第1章 引 言

20 世纪 40 年代，由于冯·诺伊曼在存储-程序计算机方面的先锋作用，编写一串代码或程序使计算机执行所需的计算已成为必要。随着计算机的广泛应用，计算机程序设计语言也从初期的机器语言发展为汇编语言，以及现在的各种高级程序设计语言。而编译技术是计算机语言发展的支柱，也是计算机科学中发展最迅速、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。其核心思想就是把同样的逻辑结构和思想从一种语言表示的程序转化为另外一种语言表示的程序。从高级语言，甚至运行于虚拟平台的高级语言，到机器语言，最终到硬件执行的物理信号，这一层层的转化，都涉及编译技术的应用。因此，编译技术是从人类智慧到机器执行的桥梁，从软件到硬件层层推进的衔接力量。使用编译技术构造的编译程序或翻译系统是程序设计语言的支撑环境，高级语言程序只有经过编译程序或翻译系统的转换，才能生成目标机器能够识别的语言程序，进而才能在目标机器上运行。

1.1 程序的翻译及运行

人类要用计算机解决问题，首先要告诉计算机解决什么问题，或许还要告诉计算机如何解决这个问题。这就牵涉到用什么样的语言描述要解决的问题。起初，人们采用机器语言(machine language)语言描述要解决的问题。机器语言即机器指令，能被计算机直接理解与执行，却不易被人们理解和接受。当然，编写这样的代码是十分费时和乏味的，这种代码形式很快就被汇编语言(assembly language)代替了。汇编语言十分接近机器语言，汇编语言中的很多语句恰好就是机器语言语句的符号表示。汇编语言的语句通常具有固定格式，这种格式将使汇编程序更易于分析这些语句，在这些汇编语句中通常不包含嵌套语句、分程序等。汇编语言大大提高了编程的速度和准确度，人们至今仍在使用着它，在编码需要极快的速度和极高的简洁程度时尤为如此。但是，汇编语言也有许多缺点：编写起来也不容易，阅读和理解很难；而且汇编语言的编写严格依赖于特定的机器，所以为一台计算机编写的代码在应用于另一台计算机时必须完全重写。随后，计算机科学家设计了一些比较习惯的语言来描述要解决的问题。这种语言表达力强，易于使用，易于为人理解和接受，称为高级程序语言。这些用机器语言对问题的描述称为程序。例如，用 Fortran 语言、ALGOL 语言、Pascal 语言、C 语言、Ada 语言、C++语言等编写的程序，都称为高级语言程序。这种程序不能直接被计算机理解与执行，必须经过等价的转换，变成机器能理解与执行的机器语言才能执行。进行这种等价转换工作的程序，称为翻译程序。

一般来说，翻译程序可以将用一种语言写的程序，等价地转换为用另一种语言写的程序。前一个程序，即被翻译的程序，叫做源程序；后一个程序，即翻译后的程序，叫做目的程序或目标程序。

与自然语言的翻译中通篇笔译和口译类似，翻译程序也有编译程序和解释程序两大类。而源程序是汇编语言程序，目标程序是机器语言程序时，翻译程序称为汇编程序。当源程序是面向对象语言程序时，翻译程序一般先对对象的特征进行分析和处理，生成另一种高级语言表示的目标程序或面向虚拟机的目标程序，然后再通过编译或解释程序生成计算机可执行的目标代码。

编译程序把用高级程序设计语言书写的源程序，翻译成等价的计算机汇编语言或机器语言书写的目标程序的翻译程序，如图1.1 所示。而并行编译程序把串行执行的高级程序语言源程序，翻译成能并行执行的汇编语言或机器语言目标程序。编译程序属于采用生成性实现途径实现的翻译程序。它以高级程序设计语言书写的源程序作为输入，而以汇编语言或机器语言表示的目标程序作为输出。编译出的目标程序通常还要经历运行阶段，以便在运行程序的支持下运行，加工初始数据，算出结

果。编译程序的实现算法较为复杂，这是因为它所翻译的语句与目标语言的指令不是一一对应关系，而是一多对应关系；同时也因为它要处理递归调用、动态存储分配、多种数据类型，以及语句间的紧密依赖关系。不过，由于高级程序设计语言书写的程序具有易读、易移植和表达能力强等特点，编译程序广泛地用于翻译规模较大、复杂性较高、且需要高效运行的高级语言书写的源程序。

