普通高校计算机类应用型本科
系列规划教材

编译原理

主编　王一宾　陈义仁

Principles of Compiler

中国科学技术大学出版社

普通高校计算机类应用型本科

系列规划教材

编译原理

主　编　王一宾　陈义仁

副主编（以姓氏拼音为序）

李　立　刘义红　谈成访

吴其林　张春梅

Principles of Compiler

中国科学技术大学出版社

内 容 简 介

本书介绍程序设计语言编译程序构造的一般原理、基本设计方法、主要实现技术和一些自动构造工具。主要内容包括：编译程序概论、文法和语言、词法分析与有限自动机、自上而下语法分析法、自下而上语法分析法、语法制导翻译和语义分析、符号表、运行时的存储组织与管理、代码优化、目标代码生成和现代编译技术概述。主要特色是：突出基础知识和基本理论，强调工程实践与应用，配有丰富的例题和习题，相关章节后配有相应的实验项目。这些特色有助于读者掌握编译程序的基本理论和设计原理，以及应用相关算法解决实际问题的能力培养。

本书可作为高等院校计算机等相关专业的本科或高职高专以及各类培训班的教材，也可作为教师、研究生、软件工程技术人员等的参考书。

图书在版编目(CIP)数据

编译原理/王一宾，陈义仁主编. —合肥：中国科学技术大学出版社，2016.8
ISBN 978-7-312-04026-9

Ⅰ.编… Ⅱ.①王… ②陈… Ⅲ.编译程序—程序设计 Ⅳ.TP314

中国版本图书馆 CIP 数据核字(2016)第 149975 号

出版 中国科学技术大学出版社
安徽省合肥市金寨路 96 号，230026
http://press.ustc.edu.cn
印刷 合肥市宏基印刷有限公司
发行 中国科学技术大学出版社
经销 全国新华书店
开本 787 mm×1092 mm 1/16
印张 17
字数 432 千
版次 2016 年 8 月第 1 版
印次 2016 年 8 月第 1 次印刷
定价 35.00 元

前　言

编译原理是计算机专业的一门核心课程，在计算机本科教学中占有十分重要的地位。编译原理课程具有很强的理论性，读者学习起来普遍感到内容抽象、不易理解。为此，本书采取由浅入深、循序渐进的方式介绍编译原理的基本概念和实现方法。在内容的组织上，将编译的基本理论与具体的实现技术有机地结合起来，既注重理论的完整性，化繁为简，又将理论融于具体的实例中，化难为易，以达到准确、清晰阐述相关概念和原理的目的。除了各章节对理论阐述具条理性之外，书中给出的例子也具有实用性与连贯性，使读者对编译的各个阶段都能有一个全面、直观的认识，从而透彻地领悟编译原理的精髓。本书采用的算法全部用C语言描述。

本书共分11章。第1章简要介绍编译程序的基本概念、工作过程和逻辑结构。第2章主要介绍文法和语言的形式化定义以及Chomsky文法分类等相关内容。第3章主要介绍词法分析器的设计原理，以及有限自动机与正规表达式的相关内容。第4章主要介绍自上而下语法分析法：首先分析自上而下分析法的一般思想及其所遇到的问题，然后介绍这些问题的解决方案，最后介绍两种自上而下语法分析算法——递归下降分析法和LL(1)分析法。第5章主要介绍自下而上语法分析法：首先介绍自下而上分析法的一般思想及其核心问题，然后根据其核心问题的不同解决方案，介绍两种自下而上语法分析算法——算符优先分析法和LR分析法。第6章介绍语法制导翻译与语义分析，包括中间代码产生的有关内容，给出了如何在语法分析的同时进行语义加工并产生中间代码的方法。第7章主要介绍符号表的作用与内容，符号表的组织与管理等相关内容。第8章主要介绍程序运行时存储空间的划分和四种常用的参数传递方式。第9章介绍代码优化的有关内容，主要涉及基本块内的局部代码优化和循环代码优化。第10章讨论目标代码生成的有关内容，讲述如何由中间代码产生最终的目标代码。第11章简要介绍面向对象编译、并行编译和网格计算编译等现代编译技术。

为了便于读者正确理解有关概念，各章还配有一定数量的习题。这些习题大多选自本科生和研究生的考试试题，也包括我们结合多年教学实践经验设计出来的典型范例，力求使读者抓住重点、突破难点，进一步全面、深入地巩固所学知识。

