编译原理

Compilers Principles

编译原理这门课程主要介绍设计和构造编译程序的基本原理和常用的技术和方法。

- 什么是编译程序
- 编译的过程
- 编译程序的结构

翻译程序与编译程序


```
#include <stdio.h>
int main(void)
{       printf("hello, world\n");
        return 0;
}
```

00 00 10 10…..

翻译程序是指这样一个程序，它把一种语言（称作源语言）所写的程序（源程序）翻译成等价的另一种语言（称作目标语言）的程序（目标程序）。

翻译程序与编译程序

高级语言程序

汇编语言或者
机器语言程序

编译程序是一种翻译程序，它将高级语言所写的源程序翻译成等价的机器语言或汇编语言的目标程序。

第1章 基本知识与技能

1.1 程序设计语言和编译程序

为了处理和解决实际问题，每一种计算机都可以实现其特定的功能，而这些功能是通过计算机执行一系列相应的操作来实现的。计算机所能执行的每一种操作对应为一条指令，计算机能够执行的全部指令集合就是该计算机的指令系统。

计算机硬件的器件特性决定了计算机本身只能直接接受由0和1编码的二进制指令和数据，这种二进制形式的指令集合称为该计算机的机器语言，它是计算机唯一能够直接识别并接受的语言。

用机器语言编写程序很不方便且容易出错，编写出来的程序也难以调试、阅读和交流。为此，出现了用助记符代替机器语言(二进制编码)的另一种语言，这就是汇编语言。汇编语言是建立在机器语言之上的，因为它是机器语言的符号化形式，所以较机器语言直观；但是计算机并不能直接识别这种符号化语言，用汇编语言编写的程序必须翻译成机器语言之后才能执行，这种“翻译”是通过专门的软件——汇编程序实现的。

尽管汇编语言与机器语言相比在阅读和理解上有了长足的进步，但其依赖具体机器的特性是无法改变的，这给程序设计增加了难度。随着计算机应用需求的不断增长，出现了更加接近人类自然语言的功能更强、抽象级别更高的面向各种应用的高级语言。高级语言已经从具体机器中抽象出来，摆脱了依赖具体机器的问题。用高级语言编制的程序几乎能够在不改动的情况下在不同种类的计算机上运行且不易出错，这是汇编语言难以做到的，但高级语言程序翻译(编译)成最终能够直接执行的机器语言程序其难度却大大增加了。

由于汇编语言和机器语言一样都是面向机器的，故相对于面向用户的高级语言来说，它们都被称为低级语言，而FORTRAN、PASCAL、C、ADA、Java这类面向应用的语言则称之为高级语言。因此，编译程序就是指这样一种程序，通过它能够将用高级语言编写的源程序转换成与之在逻辑上等价的低级语言形式的目标程序，见图1-1。

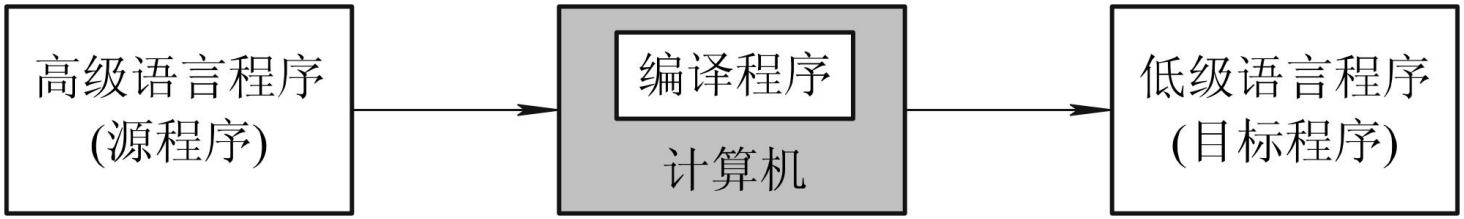


图1-1 编译程序的功能

一个高级语言程序的执行通常分为两个阶段，即编译阶段和运行阶段，如图1-2所示。编译阶段将源程序变换成目标程序；运行阶段则由所生成的目标程序连同运行系统(数据空间分配子程序、标准函数程序等)接收程序的初始数据作为输入，运行后输出计算结果。

