第 1 章　编译程序的基本概念

计算机语言的层次结构如图 1.1 所示：上层是大家编写的高级程序语言比如：C 语言、C++ 或 java 语言。中间会有一层汇编语言，不同目标机的汇编语言不一样，最底层机器执行的指令，汇编语言离机器指令已经非常非常近了，但它还不是真正的机器指令，你可以把机器指令理解成 0、1 的码，编译程序需要把高级语言转化成汇编语言，汇编程序再把汇编语言翻译成真正的目标指令。编译器所做的就是红色的部分。如果不经过汇编语言也可以直接把高级语言翻译为机器语言。或者大家经常也会遇到解释性程序，解释程序和编译程序做一样的工作，也是把高级语言翻译成目标机的指令，只不过解释程序是逐条翻译，写一句就执行一句，而编译程序是整体翻译。解释性语言比如：python、shell、perl。同样我们也可以把 java 程序转化为 python 的程序，这个叫做转换程序。当然也存在反汇编程序和反编译程序。

为了更好的解释交叉汇编程序，我们举个例子。比如，当需要在 window 环境下生成在 Linux 环境下可执行的程序。正常的做法是，如果需要的 window 环境下的可执行程序，就将源代码在 window 环境下编译然后生成可执行程序在 windows 环境下执行，如果需要的是 Linux 环境下的可执行程序，就将源代码在 Linux 环境下编译然后生成可执行程序在 Linux 环境下执行。由于可执行程序是跟操作系统以及具体的硬件和底层的指令是相关的。而所谓的交叉汇编程序是指，能够在 windows 环境下生成一个在 Linux 环境下可执行的程序，但是这个可执行程序在 windows 环境下是无法执行的。在某些特殊的场景，是需要这么去做的。当一个真实所需的环境搭建起来代价比较高，我们可以在一个简单容易搭建的环境下，去生成一个特定环境下的可执行程序。

