袁和金 黄建才 黄志强 编著

编译技术实践教程

清华大学出版社

内 容 简 介

本教材分为两章。第 1 章介绍编译程序的结构、各阶段的实现原理和构造编译程序的方法。第 2 章介绍编译程序的实验内容和实现方法。本教材给出的实验安排具有普遍性，各学校可根据自己的教学目标和学生水平酌情调节，以适应实际情况。

本书可作为大学编译技术课程设计的指导教材，也可作为编译技术或编译原理课程的配套教材。

图书在版编目（CIP）数据

编译技术实践教程 / 袁和金，黄建才，黄志强编著．—北京：清华大学出版社，2012.1
ISBN 978-7-302-27131-4

Ⅰ. ①编…　Ⅱ. ①袁…②黄…③黄…　Ⅲ. ①编译程序－程序设计－教材　Ⅳ. ①TP314

中国版本图书馆 CIP 数据核字(2011)第 213112 号

责任编辑：汪汉友　李玮琪
责任校对：胡伟民
责任印制：杨　艳

出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京鑫海金澳胶印有限公司
经　　销：全国新华书店
开　　本：185×260　　**印　张**：4.75　　**字　数**：117 千字
版　　次：2012 年 1 月第 1 版　　**印　次**：2012 年 1 月第 1 次印刷
印　　数：1～3000
定　　价：12.00 元

产品编号：037583-01

前　言

为了加强学生对编译技术相关知识的理解，建立起理论和实践的联系，提高学生的动手能力，我们编写了这本实践指导教材。本教材力图让学生从抽象的理论知识中解脱出来，通过介绍编译程序各模块的构造过程和实现例程，使学生理解和巩固编译技术的理论体系，掌握实现方法。

本教材分为两章。第1章给出了编译程序的结构、各阶段的实现原理和构造编译程序的方法。本部分是对编译技术的高度概括，从实践的角度出发，给出各个模块的实现思路，为第2章的实现过程提供理论和技术上的支持。经典的编译程序包括词法分析、语法分析、语义分析与中间代码生成、优化、目标代码生成五个阶段，而各个阶段又有其独特的理论体系和实现步骤。本章通过浓缩各个阶段的主要内容，使学生从更高的层次把握各阶段的核心，明确实现各个阶段的思路。

第2章给出了编译程序的实验内容和实现方法。本章在第1章的基础上，从实践的角度详细给出了各个阶段的实验要求和伪码实现，使学生从代码的角度理解编译程序的理论知识，并提高动手能力。本章首先介绍进行编译技术实验所需的准备知识，然后给出了词法分析、语法分析、语法制导翻译、代码生成各个阶段的实验要求、实现算法和上机安排，使学生明确实验的过程。为了突出编译技术的主要特点，简化设计过程，本章最后还给出了解释程序的实现过程。

本教材总结了多位教师多年的教学经验，并参考了国内外成功案例，从学生的角度出发介绍编译程序的构造方法，力图通过浅显易懂的讲解，使学生乐于接受所学内容，从而巩固所学的编译技术相关知识。

本教材给出的实验安排具有普遍性，各学校可根据自己的教学目标和学生水平酌情调节，以适应实际情况。

本书可作为大学编译技术课程设计的指导教材，也可作为编译技术或编译原理课程的配套教材。

由于作者水平有限，书中难免有不足之处，恳请广大读者批评指正。

作　者

2011年6月

目　录

第 1 章　课程的主要内容

1.1　概述

高级程序设计语言是 20 世纪 50 年代发展起来的编程工具，比起汇编语言、机器语言来，它比较接近于人类的自然语言，在一定程度上与具体的计算机无关，具有易学、易用、易交流、易维护的优点，所以很快得到了广泛的应用。世界上已有的高级程序设计语言有很多种，目前常用的主要有 FORTRAN、Pascal、BASIC、C、C++ 和 Java 等。

用高级程序设计语言编写程序方便，编程效率高。但由于计算机不能直接识别和执行用高级语言编写的程序，因此需要一个翻译程序将其翻译成计算机能够执行的二进制编码（称为机器语言程序）。一般说来，翻译程序是指这样一个程序：它能够将某种语言程序（称为源程序）转换成另一种语言程序（称为目标程序），而这两种语言程序在逻辑上是等价的。

从实用的角度出发，可以将翻译程序分成三 3 种：编译程序、解释程序和汇编程序。编译程序是将高级语言程序翻译成汇编语言程序或者机器语言程序的处理系统；解释程序是按高级语言程序的语句执行次序边翻译边执行相应功能的处理系统，并不产生输出代码；汇编程序是将汇编语言程序翻译成机器语言程序的处理系统，汇编语言的语句与机器语言的指令一般具有一一对应的关系。由于编译程序的构造与实现技术同样适用于解释程序和汇编程序，所以一般不对解释程序和汇编程序做单独讨论。

根据不同的用途和侧重点，编译程序可进一步分为下面几类：用于帮助程序开发和调试的编译程序称为诊断编译程序；着重于提高目标代码效率的编译程序称为优化编译程序；若一个编译程序能够在运行它的计算机上产生可在其他计算机上运行的目标语言程序，则称该编译程序为交叉编译程序。

