编译原理程序设计报告

**简易编译器的设计与实现**

**专业年级 ： 18级软件工程专业**

**小组成员 ： 姚東序10185101174**

**黄逸宏10171900143**

**侯静雯10185101139**

**苏日力格10185101175**

**指导教师 ： 琚小明**

**2020年12月**

# 设计分工

组长学号及姓名：姚東序10185101174

分工：

组员1学号及姓名：黄逸宏10171900143

分工：

组员2学号及姓名：侯静雯10185101139

分工：

组员3学号及姓名：苏日力格10185101175

分工：

# 摘要

编译器是将便于人编写，阅读，维护的高级计算机语言翻译为计算机能解读、运行的低阶机器语言的程序。编译是从源代码（通常为高阶语言）到能直接被计算机或虚拟机执行的目标代码（通常为低阶语言或机器语言）的翻译过程。

一、编译器的概述

1. 编译器的概念

编译器是将便于人编写，阅读，维护的高级计算机语言翻译为计算机能解读、运行的低阶机器语言的程序。编译器将原始程序作为输入，翻译产生使用目标语言的等价程序。源代码一般为高阶语言如Pascal、C++、Java 等，而目标语言则是汇编语言或目标机器的目标代码，有时也称作机器代码。

2．编译器的种类

编译器可以生成用来在与编译器本身所在的计算机和操作系统（平台）相同的环境下运行的目标代码，这种编译器又叫做“本地”编译器。另外，编译器也可以生成用来在其它平台上运行的目标代码，这种编译器又叫做交叉编译器。交叉编译器在生成新的硬件平台时非常有用。“源码到源码编译器”是指用一种高阶语言作为输入，输出也是高阶语言的编译器。例如: 自动并行化编译器经常采用一种高阶语言作为输入，转换其中的代码，并用并行代码注释对它进行注释（如OpenMP）或者用语言构造进行注释（如FORTRAN的DOALL指令）。

3. 本编译器概述

编译程序的工作过程一般可以分为五个阶段：词法分析、语法分析、语义分析与中间代码产生、优化、目标代码生成。每一个阶段在功能上是相对独立的，它一方面从上一个阶段获取分析的结果来进行分析，另一方面由将结果传递给下一个阶段。由编译程序的五个阶段就对应了编译系统的结构，这五个对应阶段分为编译器的前段，中间代码以及后端。

其中词法分析器利用状态转换等方法，将源程序转化成为一个一个的单词符号二元式。一般程序语言的单词符号包括关键字、运算符、常数、标识符和界符。语法分析器将这些单词符号作为输入，对它进行语法分析。语法分析采用LL1分析法，语法分析器把语法单元作为输入供语义分析器使用。在语法分析的同时进行语法分析，并产生一定的语义动作，来生成中间代码。优化和目标代码生成是针对某一种处理器而言的。代码优化是将语义分析生成的中间代码进行优化，产生执行效率更高的代码。目标代码生成最终生成可以在某种机器上运行的机器语言或者汇编语言。还要有符号表可供查询。在整个编译过程中还包括对表格的操作和对错误的处理，这些也都是非常重要的环节。

环境：

编译器整体全部使用Java编写

目标代码在8086指令集机器上运行

**关键词**：编译原理，前端，中间代码生成，词法分析，语法分析。

目录

[设计分工](#_Toc9419)

[摘要](#_Toc9079)

[1. 背景与现状分析](#_Toc1487)

[2. 相关技术介绍](#_Toc21280)

[2.1 JAVA](#_Toc8148)

[2.2 配置文件](#_Toc9909)

[2.3 JAVAFx](#_Toc27207)

[3. 功能需求分析](#_Toc21280)

[3.1 词法分析](#_Toc8148)

[3.2 语法分析](#_Toc9909)

[3.3 语义分析](#_Toc8148)

[3.4 符号表处理](#_Toc8148)

[3.5 出错处理](#_Toc9909)

[3.6 输入输出功能](#_Toc8148)

[3.7 可视化界面](#_Toc9909)

[3.8 （中间代码生成）](#_Toc8148)

[4. 功能设计](#_Toc31073)

[4.1 词法模块](#_Toc22359)

