计算机科学导论

第3章
程序设计语言

杨老师邮箱: fdteachers@163.com

第3章 程序设计语言

学习目标

- 了解计算机程序设计语言、编译原理的基本知识。
- 掌握程序的概念、高级语言程序设计的基本内容。

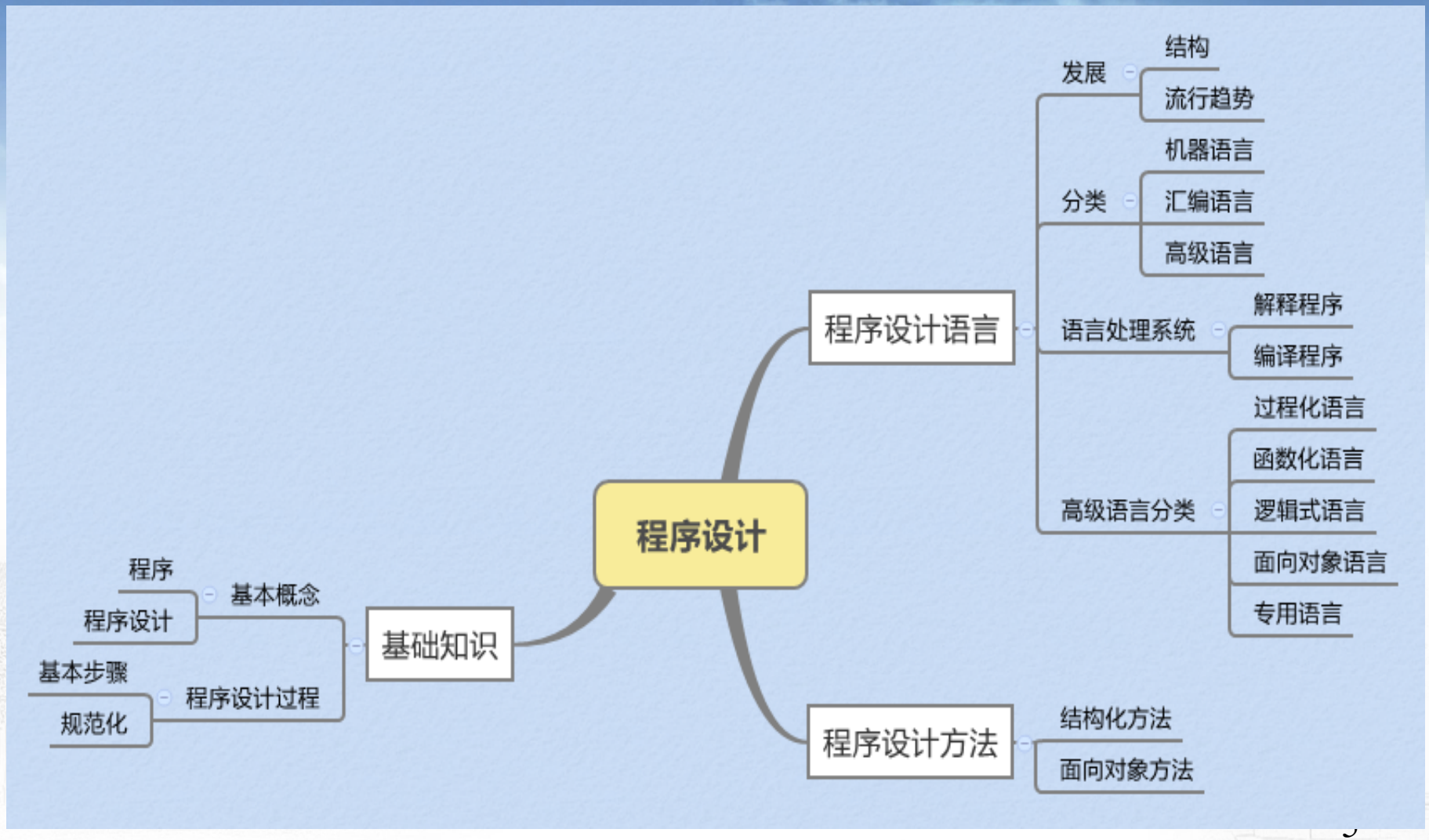
3.1 程序设计语言概述

3.1.1 程序

程序就是能够实现特定功能的一组指令序列的集合。其中，指令可以是机器指令、汇编语言指令，也可以是高级语言的语句命令，甚至还可以是用自然语言描述的运算、操作命令等。

3.1.2 计算机程序设计语言

程序设计语言使得人们能够与计算机进行交流，其种类非常繁多，总来说可以分为低级语言和高级语言两大类。


程序设计
基础知识
基本概念
程序
程序设计
程序设计过程
基本步骤
规范化
程序设计语言
发展
结构
流行趋势
分类
机器语言
汇编语言
高级语言
语言处理系统
解释程序
编译程序
高级语言分类
过程化语言
函数化语言
逻辑式语言
面向对象语言
专用语言
程序设计方法
结构化方法
面向对象方法