在计算机上执行一个高级语言程序一般分为两步，首先使用编译程序将高级语言程序翻译成汇编语言程序或者机器语言程序（如果是汇编语言程序，还需要使用汇编程序将其翻译成机器语言程序），然后运行所得的机器语言程序，以求得计算结果。

编译程序设计是一门理论性很强，实践性也很强的技术。通过课堂教学学习到的关于编译程序设计的基本原理和实现技术，常常是抽象的、孤立的，难以建立编译程序的整体概念，所学知识不能有机地结合在一起。编译技术上机实验是解决上述问题的最好方法，通过上机实验具体实现几种常用的编译技术，最好还能实现一个小而完整的编译程序（或解释程序），这样除了能够加深对编译技术的理解以外，在程序设计能力方面也将会有很大的提高。

1.2　编译程序的组成

由于高级程序设计语言程序（源程序）与汇编语言程序或机器语言程序（目标程序）的形式差别太大，往往是一条高级程序设计语言语句对应很多条机器语言语句，因此，编译程序所做的工作非常复杂。但是就其过程而言，它与自然语言之间的翻译很相似。当我们将一

段英文翻译为中文时，对于每个语句通常需要经过以下步骤：

(1) 识别出句子中的一个个单词；

(2) 分析句子的语法结构；

(3) 根据句子的含义进行初步翻译；

(4) 对译文进行修饰；

(5) 写出最后的译文。

编译程序的工作过程也可以类似地划分为 5 个阶段：词法分析、语法分析、语义分析与中间代码产生、中间代码优化、目标代码生成。每个阶段的工作由一段程序完成，分别称为词法分析器、语法分析器、语义分析与中间代码产生器、中间代码优化器、目标代码生成器，它们之间的关系如图 1-1 所示。

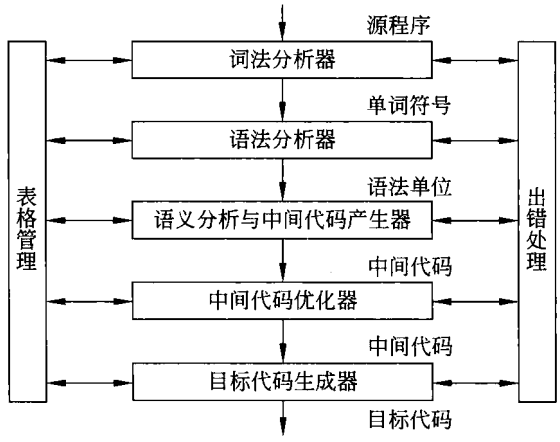

图 1-1　编译程序总框图

编译程序以源程序作为输入，通过上述 5 个阶段的加工处理，最终翻译出目标代码。编译程序的整个处理过程就像一条加工生产线，每个阶段完成特定的工作，由上一阶段的输入产生一种内部形式的代码作为中间结果输出，成为下一阶段的输入。在这个过程中，伴随着表格管理程序和错误处理程序的运行，它们分别负责源程序中各种名字的信息管理以及语法错误、语义错误的处理。

上面是按逻辑功能将编译程序分为 5 个阶段，具体实现时也可从方便实现的角度将编译程序分为几“遍”。所谓“遍”，是指对源程序或源程序的内部形式的代码(中间结果)从头到尾扫描加工一次，每一“遍”完成部分阶段或几个阶段的工作。一个编译程序分成几“遍”，与源语言、设计要求、硬件设备条件等因素有关，应该根据具体情况分析确定。

1.2.1　词法分析

词法分析属于编译程序的第一阶段，由扫描器(或称词法分析器)完成，它的工作是：将输入的源程序当做字符串进行扫描和分解，识别出其中的一个个单词(也称为单词符号或简称符号)，并将这些单词符号串作为输出。可以看出，词法分析器的输入和输出在逻辑上是等价的，只是表示形式有所不同。词法分析从理论到实现上，都已经很成熟，一般它的分析算法可以形式化地证明其正确性，甚至可以自动构造分析器。

1. 单词的种类

单词符号是程序设计语言中具有独立意义的基本符号，一般可以分为下列 5 种。

(1) 基本字

基本字也称关键字或保留字，是程序设计语言定义的具有特定意义的名字，它通常标识源程序中语法成分的功能、开始或结束，例如 PASCAL 语言中的 for、or、while、real、begin、end 等。为了使得源程序易读，提高词法分析的效率，基本字一般不被用户另行定义为标识符。

(2) 标识符

用户在设计源程序时自己定义的名字，一般用来表示变量名、数组名、过程名等。

(3) 常数

程序设计语言或用户在设计源程序时定义的常数，常数的类型一般有整型、实型、布尔型、字符及字符串型等，例如 123、3.14159、.TURE、'A'、"ABC"。

(4) 运算符

程序设计语言定义的运算符号，一般有算术运算符、逻辑运算符和关系运算符等，例如 +、-、*、/、OR、NOT、=、<。

(5) 分界符

程序设计语言定义的符号，一般有标点符号及一些特殊符号，例如","、"."、";"、"("、"/*"。

2. 单词的内部表示

词法分析器将代表源程序的单词符号串输出，作为语法分析器的输入，需要有一种内部表示形式，其形式为二元式(单词种别，单词的属性值)。其中单词种别表示单词是哪一种；单词的属性值表示单词是某一种别中的哪一个，又称为单词的自身值。