[4.1.1 按照自动机DFA来实现词法分析](#_Toc17571)

[4.1.2 扫描设计](#_Toc11149)

[4.1.3 构词缓冲设计](#_Toc17997)

[4.1.4 符号表建立与更新](#_Toc22655)

[4.1.5 关键字与标识符ID的分离.................................................](#_Toc22655)

[4.1.6 词法返回值](#_Toc22655)

[4.2 语法模块](#_Toc4502)

[4.2.1 语法分析表存储](#_Toc17571)

[4.2.2 分析堆栈构建与初始化](#_Toc11149)

[4.2.3 输出语法树构建](#_Toc17997)

[4.2.4 采用LL(1)算法的语法分析结构](#_Toc22655)

[4.2.5 符号表更新](#_Toc18373)

[4.3 语义模块](#_Toc24784)

[4.3.1 在语法树的基础上对类型进行语义分析](#_Toc6086)

[4.4符号表生成模块](#_Toc26320)

[4.4.1 代码生成方法](#_Toc27745)

[4.4.2 代码输出](#_Toc11468)

[4.5符号表](#_Toc11119)

[4.5.1 符号表的结构](#_Toc15083)

[4.5.2 符号表的查询、添加、修改等操作](#_Toc30433)

[4.6出错处理](#_Toc30690)

[4.6.1 出错信息](#_Toc16909)

[4.6.2 出错定位](#_Toc5091)

[5. 功能实现](#_Toc834)

[5.1 基本的输入、输出界面](#_Toc14832)

[5.2 界面布局与设计](#_Toc25508)

[6. 测试](#_Toc12611)

[6.1自测用例](#_Toc3171)

[6.2检查用例](#_Toc632)

[7. 总结](#_Toc18781)

[7.1组长：姚東序](#_Toc11269)

[7.2组员：黄逸宏](#_Toc31239)

[7.3组员：侯静雯](#_Toc15039)

[7.4组员：苏日力格](#_Toc31239)

[8. 附录](#_Toc4685)

[源代码：](#_Toc12465)

[8.1可执行文件链接](#_Toc20434)

[9．参考文献](#_Toc1847)

# 

1. **相关技术分析**
   1. JAVA
   2. 配置文件
   3. JAVAFx

**3.功能需求分析**

环境：

Java语言编写，在eclipse或者其他Java编译器上编译执行。

编译器整体程序流程图：



3.1 词法分析

3.1.1 功能：

1）获取待处理的源代码

2）生成二元式序列

3）采用DFA和自动机实现二元式的填写

4）将获得的二元式输入到文件中

词法分析过程是将字符序列转换为Token序列的过程。此阶段的任务是从左到右依次扫描[源程序](http://baike.baidu.com/view/546605.htm" \t "_blank)中的字符，即对构成字符流进行扫描然后根据构词规则识别Token。设计的词法分析器能相对完善地构造出不同的单词，用二元式的形式存储，简显易懂，并将新的标识符单词填入变脸表当中，实现在计算机内单词序列的统一存储。

单词符号是程序设计语言中基本的语法单元，通常分为5种：

1）关键字（又称基本字或保留字）：程序设计语言中定义的具有固定意义的英文单词，通常不能用作其他用途，如C语言中的while、if、for等都是关键字。

2）标识符：用来表示名字的字符串，如变量名、数组名、函数名等。

3）常数：常数的定义如下：

digit ← **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

intnumber ← digit+ (the maximum integer number is 231)

exponent ← (E|e) ( + | - | ε ) digit+ (the maximum exponent value is 128)

fraction ← . digit+

realnumber ← digit+ exponent | digit+ fraction ( exponent | ε )

4）运算符：又分为算术运算符，如+、-、\*、/等；关系运算符，如=、>=、>等；逻辑运算符，如 or、not、and等。

5）界符：如“，”“；”“（”“）”“：”等。

在上面所给出的5种单词符号类中，关键字、运算符和界符是程序设计语言提前定义好的，因此它们的数量是固定的，通常只有几十个或者上百个。而标识符和常数是程序设计人员根据编程需要按照程序设计语言的规定构造出来的，因此数量即便不是无穷，也是非常大的。