解释程序不同于编译程序，它不是直接将高级语言的源程序翻译成目标程序后再执行，而是一个语句一个语句地读入源程序，即边解释边执行，其工作过程如图1.2所示。

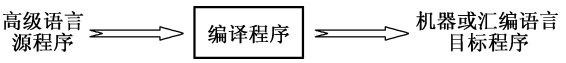

图 1.1 编译程序

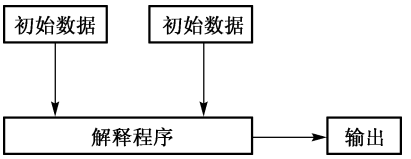

图 1.2 一个概念化的解释程序

解释程序把源程序看成自己的输入，源程序原来的输入也是解释程序的一部分输入，因而可以对源程序进行处理，就像对另一部分数据一样。程序执行时的控制点在解释程序之中，而不在用户程序中，即用户程序是消极的，这就不同于编译程序产生的可执行的用户程序，在那里它是积极的。

解释程序允许：

① 在执行用户程序时修改用户程序。因此，它提供一种直接的交互调试能力。这种修改对于 APL、BASIC 这样的非分程序结构的语言是非常容易的，因为修改个别语句并不需要重新分析整个程序。

② 对象的类型可动态地修改。随着程序的执行，符号的意义可以变化。例如，在某一点，它可以是整型变量，而在另一点，它可以是一个字符数组。这种符号意义的动态确定叫做流动绑定(Fluid Binding)，这对于编译程序是很头痛的事，编译程序很难对它进行翻译。

③ 提供良好的诊断信息。解释程序执行时，把程序的执行与源程序行文的分析交织在一起，因而可以在诊断信息中给出出错点的源程序行号、变量的符号名，并对变量交互赋值，这些工作对于编译程序是比较困难的。

④ 解释程序不依赖于目标机，因为它不生成目标代码。因此，其可移植性优于编译程序。

解释程序的突出优点是可简单地实现，且易于在解释执行过程中灵活、方便地插入修改和调试措施，但最大缺点是执行效率很低。例如，需要多次重复执行的语句，采用编译程序时只需要翻译一次；但在解释程序中却需要重复翻译，重复执行。根据这些特点，解释程序适用于如下场合。

① 有些语言中的大多数语句，如字符串加工语言中的字符串查找语句和加工语句，其执行时间比翻译时间长得多。对于这种语言，采用生成性方案效果甚微，而采用解释性方案则易于实现。

② 为了便于用户调试和修改程序，又能保证程序高效运行，很多程序设计语言配置两个加工系统，一个用于调试，另一个用于有效地运行。调试用的系统一般用解释程序实现，以便及时监视运行情况、动态地输出调试信息和灵活地修改错误。

③ 交互式会话语言(如 BASIC，APL)，要为用户提供并行、交叉编写、执行、调试和修改源程序的功能。采用解释程序易于实现这些功能。

1.2 编译过程概述

编译程序的工作，即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、语义分析与中间代码的产生、优化、目标代码的生成。

1．第一阶段：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，识别出一个个单词(又称单词符号，简称符号)，如保留字(if、for、while 等)、标识符、常数、特殊符号(标点符号、左右括号、运算符等)。例如，对于 C 语言的循环语句：

```
for （ i=1;i<= 100;i++） sum=sum+1;
```

词法分析的结果是识别出如下单词符号：

保留字　for
标识符　i ,sum
特殊符号　(，　=, ; , <=, ++ , +
整常数　1, 100

这些单词是组成上述 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也是翻译的基础。如同将英文翻译成中文的情形一样，如果你对英语单词不理解，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。

2．第二阶段：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”、“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”。语法分析所依循的是语言的语法规则。语法规则通常用上下文无关文法描述。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，在很多语言中，符号串

$$Z = X + 2*Y;$$

代表一个“赋值语句”，而其中的“$X + 2*Y$”代表一个“算术表达式”。因而，语法分析的任务就是识别“$X + 2*Y$”为算术表达式。同时，识别上述整个符号串属于赋值语句语法范畴。

3．第三阶段：语义分析与中间代码的产生

这一阶段的任务是：对语法分析所识别出的各类语法范畴，分析其含义，并进行初步翻译(产生中间代码)。这一阶段通常包括两方面的工作。首先，对每种语法范畴进行静态语义检查，例如，变量是否定义、类型是否正确等。如果语义正确，则进行另一方面的工作，即进行中间代码的翻译。这一阶段所依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”仅仅在这里才开始涉及。所谓“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。这种四元式的形式如表 1.1 所示。

表 1.1 四元式形式

运算符	第一运算量	第二运算量	结果

它的意义是：对第一和第二运算符进行某种运算，把运算所得的值作为“结果”保留下来。在采用“四元式”作为中间代码的情况下，中间代码产生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，下面的赋值语句

$$Z = (X + 3) * Y / W;$$

可翻译为如表 1.2 所示的四元式序列。

表 1.2 赋值语句的四元式序列

序 号	运 算 符	第一运算分量	第二运算分量	结 果
(1)	+	X	3	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 的值加上 3 存放其中；第二个四元式指将 T_1 的值和 Y 的值相乘存于 T_2 中；第三个四元式指将 T_2 的值除以 W 的值，结果存于 Z 中。