本书第1章、第9章由王一宾编写，第2章、第3章由李立、吴其林编写，第4章、第5章由陈义仁编写，第6章、第11章由刘义红编写，第7章、第8章和第10章由谈成访、张春梅编写，全书由王一宾统稿。

由于编者水平有限，书中难免存在一些疏漏，敬请读者批评指正。

编　者

2016年5月

目　录

前言 …… (i)

第 1 章　编译程序概论 …… (1)
1.1　编译程序的基本概念 …… (1)
1.2　编译程序的工作过程 …… (3)
1.3　编译程序的逻辑结构 …… (7)
1.4*　编译技术应用 …… (9)
1.5　本章小结 …… (14)
习题 1 …… (14)

第 2 章　文法和语言 …… (15)
2.1　符号和符号串 …… (15)
2.2　文法和语言的形式定义 …… (17)
2.3　Chomsky 文法分类 …… (21)
2.4　文法和语言的二义性 …… (23)
2.5　文法的等价及其变换 …… (26)
2.6　本章小结 …… (28)
习题 2 …… (29)

第 3 章　词法分析与有限自动机 …… (31)
3.1　词法分析器的设计思想 …… (31)
3.2　词法分析器的设计 …… (33)
3.3　单词的描述工具 …… (40)
3.4　有限自动机 …… (43)
3.5　正规文法、正规式和有限自动机的等价性 …… (51)
3.6　词法分析器的自动构造工具——LEX …… (57)
3.7　本章小结 …… (63)
习题 3 …… (64)
【实验 1】　词法分析器的设计 …… (65)

第 4 章　自上而下语法分析法 …… (67)
4.1　语法分析的任务和分析方法 …… (67)
4.2　自上而下分析法的基本思想和面临的问题 …… (68)

4.3 左递归和回溯的消除 …… (71)
4.4 LL(1)分析法 …… (75)
4.5 预测分析法 …… (79)
4.6 递归下降分析法 …… (84)
4.7 LL(1)分析中的错误处理 …… (86)
4.8 本章小结 …… (88)
习题 4 …… (88)
【实验 2】 语法分析器设计之一——预测分析器设计 …… (90)

第 5 章 自下而上语法分析法 …… (92)
5.1 自下而上分析法的一般思想和面临的问题 …… (92)
5.2 算符优先分析法 …… (100)
5.3 LR 分析法 …… (111)
5.4 语法分析器的自动产生工具——YACC …… (147)
5.5 本章小结 …… (149)
习题 5 …… (150)
【实验 3】 语法分析器设计之二——算符优先分析器设计 …… (153)
【实验 4】 语法分析器设计之三——LR 分析器设计 …… (154)

第 6 章 语法制导翻译和语义分析 …… (155)
6.1 属性文法与语法制导翻译 …… (155)
6.2 语义分析和中间代码的产生 …… (161)
6.3 简单算术表达式及赋值语句的翻译 …… (165)
6.4 布尔表达式的翻译 …… (167)
6.5 控制结构的翻译 …… (172)
6.6 说明语句的翻译 …… (177)
6.7 数组的翻译 …… (179)
6.8 过程调用语句的翻译 …… (183)
6.9 本章小结 …… (184)
习题 6 …… (185)

第 7 章 符号表 …… (187)
7.1 符号表的作用与内容 …… (187)
7.2 符号表的组织与管理 …… (190)
7.3 名字的作用域 …… (196)
7.4 本章小结 …… (199)
习题 7 …… (199)

第 8 章 运行时的存储组织与管理 …… (202)
8.1 存储分配基础及典型存储分配方案 …… (202)

8.2 参数传递方式及其实现 …… (209)
8.3 本章小结 …… (212)
习题 8 …… (213)

第 9 章 代码优化 …… (214)
9.1 代码优化概述 …… (214)
9.2 局部优化 …… (221)
9.3 循环优化 …… (229)
9.4 本章小结 …… (236)
习题 9 …… (236)

第 10 章 目标代码生成 …… (240)
10.1 代码生成概述 …… (240)
10.2 目标机器模型 …… (242)
10.3 一种简单的代码生成算法 …… (244)
10.4 本章小结 …… (253)
习题 10 …… (253)

第 11 章 现代编译技术概述 …… (255)
11.1 面向对象语言及编译技术 …… (255)
11.2 并行编译技术 …… (257)
11.3 网格计算编译技术 …… (259)
11.4 本章小结 …… (260)
习题 11 …… (260)

参考文献 …… (261)