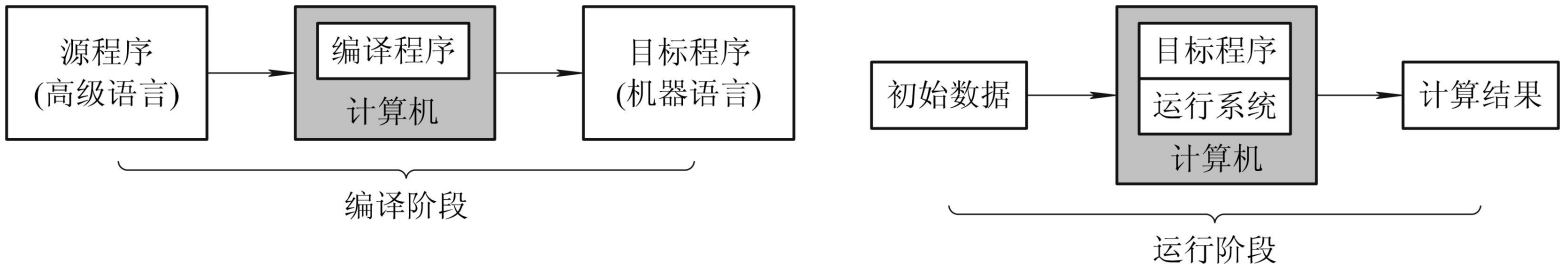


图1-2 源程序的编译和运行阶段

如果编译生成的目标程序是汇编语言形式的，那么在编译与运行阶段之间还要添加一个汇编阶段，它将编译生成的汇编语言目标程序再经过汇编程序变换成机器语言目标程序，如图1-3所示。

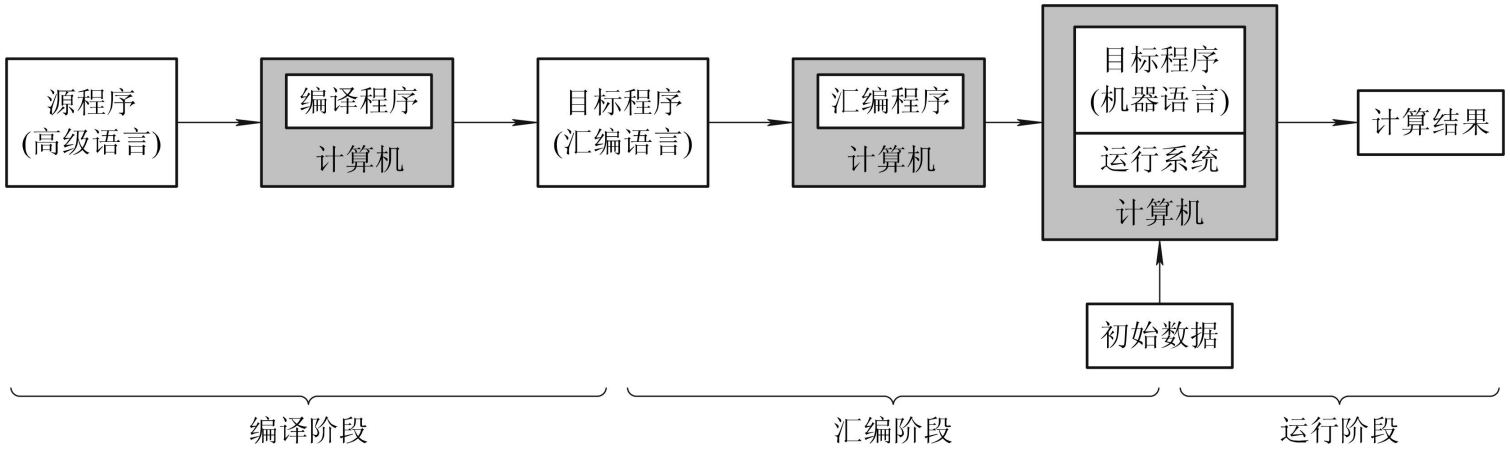
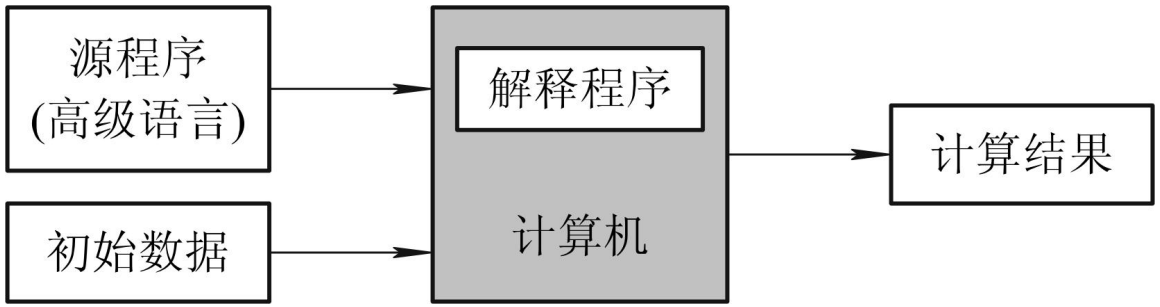


