编译原理笔记

Pionpill *

本文档为作者学习《编译原理》†一书时的笔记，

2021 年 10 月 21 日

前言：

本篇笔记用于应付学校期末考试，本人对这方面也没有进行深入研究。内容浅显，不适合考研等深入学习。

此外，本篇笔记是对原书的提炼总结，只能辅助原书进行学习，有大量例子等理解性文字并未进行说明，如有需要请购买原书。本笔记文案多为原书摘抄或个人总结，图片为本人使用 TikZ 绘制，若需进行引用，可前往下载 LaTeX 源代码[1]，请遵守 GPL-v3 协议。

2021 年 10 月 21 日

目录

*笔名：北岸，电子邮件：673486387@qq.com，Github：https://github.com/Pionpill

†《程序设计语言编译原理》: 陈火旺，国防工业出版社，2020 年印刷

[1]https://github.com/Pionpill/Notebook/tree/Pionpill/Lessons

一、引论

1.1 什么是编译原理

计算机上执行一个高级语言程序通常分为两步：第一步，用一个编译程序把高级语言翻译成机器语言程序；第二部，运行所得到的机器语言程序求计算结果。

通常所说的翻译程序是这样一个程序，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)，前后者逻辑上是等价的。这样的一个翻译程序就称为编译程序。

高级语言除了先编译后执行外，有时也可 “解释” 执行。一个源语言的解释程序就是这样的程序，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。

1.2 编译过程概述

编译程序的工作过程一般可分为五个阶段：词法分析，语法分析，语义分析与中间代码产生，优化，目标代码生成。

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。这一过程类似于英文翻译中认识每一个单词的意义。

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，如 “短语”，“子句”，“句子”,“程序段” 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。

语义分析和中间代码产生 (例子见书 P3)

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个方面的工作。首先，对每种语法范畴进行静态语义检查。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到。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。

优化

优化的的任务在于对前端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 (省时间省空间) 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，删除无用代码等等。有时为了便于 “并行运算”，还可对代码进行并行化处理。优化所依循的原则

是程序的等价变换规则。

目标代码生成

这一阶段的任务是：把中间代码 (优化处理过后) 变换成特定机器上的低级语言代码。这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。

目标代码的形式可以是绝对指令代码或可重定位的指令代码或汇编指令代码。如目标代码是绝对指令代码，则这种目标代码可立即执行。如目标代码是汇编指令代码，则需汇编器汇编后才能进行。

现代多数编译程序产生的是可重定位的指令代码。这种目标代码在运行前必须借助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 (常数) 在主存中的位置，装入内存中指定的起始地址，使之称为一个可运行的绝对指令代码程序。