词法分析程序输出的单词符号通常用二元式（单词种别，单词符号的属性值）表示。其中：

1）单词种别。单词种别表示单词种类，常用整数编码，这种整数编码又称为种别码。至于一种程序设计语言的单词如何分类、怎样编码，主要取决于技术上的方便。一般来说，基本字可“一字一种”，也可将其全体视为一种；运算符可“一符一种”，也可按运算符的共性分为几种；界符一般采用“一符一种”分法；标识符通常统归为一种；常数可统归为一种，也可按整型、实型、布尔型等分为几种。

2）单词符号的属性值。单词符号的属性值是反映单词特征或者特性的值，是编译中其他阶段所需要的信息。如果一个种别只含有一个单词符号，那么其种别编码就完全代表了自身的值，因此相应的属性值就不需要再单独给出。如果一个种别含有多个单词符号，那么除了给出种别编码之外还应给出单词符号自身的属性值，以便把同一种类的单词区别开来。例如，对于标识符，可以用它在符号表的入口指针作为它自身的值；而常数也可用它在常数表的入口指针或者其二进制值作为它自身的值。

3.1.2 流程图



3.2 语法分析

3.2.1 功能

1）获取产生式

产生式如下：

program → compoundstmt

stmt → ifstmt | whilestmt | assgstmt | compoundstmt

compoundstmt → **{** stmts **}**

stmts → stmt stmts | ε

ifstmt → **if (** boolexpr **) then** stmt **else** stmt

whilestmt → **while (** boolexpr **)** stmt

assgstmt → **ID** = arithexpr **;**

boolexpr → arithexpr boolop arithexpr

boolop → **<** | **>** | **<=** | **>=** | **==**

arithexpr → multexpr arithexprprime

arithexprprime → **+** multexpr arithexprprime | **-** multexpr arithexprprime | ε

multexpr → simpleexpr multexprprime

multexprprime → **\*** simpleexpr multexprprime | **/** simpleexpr multexprprime | ε

simpleexpr → **ID** | **NUM** | **(** arithexpr **)**

2）消除直接左递归

3）自动生成first和follow集合

4）构建分析表

5）构建分析栈

6）通过查分析表进行语法分析

7）输出动作序列

3.2.2 LL1文法

LL(1)分析法是指从左到右扫描、最左推导(LL)和只查看一个当前符号（括号中的 1）；

LL(1)分析法又称预测分析法，与递归子程序法同属于自顶向下确定性语法分析方法;

LL(1) 分析法的基本要点有三：

1. 利用一个分析表，登记如何选择产生式的知识；
2. 利用一个分析栈，记录分析过程；
3. 此分析法要求文法必须是 LL(1)文法。

语法分析是编译中的第二阶段，它的任务是识别和处理比单词更大的语法单位，判断源程序在结构上是否正确。

从形式上来说，语法分析是对一个给定的字符串，判断它是否为文法的一个句子。在我设计的语法分析器中，直接采用LL(1)分析方法。当然由于本身文法为上下文无关文法，所以纯LL1并没有使用问题，此外，对于错误的识别，该语法分析器可以输出错误信息。

3.2.3 流程图



3.3 语义分析

3.3.1 功能

语义是指源程序及其组成部分所表述的含义，和语法不同，语法是关于程序及其组成部分的构成规则的描述，是上下文无关的；而语义是关于语法结构的含义及其使用规则的描述，是上下文有关的。语法上正确，其语义不一定正确。

分为静态语义和动态语义。

|  |  |
| --- | --- |
| 静态语义 | 是指在编译阶段能够检查的语义，比如标识符未定义，类型不匹配等。 |
| 动态语义 | 是指在目标程序运行阶段能够检查的语义，比如除数为0，无效指针，数组下标越界等。 |

语义分析的任务就是对结构上正确的源程序进行上下文有关性质的审查，审查源程序是否有无语义错误，为代码生成阶段收集类型信息，主要功能包括建立符号表，进行静态语义检查，发现语义错误。