图1-3 源程序的编译、汇编和运行阶段

用高级语言编写的程序也可通过解释程序来执行。解释程序也是一种翻译程序，它将源程序作为输入，一条语句一条语句地读入并解释执行，如图1-4所示。解释程序与编译程序的主要区别是：编译程序将源程序翻译成目标程序后再执行该目标程序；而解释程序则逐条读出源程序中的语句并解释执行，即在解释程序的执行过程中并不产生目标程序。典型的解释型高级语言是BASIC语言。


源程序
(高级语言)
初始数据
解释程序
计算机
计算结果

1.2 编译程序的历史及发展

20世纪40年代，由于冯·诺伊曼在存储程序计算机方面的开创性工作，计算机可以执行编写的一串代码或程序，这些程序最初都是用机器语言(Machine Language)编写的。机器语言就是计算机能够执行的全部指令集合的二进制形式，例如：

C7 06 0000 0002

表示在IBM PC上使用的Intel 8x86处理器将数字2移至地址0000(十六进制)的指令。用机器语言编写程序很不方便且容易出错，因此这种代码形式很快就被汇编语言(Assembly Language)取代。在汇编语言中，指令和存储地址都以符号形式给出。例如，汇编语言指令

MOV X, 2

就与前面的机器指令等价(假设符号存储地址X是0000)。汇编程序(Assembler)再将汇编语言的符号代码和存储地址翻译成与机器语言相对应的二进制代码。汇编语言大大提高了编程的速度和准确性，至今人们仍在使用它，在有存储容量小和速度快的要求时尤其如此。但是，汇编语言依赖于具体机器的特性是无法改变的，这给编程和程序调试增加了难度。很明显，编程技术发展的下一个重要步骤就是用更简洁的数学定义或自然语言来描述和编写程序，它应与任何机器无关，而且也可通过一个翻译程序将其翻译为可在计算机上直接执行的二进制代码。例如，前面的汇编语言代码“MOV X, 2”可以写成一个简洁的与机器无关的形式“X=2”。

1954～1957年，IBM John Backus带领的一个研究小组对FORTRAN语言及其编译器进行了开发。但是，由于对编译程序的理论及技术研究刚刚开始，这个语言的开发付出了巨大的辛劳。与此同时，波兰语言学家Noam Chomsky开始了他的自然语言结构研究。Noam Chomsky根据文法(Grammar，产生语言的规则)的难易程度及识别它们所需的算法对语言进行了分类，定义了0型、1型、2型和3型这四类文法及其相应的形式语言，并分别与相应的识别系统相联系。

2型文法(上下文无关文法，Context-free Grammar)被证明是程序设计语言中最有用的文法，它代表着目前程序设计语言结构的标准。Noam Chomsky的研究结果最终使得编译器结构异常简单，甚至还具有自动化功能。有限自动机(Finite Automata)和正规表达式(Regular Expression)与上下文无关文法紧密相关，它们与Noam Chomsky的3型文法相对应，并引出了表示程序设计语言的单词符号形式。接着又产生了生成有效目标代码的方法——这就是最初的编译器，它们被沿用至今。随着对语法分析研究的深入，重点转移到对编译程序的自动生成的研究上。

开发的这种程序最初被称为编译程序的编译器，但因为它们仅仅能够自动完成编译器的部分工作，所以更确切地应称为分析程序生成器(Parser Generator)；这些程序中最著名的是Steve Johnson在1975年为UNIX系统编写的语法分析器自动生成工具YACC(Yet Another Compiler-Compiler)。类似地，有限自动机的研究也产生出另一种称为词法分析器自动生成工具(Scanner Generator)Lex。