第 1 章　编译程序概论

【学习目标】 掌握翻译程序、编译程序等概念；理解编译程序的工作过程和总体结构；了解编译技术的相关应用。

计算机的诞生是科学发展史上的一个里程碑。经过七十年的发展，计算机已经改变了人类生活、工作和学习的各个方面，成为我们不可缺少的工具。计算机之所以能够如此广泛地被应用，应归功于计算机语言。计算机语言之所以能由最初单一的机器语言发展到现今数千种高级程序设计语言，是因为有了编译程序。没有高级语言，计算机的推广应用是难以实现的；而没有编译程序，高级语言就无法使用。编译理论与技术是计算机科学中发展得最迅速、最成熟的一个分支，它集中体现了计算机发展的成果与精华。本章阐述编译程序的基本概念、编译程序的工作过程和逻辑结构、编译技术应用等。

1.1　编译程序的基本概念

1.1.1　程序设计语言

自然语言是人类传递信息、交流思想和感情的工具，程序设计语言则是联系人类与计算机的工具。人类正是通过程序设计语言指挥计算机按照人类的意志进行运算和操作、显示信息和输出运算结果的。

程序设计语言的发展至今已经历了三个阶段四代，如图 1-1 所示。

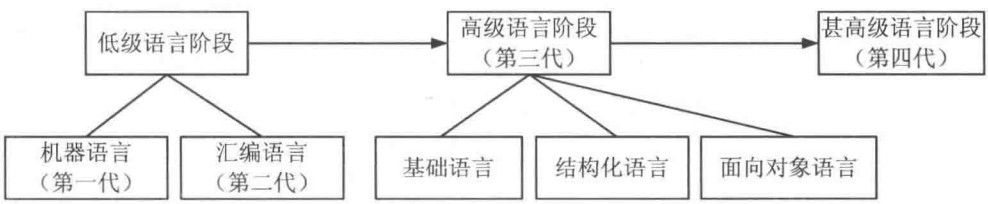

图 1-1　程序设计语言的发展和分类

1. 低级语言

低级语言包括第一代机器语言和第二代汇编语言。这两代语言依赖于机器的结构，其指令系统随机器而异，难学难用。该类语言，不仅效率低、容易出错，而且维护困难。它们在系统软件开发和计算机智能控制与处理等方面使用较为广泛。

2. 高级语言

1956年，第一种高级语言FORTRAN在美国诞生，成为最早的第三代程序设计语言(3GL)。有人统计，在同等条件下，利用高级语言一天生产的程序行数大致等于利用汇编语言一天生产的程序行数，但对于同一个问题，用高级语言写出的程序长度为用汇编语言写出的程序长度的1/7～1/3，所以两者的效率相差数倍。20世纪60年代末期，高级语言在科学计算领域的利用率已达到98%，但对于效率要求较高的实时系统，其利用率还不到10%。当时在系统软件领域，还是汇编语言独霸天下。随着现代语言特别是C语言和Ada语言的出现，汇编语言在上述两个领域的传统优势受到了严重的挑战。除了必须使用汇编语言才能满足软件需要的个别情况外，现已很少使用汇编语言编码。

从FORTRAN到Ada的大多数常用的第三代语言都是面向过程的。随着面向对象程序设计的推广，在20世纪八九十年代，一批著名的常用语言扩展了面向对象的功能，出现了C++，Object Pascal，Ada 95等面向对象的第三代语言，以及全新的面向对象高级语言，如Java，Eiffel等。它们配合面向对象软件的开发，使编码语言有了更多的选择。

3. 甚高级语言

六十余年来，高级语言的面貌发生了巨大变化，反映了人们对程序设计的认识从浅到深的过程。但是从根本上来说，上述的通用语言仍都是过程化语言，即编码的时候要详细描述问题求解的过程，告诉计算机每一步应该怎样做。要把程序员从繁重的编码劳动中解放出来，还需进一步寻求提高编码效率的新语言，这就是甚高级语言(VHLL)或第四代语言(4GL)产生的背景。

4GL迄今仍没有统一的定义。有观点认为，3GL是过程化的语言，目的在于高效地实现各种算法；4GL则是非过程化的语言，目的在于直接实现各类应用系统。前者面向过程，需要描述“怎样做”；后者面向应用，只需说明“做什么”。曾多年担任IFIP(国际信息处理联合会)数据库专家组主席的G. M. Nijssen教授则主张，语言的划代应该以数据结构为标准。他认为，前三代语言的基础注重对算法的描述，一次仅处理一个记录或数据元素，不支持对大量共享数据的处理。4GL应该以数据或知识为基础，以对集合的处理代替对单个记录或元素的处理，支持对大型数据库进行高效处理的机制。Nijssen还认为，前三代语言是工业时代的产物，4GL则标志了信息时代的开始。

