

世界著名计算机教材精选

程序设计 语言概念

John C. Mitchell

著

冯建华 王 益

廖雨果 李国良

译



CONCEPTS IN PROGRAMMING LANGUAGES

清华大学出版社



世界著名计算机教材精选

Concepts in Programming Languages

程序设计语言概念

John C. Mitchell 著

Stanford University

冯建华 王益 廖雨果 李国良 译

清华大学出版社
北 京

Originally published by Cambridge University Press in 2000.

This reprint edition is published with the permission of the Syndicate of the Press of the University of Cambridge, Cambridge, England.

THIS EDITION IS LICENSED FOR DISTRIBUTION AND SALE IN THE PEOPLE'S REPUBLIC OF CHINA ONLY, EXCLUDING HONG KONG, MACAO SAR AND TAIWAN.

本书中文翻译版由 Cambridge University Press 授权给清华大学出版社出版发行。

仅限于中华人民共和国境内（不包括中国香港、澳门特别行政区和中国台湾地区）销售发行。

北京市版权局著作权合同登记号 图字 01-2003-8325 号

版权所有，翻印必究。举报电话：010-62782989 13501256678 13801310933

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

本书防伪标签采用特殊防伪技术，用户可通过在图案表面涂抹清水，图案消失，水干后图案复现；或将表面膜揭下，放在白纸上用彩笔涂抹，图案在白纸上再现的方法识别真伪。

图书在版编目(CIP)数据

程序设计语言概念 / (美) 米切尔 (Mitchell, J. C.) 著; 冯建华等译. —北京: 清华大学出版社, 2005. 10
(世界著名计算机教材精选)

ISBN 7-302-11107-3

I. 程… II. ①米… ②冯… III. 程序语言—高等学校—教材 IV. TP312

中国版本图书馆 CIP 数据核字 (2005) 第 054271 号

出版者: 清华大学出版社 地址: 北京清华大学学研大厦
<http://www.tup.com.cn> 邮编: 100084
社总机: 010-62770175 客户服务: 010-62776969

组稿编辑: 龙啟铭

文稿编辑: 王敏稚

印刷者: 清华大学印刷厂

装订者: 三河市金元装订厂

发行者: 新华书店总店北京发行所

开本: 185 × 260 印张: 28.25 字数: 704 千字

版次: 2005 年 10 月第 1 版 2005 年 10 月第 1 次印刷

书号: ISBN 7-302-11107-3/TP · 7346

印数: 1 ~ 3000

定价: 56.00 元

前 言

最好的程序设计语言就是编程思考中的概念上的世界。

——Alan Perlis, NATO 软件工程技术会议, 罗马, 1969

程序设计语言为程序员写出一个好的程序提供了所需的抽象机制、组织原则以及控制结构。本书所介绍的是在程序设计语言中出现的概念, 即在程序设计语言的实现过程中产生的问题, 以及语言的设计方式对程序开发产生的影响。

本书分为 4 个部分:

- 第 1 部分: 函数与基本原理
- 第 2 部分: 过程、类型、内存管理与控制
- 第 3 部分: 模块、抽象与面向对象程序设计
- 第 4 部分: 并发性与逻辑编程

第 1 部分将 Lisp 作为分析程序设计语言的示例, 对其进行了简单介绍, 内容包括编译器结构、解析、朗母达演算以及指称语义。可计算性一章还涉及了编译时程序分析和优化的限制。

第 2 部分通过过程化的 Algol 系列语言和 ML, 介绍了类型、内存管理和控制结构。

第 3 部分介绍使用抽象数据类型、模块和对象来组织程序。由于目前面向对象编程广受推崇, 于是我们对几种面向对象语言进行了对比。有专门的章节对 Simula、Smalltalk、C++ 和 Java 进行研究和比较。

第 4 部分介绍了支持并发性的语言机制和逻辑编程。

本书面向的读者是有一定编程基础的大学本科高年级学生和研究生新生。他们理解 C 或其他过程化语言, 熟悉 C++ 或者其他面向对象的程序设计语言。如果读者具备一些 Lisp、Scheme 或者 ML 的经验将会对第 1 部分和第 2 部分的理解有所帮助, 但不具备这些背景知识也同样能学好这门课程。对算法和数据结构进行简单分析的经验也对理解本书有所帮助。例如, 在比较某种构造的实现方式的时候, 如果能够区分常数时间复杂性、多项式时间复杂性和指数时间复杂性将有助于理解。