3.1.2 计算机程序设计语言

低级语言

低级语言包括两种类型：机器语言和汇编语言。

(1) 机器语言

- 机器语言面向机器，可以由CPU直接识别和执行。
- 不同的机器能够识别的机器语言是不相同的。
- 机器语言指令都是用一串0、1构成的二进制位串来表示的。
- 指令系统是机器提供的机器指令的集合
- 用二进制编码表示的指令，称为机器指令，或称为机器码。
- 用机器指令编写的程序称为机器语言程序，或称为目标程序，这是计算机能够直接执行的程序。
- 机器语言难以阅读和理解，编写和修改都比较困难，而且通用性较差。

3.1.2 计算机程序设计语言

(2) 汇编语言

- 汇编语言也称符号语言。
- 指令助记符是指令英文名称的缩写，容易记忆。
- 所谓汇编语言，就是采用字母、数字和符号来代替由一个个0和1构成的指令操作码、寄存器、数据和存储地址等，并在程序中用它们代替二进制编码数，这样编写出来的程序就称为符号语言程序或汇编语言程序。
- 大多数情况下，一条汇编指令直接对应一条机器指令，少数对应几条机器指令。
- 汇编语言具有一个本质上与机器语言一一对应的指令系统。汇编语言的实质和机器语言是相同的。

3.1.2 计算机程序设计语言

低级语言的特点

机器语言和汇编语言都是低级语言。它们具有许多相同的特征。

- 都与特定的计算机硬件系统紧密相关，来自于特定系统的指令系统，可移植性差。
- 对程序员专业知识要求高，要求对计算机硬件的结构和工作原理非常熟悉。
- 每条指令的功能比较单一，程序员编写源程序时指令非常繁琐。
- 由于直接针对特定硬件编程，所以最终的可执行代码非常精炼，并且执行效率高。
- 两者主要的区别在于：机器语言编写的程序无需翻译或编译，CPU能够直接识别和执行。而汇编语言源程序必须经过汇编才能得到目标程序。

3.1.2 计算机程序设计语言

汇编与汇编程序

- 计算机CPU只能“识别”和“执行”机器语言，虽然汇编语言比机器语言更容易使用和阅读，但CPU不能“识别”和“执行”，需要汇编语言程序“翻译”成计算机能够识别的机器语言程序，该程序称为汇编程序，又称汇编语言翻译程序或汇编器，是一种把用汇编语言编写的汇编语言源程序翻译成机器语言程序的系统程序。

3.1.2 计算机程序设计语言

高级语言

- (1) 高级语言的产生
- 一个问题：如何解决程序的可移植性，即：程序员编写的源程序如何可以从一台计算机很容易地转到另一台计算机上工作。为了解决这些问题，人们引入了高级语言来编写程序。
- 所谓高级语言是一种由表达各种意义的“词”和“公式”，按照一定的“语法规则”来编写程序的语言，又称为程序设计语言或算法语言。
- 高级语言之所以“高级”，就是因为它使程序员可以完全不用与计算机的硬件打交道，可以不必了解机器的指令系统。

3.1.2 计算机程序设计语言

(2) 高级语言的常见类型

- BASIC语言
- FORTRAN语言
- COBOL语言
- PASCAL语言
- C语言
- C++和C＃语言
- 其他高级语言
- 基于视窗类操作系统的，如Visual Basic、Visual C++、Delphi、Power Builder、Java等

3.1.2 计算机程序设计语言

高级语言的优点:

- 语句的功能强，程序员编写的源程序比较短，容易学习，使用方便，可移植性较好，便于推广和交流。
- 高级语言的缺点:
- 编译程序比汇编程序复杂，而且编译出来的目标程序往往效率不高，目标程序的长度比有经验的程序员所编写的同样功能的汇编语言程序要长一半以上，运行时间也要长一些。
- 因此，在很多对时间要求比较高的系统，如某些实时控制系统或者大型计算机控制系统中，低级语言，主要是汇编语言，仍然得到了一定的应用。

3.1.3 高级语言程序设计的基本内容

- 用高级语言编写的源程序能提高程序员的开发效率，高级语言程序设计依赖于各自特定的语句和语法。
- 在高级语言中，语句是构成源程序的基本单位。

程序设计语言

程序设计基本过程

- 对复杂程度较高的问题，想直接编写程序是不现实的，必须从分析问题描述入手，经过对解题算法的分析、设计直至程序的编写、调试和运行等一系列过程，最终得到期够解决问题的计算机应用程序。

3.1.3 高级语言程序设计的基本内容

高级语言的共同特性

- 1．高级语言的基本符号

  高级语言的语法成分都是由基本符号组成的，基本符号可以分为单字符和多字符两种。单字符基本符号由单个字符组成，在高级语言中通常包括下列几种单字符基本符号。

- 字母：

  大写英文字母A～Z，小写英文字母a～z，共52个符号。

- 数字：

  0～9，共10个数字符号。

3.1.3 高级语言程序设计的基本内容

- 特殊字符:

+ (加)、- (减)、* (乘)、/ (除)、^(乘方)、= (等号)、((左括号)、) (右括号)、 > (大于)、 < (小于)、 ，(逗号)、 (空格)等。

多字符是由两个或两个以上的字符组成，如GOTO(转移)、 < = (小于或等于)、AND(与)等。

3.1.3 高级语言程序设计的基本内容

高级语言的共同特性

- 2．高级语言的基本元素

基本元素由基本符号组成，可分为数、逻辑值、名字、标号和字符串等5大类：

- 数

它由0～9共10个基本数字和其他一些符号(如小数点“.”、正负号“+、-”及指数符号“E”等所构成。

- 逻辑值

由真(True)和假(False)两个值构成。

3.1.3 高级语言程序设计的基本内容

- 名字

由字符组成，一般约定名字的开头是字母或者下划线，其后可为字母或数字，如XYZ、A123、_C等。名字可用来定义常量、变量、函数、过程或子程序的，也被用来定义成某些东西，故也称为标识符。在高级语言中，一般还规定了组成名字的字符的长度，即字符个数。

- 标号

是在高级语言中的程序语句前所加的一个名字，主要用来指示程序可能的转移方向。

- 字符串

由一串字符所组成。在不同的高级语言中，字符串中的多个字符放在一对单引号或双引号中。

3.1.3 高级语言程序设计的基本内容

高级语言的共同特性

- 3．基本的数据类型

任何一个计算机程序都不可能没有数据，数据是程序操作的对象。通常，一种高级语言都会定义一些基本的数据类型，通常包括整数类型、实数类型和字符类型等。

高级语言中，在使用变量前，必须为每个变量分配所需大小的内存单元空间。因此，几乎任何一种高级语言都要求变量必须先定义后使用。

3.1.3 高级语言程序设计的基本内容

高级语言的共同特性

4．结构数据类型

结构数据类型是在基本数据类型的基础上构造出来的数据类型，数组和结构体是大多数高级语言都支持的两种最基本的结构数据类型：

- 数组类型

数组是若干个相同类型数据的集合。

- 用户自定义的结构体类型

结构体是隶属于同一个事物的多个不同类型数据的集合，用来表示具有若干个属性的一个事物

除了以上两种最基本的结构数据类型外，许多高级语言还有比如枚举、集合，以及更复杂的队列、堆栈等多种数据类型。结构数据类型在使用时也必须定义相应类型的“变量”名字。

3.1.3 高级语言程序设计的基本内容

高级语言的共同特性

5．运算符与表达式

高级语言的表达式由基本符号、基本元素和各种数据通过运算符连接而成，运算符大致包括以下几类：

- 逻辑运算：与、或、非、异或等。
- 算术运算：加、减、乘、除、取模等。
- 数据比较：大于、小于、等于、不等于等。
- 数据传送：输入、输出、赋值等。
- 通过各种运算符连接而得到的表达式有以下几种类型:
- 算术表达式：表达式的运算结果是数值，非常近似于日常的数学计算公式。
- 关系运算表达式：表达式的运算结果是逻辑值。
- 字符串表达式：表达式的运算结果是字符串。

3.1.3 高级语言程序设计的基本内容

高级语言的共同特性

6．语句

- 语句是构成高级语言源程序的基本单位，是由基本元素、运算符、表达式等组成。任何一种高级语言往往都支持赋值、条件判断、循环、输入输出等语句。程序员利用这些语句的结合，能够很方便地编制出功能强大的程序。

3.1.3 高级语言程序设计的基本内容

高级语言的共同特性

7．库函数和用户自定义的函数

- 为了支持用户编写出功能强大的源程序，几乎所有的高级语言都为用户提供了丰富的库函数，这些库函数能够实现某些特定的功能，比如计算一个比较复杂的数学函数。
- 在源程序中，用户也可以自己定义自己的函数 (子程序或过程)，以便以后可以反复调用这些代码集合。

3.1.3 高级语言程序设计的基本内容

高级语言的共同特性

8．注释

- 任何一种程序设计语言都强调注释的重要性。源程序所包含的代码往往比较冗长，添加必要的注释不仅有助于阅读程序，更重要的是，在需要对程序功能进行扩充时，注释可以极大地帮助程序员对原始程序的理解。
- 经常会出现这样一种情况，程序员自己编写的程序，经过一段时间后，可能就是半年或者几个月以后，程序员自己也读不懂自己的程序了。况且，程序不仅要自己看得懂，更重要的是也要让别人能够看懂。

3.1.3 高级语言程序设计的基本内容

高级语言的共同特性

9．程序设计风格

- 程序不仅仅要求能够在机器上执行并给出正确结果，而且还要求便于调试和维护。在程序设计过程中，程序员应该尽量保持程序结构的合理和清晰，养成良好的编程习惯。好的程序设计风格有助于提高程序的正确性、可读性、可维护性和可用性。