计算机中语言的层次体系

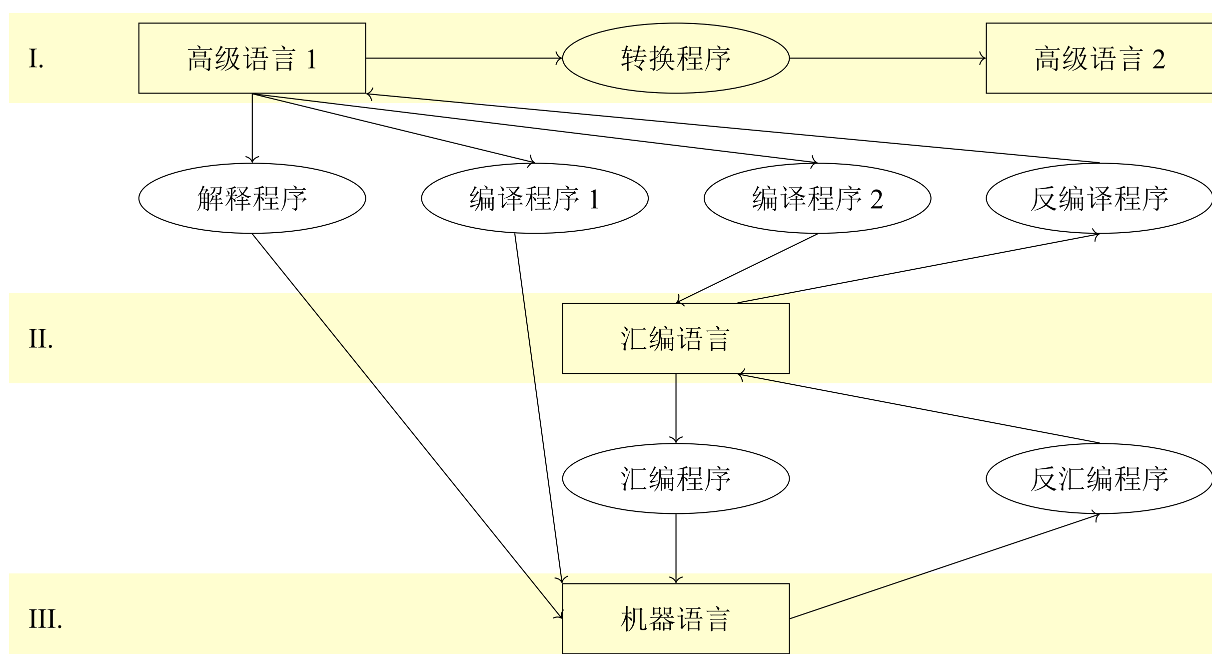


图 1.1: 计算机中语言的层次体系

1.1 什么是编译程序？

编译程序（compiler）是一种翻译程序，它特指把某种高级程序设计语言翻译成具体计算机上的低级程序设计语言。

高级语言执行过程有两个阶段：第一个阶段叫做编译的阶段：从源语言经过编译程序生成目标语言。这个目标语言就是要生成的语言，如果我们要把汇编语言做为目标语言的话，那么汇编语言就是编译的结果，编译器做的工作就是把高级语言转化为汇编语言。拿到一段程序后，运行程序时可能与外界有数据的交互，最终通过这些交互会把程序的结果输出出来。这门课程给大家介绍的是编译程序这部分。