一般情况下，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、简接三元式、逆波兰式和抽象语法树等。

4. 第四阶段：优化

优化的任务在于对前阶段产生的中间代码进行加工变换，以期在最后阶段产生出更为高效(节省时间和空间)的目标代码。优化的主要方面有：公共子表达式的提取、循环优化、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优化处理。优化所依循的原则是程序的等价变换规则。

5. 第五阶段：目标代码的生成

这一阶段的任务是：把中间代码(或经优化处理后的代码)变换成特定机器上的低级语言代码。本阶段实现了最后的翻译，它的工作有赖于硬件系统结构和机器指令含义。本阶段工作非常复杂，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。如何产生充分发挥硬件效率的目标代码，是一件非常不容易的事情。

目标代码的形成可以是绝对指令代码、可重定位的指令代码或汇编指令代码。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，确定程序变量在主存中的位置，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。

上述编译过程的五个阶段是一种典型的分法。事实上，并非所有编译程序都分为这五个阶段。有些编译程序对优化没有什么要求，优化阶段就可省去。在某些情况下，为了加快编译速度，中间代码产生阶段也可以去掉。有些最简单的编译程序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像上面所说的五个阶段。有时编译过程还可以分为六个阶段，即把语义分析和中间代码产生分为两个阶段。

1.3　编译程序的结构框图

典型的编译程序结构框图如图1.3所示。

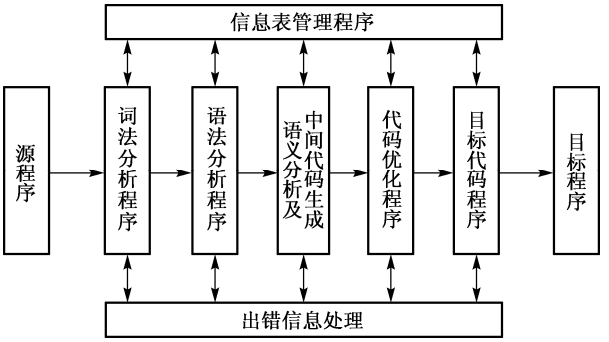

图 1.3　典型的编译程序结构框图

不同的编译程序其结构不同，图中给出的只是编译程序各部分的逻辑结构图，并不代表时间上的执行顺序。有些编译程序可能恰好按图中顺序执行这些逻辑过程，有些编译程序可能按平行、互锁方式执行这些逻辑过程。

采用哪种执行方式，视语种、机型等因素的不同而不同。

1.4　编译程序的开发

编译程序是一个非常复杂的软件系统，虽然编译理论和编译技术不断发展，已使编译程序的生产周期不断缩短，但是目前要研制一个编译程序仍需要相当长的时间，而且工作相当艰巨，因此如何高效地生成一个高质量的编译程序一直是人们追求的目标。本节将简单介绍编译程序的开发步骤、开发技术和程序的自动生成。

1.4.1　编译程序的开发步骤

从软件工程的理论和方法来分析，编译程序的开发过程大致分为以下几个阶段。

① 认真做好需求分析并合理分工，编译程序要把源程序翻译成某台计算机上的目标程序，用户首先要熟悉某种源语言，对源语言的语法和语义要有准确无误的理解；其次确定对编译程序的要求，同时很重要的一点是，对目标机要有深刻的可行性研究，否则将会生成质量较低的目标程序；最后要根据编译程序的规模进行划分，并组织合理分工。

② 系统设计，选择算法并确定方案。确定方案是编译开发过程中最关键的一步。在系统设计选择算法中最重要的是使编译程序具有易读性和易改性，以便将来对编译程序的功能进行更新扩充。

③ 选择语言，认真编写程序。根据所设计的算法慎重选用某种语言(低级或高级语言)编写编译程序，最终得到机器语言级的编译程序。

④ 测试程序，确保质量。通过大量实例对编写好的编译程序进行测试。为了检查编译程序各部分的功能，要选用各种不同的程序，尤其要有意在源程序中设计各种类型的错误障碍，让编译程序进行编译。在以大量实例对编译程序测试的过程中不断修改完善编译程序，以确保质量。

⑤ 建立、整理文档资料。根据前面的工作整理出有关编译程序的一套资料，形成文档，其中包

括源程序的语法、目标机器指令系统、编译算法、出错信息表及编译程序的使用说明等，以供用户在使用该编译程序时查阅。

1.4.2 编译程序的开发技术

1．系统程序设计语言

20 世纪 70 年代以前，几乎所有的编译程序都是用机器语言或汇编语言编写的，这样不仅工作量大，而且编译程序的可靠性比较差，难以维护，质量也难以保证。从 20 世纪 80 年代开始，大部分编译程序都是用高级语言编写的，这样不仅减少了开发的工作量，而且缩短了开发周期。