3.1.3 高级语言程序设计的基本内容

高级语言的共同特性

10．高级语言程序的运行

使用高级语言编写程序的一般过程可以归纳为以下几个步骤：

- 使用文本编辑工具，逐条编写源程序的语句。保存源程序的文件时，文件的后缀名与所用的高级语言有关。
- 编译源程序文件，生成目标文件，文件后缀名通常为obj。
- 链接目标文件，生成可执行文件，文件后缀名通常为exe。
- 在计算机上运行可执行程序，并进行调试和维护。

3.1.4 高级语言的数据表示手段

1．常量

- 常量也称常数，是一种恒定的、不随时间改变的数值或数据项。

2．变量

- 变量是指在程序的运行过程中可以发生改变的量，是程序中数据的临时存放场所。

3．数据类型

- 用来约束数据的解释。

4．表达式

- 表达式是操作符、操作数和标点符号组成的序列，其目的是用来说明一个计算过程。

3.2 声明和类型

1．声明

- 声明用于说明每个标识符的含义，而不必为每个标识符预留存储空间。预留存储空间的声明称为定义。

2．类型

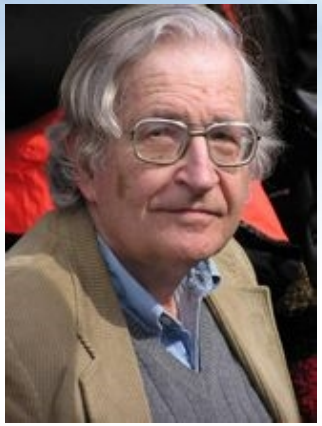
- 类型检查是利用一组逻辑规则来推理一个程序在运行时刻的行为。

3.3 类型系统

- 类型系统用于定义如何将程序语言中的数值和表达式归类为许多不同的类型、如 何操作这些类型、这些类型如何互相作用。
- 类型可以确认一个值或者一组值，具有特定的意义和目的(虽然某些类型，如抽象类型和函数类型，在运行程序中，可能不表示为值)。

3.4 编译原理

3.4.1 编译程序



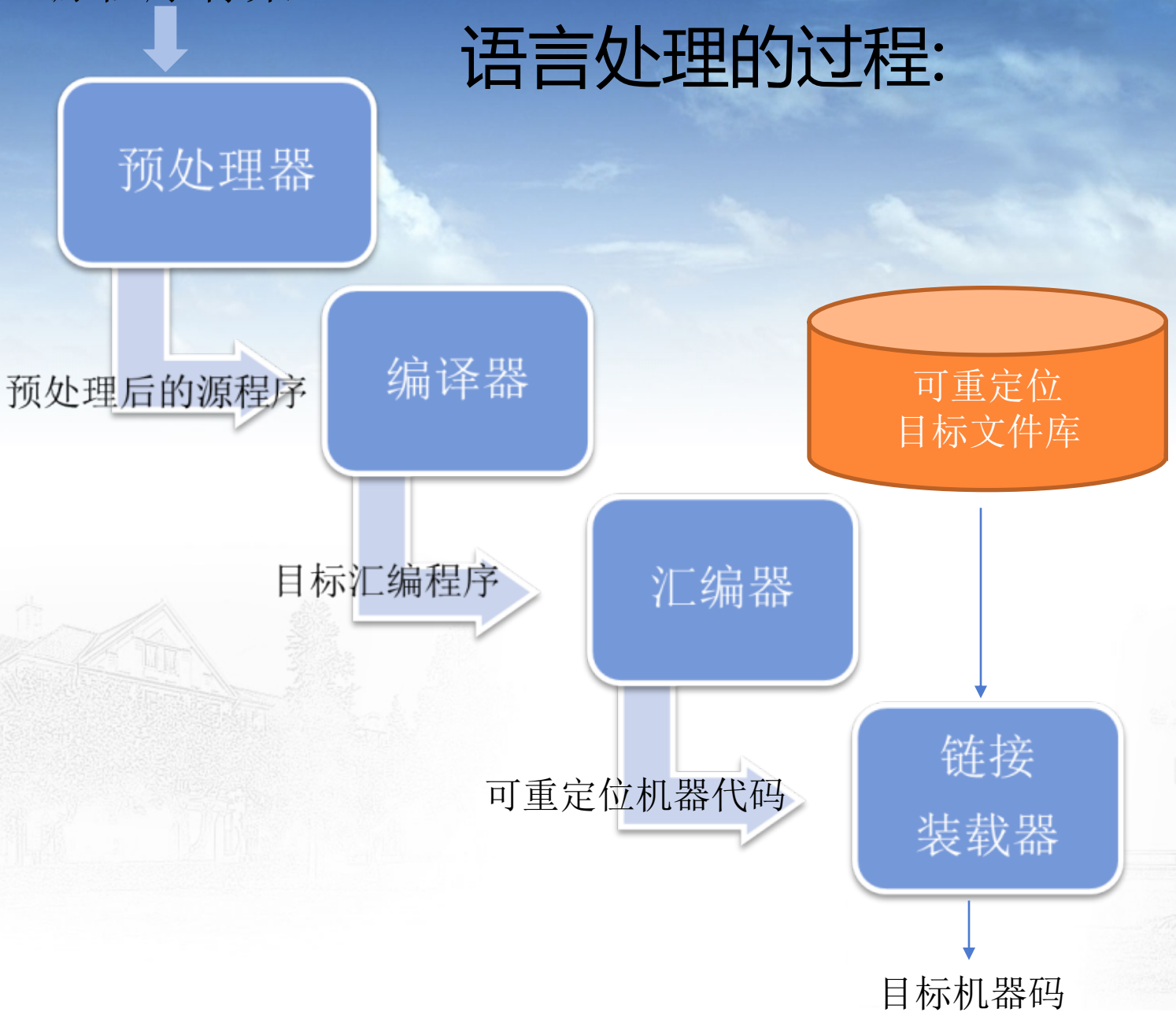
20世纪50年代，IBM的约翰·巴克斯带领一个研究小组对FORTRAN语言及其编译器进行开发。受限于人们对编译理论知识的缺乏，开发工作变得既复杂又艰苦。与此同时，诺姆·乔姆斯基(Noam Chomsky，1928~)开始对自然语言结构的研究。他的发现最终使得编译器的结构异常简单，甚至还有一些自动化形式。