在学习了本书之后, 读者将会对过去 40 年中所使用过的各种程序设计语言有更好的理解, 对程序设计语言的设计过程中出现的问题和折衷有更深入的认识, 也会对所使用的程序设计语言的利弊有更透彻的了解。由于不同的语言体现了不同的编程概念, 把其他语言中的思想引入到自己所编写的程序中将会提高读者的编程能力。

致谢

这本书的手稿源于我从 1993 年开始开设的一门程序设计语言课程 (Stanford CS 242) 的笔记。每年都有精力充沛的助教帮助我调试课程的示例程序, 设计课程作业和准备解决方案模型。该课程的组织和内容都受益于他们的建议。特别感谢 Kathleen Fisher, 他在 1993

年和 1994 年担任助教，并于 1995 年我不在校的时候教授课程。Kanthleen 早些年帮我组织材料，并在 1995 年将我的手稿转录成在线文档。感谢 Amit Patel 主动组织布置作业和解决方案，感谢 Vitaly Shmatikov 对程序设计语言术语表做出的不懈努力。Anne Bracy、Dan Bentley 和 Stephen Freund 仔细地校对了许多章节。

剑桥大学出版社的 Lauren Cowles、Alan Harvey 和 David Tranah 给予我支持和帮助。我要特别感谢 Lauren 对草稿的所有 12 章都仔细阅读并详细做注。同时也要感谢他们邀请的校阅者，他们对本书的早期版本提出了很多宝贵的建议。Zena Ariola 从本书的初稿开始就连续几年在俄勒冈州大学教授此书，并提出了很多很好的建议；还有很多其他讲师也提供了很多建议。

最后，特别感谢 Krzysztof Apt 对“逻辑编程”一章做出的贡献。

John C. Mitchell

目 录

第 1 部分 函数与基本原理

第 1 章 导言	2
1.1 程序设计语言	2
1.2 目标	3
1.2.1 总体目标	3
1.2.2 特殊主题	3
1.3 程序设计语言的历史	4
1.4 组织: 概念和语言	5
第 2 章 可计算性	7
2.1 部分函数与可计算性	7
2.1.1 表达式、错误和非终止符	7
2.1.2 部分函数	8
2.1.3 可计算性	9
2.2 本章小结	11
习题	11
第 3 章 Lisp 语言: 函数、递归和列表	13
3.1 Lisp 语言的历史	13
3.2 好的语言设计	13
3.3 语言简述	15
3.4 Lisp 设计中的创新	18
3.4.1 语句和表达式	18
3.4.2 条件表达式	19
3.4.3 Lisp 抽象机	20
3.4.4 把程序作为数据	23
3.4.5 函数表达式	24
3.4.6 递归	25
3.4.7 高阶函数	25
3.4.8 垃圾收集	26
3.4.9 纯 Lisp 与副作用	29
3.5 本章小结	30
习题	30

第 4 章 基本原理	38
4.1 编译器和语法	38
4.1.1 一个简单编译器的结构	38
4.1.2 文法和解析树	41
4.1.3 解析和优先级	43
4.2 朗母达演算	44
4.2.1 函数和函数表达式	44
4.2.2 朗母达表达式	45
4.2.3 朗母达演算编程	49
4.2.4 归约、汇合和范式	51
4.2.5 朗母达演算的重要特征	52
4.3 指称语义	52
4.3.1 目标语言和元语言	53
4.3.2 二进制数的指称语义	54
4.3.3 While 程序的指称语义	55
4.3.4 透视和非标准语义	58
4.4 函数型语言和命令型语言	60
4.4.1 命令语句和声明语句	60
4.4.2 功能型程序和命令型程序	61
4.5 本章小结	65
习题	66