1.3 编译程序的结构

1.3.1 编译程序总框

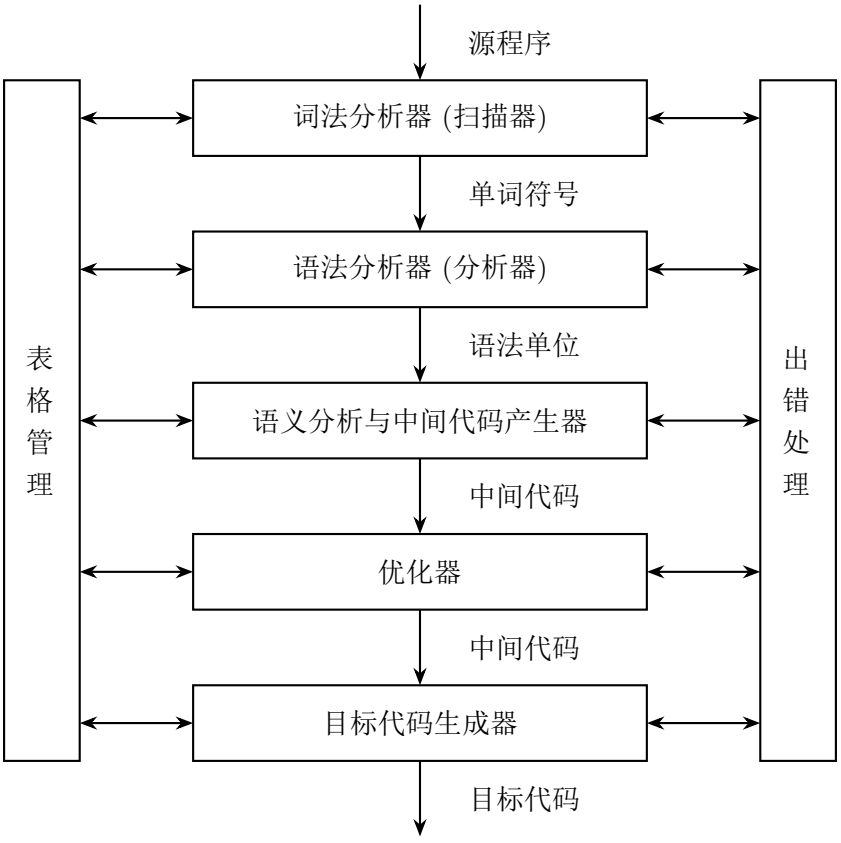

图 1.1 编译程序总框

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中间代码产生。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，分析器和中间代码

产生器并非截然分开，而是相互穿插。

1.3.2 表格与表格管理

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情况。在编译程序使用的各种表格中，最重要的是符号表。它用来登记源程序中出现的每个名字以及名字的各种属性。

编译各阶段都涉及到构造，查找或更新有关表格。

1.3.3 出错处理

编译程序应能对出现在源程序中的错误进行处理。这部分工作由专门的一组程序 (出错处理程序) 完成。一个好的编译程序应找到错误，并最小化错限制错误造成的影响，使得源程序的其他部分能够继续被编译，以便进一步发现其他可能的错误。

编译过程的每一阶段都可能检测出错误，绝大部分错误出现在编译的前三阶段。源程序中的错误通常可分为语法错误与语义错误两大类。

- 语法错误

指源程序中不符合语法 (词法) 规则的错误，可在语法分析与词法分析时检测出来。例如非法字符，括号不匹配等。

- 语义错误

指源程序中不符合语义规则的错误，一般在语义分析时检测出来，有的语义错误需要在运行时检测出来。例如类型不一致，作用域错误等

1.3.4 遍

编译过程具体实现时，受不同限制，往往将编译程序组织为若干遍 (pass)。所谓“遍”就是对源程序或源程序中间结果从头到尾扫描一次，并作有关的加工处理，生成新的中间结果或目标程序。通常，每遍地工作由从外存上获得地前一遍地中间结果开始，完成它所含地有关工作之后，再把结果记录于外存。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。

遍数多一点有一个好处，即整个编译程序的逻辑结构可能清晰一点。但遍数多势必增加输入/输出所消耗的时间。

1.3.5 编译前端与后端

概念上，我们有时把编译程序分为编译前端和编译后端。前端主要由与源语言有关但与目标无关的部分组成。包括：词法分析，语法分析，语义分析与中间代码产生，有的代码优

化工作也能包含在内。后端包括编译程序中与目标机有关的那部分，如与目标机有关的代码优化和目标代码生成等。通常，后端不依赖于源语言而仅依赖于中间语言。

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。也可以将几种源语言编译成相同的中间语言，然后为不同的前端配上相同的后端。