以上简述了程序设计语言的发展，下面再补充一点。在高级语言的应用中，人工智能语言也占有一席之地。最早的人工智能语言可追溯到20世纪50年代的IPL语言。随后出现的LISP(20世纪50年代后期)和PROLOG(20世纪70年代前期)等语言，都具有很高的知名度。有人曾称PROLOG为第五代语言，但它主要是为新一代人工智能计算机设计的语言，故不属于本书的讨论范围。

特别注意：程序设计语言的发展经历了低级语言—高级语言—甚高级语言三个阶段。

1.1.2 编译程序的定义

在使用现代计算机时，多数用户应用高级语言来实现他们所需要的计算。现代计算机系统一般都含有的高级语言编译程序不止一个，甚至对有些高级语言配置了几个不同性能

的编译程序，供用户按不同需要进行选择。高级语言编译程序是计算机系统软件最重要的组成部分之一，也是用户直接最关心的计算机工具之一。

在计算机上执行一个高级语言程序一般要分为两步：第一步，用一个编译程序把高级语言翻译成机器语言程序；第二步，运行所得的机器语言程序求得计算结果。

通常所说的翻译程序是指这样的一个程序，它能够把某一种语言程序（称为源语言程序）转换成另一种语言程序（称为目标语言程序），而后者与前者在逻辑上是等价的。高级语言的翻译程序主要有两种，一种是编译程序，另一种是解释程序。

如果一个翻译程序的源语言是诸如 FORTRAN，ALGOL，Pascal，C，C++或 Ada 这样的高级语言，而目标语言是诸如汇编语言或机器语言之类的低级语言，则这个翻译程序就称为编译程序。世界上第一个编译程序——FORTRAN 语言编译程序——是 20 世纪 50 年代中期研制成功的。当时，人们普遍认为设计和实现一个编译程序是一件十分困难、令人生畏的事情。经过六十多年的努力，编译理论与技术得到迅速的发展，现已形成一套比较成熟的、系统化的理论与方法，并且开发出了一些好的编译程序的实现语言、环境与工具。在此基础上设计并实现一个编译程序不再是高不可攀的事情。

一个源语言的解释程序是这样的程序，它以该语言写的源程序作为输入，但不产生目标程序，而是边解释边执行源程序本身。Basic 语言是一种交互式语言，它的程序是一种行结构，因而对 Basic 程序的执行宜采用解释方式。实际上，许多编译程序的构造与实现技术同样适用于解释程序。

特别注意：编译程序实际上是一个源语言是高级语言，而目标语言是低级语言的翻译程序。计算机执行高级语言编写的程序通常有编译和解释两种方式，它们的区别在于是否生成目标代码。

1.2 编译程序的工作过程

编译程序的工作过程指从输入源程序开始到输出目标程序为止的整个过程，是非常复杂的。但就其过程而言，它与人们进行自然语言之间的翻译有许多相近之处。当我们把一种文字翻译为另一种文字（如把一段英文翻译为中文）时，通常需经过下列步骤：

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。

类似地，编译程序的工作过程一般也可以划分为五个阶段：词法分析、语法分析、语义分析和中间代码产生、代码优化、目标代码生成。下面分别来介绍这五个阶段。

1.2.1 词法分析

词法分析的任务是：输入源程序，对构成源程序的字符串进行扫描和分解，识别出一个

个的单词(亦称单词符号,简称符号),如基本字(if,switch,for,while 等)、标识符、常数、算符和界符(标点符号、左右括号等)。例如,对 C 语言的一条循环语句

```
for(i=1;i<10;i++)
```

进行词法分析,识别的结果是如图 1-2 所示的单词符号。

基本字	for	界符	(	标识符	i	算符	=
常数	1	界符	;	标识符	i	算符	<
常数	10	界符	;	标识符	i	算符	++
界符	)						

图 1-2

单词是组成上述 C 语句的基本符号。单词符号是语言的基本组成成分,是人们理解和编写程序的基本要素。识别和理解这些要素无疑是翻译的基础。如同将英文翻译成中文的情形一样,如果对英语单词不理解,那就谈不上进行正确的翻译。在词法分析阶段的工作中所遵循的是语言的词法规则(或称构词规则)。描述词法规则的有效工具是正规式和有限自动机。