20世纪70年代后期和80年代初，大量的研究都关注于编译器其他部分的自动生成，其中包括代码生成。这些努力并未取得多少成功，这是由于这部分工作过于复杂，对其本质不甚了解，在此不再赘述。

现今编译器的发展包括了更为复杂的算法应用程序，它用于简化或推断程序中的信息，这又与具有此类功能的更为复杂的程序设计语言发展结合在一起。其中典型的有用于函数语言编译Hindley-Milner类型检查的统一算法。目前，编译器已经越来越成为基于窗口的交互开发环境(Interactive Development Environment，IDE)的一部分，这个开发环境包括了编辑器、链接程序、调试程序以及项目管理程序。尽管近年来对IDE进行了大量的研究，但是基本的编译器设计在近40年中都没有多大的改变。

1.3 编译过程和编译程序结构

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，识别出一个个单词符号，如基本字(if、for、begin等)、标识符、常数、运算符和界符(如“(”、“)”、“=”、“;”)等，将所识别出的单词用统一长度的标准形式(也称内部码)来表示，以便于后继语法工作的进行。因此，词法分析工作是将源程序中的字符串变换成单词符号流的过程，词法分析所遵循的是语言的构词规则。

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(文法规则)把单词符号流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”。通过语法分析可以确定整个输入串是否构成一个语法上正确的“程序”。语法分析所遵循的是语言的语法规则，语法规则通常用上下文无关文法描述。

3．语义分析和中间代码生成

语义分析和中间代码生成的任务是对各类不同语法范畴按语言的语义进行初步翻译，包含两个方面的工作：一是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；二是在语义检查正确的情况下进行中间代码的翻译。注意，中间代码是介于高级语言的语句和低级语言的指令之间的一种独立于具体硬件的记号系统，它既有一定程度的抽象，又与低级语言的指令十分接近，因此转换为目标代码比较容易。把语法范畴翻译成中间代码所遵循的是语言的语义规则，常见的中间代码有四元式、三元式、间接三元式和逆波兰记号等。

4. 优化

优化的任务主要是对前阶段产生的中间代码进行等价变换或改造(另一种优化是针对目标机即对目标代码进行优化)，以期获得更为高效(节省时间和空间)的目标代码。常用的优化措施有删除冗余运算、删除无用赋值、合并已知量、循环优化等。例如，其值并不随循环而发生变化的运算可提到进入循环前计算一次，而不必在循环中每次循环都进行计算。优化所遵循的原则是程序的等价变换规则。

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)变换成特定机器上的机器语言程序或汇编语言程序，实现最终的翻译工作。最后阶段的工作因为目标语言的关系而十分依赖硬件系统，即如何充分利用机器现有的寄存器，合理地选择指令，生成尽可能短且有效的目标代码，这些都与目标机器的硬件结构有关。

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分模块进行设计。编译程序的结构示意如图1-5所示。

编译过程中源程序的各种信息被保留在不同的表格里，编译各阶段的工作都涉及构造、查找或更新有关的表格，编译过程的绝大部分时间都用在造表、查表和更新表格的事务上，因此，编译程序中还应包括一个表格管理程序。