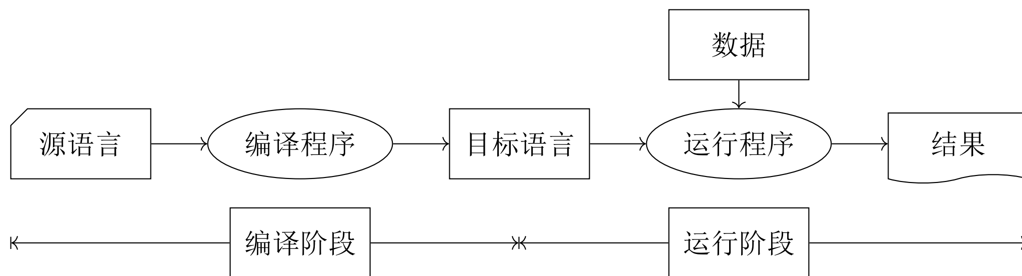


图 1.2: 高级语言的执行过程

⋇ 什么是解释程序？

解释程序（interpreter）也是一种翻译程序，将某高级语翻译成具体计算机上的低级程序设计语言。

解释程序会逐条读取代码，读取之后就会把代码翻译成目标程序，直接进行执行，这个过程会反复执行，所以在执行 perl 的时候不会产生目标代码。

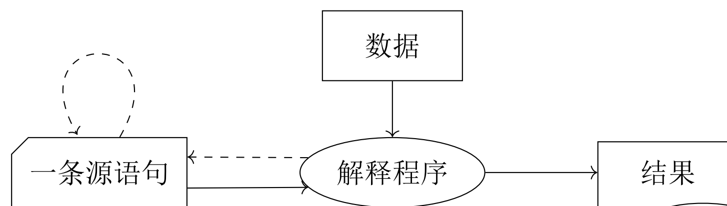


图 1.3: 解释程序的执行过程

编译程序和解释程序有两个重要的区别：

1. 前者有目标程序而后者无目标程序，但是解释性程序执行比较慢；
2. 前者运行效率高而后者便于人机对话，这种解释性程序一句一句执行便于调试。

编译程序和解释程序的目的都是为了帮助用户执行程序，实现二者的算法是类似的。

1.2 编译程序结构

编译程序包括词法分析、语法分析、语义分析、优化处理和代码生成五个阶段，如图 1.4 所示。

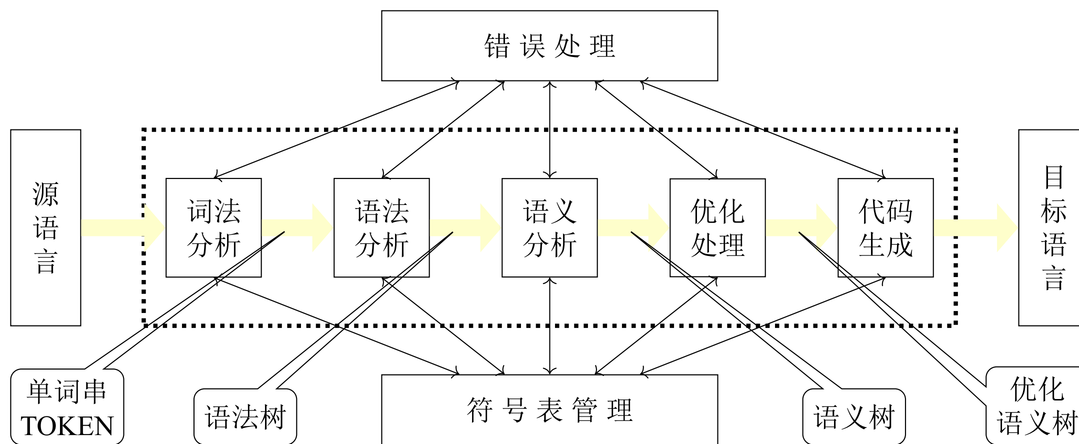


图 1.4: 编译程序总体结构框图

上图中左侧表示输入的源程序，右侧表示经过编译处理得到的目标程序。比如输入 C 语言源程序，经过编译处理，输出该源程序对应的汇编语言或者是可执行的机器代码。其中会包含若干个步骤分别是：词法分析、语法分析、语义分析、优化处理、代码生成。这五个步骤构成了编译器最核心的部件。首先拿到源语言会进行词法分析生成单词 TOKEN 串，词法分析的作用是识别在编译中什么是一个词；拿到了 TOKEN 串后生成语法树，语法分析是最重要的内容，语法分析的作用是告诉我们一句话的结构是什么，比如中文的主谓宾定状补；之后我们对语法分析的结果进行语义分析，语义分析告诉我们每一句高级语言的句子是什么意思，举个例子：$a = b$ 代表的是把 b 的值赋值给 a 这个变量，据此我们可以得到中间代码；中间代码并不是最高效的，中间步骤可能会存在冗余，我们要执行优化处理得到优化后的代码；最终优化后的代码会生成我们的目标代码，也就是我们所谓的汇编语言程序或者机器的指令。

这些步骤的执行中有两个问题需要注意：第一个问题：待编译的程序出错了怎么办？如何告诉我们错误在哪？错误处理模块就是负责这部分工作，所有编译程序里都有错误处理模块。第二个问题：符号表管理，我要知道这个程序里有哪些函数，每个函数有哪些形参和局部变量，每个变量都是什么类型，存在什么位置等？比如在 C 语言中变量声明语句 $inta$，我们是如何知道 a 被定义为一个整形的变量？C 语言是典型的强数据类型的语言，这种语言需要在你使用变量之前对变量的类型进行定义，而且这个类型在 C 语言中是不允许修改的，但像 python 就是弱数据类型的语言，对变量的类型不需要定义。这些变量及其语义信息（如变量的数据类型和存储位置等）都存放在符号表里。