3.4 符号表处理

* + 1. 功能

1. 构建活动记录表
2. 构建符号表
3. 构建函数表
4. 构建长度表
5. 添加活动记录
6. 添加变量表
7. 符号表预处理
8. 添加函数表
9. 添加参数表

10）构建活动记录与变量表（二合一）

符号表是在目标代码生成中不可以缺少的提供查询服务的表，在其中记录了程序收集，记录于使用的源程序的语法符号的类型、特征等相关信息，信息集中反映了标识符的语义特征属性。在词法分析及语法在分析过程中不断积累和更新表中的信息，并在词法分析到代码生成的各阶段，按各自的需要从表中获取不同的属性信息。在本次课设中，符号表的作用和地位是重要关键的。

符号表是一个编译器的核心数据结构，它代表了标识符的动态语义词典，属于编译中语义分析的知识库。符号表中所登记的信息在编译的不同阶段都要用到，对于一个多遍扫描的编译程序，不同遍所用的符号表也往往各有不同，因为每遍所关心的信息各有差异。符号表的组织方式也决定了未来处理符号表内容时的效率。我所设计的符号表生成器能够在定义标识符时把对应的语义信息填入符号表中；当在语句中使用该标识符时，通过查找相应的表项来判断该标识符是否存在且使用正确。符号表常见的功能有定义和重定义检查、类型匹配校验、数据的越界和溢出检查、值单元存储分配信息和函数的参数传递与校验。由于时间有限，我设计的符号表系统只完成了上述功能中的一部分。

符号表是标识符的动态语义词典，属于编译中语义分析的知识库；主要内容：

1. 名字 — 标识符源码，用作查询关键字；
2. 类型 -- 该标识符的数据类型及其相关信息；
3. 种类 -- 该标识符在源程序中的语义角色；
4. 地址 -- 与值单元相关的一些信息；

3.4.2 流程图



3.5 出错处理

3.5.1 功能

1）诊察各类错误的能力。

2）报错及时准确（出错位置，错误性质）。

3）一次编译找出错误的多少。

4）改正错误的能力。

5）遏制重复错误信息的能力。

3.5.2 目标与诊察

目标：编译程序对于语法和语义正确的源程序要正确地编译生成等价的目标代码；而对于错误的源程序不能一发现就停止，而是要能检查出错误的性质和出错位置，并使编译能继续下去，同时尽可能多而准确地发现错误和指出各种错误。

诊察：（1）违反语法和语义规则以及超过编译系统限制的错误。由编译程序在语法和语义分析过程中诊察出来。（语义分析要借助符号表）（2）下标越界、计算结果溢出以及动态存储数据区溢出等在目标程序运行时才能检测，因此由目标程序诊察。

对此，编译程序要生成相应的目标程序代码进行检查并处理。

3.5.2 错误分类

错误分类：语法错误、语义错误。

语法错误：程序结构不符合语法（包括词法）规则的错误。如漏掉分号、小数点写成逗号等错误。

语义错误：程序不符合语义规则或超越具体计算机系统的限制。如标识符先声明再使用、溢出错误等。

3.5.3 处理方式

（1）错误改正：指编译诊察出错误以后，根据文法进行错误改正。实际做起来很难，尽量改正如缺少分号这样的小错误。

（2）错误局部化处理：发现错误后吧错误控制在局部范围，避免错误扩散，影响其他部分分析。

一般原则：直接跳过。

错误局部化处理的实现：递归下降分析法。

提高错误局部化程度的方法：合法后继符号集与定制符号集。

* 1. 输入输出功能

输入：源程序；

输出：单词符号。词法分析器的单词符号常常表示成以下的二元式(单词种别码，单词符号的属性值)。

* 1. 可视化界面

算法功能模块和可视化模块：算法功能模块与普通编译器相似， 分为 “词法分析器 ”、 “语法分析器 ”、“语义分析器”和“中间代码生成器”四个部分；可视化模块与算法功能模块一一映射，将传统编译器中的算法实现的输入与输出结果动态地展现在可视化界面上。