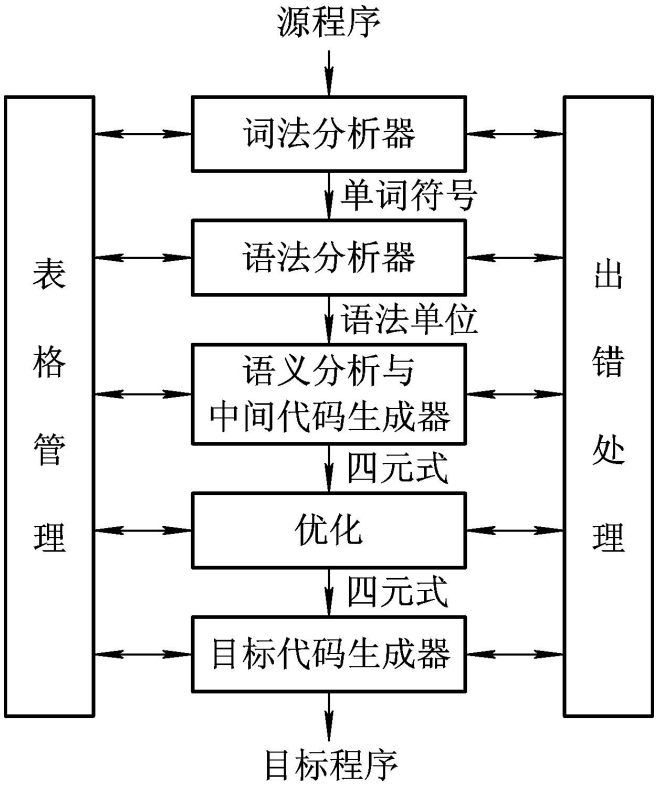


图1-5 编译程序的结构示意

出错处理与编译的各个阶段都有联系，与前三个阶段的联系尤为密切。出错处理程序应在发现错误后，将错误的有关信息如错误类型、出错地点等向用户报告。此外，为了尽可能多地发现错误，应在发现错误后还能继续编译下去。

一个编译过程可分为一遍、两遍或多遍完成，每一遍完成所规定的任务。例如，第一遍只完成词法分析的任务，第二遍完成语法分析和语义加工工作并生成中间代码，第三遍再实现代码优化和目标代码生成。当然，也可一遍即完成整个编译工作。至于一个编译程序究竟应分为几遍，如何划分，这和源程序语言的结构与目标机器的特征有关。分多遍完成编译过程可以使整个编译程序的逻辑结构更清晰，但遍数多势必增加读写中间文件的次数，从而消耗过多的时间。

词法规则

词法规则是单词符号的形成规则，它规定了哪样的字符串构成一个单词符号。

例如 float r，h，s；

s = 2*Π*r*(h＋r)；

词法规则

上述源程序通过词法分析识别出如下单词符号：

基本字 **float**

标识符 **r、h、s**

常数 **Π、2**

算符 **＊、＋**

界符 **（、）、；、，、＝**

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从单词符号串中识别出各种语法单位（如表达式、说明、语句等），并进行语法检查，即检查各种语法单位在语法结构上的正确性。

语法规则

语言的语法规则规定了如何从单词符号形成语法单位，语法规则是语法单位的形成规则。

例如 float r，h，s；

s = 2*Π*r*(h＋r)；

语法规则

单词符号串 s=2*Π＊r＊(h＋r)中，“s”是<变量>，单词符号串

“2＊Π＊r＊(h＋r)”

组合成<表达式>这样的语法单位，由<变量>=<表达式>构成<赋值语句>这样的语法单位。

3. 语义分析和中间代码生成

语义分析的任务是首先对每种语法单位进行静态的语义审查，然后分析其含义，并用另一种语言形式（比源语言更接近于目标语言的一种中间代码或直接用目标语言）来描述这种语义。

例如，前例中

将s ＝2*Π＊r *(h＋r)翻译成如下形式的四元式中间代码：

(1) (*, 2, 3.1416, T_1)

(2) (*, T_1, r, T_2)

(3) (+, h, r, T_3)

(4) (*, T_2, T_3, T_4)

(5) (=, T_4, __, s)

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等价变换或改造，以期获得更为高效即省时间和空间的目标代码。优化主要包括局部优化和循环优化等，例如上述四元式经局部优化后得：

(1) (*, 6.28, r, T_2)

(2) (+, h, r, T_3)

(3) (*, T_2, T_3, T_4)

(4) (=, T_4, ___, s)