1.4 编译程序与程序设计环境

程序设计环境：编译程序与程序设计工具一起构成。程序设计工具包括：编辑程序，连接程序，调试工具等。

集成化的程序设计环境：特点是将相互独立的程序设计工具集成起来，以便为程序员提供完整的，一体化的支持，从而进一步提高程序开发效率，改善程质量。

下面以 Ada 语言的程序设计环境 APSE 为例，介绍程序设计环境的基本构成与主要工具。

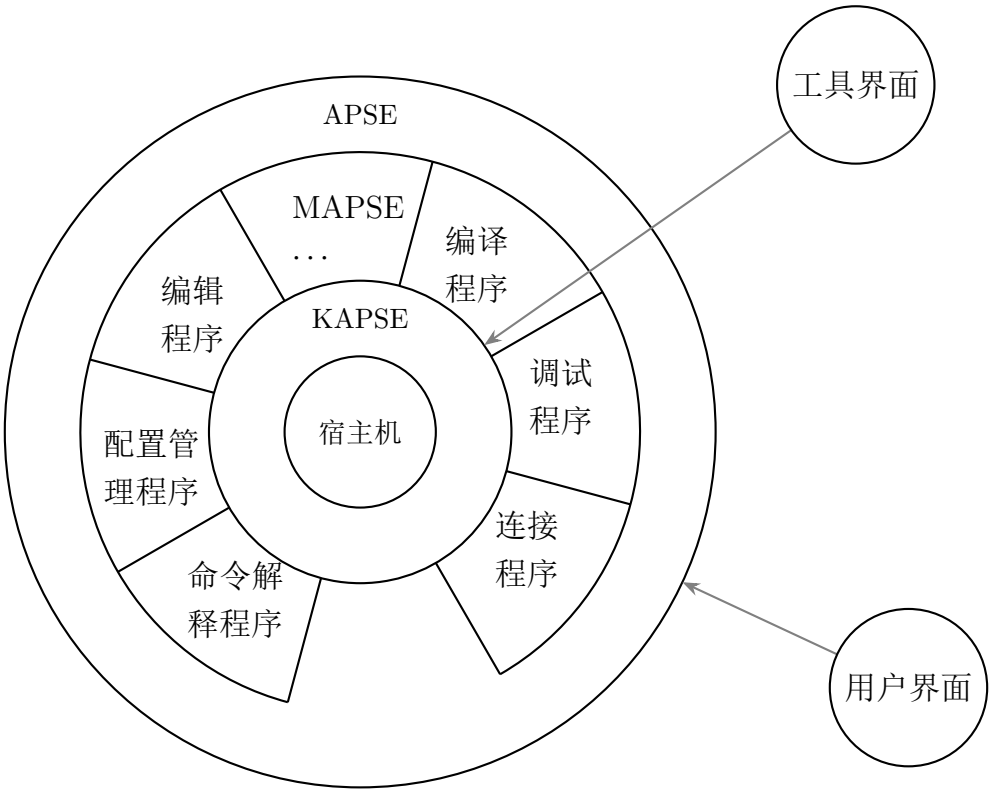

图 1.2 Ada 程序设计环境

最内层是宿主计算机系统,包括硬件,宿主操作系统和其他支撑软件;第一层是核心 APSE (KAPSE)，包含环境数据库，通信即运行时支撑功能等。第二层，最小 APSE (MAPSE)，包含 Ada 程序开发及维护的基本工具，这些工具包括编译程序，编辑程序，连接程序，调试程序等。第三层，APSE，在 MAPSE 外面再机上更广泛的工具构成完整的 APSE。

在一个程序设计环境中，编译程序起着中心作用。连接程序，调试程序等的工作直接依赖于编译程序所产生的结果。而其他工具的构造常常要用到编译的原理，方法和技术。

1.5 编译程序的生成

我们用一种 T 形图来表示源语言 S，目标语言 T 和编译程序实现语言 I 之间的关系。

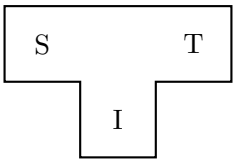

图 1.3 T 形图

如果 A 机器上已有一个用 A 机器代码实现的某高级语言 L_1 的编译程序，则我们可以用 L_1 语言编写另一种高级 L_2 的编译程序，把学好的 L_2 编译程序经过 L_1 编译程序编译后就可得到 A 机器代码实现的 L_2 编译程序。

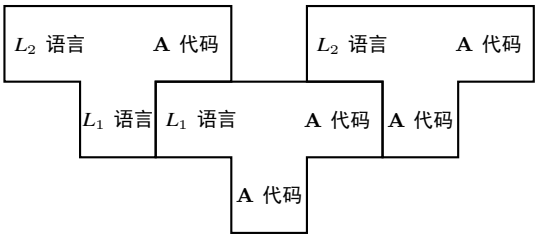

图 1.4 用 L_1 语言编写编译程序

采用一种所谓 “移植” 方法，我们可以利用 A 机器上已有的高级语言 L 编写一个能够在 B 机器上运行的高级语言 L 的编译语言，然后把该源程序经过 A 机器上的 L 编译程序编译后得到能在 A 机器上运行的产生 B 机器代码的编译程序，用这个编译程序再一次编译上述编译程序源程序就得到了能在 B 机器上运行的产生 B 机器代码的编译程序。

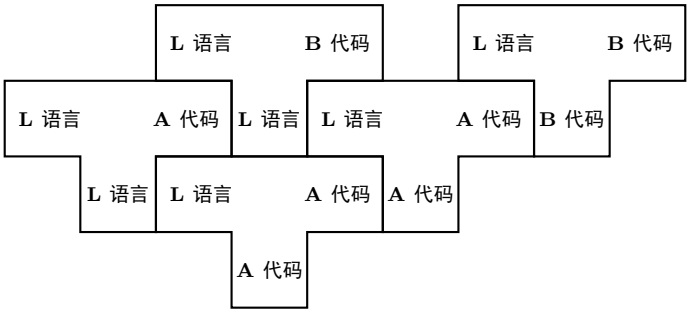

图 1.5 编译程序 “移植”