除了实现传统编译器的词法分析、语法分析、语义分析、中间代码生成外，

还包含对应功能模块的可视化功能：

1) 词法分析过程中动态显示构词缓冲区中指针的改变，高亮显示正在分析的词素，被分析过的词素按照不同的类型呈现不同的颜色；

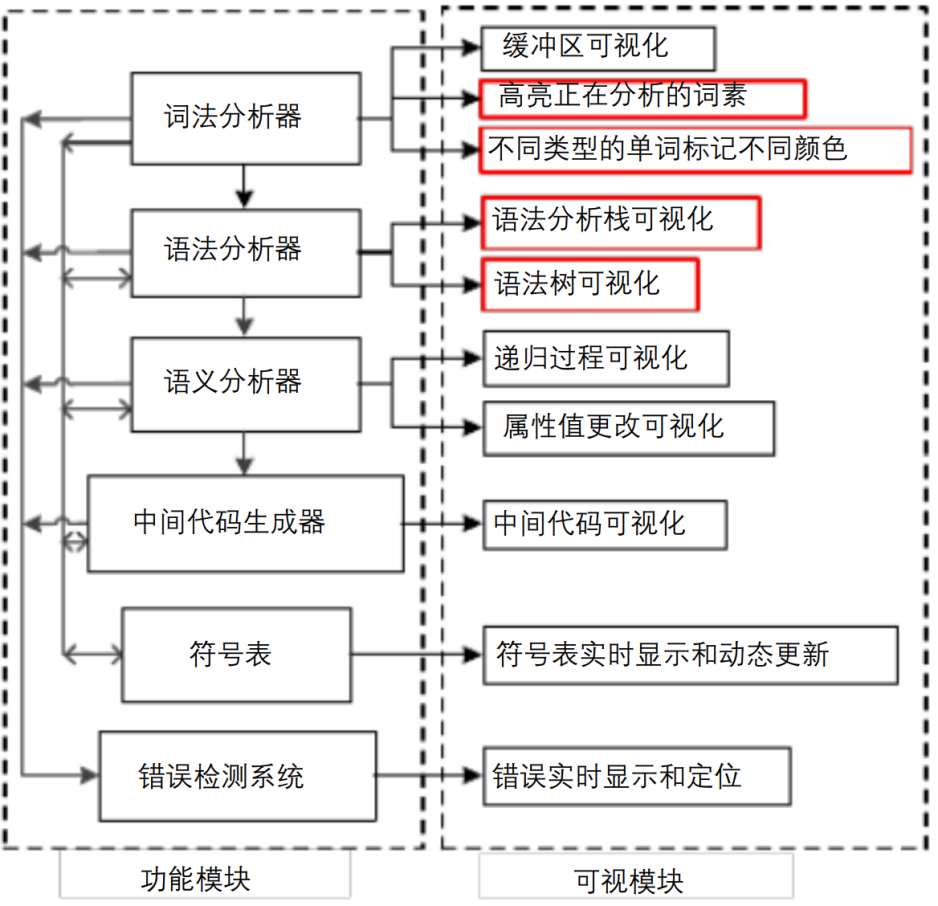
2) 语法分析过程中动态展示语法分析栈的变化，包括堆栈内容、堆栈指针以及堆栈生长方向；

3) 语法分析过程中动态构造语法树， 语义分析过程中动态改变语法树上被调用节点的颜色和属性值；

4) 中间代码生成过程中动态展现正在分析的语句和正在生成的三地址代码；

5) 符号表中的值随着编译的进行动态改变；

6) 实时产生和定位的错误处理系统。



* 1. 中间代码生成

3.8.1 功能

1）构建语义栈

2）重新导入二元式

3）获取动作序列

4）初始化四元式

5）获取动作序列位置

6）在已有动作序列的基础上生成四元式（自底向上）

7）输出四元式

8）四元式的预处理及优化

中间代码是高级程序语言中，各种语法成分的语义结构表示；它介于源语言和目标语言之间。中间代码设置的目的是便于编译的后期处理（优化和目标代码生成）。因此生成四元式是制作一个编译器必不可少的一环。