5.目标代码生成

目标代码生成的任务是将中间代码变换成特定机器上的绝对指令代码或可重定位的指令代码或汇编指令代码。

表格管理和错误处理

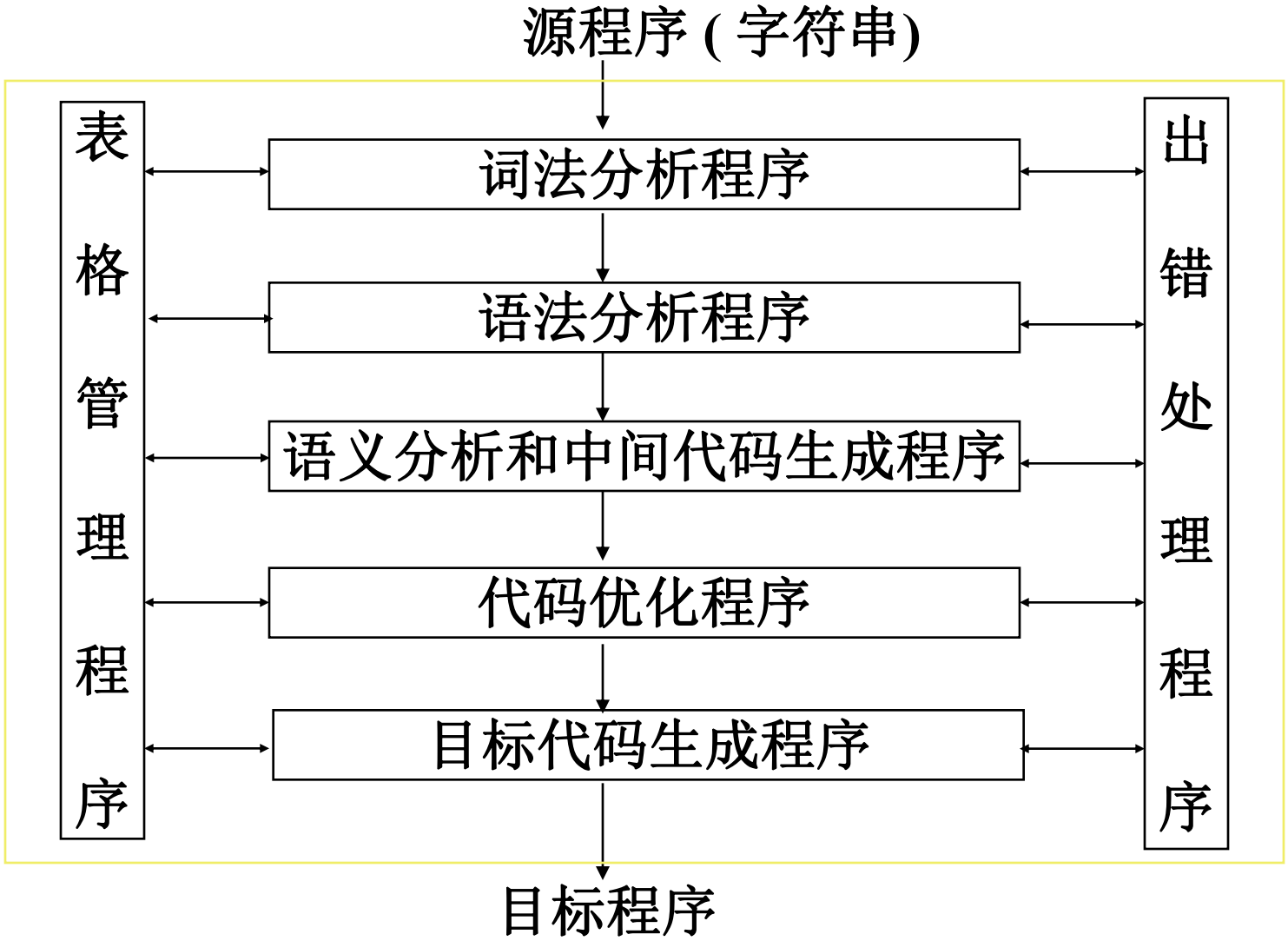
在编译程序的各个阶段中，都要涉及表格管理和错误处理。

编译程序在工作过程中需要建立一些表格，以登记源程序中所提供的或在编译过程中所产生的一些信息，编译各个阶段的工作都涉及到构造、查找、修改或存取有关表格中的信息，因此，在编译程序中必须有一组管理各种表格的程序。

表格管理和错误处理

一个好的编译程序在编译过程中，应具有广泛的程序查错能力，并能准确地报告错误的种类及出错位置，以便用户查找和纠正，因此，在编译程序中还必须有一个出错处理程序。

编译程序的结构


源程序 (字符串)
词法分析程序
语法分析程序
语义分析和中间代码生成程序
代码优化程序
目标代码生成程序
目标程序
表格管理程序
出错处理程序

1.4 编译程序的开发

由于计算机语言功能的完善、硬件结构的发展、环境界面的友好等都对编译程序提出了更多、更高的要求，因而构造一个编译系统并非易事。虽然编译理论和编译技术的不断发展已使编译程序的生产周期不断缩短，但是要研制完成一个编译程序仍需要相当长的时间，工作也相当艰巨。因此，如何高效、高质量地生成一个编译程序一直是计算机系统设计人员追求的目标。