3.4.1 编译程序

高级语言编写的源程序需要“翻译”成计算机能够识别的机器语言，机器才能执行，这种“翻译”程序被称为语言处理程序。

源程序骨架

语言处理的过程:



3.4.1 编译程序

- 一个翻译程序能够把诸如FORTRAN、Pascal、C、Ada、Smalltalk或Java这样的“高级语言”编写的源程序转换成逻辑上等价的诸如汇编语言之类的“低级语言”的源程序，这样的翻译程序则称之为编译程序。
- 编译程序的功能如图所示:

3.4.1 编译程序

编译程序

将高级语言编写的源程序翻译为机器语言程序的方式有解释和编译:

(1)解释

- 解释程序在处理源程序时，执行方式类似于日常生活中的“同声翻译”。
- 解释一句、执行一句，立即产生运行结果。解释程序不产生目标代码，不能脱离其语言环境独立执行。
- 解释程序对源程序的解释执行比编译程序产生的目标代码程序的执行速度要慢。

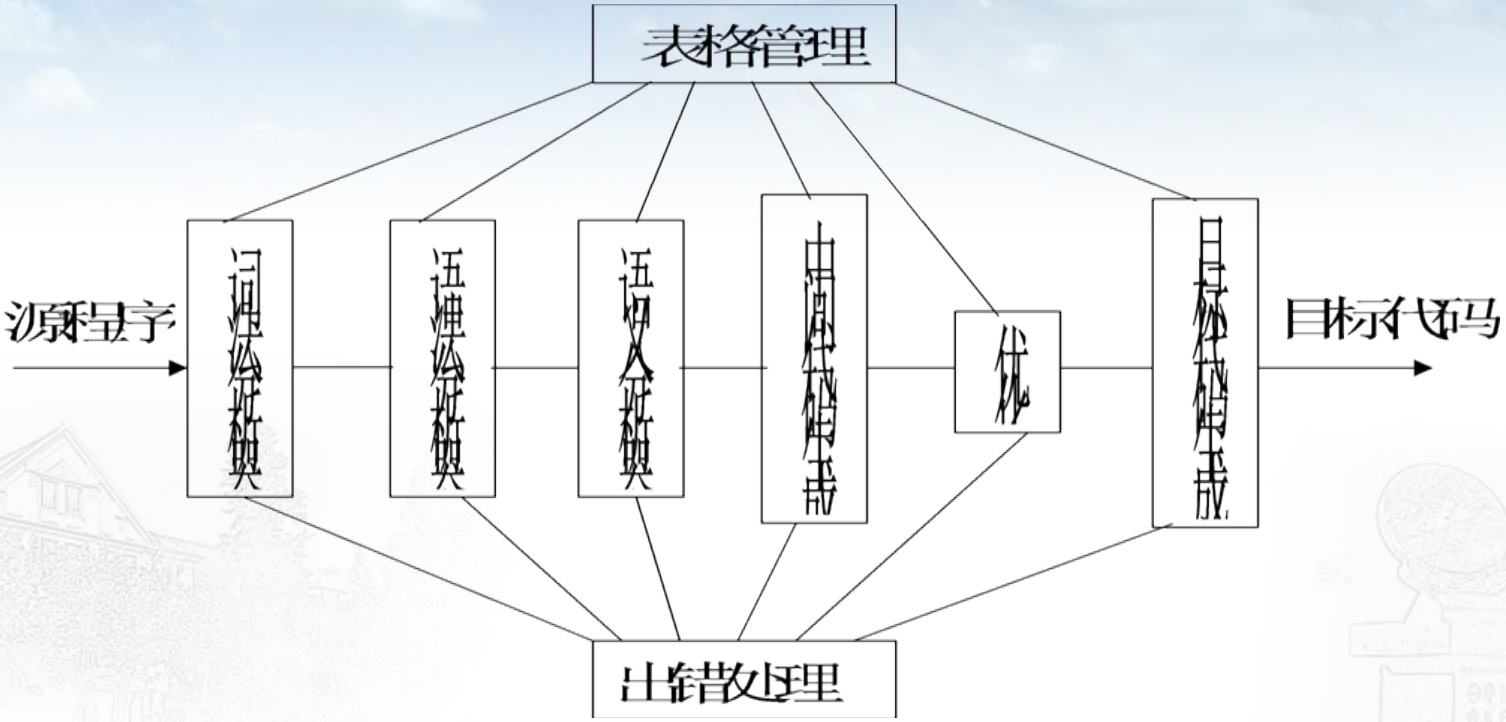
3.4.1 编译程序

(2)编译

- 编译程序是把高级语言程序(源程序)作为一个整体来处理，首先将程序源代码“翻译”成目标代码(机器语言)，编译后与系统提供的代码库链接，形成一个完整的可执行的机器语言程序(目标程序代码)。
- 目标程序可以脱离其语言环境独立执行，使用比较方便、效率较高。相应地，由于每次执行之前必须通过编译得到可执行程序，所以，可执行程序一旦需要修改，必须先修改源代码，再重新编译生成新的目标文件(*.obj)才能执行。

3.4.1 编译程序

编译程序的工作过程



3.4.2 词法分析

- 其任务是从左到右一个字符、一个字符地对源程序进行扫描，读入源程序，对构成源程序的字符流进行扫描和分解，通过词法分析从而识别出一个个单词(也称单词符号或符号)。

例1 对表达式：position := initial + rate * 100 ；进行词法分析。

对其进行词法分析后得到以下结果：

3.4.2 词法分析

单词类型	单词值
标识符1(id1)	position
算符(赋值)	:=
标识符2(id2)	initial
算符(加)	+
标识符3(id3)	rate
算符(乘)	*
整数	100
分号	;

3.4.3 语法分析

语法分析是编译过程的第二个阶段，任务是在词法分析的基础上将单词序列分解成各类语法短语，如“程序”、“语句”、“表达式”等等。一般这种语法短语也称为语法单位。

3.4.3 语法分析

例2 按照例1的结果，对表达式：position := initial + rate * 100；进行语法分析。

语法规则：

<赋值语句>::=<标识符> “:=” <表达式>

<表达式>::=<表达式> “+” <表达式>

<表达式>::=<表达式> “*” <表达式>

<表达式>::= “(” <表达式> “)”