中间代码生成属于语义分析的阶段，中间代码是高级程序语言中，各种语法成分的语义结构表示，它介于源语言和目标语言之间。设置中间代码有以下三点好处：

1）便于进行与机器无关的代码优化工作。

2）是编译程序改变目标机更容易。

3）使编译程序的结构在逻辑上更为简单明确。

以中间语言为界面，编译前端和后端的接口更清晰。我设计的四元式生成器是在语法分析的基础上，直接获取动作序列，而后直接基于词法序列进行分析，采用了自底向上的LR0分析法，对于每一个作用域都生成唯一的四元式结构体数组，便于接下来中间代码的优化，作用域和作用域之间的四元式序列互不干扰。

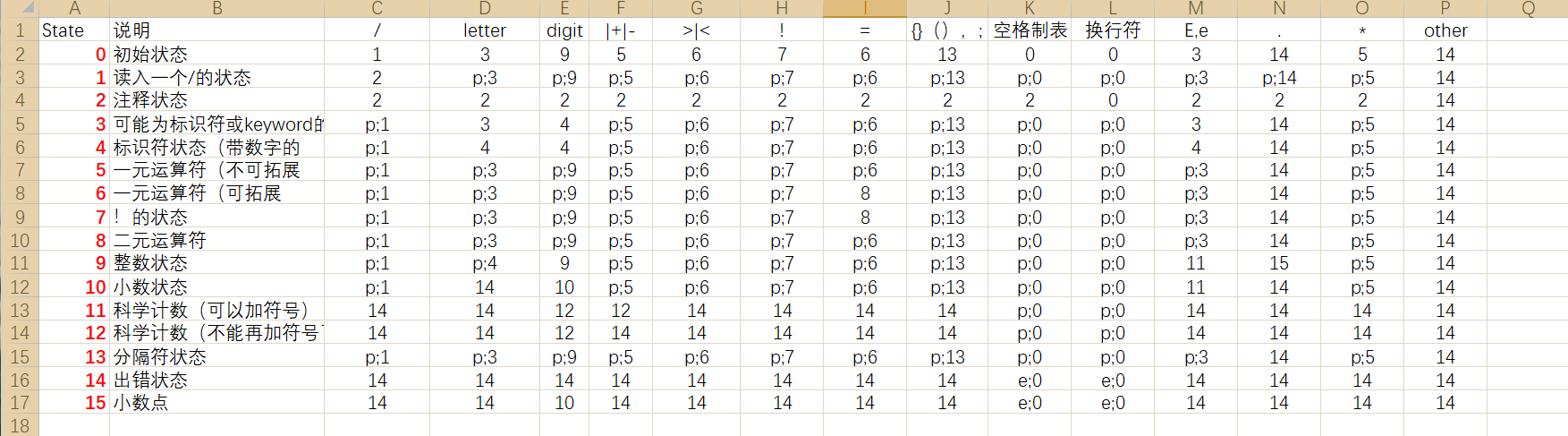
3.8.2 流程图



**4 功能设计**

4.1 词法模块

4.1.1 按照自动机DFA来实现词法分析



在进行词法分析之前，首先进行构表，对状态转换进行解析

上图是我们针对于Assignment1的要求做出来的DFA词法分析表，一共分为15个状态。并对不同的输入进行分类。

以下就词法分析遇到的单词处理方法进行分情况解释：

注释：在词法分析之前有一段的注释处理部分，其相应方案是在遇到/符时暂时不进行注释处理，即不判断是注释状态，而后再次遇到/符时候调到注释状态。而后在输入任何字符的时候，始终把状态设为注释状态，直到读进换行状态的时候，才把状态设为初始状态。

关键字：在assignment1中，所需要处理的关键字有六个。int real if then else while 。在进行词法分析的时候，一开始将标识符和关键字设为一类一起读进来。然后在后面会进行一轮标识符和关键字的判断，如果不是int real if then else while 这六个关键字之一的就分类到标识符中，反之输出相应的关键字。

标识符：由于关键字也是一些字符串，所以一般放在一起来判断，若不是关键字则为标识符。由于关键字和标识符都必须是以字母开头，所以用if为字符来判断，为真接着判断，while循环来识别完整个标识符或关键字的部分与识别字符串相同，也是装入字符数组，但比字符串部分多出一点就是标识符可以有下划线，所以中间判断条件会变成数字字符下划线均可。While循环结束后需要在字符数组的最后一位后加上‘\0’。识别完毕后就需要判断该字符串到底是标识符还是关键字了，关键字个数是固定的，挨个匹配就可以了。