单词种别一般用整数编码来表示，这里单词种别并不是指前述单词符号的 5 个种类，而是根据编译程序在技术处理上的具体需要自行决定的。例如，基本字可以作为一种，也可以一字作为一种；标识符可以作为一种，也可以分为变量名、数组名和过程名 3 种；运算符可以一字作为一种，也可以按性质分为几种；分界符可以一字作为一种等等。

如果单词一个种别只有一个，则单词的属性值也就失去了作用；若一个种别含有多个单词，则要给出单词的属性值。单词的属性值是指单词符号的特性。例如，对于标识符或常数，通常将保存其信息的符号表中的登记项指针值(在符号表中的相对地址)作为属性值，这样在编译程序后期对其引用时，就极大地加快了查找符号表的速度(可以直接访问)。

3. 词法分析器作为一个独立的子程序

把词法分析器设计为语法分析器的一个子程序，可以避免在存储器中保存代表整个源程序的单词符号串，使得编译程序的结构更简洁、清晰和条理化。每当语法分析器需要一个单词时就调用这个子程序，词法分析器就从源程序字符串中识别出一个单词符号，交给语法分析器。下文关于词法分析器设计的阐述中，都假定它是按这种方式工作的。

4. 词法分析器的设计

词法分析器的结构如图 1-2 所示。首先将源程序字符串读入到输入缓冲区，经预处理子程序预处理一定长度的字符串后送入扫描缓冲区，扫描器从扫描缓冲区识别出一个个单词，在扫描缓冲区中的字符串处理完后，再调用预处理子程序将一段新的源程序字符串送入

扫描缓冲区。其中，预处理子程序的工作是删除源程序中无用的空格、回车符、换行符和注释等。这些字符对执行源程序没有任何作用，删除后可以使得单词、语句的组成更加有规律，从而便于词法和语法分析的实现。

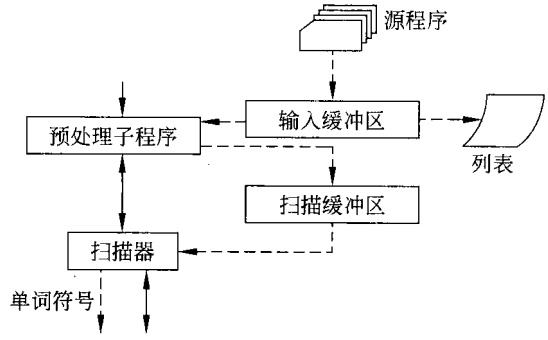

图 1-2　词法分析器的结构

设计词法分析器的一种非常好的方法是使用状态转换图。状态转换图既可以描述单词的结构，又能够识别(或接受)单词。用程序实现了识别单词的状态转换图，也就得到了词法分析器。基于状态转换图的词法分析器的设计方法如下。

(1) 构造状态转换图

先从描述单词结构的文法出发，构造一个描述单词结构的状态转换图。形式语言与有限自动机理论的研究表明，目前所遇到的程序设计语言的单词都能够用状态转换图描述，它具有一个初态和多个不同的终结状态，这些终结状态分别对应不同种别的单词。

(2) 实现状态转换图

在一个描述单词结构的状态转换图上，从初态出发经过状态转换若能到达终态，则识别了一个单词。设计一个程序实现状态转换图的功能，即得到了词法分析器。状态转换图与程序有一定的对应关系，其具体做法是：对每一个状态设计一段程序，这段程序负责完成接受当前输入字符后所做的工作；再设计一个控制程序，它负责根据当前状态和当前输入的字符决定执行哪一段程序。

1.2.2　语法分析——自上而下分析

语法分析属于编译程序的第二阶段，由语法分析器完成。它的工作是：在词法分析器识别出单词符号串的基础上，分析并判别源程序的语法结构是否符合语法规则。这里所说的语法规则是指上下文无关文法(以下简称文法)。给出一个单词符号串后，语法分析器判别它是不是合乎语法规则的句子，如果是合乎语法规则的句子则构造它的语法树，否则指出语法错误。

语法分析从理论到实现都已经很成熟，语法分析程序的基本构成是一个分析表和一个实现分析算法的程序(需使用分析栈)。一般来说，可以形式化地证明分析算法的正确性，并自动构造其分析表。按照语法树的构造方法，语法分析可以分成两大类，一类是自上而下的分析方法，另一类是自下而上的分析方法，它们都是从左向右扫描输入串(单词符号串)，逐步建立输入串的语法树。

1. 自上而下语法分析方法介绍

自上而下语法分析法是从文法开始符号(作为根结点)出发,自上而下,从左向右地构造输入串的语法树,它等同于建立一个最左推导。

自上而下语法分析的一种一般方法是带"回溯"的分析,即对一个输入串,从文法开始符号出发,试图用一切可能的方法选择候选式,去构造输入串的语法树。对于一个非终结符号,当选择它的一个候选式去匹配输入串的最左子串时,如果不成功就会产生"回溯",然后再尝试选择其他候选式去匹配。由于工作中的这种试探性而产生的"回溯"给语法分析的实现(以至于翻译的实现)带来了很大困难,因此这种方法并不实用,一般来说,只是在理论上有一定价值。