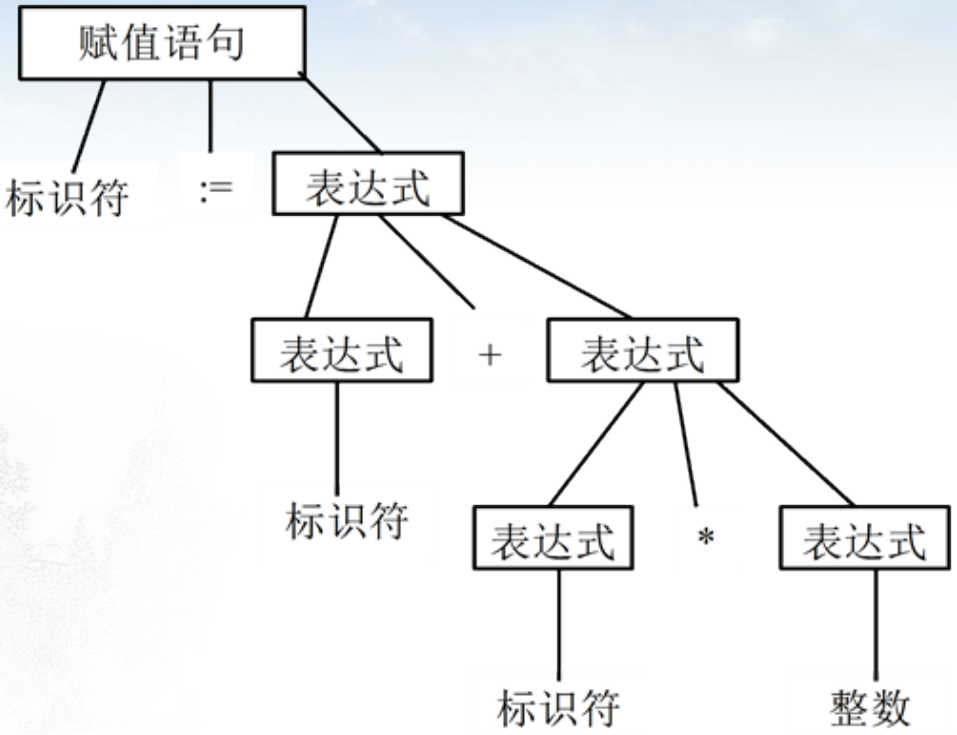
<表达式>::=<标识符>

<表达式>::=<整数>

<表达式>::=<实数>

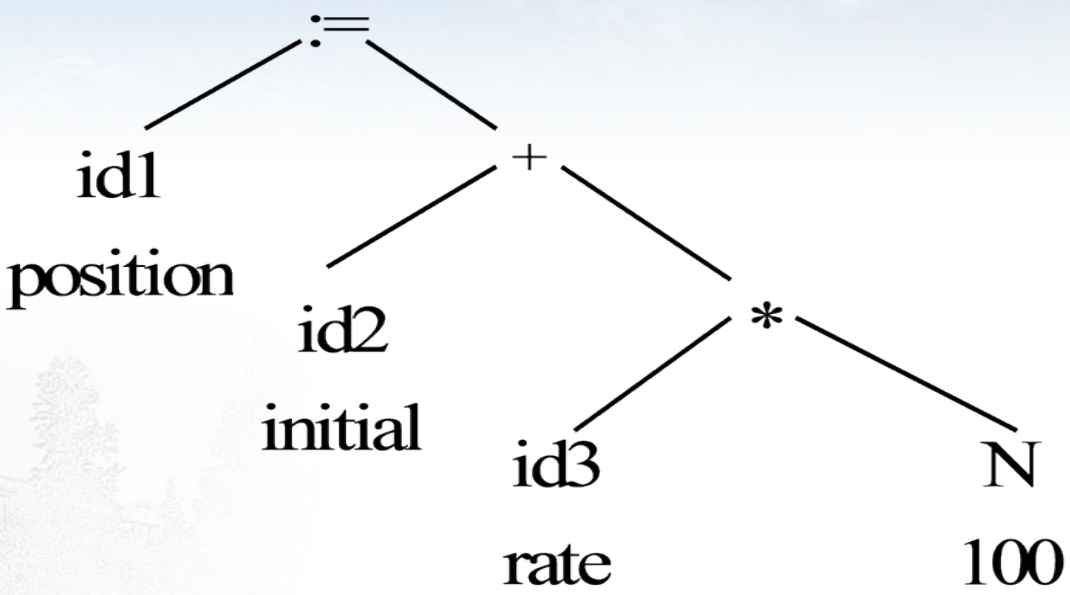
3.4.3 语法分析

依据源程序的语法规则把源程序的单词序列组成语法短语(表示成语法树)，见图:



3.4.3 语法分析

把id1:=id2+id3*N转换成语法树见图:



3.4.4 语义处理

在词法分析程序和语法分析程序对源程序的语法结构进行分析之后，一般要由语法分析程序调用相应的语义子程序进行语义处理。

编译过程中的语义处理实现两个功能：

审查每个语法结构的静态语义，即验证语法结构合法的程序是否真正有意义，有时把这个工作称为静态语义分析或静态审查。

如果静态语义正确，则语义处理要执行真正的翻译，要么生成程序的一种中间表示形式(中间代码)，要么生成实际的目标代码。

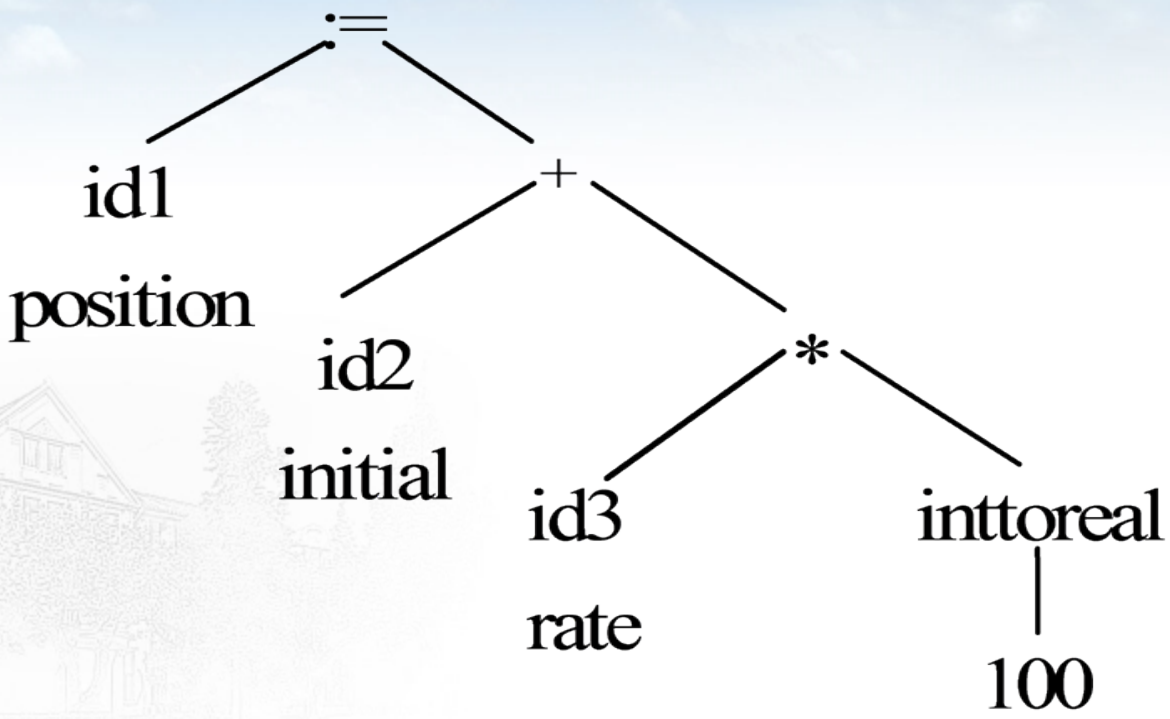
3.4.4 语义处理

例3 按照例1和2的结果，对表达式：position := initial + rate * 100；进行语义处理。

```
Program p();
Var  rate:real;
Var  initial :real;
Var  position :real ;
…
position := initial + rate * 100 ;
```

3.4.4 语义处理

下图是得到的语义分析树：



3.4.4 语义处理

- 语义分析阶段审查源程序有无语义错误，为代码生成阶段收集类型信息。比如语义分析的一个工作是进行类型审查，审查每个算符是否具有语言规范允许的运算对象，当不符合语言规范时，编译程序亦报告错误。如有的编译程序要对实数用作数组下标的情况报告错误。又比如某些语言规定运算对象可被强制，那么当二目运算施于整型和实型时，编译程序应将整型转换成实型而不能认为是源程序的错误。

3.4.5 中间代码生成

- 所谓“中间代码”是一种结构简单、含义明确的记号系统，这种记号系统可以设计为多种多样的形式，重要的设计原则为两点：一是容易生成；二是容易将它翻译成目标代码。
- 常用的中间代码形式有逆波兰式、三元式和四元式。

3.4.5 中间代码生成

- 采用三地址指令表示t2 := id3 * t1，会得到以下的结果：

(　*　id3　t1　t2)

- 表达式id1:= id2 + id3 * 100按照三地址指令生成中间代码：

(1) (inttoreal, 100 - t1)

(2) (* id3 t1 t2)

(3) (+ id2 t2 t3)

(4) (:= t3 - id1)

3.4.6 中间代码优化