1.2.2 语法分析

语法分析的任务是:在词法分析的基础上,根据语言的语法规则,把单词符号串分解成各类语法单位(语法范畴),如短语、子句、句子(语句)、程序段和程序等。通过语法分析,确定整个输入串是否构成语法上正确的"程序"。语法分析所遵循的是语言的语法规则。语法规则通常用与上下文无关的文法描述。词法分析是一种线性分析,而语法分析是一种层次结构分析。例如,在 C 语言中,符号串

```
Z=X+0.618 * Y
```

代表一个赋值语句,而其中的 X+0.618 * Y 代表一个算术表达式。因而,语法分析的任务就是识别 X+0.618 * Y 为算术表达式,同时,识别上述整个符号串为赋值语句。

1.2.3 语义分析和中间代码产生

这一阶段的任务是:分析语法分析所识别出的各类语法范畴的含义,并进行初步翻译(产生中间代码)。这一阶段通常包括两个方面的工作。首先,对每种语法范畴进行静态语义检查,如变量是否定义、类型是否正确等。如果语义正确,则进行另一方面工作,即进行中间代码的翻译。这一阶段所遵循的是语言的语义规则。通常使用属性文法描述语义规则。

翻译在这里才刚开始涉及。所谓的中间代码是一种含义明确、便于处理的记号系统,它通常独立于具体的硬件。这种记号系统或者与现代计算机的指令形式有某种程度的接近,或者能够比较容易地被变换成现代计算机的机器指令。例如,许多编译程序采用了一种与三地址指令非常近似的四元式作为中间代码。这种四元式的形式是:

算符	左操作数	右操作数	结果

它的意义是:对左、右操作数进行某种运算(由算符指明),把运算所得的值作为结果保留下

来。在采用四元式作为中间代码的情形下，中间代码的任务就是按语言的语义规则把各类语法范畴翻译成四元式序列。例如，赋值句

$$Z=(X+0.418)*Y/W$$

可被翻译为如下的四元式序列：

序号	算符	左操作数	右操作数	结果
①	+	X	0.418	T_1
②	*	T_1	Y	T_2
③	/	T_2	W	Z

其中 T_1 和 T_2 是编译期间引进的临时工作变量；第一个四元式意味着把 X 的值加 0.418 的结果存于 T_1 中；第二个四元式是将 T_1 的值和 Y 的值相乘的结果存于 T_2 中；第三个四元式是将 T_2 的值除以 Y 的值的结果存于 Z 中。

一般而言，中间代码是一种独立于具体硬件的记号系统。常用的中间代码，除了四元式之外，还有三元式、间接三元式、逆波兰式和树形表示等。

1.2.4 代码优化

代码优化的任务是：对前阶段产生的中间代码进行加工变换，以期在最后阶段能产生更为高效(省时间和空间)的目标代码。代码优化的主要方法有：公共子表达式的提取、循环代码优化、删除无用代码等。有时，为了便于并行运算，还可以对代码进行并行化处理。代码优化所遵循的是程序的等价变换规则。例如，如果我们把程序片断

```
for(K=1;K<=100; K++)
{
  M=I+10 * K;
  N=J+10 * K;
}
```

的中间代码

序号	OP	ARG1	ARG2	RESULT	注　　解
①	=	1		K	K=1
②	J>	K	100	⑨	若 K>100 转至第⑨个四元式
③	*	10	K	T_1	$T_1=10*K$；T_1 为临时变量
④	+	I	T_1	M	$M=I+T_1$
⑤	*	10	K	T_2	$T_2=10*K$；T_2 为临时变量
⑥	+	J	T_2	N	$N=J+T_2$
⑦	+	K	1	K	K=K+1
⑧	J			②	转至第②个表达式
⑨					

转换成如下的等价代码：

序号	OP	ARG1	ARG2	RESULT	注　解
①	=	I		M	M=I
②	=	J		N	N=J
③	=	1		K	K=1
④	J>	K	100	⑨	if (K>100) goto ⑨
⑤	+	M	10	M	M=M+10
⑥	+	N	10	N	N=N+10
⑦	+	K	1	K	K=K+1
⑧	J			④	goto ④
⑨					

那么最终所得的目标程序的执行效率肯定会提高很多。因为对于前者，在循环中需做 300 次加法和 200 次乘法；对于后者，在循环中只需做 300 次加法。而且在多数硬件中，做乘法的时间比做加法的时间要长得多。