自编译方法：先对语言的核心部分构造一个小小的编译程序 (可用低级语言实现)，再以它为工具构造一个能够编译更多语言成分的较大编译程序。如此扩展下去，就能像滚雪球一样，越滚越大。通过这一系列自展途径而形成编译程序的过程叫做自编译过程。

在某一台机器上为某种语言构造一个编译程序，必须掌握以下内容：

1. 源语言：对被编译的源语言，要深刻理解其结构 (语法) 和含义 (语义)。

2. 目标语言：假定目标语言是机器语言，那么就必须搞清楚硬件的系统结构和操作系统的功能。

3. 编译方法：把一种语言程序翻译为另一种语言程序方法很多，但必须准确地掌握一二。

二、高级语言及其语法描述

2.1 程序语言的定义

2.1.1 语法

任何语言程序就可看成是一定字符集 (字母表) 上的一字符串 (有限序列)。所谓的语法是指这样的一组规则，用它可以形成和产生一个合适的程序。这些规则的一部分称为词法规则，另一部分称为语法规则 (产生规则)。

如：0.5 * X1 + C。可看作常数 0.5，标识符 X1，C。运算符 *，+。这些为单词符号。表达式称为语法范畴，或语法单位。

语言的单词符号是由词法规则所确定的。词法规则规定了字母表中哪样的字符串是一个单词符号。

词法规则是指单词符号的形成规则。包括各类型的常数，标识符，基本字，算符等。语法规则规定了如何从单词符号形成更大的结构 (即语法单位)，换言之，语法规则是语法单位的形成规则。一般程序语言的语法单位由：表达式，语句，分程序，函数等。

2.1.2 语义

语义问题：对于一个语言来说，不仅要给出它的词法，语法规则，而且要定义它的单词符号和语法单位的意义。

所谓一个语言的语义是指这样一组规则，使用它可以定义一个程序的意义。这些规则称为语义规则。现在还没有一种公认的形式系统，借助于它可以自动地构造出使用的编译程序，书上介绍的是基于属性文法的语法制导翻译方法。

一个程序的层次结构大体上如下：

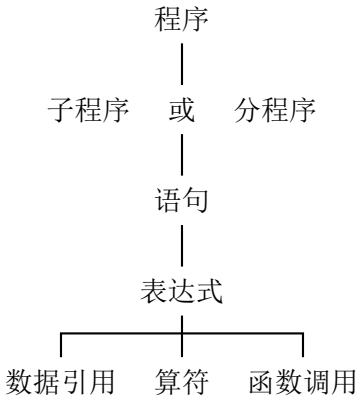

图 2.1 程序的层次结构

2.2 高级语言的一般特性

2.2.1 高级语言的分类

从语言范型分类，当今的大多数程序设计语言可分为四类。

- 强制式语言

强制式语言也称过程式语言。其特点是命令驱动，面向语句。一个强制式语言程序由一系列的语句组成，每个语句的执行引起若干存储单元中的值得改变。

```
语句1;
语句2;
......
语句n;
```

这类语言有：C，Pascal，FORTRAN，Ada

- 应用式语言

应用式语言更注重程序所表示的功能，而不是一个语句接着一个语句地执行。程序的开发过程从前面已有的函数出发构造出更复杂的函数。

```
函数n(...函数2(函数1(数据))...)
```

这类语言有：LISP，ML

- 基于规则的语言

基于规则的语言程序的执行过程式：检查一定的条件，当它满足值，则执行适当的动作。也称逻辑程序设计语言。

```
条件1 -> 动作1
条件2 -> 动作2
......
条件n -> 动作n
```

- 面向对象语言

面向对象语言如今已成为最流行、最重要的语言。它主要的特征是支持封装性、继承性和多态性等。把复杂的数据和用于这些数据的操作封装在一起，构成对象；对简单对象进行扩充继承简单对象的特性，从而设计出复杂的对象。通过对象的构造可以使面向对象程序获得强制式语言的有效性，通过作用于规定数据的函数的构造可以获得应用式语言的灵活性和可靠性。