编译程序的任务是把源程序翻译成某台计算机上的目标程序，因此，开发人员首先要熟悉这种源程序语言，对源程序语言的语法和语义要有准确无误的理解。此外，开发人员还需确定编译程序的开发方案及方法，这是编译开发过程中最关键的一步，其作用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和扩充。选择合适的语言编写编译程序也是非常重要的，语言选择不当会使开发出来的编译程序的可靠性变差，难以维护且质量也无法保证。目前大部分编译程序都是用C语言之类的高级语言编写的，这不仅减少了开发的工作量，也缩短了开发周期。最后，开发人员对目标机要有深入的研究，这样才能充分利用目标机的硬件资源和特点，产生质量较高的目标程序。

1. 自编译

用某种高级语言编写自己的编译程序称为自编译。例如，假定A机器上已有一个PASCAL语言可以运行，则可以用PASCAL语言编写出一个功能更强的PASCAL语言编译程序，然后借助于原有的PASCAL编译程序对新编写的PASCAL编译程序进行编译，从而在编译后即得到一个能在A机器上运行的功能更强的PASCAL编译程序。

2．交叉编译

交叉编译是指用A机器上的编译程序来产生可在B机器上运行的目标代码。例如，若A机器上已有C语言可以运行，则可用A机器中的C语言编写一个编译程序，它的源程序是C语言程序，而产生的目标程序则是基于B机器的，即能够在B机器上执行的低级语言程序。

以上两种方法都假定已经有了一个系统程序设计语言可以使用，若没有可使用的系统程序设计语言，则可采用自展或移植的方法来开发编译程序。

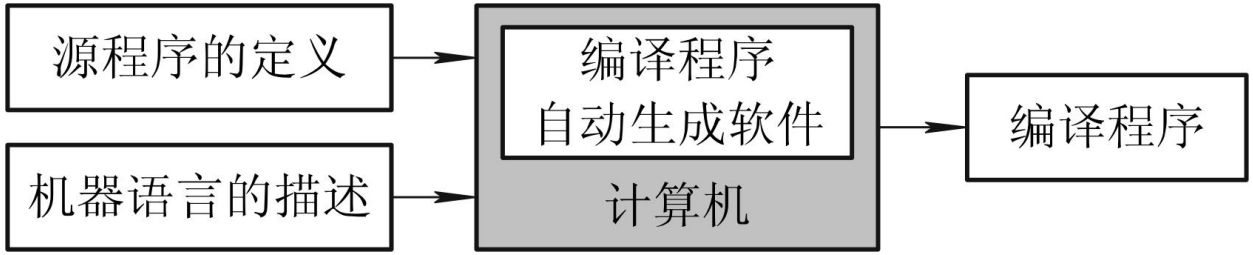
3．自展

自展的方法是：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编写出它的编译程序T_0；再把语言L_0扩充到L_1，此时有$L_0 \subset L_1$，并用L_0编写L_1的编译程序T_1(即自编译)；然后再把语言L_1扩充为L_2，此时有$L_1 \subset L_2$，并用L_1编写L_2的编译程序T_2……这样不断扩展下去，直到完成所要求的编译程序为止。

4．移植

移植是指A机器上的某种高级语言的编译程序稍加改动后能够在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。移植具有一定的局限性。

用系统程序设计语言来编写编译程序虽然缩短了开发周期并提高了编译程序的质量，但实现的自动化程度不高。实现编译程序的最高境界是能够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以及机器语言的描述输入到该软件中，就能自动生成这个语言的编译程序，如图1-6所示。


源程序的定义
机器语言的描述
编译程序
自动生成软件
计算机
编译程序

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进行了大量研究工作，随着形式语言学研究的发展和编译程序自动生成工作的进展，已经出现了一些编译程序中某一部分的自动生成系统，如UNIX操作系统下的软件工具Lex和YACC等。

1.5 构造编译程序所应具备的知识内容

要在某一台机器上为某种语言构造一个编译程序，必须掌握下述三方面的内容。

(1) 对被编译的源语言(如C、PASCAL等)，要深刻理解其结构(语法)和含义。例如，下面的for循环语句：

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

就存在对循环终值的理解问题。一种理解是以第一次进入for循环的i值计算出循环终值，此循环终值在循环中不再改变，也即循环终值为11；而另一种理解则是循环终值表达式10+i中的i值随循环在不断地改变，此时for语句将出现死循环。如TURBO PASCAL就是按第一种语义进行翻译的，而TURBO C和VC++6.0则是按第二种语义进行翻译的。此外，如果出了循环后还要引用i值，那么这个i值究竟是循环终值还是循环终值加1？因此，对语义的不同理解可以得到不同的编译程序。