一种实用的分析方法是不带"回溯"的自上而下语法分析——$LL(1)$分析方法。这种方法从文法开始符号出发构造输入串的语法树时,要求每个非终结符号都能够针对当前输入符号准确地做出选择,即如果能选择出合适的候选式(是唯一的),则用它去发展语法树;如果选择不出合适的候选式,则说明输入串不是句子,有语法错误。实现 $LL(1)$语法分析方法对文法有一定的要求,要求文法是 $LL(1)$文法。

2. $LL(1)$文法

如果一个文法 G 满足以下 3 个条件,则称该文法 G 为 $LL(1)$文法。

(1) 文法不含左递归;

(2) 对文法中每一个非终结符 A 的任何两个不同的侯选式的终结符首符集不相交;

(3) 对文法中每一个非终结符 A,若它存在某个侯选式的终结符首符集包含 ε,则 $\mathrm{FIRST}(A)\cap\mathrm{FOLLOW}(A)=\Phi$。

其中,FIRST(A)和 FOLLOW(A)的定义如下:

设 A 是文法 G 的一个符号串,$A\in(V_N\cup V_T)^*$,定义:

$$\mathrm{FIRST}(A)=\{c\mid A\overset{*}{\Rightarrow}cB,c\in V_T,B\in(V_N\cup V_T)^*\}$$

特别地,若有 $A\overset{*}{\Rightarrow}\varepsilon$,则 $\varepsilon\in\mathrm{FIRST}(A)$,即 FIRST($A$)是从 A 可推导出的所有首终结符或可能的 ε。

设 S 是文法的开始符号,$A\in V_N$,定义:

$$\mathrm{FOLLOW}(A)=\{b\mid S\overset{*}{\Rightarrow}xAby,b\in V_T,x,y\in(V_N\cup V_T)^*\}$$

若 $S\overset{*}{\Rightarrow}xA$,则规定 $\#\in\mathrm{FOLLOW}(A)$,即 FOLLOW(A)是文法的所有句型中紧接在 A 之后出现的终结符或 $\#$。

例 1:设有 $LL(1)$文法 G 如下:

(1) $S\to TE'$;

(2) $E'\to +TE'$;

(3) $E'\to\varepsilon$;

(4) $T\to FT'$;

(5) $T'\to *FT'$;

(6) $T'\to\varepsilon$;

(7) $F\to(E)\mid id$;

构造各非终结符的 FIRST 和 FOLLOW 集。

解答：根据上述 FIRST 和 FOLLOW 的构造方法，有：

$$\text{FIRST}(E)=\text{FIRST}(T)=\text{FIRST}(F)=\{(,id\}$$
$$\text{FIRST}(E')=\{+,\varepsilon\}、\text{FIRST}(T')=\{*,\varepsilon\}$$
$$\text{FOLLOW}(E)=\text{FOLLOW}(E')=\{),\#\}$$
$$\text{FOLLOW}(T)=\text{FOLLOW}(T')=\{+,),\#\}$$
$$\text{FOLLOW}(F)=\{+,*,),\#\}$$

对于 $LL(1)$文法，可以实现不带“回溯”的自上而下语法分析，其常用的实现方法有预测分析法和递归下降分析法。

3. 预测分析法

如果一个文法是 $LL(1)$文法，则可以构造不带“回溯”的自上而下语法分析程序——预测分析程序。构造预测分析程序需要做两项工作：构造一个预测分析表，称为 $LL(1)$分析表；设计一个总控程序实现预测分析算法。一般来说，不同的文法对应不同的预测分析表，而总控程序则是一个通用程序，它适用于一类文法——$LL(1)$文法。

(1) 预测分析表

它是一个 $M[A,a]$形式的二维矩阵，其中 A 为非终结符号，a 为终结符号或“#”。可将“#”看做是一个特殊的终结符号，它标记输入串的开始和结尾。预测分析表 M 有 $m\times n$ 个元素，其中 m 是文法中非终结符号的个数，n 是终结符号的个数+1。矩阵的元素 $M[A,a]$中可能存放着一条关于非终结符号 A 的产生式 $A\to\alpha$(或侯选式 α)，它指出当使用 A 去发展语法树(匹配当前输入串的最左子串)而面临的当前输入符号为终结符号 a 时所采用的侯选式；$M[A,a]$也可能为空(为了便于检查语法错误，可以在此位置填上“出错标志”)，它指出，输入串如果是句子，当使用 A 去发展语法树时，不可能面临输入符号 a，所以输入串有语法错误，不是句子。

给出一个文法 G，其终结符号集合为 V_T，非终结符号集合为 V_N，构造相应文法 G 的预测分析表，需要先构造 FIRST 集、FOLLOW 集，再构造预测分析表，然后判别文法 G 是否为 $LL(1)$文法。

① 对文法 G 中每个文法符号 $X\in V_T\cup V_N$ 构造 FIRST(X)，其构造方法是：反复使用如下规则，直至 FIRST 集合不再扩大为止。