运算符：判断方法简单粗暴，直接switch即可，不过要注意比如》=这种需要多判一次，即单词读取应向后延伸一位。所以在构造DFA词法表的时候，我们把界符分为两类，一类是一元运算符，一类是二元运算符。在读入一个字符的时候，如果该字符是运算符，则直接归类为运算符，并且将状态调整。如果输入到一个疑似二元运算符的字符，如>, |, <等字符，则将状态置为“一元运算符（可拓展”。

数字：数字的识别，首先，识别数字直接进行if比较，接着同字符串一样进入循环识别整个数字，都存入整形数组。由于要存入常数表的应该是一个数而不是整形数组，所以应做出一个变量用于计算置位，大致思想就是用循环来将数组中的数乘10，数组的第一位先乘10，每循环一次增加数组下一位。倘若在识别了若干个数字的过程中，突然进来了一个非数字的字母，则对该token进行错误处理。

将整数部分都识别完了后就要判断是否为小数或者科学计数法，若ch为小数点就进入小数的判断部分，小数部分从数组转变为小数的程序就是将上述的函数中的乘便为除，然后再识别符号后面的数，转化为整数，将之前识别出来的数除或乘该整数次数的10，而后用数字转字符串函数存入二元式中，二元式种类写一个常数种别码。

倘若是判断到E或者e的科学计数法字符，则将后面的数字当做for循环的次数做一次for循环，每做一次for循环乘以（或者除以）10。当然，E字符后面只能跟+（或者-）号和digit数字，倘若跟了别的（例如小数点）等字符也需要进行词法报错处理。

界符：最后是界符的识别。也就是类似于{、}、(、)、;等等界符符号的判断。界符判断较为简单，只要处理的时候碰到界符符号，就将前面缓冲字符当做一个token输出，并且将状态置为“分隔符状态”。

4.1.2 扫描设计

扫描设计的基本思路是：将文件里面的字符一个一个读进程序中，然后程序根据DFA表配置文件，对状态进行跳转。DFA词法表在4.1中已经给出。我们可以看到，在词法表中，有一些状态前面含有p字母，代表着需要将前面缓存好的字符串当成一个token进行处理；有一些状态前面含有e字母，代表需要进行出错处理。当读入字符时，按照目标状态跳转到相应状态位置，通过Integer.parseInt(action)函数将表中的相应状态继续处理字符；如果遇到p开头的目标状态，首先执行printQuad();函数将前面缓冲的token输出。如果遇到e开头的目标状态，执行error();进行错误处理。



4.1.3 构词缓冲设计

StringBuffer buffer=new StringBuffer();//构建buffer

首先构造一个缓冲区，存放还没有解析为token的已读字符。

public void setPosition(int line,int offset){

quad.setLines(line);

quad.setOffset(offset);

buffer.delete(0,buffer.length());

}//清除缓冲区

如果遇到了类似空格或者其他可以确定分割token的字符，则将buffer中的字符转换成token，并且清除。

if(!Utils.isBlank(ch)&&!Utils.isNewLine(ch)){

buffer.append((char)ch);

}//加入字符

如果读入不是空格或者换行符的字符，一律先放到缓冲区中存储。

void error(){

stderr.add("ERROR"+LineInfo()+":"+buffer.toString());

buffer.delete(0,buffer.length());