⋇ 编译程序与机器翻译的类比：

我们用自然语言的处理来类比一下计算机的编译过程。

机器翻译是指计算机把一种自然语言翻译成另一种自然语言。下面通过汉译英示例分析机器翻译的过程。

例 1.1 我们用树叶和颜料能够制作美丽的图画

1. 词法分析：首先分词, 图中 r 代表代词,p 代表介词,n 代表名词。

我们 用 树叶 和 颜料 能够 制作 美丽的 图画

I. 词法分析：r() p() n() c() n() u() v() a() n().

图 1.5: 一个简单的翻译过程例子

2. 句法分析：句子的结构用树来表示，称之为句法树，描述了整个句子的结构。

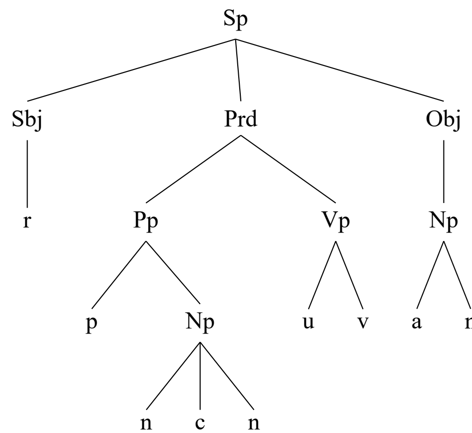


图 1.6: 语法树

3. 语义分析：除了这个结构之外，我们还要知道这个句子表达什么动作执行了什么事情。

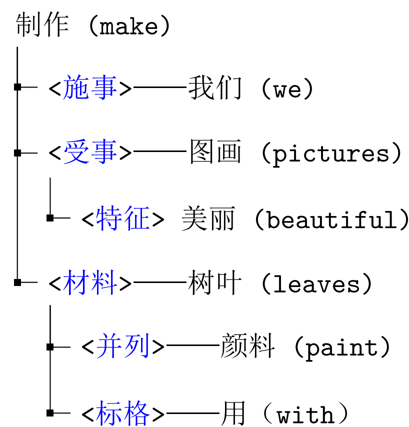


图 1.7: 语义网

4. 优化处理：
5. 目标生成：We can make beautiful pictures with leaves and paint.

在上述英译汉的各个阶段中，我们需要一些词典和知识库的支持，例如汉英词典等。

1.3 编译程序的实现机制

类比机器翻译，我们是不是也可以做高级程序设计语言的翻译呢？在做编译程序的时候有一个概念叫遍。

定义 1.1 遍：编译程序对源程序或等价程序从头至尾扫描的次数。

一般来说我们现在使用的编译器是两遍的编译器：

第一遍：词法分析、语法分析和语义分析；

第二遍：目标代码生成和目标代码优化。

当一遍中包含若干阶段时，各阶段的工作是穿插进行的。由于语法分析在整个编译器中起到核心的作用，我们会提到语法制导的翻译或语法制导的编译。所谓语法制导的编译，用语法分析来做引导的一种编译器。例如：使语法分析器处于核心位置。当语法分析需要下一个单词时，就调用扫描器，识别一个单词并生成相应 Token；一旦识别出一个语法单位时，就调用中间语言生成器，完成相应的语义分析并产生中间代码。

1.4 编译过程实例

Listing 1.1: Pascal 程序片段

```
1   Var $a,b$:integer;
2       ...
3       B:=a+2*5
```

编译过程如下：

1. 词法分析：识别单词并分类

单词类码

(1) 关键字（k）—var，integer;

(2) 标识符（i）—a，b;

(3) 常　数（c）—2，5;

(4) 界　符（p）—，；: := +

图 1.8: 词法分析

2. 语法分析：组词成句及语法错误检查

例：$b := a + 2 * 5$ 的分析过程如下所示：

算数表达式的层次结构

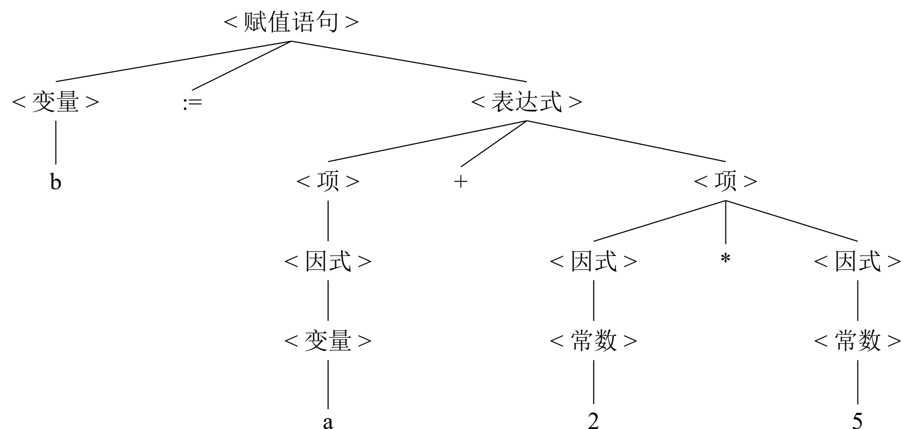


图 1.9: 赋值语句 b:=a+2*5 的语法树

画出这个句子的语法树；第一步知道这个句子是一个赋值语句，第二步知道把表达式赋值给变量，表达式由两部分组成，第一项是一个由变量 a 构成的因式，第二项是由常数 2 因式和常数 5 因式构成。这些就构成了整个赋值语句。算数表达式的层次结构为：常数、因式、项。