- 若 $X\in V_T$，则 FIRST$(X)=\{X\}$；
- 若 $X\in V_N$，有产生式 $X\to a\cdots$，则把 a 加入到 FIRST(X)中；
- 若 $X\in V_N$，有产生式 $X\to\varepsilon$，则把 ε 加入到 FIRST(X)中；
- 若 $X\in V_N$，有产生式 $X\to Y_1Y_2\cdots Y_n$，则把 FIRST$(Y_1)-\{\varepsilon\}$加入 FIRST(X)中；若 $\varepsilon\in$FIRST(Y_1)，则把 FIRST$(Y_2)-\{\varepsilon\}$加入 FIRST(X)中；若 $\varepsilon\in$FIRST$(Y_1)\wedge\varepsilon\in$FIRST$(Y_2)$，则把 FIRST$(Y_3)-\{\varepsilon\}$加入 FIRST$(X)$中；依次类推；若 $\varepsilon\in$FIRST$(Y_1)\wedge\varepsilon\in$FIRST$(Y_2)\wedge\cdots\wedge\varepsilon\in$FIRST$(Y_n)$，则把 ε 加入 FIRST(X)中。

② 对于文法 G 的任何一个文法符号串 $\alpha=X_1X_2\cdots X_n$ 构造 FIRST(α)集合，构造方法是：先将 FIRST$(X_1)-\{\varepsilon\}$加入 FIRST(α)；若 $\varepsilon\in$FIRST(X_1)，则把 FIRST$(x_2)-\{\varepsilon\}$加入 FIRST(α)中；若 $\varepsilon\in$FIRST$(X_1)\wedge\varepsilon\in$FIRST$(X_2)$，则把 FIRST$(X_3)-\{\varepsilon\}$加入 FIRST$(\alpha)$中；依次类推；若 $\varepsilon\in$FIRST$(X_1)\wedge\varepsilon\in$FIRST$(X_2)\wedge\cdots\wedge\varepsilon\in$FIRST$(X_n)$，则把 ε 加入 FIRST(α)中。

③ 对于文法 G 的每个非终结符号 $A\in V_N$ 构造 FOLLOW(A)，构造方法是：反复使用

如下规则，直至 FOLLOW 集合不再扩大为止。

- 对于文法开始符号 S，将 # 加入到 FOLLOW(S)中；
- 若有产生式 $A\rightarrow\alpha B\beta$，则把 FIRST($\beta$)－{$\varepsilon$}加入到 FOLLOW($B$)中；
- 若有产生式 $A\rightarrow\alpha B$ 或者有产生式 $A\rightarrow\alpha B\beta$ 且 $\varepsilon\in$FIRST(β)，则把 FOLLOW(A)加入到 FOLLOW(B)中。

④ 构造文法 G 的预测分析表 M。对于文法 G 的任何一个产生式 $A\rightarrow\alpha$ 构造出集合 FIRST(α)，每个非终结符号 A 构造出 FOLLOW(A)后，就可以构造文法 G 的预测分析表 M。构造算法是，对于文法 G 中的每个产生式 $A\rightarrow\alpha$ 执行以下步骤：

- 对每个终结符号 $a\in$FIRST(α)，把 $A\rightarrow\alpha$ 加入 $M[A,a]$中；
- 若 $\varepsilon\in$FIRST(α)，则对任何 $b\in$FOLLOW(A)，把 $A\rightarrow\alpha$ 加入 $M[A,b]$中；
- 把所有无定义的 $M[A,a]$标上“出错标志”。

⑤ 判别文法 G 是否为 $LL(1)$文法。

如果一个文法 G 的预测分析表 M 不含多重定义入口(即任何 $M[A,a]$的值都是一个产生式，或者是“出错标志”)，则称 G 为 $LL(1)$文法。由于直接使用 $LL(1)$文法的定义判别一个文法是否为 $LL(1)$文法比较困难，因此，通常可以用构造预测分析表 M 的方法来判别。

例 2：对如下的 $LL(1)$文法：

(1) $S\rightarrow A$；

(2) $A\rightarrow BA'$；

(3) $A'\rightarrow iBA'|\varepsilon$；

(4) $B\rightarrow CB'$；

(5) $B'\rightarrow+CB'|\varepsilon$；

(6) $C\rightarrow)A*|($；

构造分析表。

解答：对于产生式 $S\rightarrow A$，因为 FIRST(A)＝{(,)}，所以 $M[S,(]=S\rightarrow A$，$M[S,)]=S\rightarrow A$。

对于产生式 $A\rightarrow BA'$，因为 FIRST(B)＝{(,)}，所以 $M[A,(]=A\rightarrow BA'$，$M[A,)]=A\rightarrow BA'$。

对于产生式 $A'\rightarrow iBA'$，因为 FIRST(iBA')＝{i}，所以 $M[A',i]=A'\rightarrow iBA'$。

对于产生式 $A'\rightarrow\varepsilon$，因为 FOLLOW(A')＝{*，#}，所以 $M[A',*]=A'\rightarrow\varepsilon$，$M[A',\#]=A'\rightarrow\varepsilon'$。

对于产生式 $B\rightarrow cB'$，因为 FIRST(C)＝{(,)}，所以 $M[B,(]=B\rightarrow CB'$，$M[B,)]=B\rightarrow CB'$。

对于产生式 $B'\rightarrow+CB'$，因为 FIRST($+CB'$)＝{+}，所以 $M[B',+]=B'\rightarrow+CB'$。

对于产生式 $B'\rightarrow\varepsilon$，因为 FOLLOW(B')＝{i，*，#}，所以 $M[B',i]=B'\rightarrow\varepsilon$，$M[B',*]=B'\rightarrow\varepsilon$，$M[B',\#]=B'\rightarrow\varepsilon$。