并非所有的高级语言都适合于编写编译程序，通常把能够编写编译程序或其他系统软件的高级语言称为系统程序设计语言，如 Pascal、C 和 Ada 等都可以作为系统程序设计语言，而 Fortran 等语言则不宜用做系统程序设计语言。

此外，像 Pascal 和 Ada 这样的高级语言，不仅可以用来编写其他高级语言的编译程序，而且还可以用来编写自己的编译程序，因此这种语言又称为自编译语言。

2．编译程序的开发技术

编译程序的开发常常采用自编译、交叉编译、自展和移植等行之有效的技术。

① 自编译，某种高级语言书写自己的编译程序称为自编译。

例如，假如 A 机器上已有一个 Pascal 语言可以运行，则可以用 Pascal 语言编写 Pascal 语言的编译程序，然后借助于原有的 Pascal 编译程序对编写的 Pascal 编译程序进行编译，从而编译后即得到一个能在 A 机器上运行的 Pascal 编译程序。这种编译系统的程序设计语言是其本身。

② 交叉编译。交叉编译是指 A 机器上的编译程序能产生 B 机器上的目标代码。

若 A 机器上已有的 Pascal 语言可以运行，则可用 A 机器中的 Pascal 语言写一个编译程序，它的源语言是 Pascal，目标语言是 B 代码。这种在 A 机器上运行，而产生在 B 机器上的目标代码也称为交叉编译。

对于以上两种方法，都假定已经有了一个系统编译程序语言可以使用，否则采用自展和移植方法。

③ 自展。自展是首先确定一个非常简单的核心语言 L_0，然后用机器语言或汇编语言写出它的编译程序 T_0，再把语言 L_0 扩充到 L_1，此时

$$L_1 \supset L_0$$

并用 L_0 编写 L_1 的编译程序 T_1，再把语言 L_1 扩充为 L_2，则有

$$L_2 \supset L_1$$

并用 L_1 编写 L_2 的编译程序 T_2……如此逐步扩展下去，直到完成所要求的编译程序。

这种方法中最初的核心语言的编译程序比较简单，以后各级编译程序(T_1，T_2，…)均可用前一级已实现的较高级的语言编写，后一级建立在前一级语言的基础上，级级升高，这样大大减少了开发编译程序的工作量。

④ 移植。移植是将 A 机器上的某高级语言的编译程序移植到 B 机器上运行。一个程序若能较容易地从 A 机器上搬到 B 机器上运行，则称该程序是可移植的。

1.4.3 编译程序的自动生成

用系统程序设计语言来书写编译程序，虽然缩短了编译程序的开发周期，提高了编译程序的质量，但自动化程序仍然不高，它需要开发者熟悉各种编译技术，根据语言的要求设计算法。人们的最大愿

望是能有一个自动生成编译程序的软件工具，只要把源程序的定义及机器语言的描述输入这种软件中去，就能自动生成该语言的编译程序，这就是编译程序的自动生成，如图1.4所示。

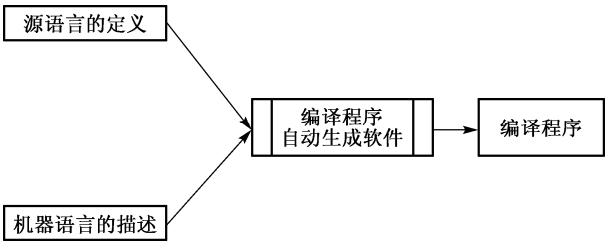

图 1.4　编译程序的自动生成

计算机科学家和软件工作者为了实现编译程序的自动生成做了大量的研究工作，近年来随着形式语言学研究的发展，大大推动了编译程序的自动生成研究工作，并已出现了一些编译程序的自动生成系统。如 UNIX 操作系统下的软件工具 LEX 和 YACC 等。本书第 13 章简单介绍了编译程序的自动生成工具，关于这方面的详细讨论，有兴趣的读者可参考有关文献。

习题 1

1.1　分别指出编译程序、汇编程序和解释程序的含义。

1.2　什么叫系统程序设计语言?

1.3　“解释方式与编译方式的区别在于解释程序对于源程序并没有真正进行翻译”。这种说法对吗?

1.4　编译程序的开发过程大致分为几个阶段?

1.5　编译程序的开发技术有哪几种？并分别叙述其基本思想。

1.6　有人认为编译程序的五个组成部分缺一不可，这种看法正确吗?