1.2.5　目标代码生成

目标代码生成的任务是：把中间代码（或经优化处理之后的代码）变换成特定机器上的低级语言代码。这个阶段能实现最后的翻译，它的工作有赖于硬件系统结构和机器指令含义。这个阶段的工作非常复杂，涉及硬件系统功能部件的运用，机器指令的选择，各种数据类型变量的存储空间分配，以及寄存器和后援寄存器的调度等。产生足以充分发挥硬件效率的目标代码是一件非常不容易的事情。

目标代码的形式可以是绝对指令代码、可重定位的指令代码以及汇编指令代码。如果目标代码是绝对指令代码，则这种目标代码可立即执行。如果目标代码是汇编指令代码，则需汇编器汇编之后才能运行。必须指出，现代多数实用编译程序所产生的目标代码都是一种可重定位的指令代码。这种目标代码在运行前必须借助于一个连接装配程序把各个目标模块（包括系统提供的库模块）连接在一起，确定程序变量（或常数）在内存中的位置，装入内存中指定的起始地址，使之成为一个可以运行的绝对指令代码程序。

上述编译过程划分的五个阶段是一种典型的分法。事实上，并非所有的编译程序都分成这五个阶段。有些编译程序对代码优化没有什么要求，代码优化阶段就可省去。在某些情况下，为了加快编译速度，语义分析和中间代码产生阶段也可以省去。有些最简单的编译程序是在语法分析的同时产生目标代码。但是多数实用编译程序的工作过程大致都可分为上述五个阶段。

特别注意：典型的编译程序的逻辑结构一般划分为五个阶段：词法分析、语法分析、语义分析和中间代码产生、代码优化、目标代码生成。

1.3 编译程序的逻辑结构

1.3.1 编译程序的总体框架

上述编译过程的五个阶段是编译程序工作时的动态特征。编译程序的结构可以按照这五个阶段的任务进行分模块设计。图 1-3 给出了编译程序的总体框架。

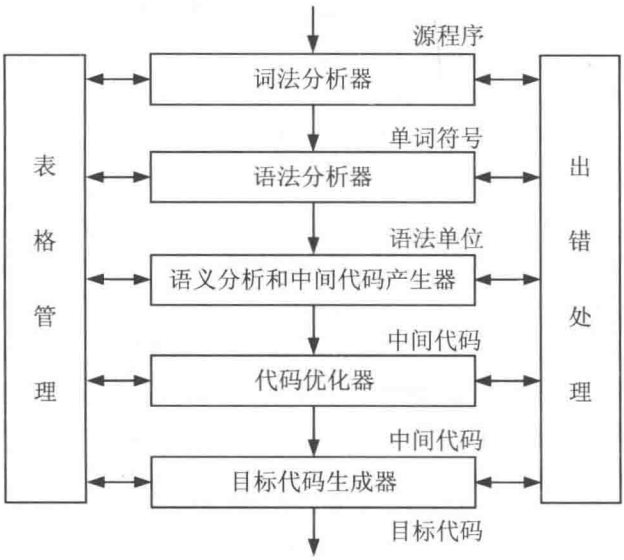

图 1-3 编译程序的总体框架

① 词法分析器，又称扫描器，输入源程序，进行词法分析，输出单词符号。

② 语法分析器，简称分析器，对词法分析阶段产生的单词符号串进行语法分析（根据语法规则进行推导或归约），识别出各类语法单位，最终判断输入串是否构成语法正确的程序。

③ 语义分析和中间代码产生器，按照语义规则对语法分析器归约出（或推导出）的语法单位进行语义分析，并把它们翻译成一定形式的中间代码。

有的编译程序在识别出各类语法单位后，构造并输出一棵表示语法结构的语法树，然后根据语法树进行语义分析和产生中间代码。还有许多编译程序在识别出语法单位后并不真正构造语法树，而是调用相应的语义子程序。在这种编译程序中，扫描器与分析器和中间代码产生器二者并非是截然分开的，而是相互穿插的。

④ 代码优化器，对中间代码进行优化处理。

⑤ 目标代码生成器，把中间代码翻译成目标程序。

除了上述五个功能模块外，一个完整的编译程序还应包括表格管理和出错处理两个部分。

1.3.2 编译程序的表格管理

编译程序在工作过程中需要保持一系列的表格，以登记源程序的各类信息和编译各阶段的进展状况。合理地设计和使用表格是编译程序构造的一个重要问题。在编译程序使用的表格中，最重要的是符号表。它用来登记源程序中出现的每个名字（标识符）以及名字的各种属性。例如，一个名字是常数名、变量名还是过程名等；如果是变量名，它的类型是什么、所占内存是多大、地址是什么等。通常，编译程序在处理到名字的定义性出现时，要把名字的各种属性填入符号表中；当处理到名字的使用性出现时，要对名字的属性进行查证。