对于产生式 $C\rightarrow)A*$，因为 FIRST()$A*$)＝{)}，所以 $M[C,)]=C\rightarrow)A*$。

对于产生式 $C\rightarrow($，因为 FIRST(()＝{(}，所以 $M[C,(]=C\rightarrow($。

该文法的分析表如表 1-1 所示。

表 1-1　分析表 M

非终结符号	终结符					
	i	+	*	(	)	#
S				$S\rightarrow A$	$S\rightarrow A$	
A				$A\rightarrow BA'$	$A\rightarrow BA'$	
A'	$A'\rightarrow iBA'$		$A'\rightarrow\varepsilon$			$A'\rightarrow\varepsilon$
B				$B\rightarrow CB'$	$B\rightarrow CB'$	
B'	$B'\rightarrow\varepsilon$	$B'\rightarrow+CB'$	$B'\rightarrow\varepsilon$			$B'\rightarrow\varepsilon$
C				$C\rightarrow($	$C\rightarrow)A*$	

(2) 总控程序

程序使用的主要数据结构是一个分析栈 STACK，用于存放文法符号，它的栈顶元素总保持当前发展语法树的文法符号(匹配当前输入串最左子串的文法符号)，在分析开始时，栈底先放入一个“#”，然后放入文法开始符号 S，表示从 S 出发去发展语法树(匹配整个输入串)。同时在输入串最后边加入一个“#”，标识输入串的结束。当 STACK 栈顶文法符号为 X，而当前输入符号为 a 时，总控程序执行下述三种可能的动作之一：

① 若 $X=a='\#'$，则分析成功，输入串是句子，分析结束。

② 若 $X=a\neq'\#'$，则 X 从 STACK 栈顶出栈，让 a 指向下一个输入符号。

③ 若 X 是一个非终结符号，则查看分析表 M。若 $M[X,a]$中存放着一条关于 X 的产生式 $X\rightarrow X_1X_2\cdots X_n$，则 X 从 STACK 栈顶出栈，然后，产生式的右部 $X_1X_2\cdots X_n$ 按反序进栈(产生式的右部若是 ε，则无符号进栈)；若 $M[X,a]$为出错标志，则输入串不是句子，转向出错处理程序。

例 3：按照 $LL(1)$分析器总控算法，对上例中的文法符号串'$i($'进行分析。

解答：分析过程如表 1-2 所示。

表 1-2　符号串分析过程

步　骤	符号栈 S[i]	输入串 str[j]	产生式
1	#S	i(#	
2	#A	i(#	$S\rightarrow A$
3	#A'B	i(#	$A\rightarrow BA'$
4	#A'B'C	i(#	$B\rightarrow CB'$
5	#A'B'(	i(#	$C\rightarrow($
6	#A'B'	i(#	
7	#A'	i(#	$B'\rightarrow\varepsilon$
8	#A'Bi	i(#	$A'\rightarrow iBA'$
9	#A'B	(#	

续表

步　骤	符号栈 S[i]	输入串 str[j]	产生式
10	#$A'B'C$	(#	$B\rightarrow CB'$
11	#$A'B'$(	(#	$C\rightarrow($
12	#$A'B'$	#	
13	#A'	#	$B'\rightarrow\varepsilon$
14	#	#	$A'\rightarrow\varepsilon$

4. 递归下降分析法

如果一个文法是 $LL(1)$ 文法，可以构造不带“回溯”的自上而下语法分析程序——递归下降分析程序。这个分析程序由一组递归过程组成(因为文法是递归的)，每个非终结符对应一个过程，进行语法分析是调用对应文法开始符号 S 的过程。

递归下降分析法是在求出文法中每个候选式的 FIRST 集以及每个非终结符号的 FOLLOW 集的基础上进行的。其处理思想是，根据分析的进展和当前的输入符号确定下一步动作，如果用于匹配当前输入串的最左子串的是非终结符号 A，则使用 A 的候选式的 FIRST 集或 A 的 FOLLOW 集，根据当前输入符号选择一个合适的候选式去发展语法树；如果用于匹配当前输入串的最左子串的是终结符号 a，则用 a 自身去匹配当前输入符号。

如果使用的程序设计语言允许递归，则递归下降分析器实现起来很方便，但是其运行效率较低。

1.2.3 语法分析——自下而上分析

1. 自下而上语法分析方法介绍

自下而上语法分析是从左向右地读入输入串，从叶结点开始，逐步向上构造语法树。或者说，逐步进行“归约”，直至归约到文法的开始符号 S。自下而上语法分析通常是一个最右推导的逆过程——最左归约。

自下而上语法分析的一般方法是规范归约的自下而上语法分析，即对一个输入串假设它是句子，找出它的“句柄”，然后，使用一个产生式将其归约为一个非终结符号，得到一个新的句型。反复执行以下步骤：找出当前句型的“句柄”并使用产生式进行“归约”，直至得到文法的开始符号 S，则说明输入串是句子。在实现这种分析方法时，因找“句柄”的方法不同而派生出不同的自下而上语法分析方法，常用的实现方法有算符优先分析法和 LR 分析法。