- 中间代码优化的任务是对中间代码进行变换或进行改造，目的是使生成的目标代码更为高效，即节省时间和空间。
- 常用的优化技术有删除多余运算、强度削弱、变换循环控制条件、合并已知量与复写传播、删除无用赋值等。
- 将3.5.5小节的中间代码进行优化，结果为：

(1) (* id3 100.0 t1)
(2) (+ id2 t1 id1)

3.4.7 目标代码生成

- 目标代码生成阶段的任务是把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。这是编译的最后阶段，它的工作与硬件系统结构和指令含义有关。
- 将3.5.6小节优化后的中间代码生成目标代码：

```
movf    id3,R2
mulf    #100.0,R2
movf    id2,R1
addf    R2,R1
movf    R1,id1
```

3.4.7 目标代码生成

- 由于一个高级程序设计语言的目标代码需反复使用，因而代码生成器的设计要着重考虑目标代码的质量问题。衡量目标代码的质量主要从占用空间和执行时间两个方面综合考虑。到底产生什么样的目标代码取决于具体的机器结构、指令格式、字长及寄存器的个数和种类，并与指令的语义和所用操作系统、存储管理等都密切相关。
- 编译过程的阶段划分是一种典型处理模式，事实上并非所有的编译程序都可分成这样几个阶段，有些编译程序并不要生成中间代码，有些编译程序不进行优化，有些最简单的编译程序在语法分析的同时生成目标指令代码。不过多数实用的编译程序都具有上述几个阶段。

3.4.8 编译技术的新发展

- 并行编译技术
- 交叉编译技术
- 硬件描述语言及其编译技术

3.5 程序设计语言的设计

- 程序设计语言的设计是一门技术，需要相应的理论、技术、方法和工具来支持。
- 除了好的程序设计方法和技术之外，程序设计风格也是很重要的。

本章小结

- 本章介绍了程序设计语言的相关知识，包括程序设计语言的种类，声明和类型的含义，类型系统，程序编译原理和设计。程序设计语言是计算机中的一个重要概念，特别是读者需要使用程序设计语言来实现特定功能，必须理解和掌握程序设计的核心思想。
- 通过本章的学习，读者应该了解程序设计的基本概念和基本原理，对声明和类型系统有基本的认识，初步掌握编译过程中从词法分析直到目标代码生成的完整过程。

谢谢大家！