其次，C语言中的for循环语句还可以写成下面的形式：

```
for ( i=0, j=5; i<3, j>0; i++, j-- )
    printf ("%d,%d\n", i, j );
```

逗号表达式“i<3, j>0”作为循环终值表达式，C语言规定该表达式的值取逗号表达式中最右一个表达式的值，因此当i值为3时循环并不终止，只有当j值为0时循环才结束。不了解C语言的逗号表达式的功能就会得到错误的运行结果。

此外，下面C语言的函数：

```
#include<stdio.h>
int f(int x,int y)
{
    return x+y;
}
void main()
{
    int i=3;
    printf("%d\n",f(i,++i));
}
```

C编译程序对函数参数传递的处理是由右向左进行的，因此，先传递的是第二个参数++i，即i值先进行自增，由3变为4，也即这时函数f的两个实参的值都为4，最终程序的运行结果是8，而不是7。不了解C语言的函数传递方式就很容易得到错误的结果。

即使是同一个C语言语句，不同版本的编译系统翻译的结果也是不一样的。例如：

```
i=3;
k=(++i)+(++i)+(++i);
```

对VC++ 6.0版本的C语言来说，执行语句“k=(++i)+(++i)+(++i);”的过程是先执行前两个“++i”，即先对i进行两次自增，i值变为5，然后相加得到10；接下来再将这个10与第三个“++i”相加，也是先对i进行自增，其值由5变为6，最后10加6得到16，所以k值最终为16。对Turbo C版本的C语言来说，执行语句“k= (++i)+(++i)+(++i);”的过程是先执行三次“++i”，即先对i进行三次自增，i值变为6，然后再将自增后的三个i值相加，其结果为18，所以k值最终为18。也即，不同的C编译程序给出了不同的解释。

(2) 必须对目标机器的硬件和指令系统有深刻的了解。例如，两个数相加的指令在8086/8088汇编中假定用下面两种指令实现：

ADD AX，06或ADD BX，06

粗略看来，这两条加法指令除了寄存器不同外没有本质上的差别，其实不然。由于AX是累加器，因此，从机器指令的代码长度来说(见附录1)，第一条指令比第二条指令节省一个字节。此外，从PC硬件结构来看，AX本身就是累加器且相加的结果也在累加器中，这就节省了传送的时间；而第二条指令则先要将BX中的值送到累加器中，相加后又要从累加器中取出结果再送回寄存器BX中。显然，第二条指令要比第一条指令费时，因此，在可能的情况下，应尽量生成像第一条指令这样的目标代码。

(3) 必须熟练掌握编译方法，编译方法掌握的如何将直接影响到编译程序的成败，一个好的编译方法可能得到事半功倍的效果。

由于编译程序是一个极其复杂的系统，故在讨论中只好把它分解开来，一部分一部分地进行研究。在学习编译程序的过程中，应注意前后联系，切忌用静止的、孤立的观点看待问题；作为一门技术课程，学习时还必须注意理论联系实际，多练习、多实践。

习 题 1

1.1 完成下列选择题：

(1) 下列叙述中正确的是_____。

A．编译程序是将高级语言程序翻译成等价的机器语言程序的程序

B．机器语言因其使用过于困难，所以现在计算机根本不使用机器语言

C．汇编语言是计算机唯一能够直接识别并接受的语言

D．高级语言接近人们的自然语言，但其依赖具体机器的特性是无法改变的

(2) 将编译程序分成若干个“遍”是为了　　。

A．提高编译程序的执行效率

B．使编译程序的结构更加清晰

C．利用有限的机器内存并提高机器的执行效率

D．利用有限的机器内存但降低了机器的执行效率

(3) 构造编译程序应掌握　　。

A．源程序　　B．目标语言

C．编译方法　　D．以上三项

(4) 编译程序绝大多数时间花在　　上。

A．出错处理　　B．词法分析

C．目标代码生成　　D．表格管理

(5) 编译程序是对　　。

A．汇编程序的翻译

B．高级语言程序的解释执行

C．机器语言的执行

D．高级语言的翻译

1.2 计算机执行用高级语言编写的程序有哪些途径？它们之间主要区别是什么？

1.3 请画出编译程序的总框图。如果你是一个编译程序的总设计师，设计编译程序时应当考虑哪些问题？