这类语言有：JAVA，C++

2.2.2 程序结构

一个高级语言程序通常由若干子程序段构造，许多语言还引入了类，程序包等更高级的结构。下面以说明 JAVA [2]语言程序结构。

JAVA 是一种面向对象的高级语言，它很重要的方面是类及继承的概念，同时支持多态性和动态绑定特征。

相信各位都能了解 JAVA，这里不写了。

2.2.3 数据类型与操作

一个数据类型通常包括以下三种要素。

- 用于区别这种类型的数据对象的属性。
- 这种类型的数据对象可以具有的值。
- 可以作用于这种类型的数据对象的操作。

初等数据类型

一个程序语言必须提供一定的初等类型数据成分，并定义对于这些数据成分的运算。常见的初等数据类型有：

- **数值数据**：如整数，实数，复数以及这些类型的双长 (或多倍长) 精度数，对他们可施行算数运算 (+,-,*,/等)。
- **逻辑数据**：多数语言有逻辑型 (布尔型) 数据，对它们可施行逻辑运算 (and,or,not 等)。
- **字符数据**：字符型或字符串型数据。
- **指针类型**：指针的值指向另一些数据。

标识符：由字母或数字组成的以字母为开头的一个字符串。

计算机的名字仅代表一个抽象的存储单位，还必须指出它的属性。名字还包含类型和作用域等，相信读者都明白，这里不写了。

这小节内容多为数据结构的内容以及高级语言的基础知识，不写了。

2.3 程序语言的语法描述

在本节开始之前，首先介绍几个概念。

设 Σ 是一个有穷字母表，它的每个元素称为一个符号。Σ 上的一个符号串是指由 Σ 中的符号所构成的一个有穷序列。不包含任何符号的序列被称为空字，记作 ϵ。用 Σ^* 表示 Σ 上所

[2]FORTRAN 等语言例子见书 P16，本人没学过故不写

有符号串的全体。

Σ^* 的子集 U 和 V 的 (连接) 积定义为:

$$UV = \{\alpha\beta | \alpha \in U \& \beta \in V\}$$

即集合 UV 中的符号串是由 U 和 V 的符号串连接而成的。V 自身的 n 次 (连接) 积记为：

$$V^n = \underbrace{VV...V}_{n}$$

规定 $V^0 = \{\epsilon\}$。令

$$V^* = V^0 \cup V^1 \cup V^2 \cup V^3 \cdots$$

称 V^* 是 V 的闭包[3]。记 $V^+ = VV^*$，称 V^+ 是 V 的正则闭包。

闭包 V^* 中的每个符号串都是由 V 中的符号串经有限次连接而成的。

2.3.1 上下文无关法

文法是描述语言的语法结构的形式规则 (即语法规则)。所谓上下文无关法是指这样一种文法：它所定义的语法范畴 (语法单位) 是完全独立于这种范畴可能出现的环境的。也即遇到某个语法单位时，完全不考虑它的上下文环境，而直接进行处理 (例如运算)。下文所指的文法无特别说明都是上下文无关文法。

例如我们将下面自然语言进行文法分析并绘制分析树。

He gave me a book.

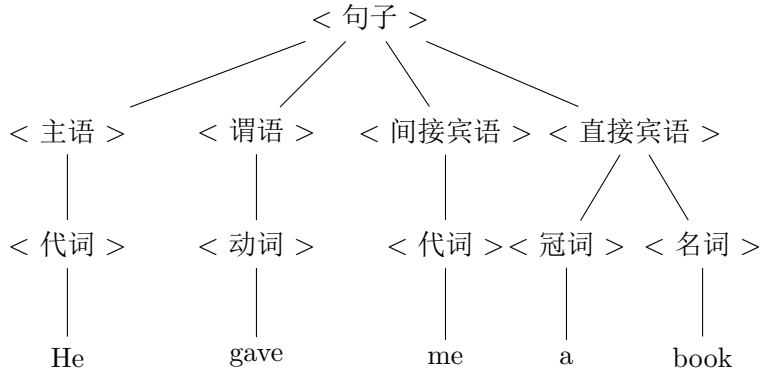

图 2.2 语法树：He gave me a book.

一个上下文无关法包括四个组成部分：一组终结符号，一组非终结符号，一个开始符号，一组产生式。在上例中，它们分别是：

- 终结符号：组成语言的基本符号

He,gave,me,a,book

[3]闭包：包含指定集合的满足在某个运算下闭合的最小集合。闭合：在一个集合上执行某种运算，得到的结果还是这个集合的元素。

- 非终结符号：代表语法范畴，语法概念

< 句子 >,< 主语 >,< 代词 > 等

- 开始符号：代表所定义语言中我们最感兴趣的语法范畴

< 句子 >

- 产生式：定义语法范畴的一种书写规范

< 直接宾语 > → < 冠词 > < 名词 >