}//错误输出

如果处理过程中判断有错误发生，则将缓冲区中的所有字符清除掉，并且输出错误日志。



状态转换的过程中，需要结合DFA词法表对buffer中的字符串判断再删除更新。具体实现如上图。

4.1.4 符号表建立与更新

# 7.附录

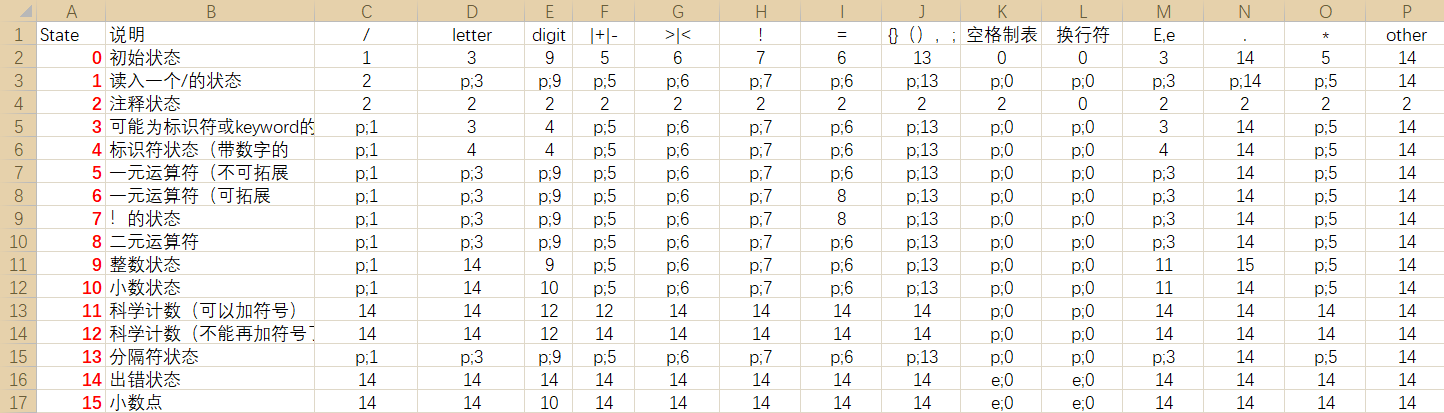
## 源代码：

### 7.1可执行文件链接

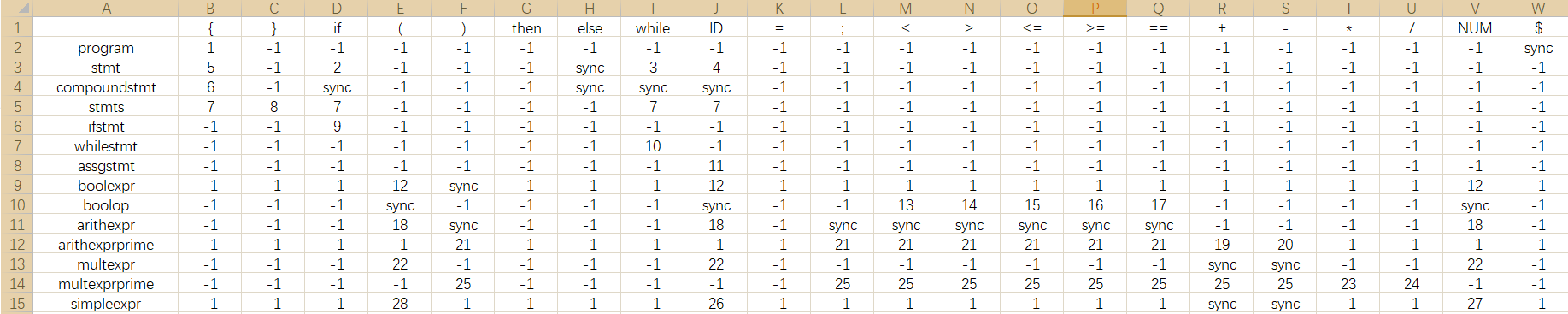
项目文件已上传至github，可直接下载阅读，下面附上github链接地址。

<https://github.com/trayfour34/Compile>

词法分析表如下（注：分析表过大，建议直接用excel打开观看）



语法分析表如下（注：分析表过大，建议直接用excel打开观看）



完成版文法文件如下：

[编译器\完成版文法.docx](编译器/完成版文法.docx)

源文件可执行文件链接如下

# 

# 8、参考文献

1、美 Alfred V.Aho Ravi Sethi Jeffrey D. Ullman著.李建中，姜守旭译.《*编译原理*》.北京：机械工业出版社.2003.

2、美 Kenneth C.Louden著.冯博琴等译.《*编译原理及实践*》.北京：机械工业出版社.2002.

3、金成植著.《*编译程序构造原理和实现技术*》. 北京：高等教育出版社. 2002.