2. 算符优先分析方法

算符优先分析法的运行效率很高，特别适用于各类表达式的语法分析，如果对文法稍加修改，也适用于复杂的程序设计语言。构造算符优先分析程序需要做两项工作：构造一个算符优先分析表和设计一个算符优先分析程序。一般来说，不同的文法对应不同的算符优先分析表，而算符优先分析程序是一个通用程序，它适用于一类文法——算符优先文法。

(1) 算符优先文法

假设文法 G 是一个不含 ε 一产生式的算符文法，对于任何一对终结符号 (a,b) 定义如下优先关系：

① $a \doteq b$：当且仅当文法 G 中含有形如 $P \to \cdots ab \cdots$ 或 $P \to \cdots aQb \cdots$ 的产生式；

② $a \lessdot b$：当且仅当文法 G 中含有形如 $P \to \cdots aR \cdots$ 的产生式，而 $R \overset{+}{\Rightarrow} b \cdots$ 或 $R \overset{+}{\Rightarrow} Qb \cdots$；

③ $a \gtrdot b$：当且仅当文法 G 中含有形如 $P \to \cdots Rb \cdots$ 的产生式，而 $R \overset{+}{\Rightarrow} \cdots a$ 或 $R \overset{+}{\Rightarrow} \cdots aQ$。

如果一个文法 G 中的任何终结符号对 (a,b) 至多满足以上三种优先关系之一，则称 G 为算符优先文法。

(2) 算符优先关系表

算符优先关系表是一个 $F[a,b]$ 形式的二维矩阵，其中 a、b 为文法 G 中的任何终结符号或“#”。这里将“#”看做一个特殊的终结符号，它标记输入串的开始和结尾，预测分析表 F 有 $n \times n$ 个元素，n 是文法 G 的终结符号个数 $+1$。矩阵的元素 $F[a,b]$ 中存放着终结符号对 (a,b) 之间的优先关系。

给出一个不含 ε 一产生式的算符文法 G，构造相应的算符优先关系表需要先构造 FIRSTVT 集合、LASTVT 集合，再构造算符优先关系表，然后判别文法 G 是否为算符优先文法。

① 构造 FIRSTVT 集合

对算符文法 G 中每一个非终结符号 $P \in V_N$ 构造 FIRSTVT(P)，构造方法是：反复使用如下规则，直至 FIRSTVT 集合不再扩大为止。

- 若有产生式 $P \to a \cdots$ 或者 $P \to Qa \cdots$，$a \in V_T$，$Q \in V_N$，则把 a 加入到 FIRSTVT(P)中；
- 若有产生式 $P \to Q \cdots$，$Q \in V_N$，则把任何 $a \in$ FIRSTVT(Q)加入到 FIRSTVT(P)中。

② 构造 LASTVT 集合

对算符文法 G 中每个非终结符号 $P \in V_N$ 构造 LASTVT(P)，构造方法是：反复使用如下规则，直至 LASTVT 集合不再扩大为止。

- 若有产生式 $P \to \cdots a$ 或者 $P \to \cdots aQ$，$a \in V_T$，$Q \in V_N$，则把 a 加入到 LASTVT(P)中；
- 若有产生式 $P \to \cdots Q$，$Q \in V_N$，则把任何 $a \in$ LASTVT(Q)加入到 LASTVT(P)中。

③ 构造算符优先关系表

对算符文法 G 的每一个非终结符号 $P \in V_N$ 构造出 FIRSTVT(P)和 LASTVT(P)后，就可以构造 G 的算符优先关系表。构造方法是：对于文法 G 中的每一个产生式，执行以下步骤。

- 若有产生式 $P \to \cdots ab \cdots$ 或者 $P \to \cdots aQb \cdots$，则把 $a \doteq b$ 加入到 $F[a,b]$ 中；
- 若有产生式 $P \to \cdots aA \cdots$，则对于任何 $b \in$ FIRSTVT(A)，把 $a \lessdot b$ 加入到 $F[a,b]$ 中；
- 若有产生式 $P \to \cdots Ab \cdots$，则对于任何 $a \in$ LASTVT(A)，把 $a \gtrdot b$ 加入到 $F[a,b]$中；
- 把所有无定义的 $F[a,b]$标上“出错标志”。

④ 判别文法 G 是否为算符优先文法

在以上构造的算符优先关系表中，如果任何元素 $F[a,b]$的值是三种优先关系之一，或者是“出错标志”，则文法 G 是算符优先文法，可以用于算符优先分析。

例 4：对文法：$S \to (R) \mid a \mid b$、$R \to T$、$T \to S; T \mid S$ 构造其 FIRSTVT、LASTVT 集合和优先关系表，并说明它是否为算符优先文法。

解答：根据 FIRSTVT、LASTVT 集合和优先关系表的构造方法可得：

FIRSTVT(S)＝{(,a,b}、FIRSTVT(T)＝FIRSTVT(R)＝{(,a,b,;}

LASTVT(S)＝{a,b,)}、LASTVT(R)＝LASTVT(T)＝{a,b,);}，其算符优先表如表 1-3 所示。

表 1-3　优先关系表

	(	)	a	b	;	#
(	⋖	≐	⋖	⋖	⋖	
)		⋗			⋗	⋗
a		⋗			⋗	⋗
b		⋗			⋗	⋗
;	⋖	⋗	⋖	⋖	⋖	
#	⋖		⋖	⋖		≐