扫描器识别出一个名字后，把该名字填入符号表中。但这时不能完全确定名字的属性，它的各种属性要在后续的各阶段才能确定。例如，名字的类型要在语义分析时才能确定，而名字的地址可能要到目标代码生成时才能确定。

由此可见，编译各阶段都涉及与构造、查找或更新有关的表格。

1.3.3 编译程序中的错误及出错处理

一个编译程序不仅应能对书写正确的程序进行翻译，而且应能对出现在源程序中的错误进行处理。如果源程序有错误，编译程序应设法发现错误，把有关错误信息报告给用户。这部分工作是由专门的一组程序——出错处理程序——完成的。一个好的编译程序应能最大限度地发现源程序中的各种错误，准确地指出错误的性质和错误发生的地点，并且能将错误所造成的影响限制在尽可能小的范围内，使得源程序的其余部分能继续被编译下去，以便进一步发现其他可能的错误。如果不仅能够发现错误，而且还能自动校正错误，那当然就更好了。但是自动校正错误的代价是非常高的。

编译过程的每一阶段都可能检测出错误，其中绝大多数错误可以在编译的前三个阶段检测出来。源程序中的错误通常分为语法错误和语义错误两大类。语法错误是指源程序中不符合语法（或词法）规则的错误，它们可在词法分析或语法分析时被检测出来。例如，词法分析阶段能够检测出“非法字符”之类的错误；语法分析阶段能够检测出诸如“括号不匹配”“缺少；”之类的错误。语义错误是指源程序中不符合语义规则的错误，这些错误一般在语义分析时被检测出来，有的语义错误要在运行时才能被检测出来。语义错误通常包括：说明错误、作用域错误、类型不一致等。关于错误检测和处理方法，将穿插在有关章节中介绍。

特别注意：从编译程序的角度看，源程序中的错误通常分为语法错误和语义错误两大类。

1.3.4 编译程序的分遍处理

前面介绍的编译程序的工作过程的五个阶段仅仅是逻辑功能上的一种划分。其具体实现，受不同源语言、设计要求、使用对象和计算机条件（如内存容量）的限制，往往将编译程序组织为若干遍（pass）。所谓的“遍”就是对源程序或源程序的中间结果从头到尾扫描一次，并做有关的加工处理，生成新的中间结果或目标程序。通常，每一遍的工作由从外存上获得的前一遍的中间结果开始（对于第一遍而言，从外存上获得源程序），完成它所含的有关工作之后，再把结果记录于外存中。操作中，既可以将几个不同阶段合为一遍，也可以把一个阶

段的工作分为若干遍。例如，词法分析这一阶段可以单独作为一遍，但更多的时候是把它与语法分析合并为一遍；为了便于处理，语法分析与语义分析和中间代码产生又常常被合为一遍。在对代码优化要求很高时，往往还可把代码优化阶段分为若干遍来实现。

当一遍中包含若干阶段时，各阶段的工作是穿插进行的。例如，我们可以把词法分析、语法分析及语义分析和中间代码产生这三个阶段安排成一遍。这时，语法分析器处于核心位置，当它在识别语法结构而需要下一个单词符号时，它就调用词法分析器，一旦识别出一个语法单位，它就调用中间代码产生器，完成相应的语义分析并产生相应的中间代码。

一个编译程序究竟应分成几遍，如何划分，是与源语言、设计要求、硬件设备等诸因素有关的，因此难以统一划定。遍数多一点的好处是整个编译程序的逻辑结构可能会清晰一点。但遍数多势必增加输入/输出所消耗的时间。因此在内存足够的前提下，一般还是遍数尽可能少一点为好。应当注意的是，并不是每种语言都可以用单遍编译程序实现。

1.3.5 编译前端与后端

概念上，有时我们把编译程序划分为编译前端和编译后端。编译前端主要由与源语言有关但与目标机无关的那些部分组成。这些部分通常包括词法分析、语法分析、语义分析和中间代码产生，有的代码优化工作也被包括在编译前端中。编译后端包括编译程序中与目标机有关的那些部分，如与目标机有关的代码优化和目标代码生成等。通常，编译后端不依赖于源语言，而仅仅依赖于中间语言。

