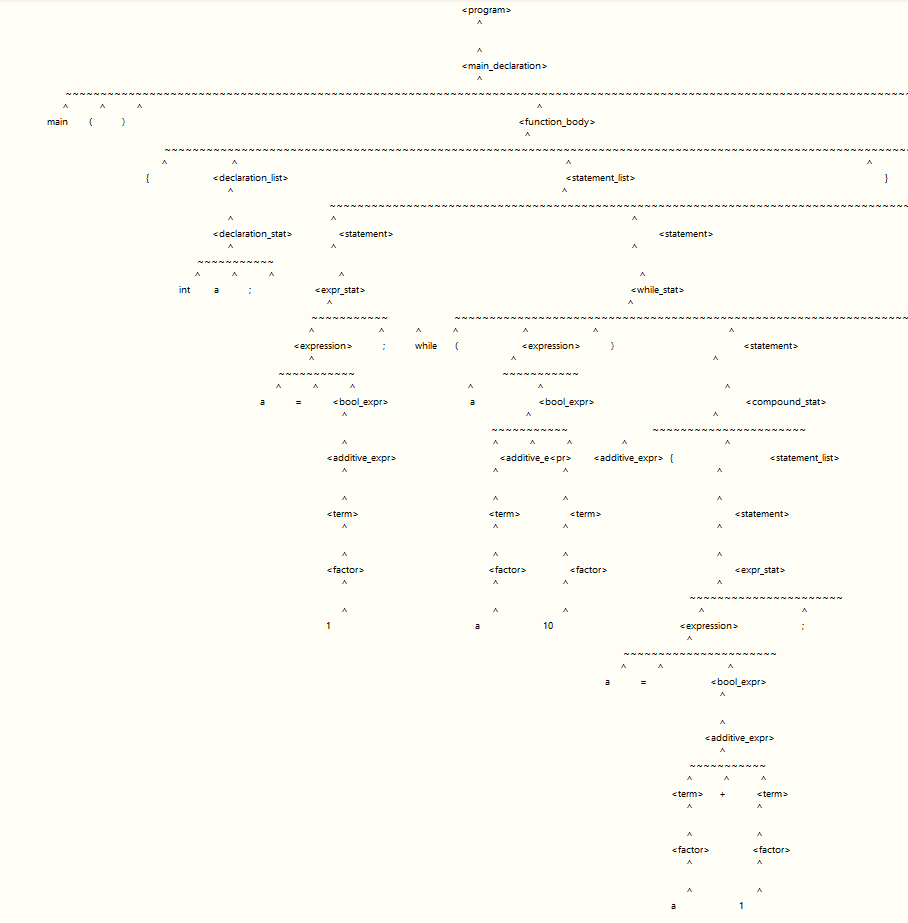
  FOLLOW (<加法运算符>)={ <标识符>， <无符号整数>， ( }

  FOLLOW (<乘法运算符>)={ <标识符>， <无符号整数>， ( }

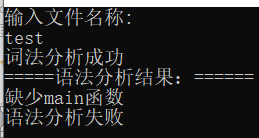
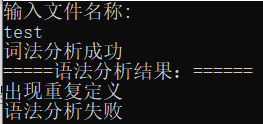
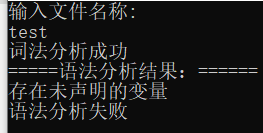
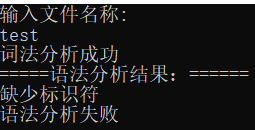
到此，可以根据TEST源程序完成语法分析的过程。任意给定一个TEST源程序如下（该源程序没有词法错误，也不存在语法错误）：