上述句法分析告诉我们这个句子是什么样的结构

3. 语义分析：分析各种语法成分的语义特征

构建标识符的语义辞典——符号表:

符号表

var a,b:integer;
...
b:=a+2*5;

名字	类型	种类	地址
a	i	v	
b	i	v	

数据区

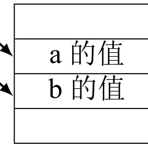
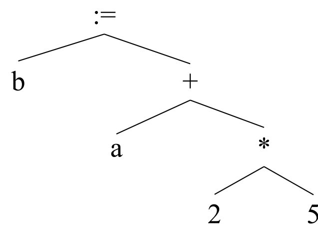


图 1.10: 构建符号表

我们要得到句子的语义动作信息，这个动作是什么？a 是什么？b 是什么？a 变量的值放在物理内存的哪个位置？符号表会告诉我们这些事情。

如: b:=a+2*5



或

(1)(* 2 5 t_1)

(2)(+ a t_1 t_2)

(3)(:= a t_1 t_2)

图 1.11: 语法树

之后我们会把上述赋值语句写成这种树的形式，我们就知道了这个式子的意思是把 2 和 5 相乘再与 a 相加最后赋值给 b。当然我们也可以把这个树写成四元式的形式，第一元表示算符，第二元和第三元是运算的对象，最后一元是运算的结果。

4. 优化：提高目标程序质量的工作

例: $b := a + 2 * 5$

经常数合并，可分别获得优化后的中间代码：

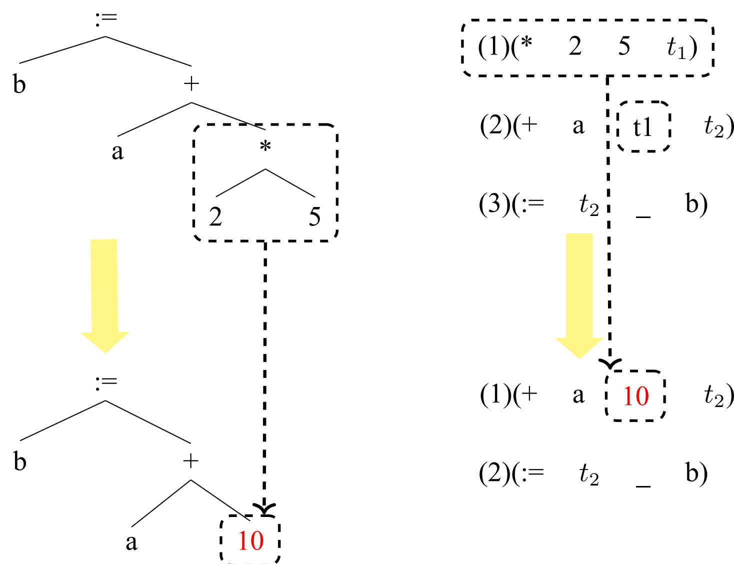


图 1.12: 优化后的中间代码

对计算机来说不必反复计算 2*5 这个过程，经过优化四元式由三条变为两条，这个是运行前的优化，而寄存器优化是负责运行时的优化。

1. 目标代码生成：产生计算机可识别的语言

   通常，是把中间语言转换成目标代码，把之前的四元式生成汇编语言

(1)(+ a 10 t_2)

(2)(:= t_2 _ b)

(1)LD R , a

(2)ADD R , 10

(3)ST R , b

图 1.13: 生成汇编语言

1.5 本章小结

本章介绍了编译原理的课程目标，即设计编译器以实现对高级程序语言的理解、分析和处理，同时为解决其他复杂工程问题提供一种新思路。给出编译程序的定义及其逻辑结构，并分析了编译程序实现机制，最后通过实例分析编译过程。