该文法没有连续的两个非终结符，而且其中每一对终结符之间最多只有一种优先关系，所以它是一个算符优先文法。

(3) 算符优先分析算法

算符优先分析不是规范归约的自下而上语法分析方法，它每一次归约的不是真正的“句柄”，而是“最左素短语”。“最左素短语”在算符优先分析中起着与“句柄”相同的作用，也正是因为这样，使得算符优先分析方法效率很高。素短语是一个短语，它至少含有一个终结符号，并且除它自身之外不再含有更小的素短语。最左素短语是句型中最左边的素短语。

假设算符优先文法 G 的句型的一般形式为：

$$\#N_1a_1N_2a_2\cdots N_na_nN_{n+1}\#$$

其中，$a_i\in V_T$，$N_i\in V_N$ 或者没有。在一个算符优先文法 G 的句型中，最左素短语是满足如下条件的最左子串：

$$N_ja_j\cdots N_ia_iN_{i+1}$$

$$a_{j-1}\lessdot a_j$$

$$a_j\doteq a_{j+1},\cdots,a_{i-1}\doteq a_i$$

$$a_i\gtrdot a_{i+1}$$

从以上结论可以看出，寻找最左素短语与非终结符号无关，只与文法中终结符号对之间的优先关系有关。实际上，当由文法产生了终结符号对之间的优先关系表之后，文法对于语法分析也就失去了作用。

根据最左素短语在句型中的形式，可以设计出算符优先分析程序。在程序中使用的主要数据结构有一个分析栈 STACK，用于存放文法符号，还有一个寄存器 a，用于存放当前输入符号。

算符优先分析算法：

(1) “#”进入分析栈 STACK；

(2) 重复：当前输入符号读入寄存器 a；

① 若栈顶第一个终结符号$\gtrdot a$，则找到了最左素短语的“尾”，然后转入以下步骤。

从 STACK 栈顶向下找到最左素短语的“头”(即找出满足“$\lessdot$”优先关系的相邻终结符号对);

- 将栈顶的最左素短语归约为一个非终结符号 N。

② 若栈顶第一个终结符号$\lessdot a$或$\doteq a$,则 a 进栈;

③ 否则栈顶第一个终结符号与 a 没有优先关系,则输入串不是句子,转向错误处理程序。

(3) 直到 $a='\#'$,分析结束。

算法结束时,若分析栈 STACK 呈现出“$\#N\#$”,则说明输入串是句子,否则输入串不是句子。

3. *LR* 分析方法

LR 分析属于规范的自下而上语法分析方法,这种方法更具有一般性(对文法限制较少),并且适于自动生成语法分析器。*LR* 分析解决的主要问题是找句柄,在分析过程中使用的主要数据结构是一个分析栈,其处理思想是:在分析中,一方面记住已移进栈和归约出的整个符号串,即记住“历史”;另一方面根据所用的产生式推测未来可能遇到的输入符号,即对未来进行“展望”。当一个像是句柄的符号串呈现在分析栈顶时,可以通过“历史”、“展望”和“现实”的输入符号等三方面的信息,来确定当前分析栈顶的符号串是否构成了句柄。实际上,*LR* 分析器是一个带下堆栈的确定的有限自动机(即确定的下推自动机)。

实现 *LR* 语法分析程序需要做两项工作:构造一个 *LR* 分析表、设计一个 *LR* 分析程序。一般来说,不同的文法对应不同的 *LR* 分析表,而 *LR* 分析程序是一个通用程序,它适用于一类文法——*LR* 文法。

(1) *LR* 分析表与 *LR* 文法

LR 分析表包括两个子表,一个是动作表 ACTION,另一个是状态转换表 GOTO。它们都是二维矩阵。ACTION$[S,a]$规定了状态 S 面临输入符号 a 时应采取什么行动;GOTO$[S,X]$规定了状态 S 面临文法符号 X 时下一个状态是什么。$a_i\in V_T$, $X\in V_T\cup V_N$。

对于一个文法 G,如果其对应分析表的每个入口都是唯一确定的,则称 G 为 *LR* 文法。

(2) *LR*(0)分析表的构造

LR 分析器需要“展望”和检查未来的 k 个输入符号才能决定采取什么样的分析策略。一般来说,对于一个文法 G,如果能用于向前检查 k 个输入符号的 *LR*(k)分析器,则这个文法 G 称为 *LR*(k)文法。对于多数程序设计语言 $k\leqslant 1$ 就够了,在这里仅假设 $k=0$。*LR*(0)分析虽然具有一定的局限性,但它是建立一般 *LR* 分析的基础。

给出一个文法 G 构造相应的 *LR*(0)分析表,需要从其拓广文法 G' 出发,先构造识别规范句型活前缀的有限自动机 FA,再构造 *LR*(0)分析表,然后判别文法 G 是否为 *LR*(0)文法。

① 文法 G 的拓广文法 G'

假设文法 G 的开始符号为 S,构造文法 G',使它包含整个文法 G,并且引进一个新的非终结符号 S'作为 G'的开始符号,再增加一个新的产生式 $S'\rightarrow S$,则 G'称为 G 的拓广文法。

② 规范句型的活前缀

规范句型的活前缀是规范句型的一个前缀,这种前缀不包括句柄之后的任何符号。在其后面添加一些终结符号之后,就可以成为一个规范句型。