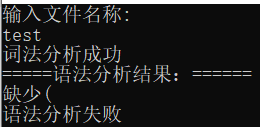
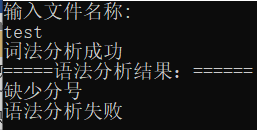
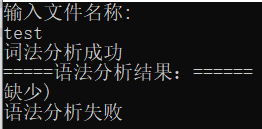
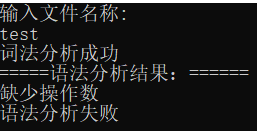


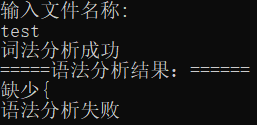
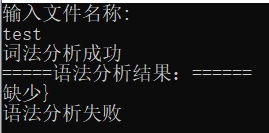
而要想生成语法树，需要从根节点开始递归画出其所有子树，每个子树在到叶节点时返回，对于同属一个父节点的子节点需要进行连接从而加以区分。上述TEST源程序经过成功的语法分析程序所生成的语法树如下：



若TEST程序中存在语法错误，则会出现相应的提示，部分错误情况如下：

针对错误处理，一次运行应尽可能多地发现语法错误，并且在遇到错误后便立即调用中断程序执行中断。对于常见、可预见的错误，可以在程序对应处作出相应的修改或扩充；而针对无法遇见的、多变的错误则可以旋转跳过部分原文使其在适当的地方停止，然后再继续分析。在识别出一个错误之后，要想跳过原文的某部分，则语言必定包含某些关键字，这些关键字出错的概率很小，因而被调用的语法分析过程总能够正常结束。

# 实验总结

此次实验，让我对编译原理的基本知识有了深入的了解，加强了对语法分析的认识。代码的编写过程中用到了一些以前从未用过的函数，都借鉴了网上的宝贵经验对其现学现用，其中部分掌握得还不是很深。在代码调试过程中也出现许多无法理解的错误，但仍旧坚持不断改正错误，最终调试出了结果。也正因此，我的编程实践能力得到很大的提高。

递归下降分析法是确定的自上而下分析法，这种分析法要求文法是LL（1）文法。它的基本思想是：对文法中的每个终结符编写一个函数，每个函数的功能是识别由该非终结符所表示的语法成分。由于描述语言的文法常常是递归定义的，因此相应的这组函数必然以相互递归的方式进行调用。也正是通过此次实验，我对于递归下降法的抽象定义，有了更加具体的理解和感受，也从实践中明白了其运用的重要意义。

实验最开始盲目照抄书后的代码，每次的运行结果都和词法分析的输出一致，而无法生成抽象语法树。经过查阅大量网上的资料，我尝试在每个函数中都添加一个指针变量，并将所有printf语句删除，避免语法分析的结果以二元组的形式输出，同时定义了一个结构体用于存储树，该结构体定义如下：

struct node

{

string st;

vector<node\*>son;

int wide;

node\* addSon(string s)

{

node\* Son=new node();

Son->st=s;

son.push\_back(Son);

return Son;

}

} \*Root;

这样一来，每分析完一个单词，就能够将其输出到语法树中并存储起来，最后若语法分析通过，则将存储的所有信息打印输出，便实现了生成语法树的功能。为了让语法树能够更加美观，首先使得根节点处在居中的位置，然后开始向下画子树。父子节点之间用符号^进行连接，以表达“该子节点属于其父节点”的这一关系，针对同一父节点的子节点，用横线绘制以集中在一起，从而使得语法树逻辑清晰，具体形象。

但是本程序没有解决连续查错的问题，也不能定位错误位置，这些都是有待改进的地方。若想实现连续查错，则每次函数调用结束后不能return es，否则一旦遇到错误便不会往下进行，但若不使用return无法完成所有的单词流输入。而对于错误的定位，我尝试效仿词法分析，定义一个全局变量，每当读到换行符则使该变量+1，最后在遇到错误时输出的数即为错误所在行数，但是由于每个函数均返回es的值，已无法再返回另一个值，因此我没有找到合适的办法将错误所在行数输出，这也看出我的编程水平仍需要进一步的提升。