我们可以取编译程序的前端，改写其后端，以生成不同目标机上的相同语言的编译程序。如果编译后端的设计是经过精心考虑的，那么将用不了太大工作量就可实现编译程序的目标机改变。我们也可以设想将几种源语言编译成相同的中间语言，然后为不同的编译前端配上相同的编译后端，这样就可为同一台机器生成不同语言的编译程序。然而，由于不同语言存在某些微妙的区别，所以目前在这方面所取得的成果还非常有限。

1.4* 编译技术应用

虽然真正从事主流编程语言编译器设计工作的只有极少数一部分人，但是编译技术有着其重要的应用，这是学习编译技术的主要理由。此外，编译器的设计影响着计算机科学的其他一些领域。本节讲述编译器设计与计算机科学的其他主要领域之间最重要的相互影响以及编译技术的重要应用。

1.4.1 高级语言的实现

一种高级编程语言定义了一种编程抽象：程序员用这种语言表达算法，编译器将程序翻译成目标语言。一般来说，高级编程语言易于编程，但是所得到的程序运行较慢。用低级语言编程时可以在程序中实施更有效的控制方式，原则上说，能得到更有效的代码。不幸的是，低级语言程序难编写、易出错，更糟糕的是其难维护、缺乏移植性。代码优化编译器包括

了很多改进后所生成代码性能的技术，从而弥补了高级抽象导入低效的缺点。

历史上，主流编程语言的大多数改变都是朝着提高抽象级别的方向进行的。20 世纪 80 年代 C 语言占据了系统编程的统治地位，90 年代启动的更多项目选择 C++，1995 年问世的 Java 语言在 20 世纪 90 年代后期迅速流行，现已成为应用开发使用的主流语言。每一轮编程语言新特征的出现都促进了编译器优化的新研究。下面回顾激励编译技术显著推进的主要语言特征。

所有通用编程语言，包括 C，FORTRAN 和 COBOL，都支持用户定义的聚合数据类型（如数组和记录），也都支持高级控制流（如循环和过程调用）。如果逐句把程序直接翻译成机器代码，则非常低效，所生成代码的效率相当于有经验的程序员用低级语言写出程序的效率。数据流优化至今仍在被广泛研究。

面向对象的概念于 1967 年首次出现在 Simula 67 语言中，以后逐步被加入到各种语言中，如 Smalltalk，C++，C # 和 Java。面向对象的主要概念是数据抽象和性质继承，它们都使得程序更加模块化并易于维护。面向对象的程序不同于用其他语言编写的程序，其中类的定义中可能包含许多较小的过程（在面向对象语言中称为方法）。因此对于源程序中跨越过程边界的操作，编译器优化必须能够适当处理才能保证效率。在这个场合，用过程体替换过程调用的过程内联特别有用。加速虚方法调遣的优化也一直是研究的热点。

Java 有很多使编程变得容易的特征，其中大部分已存在于先前的语言中。Java 语言是类型安全的，它使一个对象不会被当作一个不相关类型的对象来使用。所有数组访问都被检查，以保证这些访问在相应数组的边界内。Java 没有指针，也不允许指针算术运算。它用无用单元收集机制来自动地回收那些不再使用的变量所占的内存空间（俗称垃圾收集）。这些特征虽然使编程变得容易，但是它们会引起运行开销的增加。相应地，出现了一些用于降低开销的编译器优化技术。例如，删除不必要的边界检查，把离开过程后不再访问的对象分配在栈上而不是堆上，现也一直在开发极小化无用单元收集开销的算法。

此外，Java 支持代码移植和代码移动。程序以 Java 字节码的形式分发，字节码必须被解释或被动态地翻译成本地代码。在运行时能够收集运行信息的场合，动态编译可用来生成较好优化的代码。在动态代码优化中，很重要的一点是极小化编译所需的时间，因为它是执行开销的一部分。现在通用的一种技术是仅仅编译和代码优化程序中经常执行的部分。

1.4.2 针对计算机体系结构的优化

计算机体系结构的迅速演化对编译器技术不断提出新的要求，几乎所有的高性能系统都在利用两种基本技术——并行和内存分层。并行可以在指令级和处理器级分别发掘；内存分层针对这样的基本局限：构造非常快的存储器或者非常大的存储器是可能的，但是构造不出既快又大的存储器。

1. 并行

所有现代微处理器都开拓指令级的并行。这种并行对程序员是隐蔽的，程序员把它理解为串行执行的指令序列，会被硬件动态地检查其中的相关性，并尽可能并行发出这些指令。在有些情况下，计算机用一个硬件调度器改变指令的排序以增加程序中的并行。不管硬件调度器是否重排指令的次序，编译器总可以重新整理指令，使得指令级的并行更有效。