第 2 部分 过程、类型、内存管理与控制

第 5 章 Algol 与 ML 语言	74
5.1 Algol 家族的程序语言	74
5.1.1 Algol 60	74
5.1.2 Algol 68	76
5.1.3 Pascal	77
5.1.4 Modula	78
5.2 C 语言的发展	78
5.3 LCF 系统和 ML	80
5.4 ML 程序设计语言	82
5.4.1 交互会话和运行时环境	82
5.4.2 基本类型和类型构造器	85
5.4.3 模式、声明、函数表达式	89
5.4.4 ML 数据类型的声明	92
5.4.5 ML 的引用单元与赋值	94
5.4.6 ML 小结	97

5.5 本章小结	98
习题	98
第 6 章 类型系统和类型推测	105
6.1 程序设计中的类型	105
6.1.1 程序的组织和文档	105
6.1.2 类型错误	106
6.1.3 类型与优化	107
6.2 类型安全和类型检查	108
6.2.1 类型安全	108
6.2.2 编译时和运行时的类型检查	108
6.3 类型推测	110
6.3.1 第一个类型推测的示例	110
6.3.2 类型推测算法	111
6.4 多态和重载	118
6.4.1 参数多态	118
6.4.2 参数多态的实现	120
6.4.3 重载	122
6.5 类型声明和类型等价性	123
6.5.1 透明的类型声明	123
6.5.2 C 语言的声明和结构	124
6.5.3 ML 类型声明	125
6.6 本章小结	126
习题	127
第 7 章 作用域、函数和存储管理	133
7.1 块结构的语言	133
7.2 内嵌块	135
7.2.1 活动记录和局部变量	135
7.2.2 全局变量和控制链	138
7.3 函数和子程序	139
7.3.1 函数的活动记录	139
7.3.2 参数传递	141
7.3.3 全局变量（一阶情况）	144
7.3.4 末端递归（一阶情况）	146
7.4 高阶函数	148
7.4.1 一阶函数	148
7.4.2 将函数传递给函数	149
7.4.3 从嵌套作用域中返回函数	152

7.5 本章小结	154
习题	155
第 8 章 顺序语言中的控制	168
8.1 结构化控制	168
8.1.1 意大利面条式的代码	168
8.1.2 结构化控制	168
8.2 异常	169
8.2.1 异常机制的目的	169
8.2.2 ML 异常	171
8.2.3 C++ 异常	173
8.2.4 关于异常的更多内容	175
8.3 延续	179
8.3.1 表示“程序其余部分”的函数	179
8.3.2 延续传递形式和末端调用	180
8.3.3 延续的编译	183
8.4 函数和求值顺序	183
8.5 本章小结	186
习题	187
第 3 部分 模块、抽象与面向对象程序设计	
第 9 章 数据抽象和模块化	192
9.1 结构化程序设计	192
9.1.1 数据细化	193
9.1.2 模块化	194
9.2 支持抽象机制的语言	196
9.2.1 抽象	197
9.2.2 抽象数据类型	198
9.2.3 ML 抽象数据类型	198
9.2.4 表达无关性	201
9.2.5 数据类型介绍	202
9.3 模块	204
9.3.1 Modula 和 Ada	205
9.3.2 ML 模块	207
9.4 一般抽象	210
9.4.1 C++ 函数模板	210
9.4.2 标准的 ML 算符	212
9.4.3 C++ 标准模板库	215
9.5 本章小结	218

习题	220
第 10 章 面向对象语言的概念	226
10.1 面向对象设计	226
10.2 面向对象语言中的 4 个基本概念	227
10.2.1 动态查找	227
10.2.2 抽象	229
10.2.3 子类型	231
10.2.4 继承	232
10.2.5 作为对象的闭包	233
10.2.6 继承不是子类型	234
10.3 编程结构	235
10.4 设计模式	236
10.5 本章小结	239
10.6 展望: Simula、Smalltalk、C++、Java	239
习题	240
第 11 章 对象的历史: Simula 和 Smalltalk	246
11.1 Simula 面向对象机理	246
11.1.1 对象和仿真	246
11.1.2 Simula 的主要概念	247
11.2 Simula 中的对象	247
11.2.1 Simula 中面向对象的基本特点	248
11.2.2 一个点线圆的例子	248
11.2.3 示例代码和对象表示	250
11.3 Simula 中的子类 and 继承	251
11.3.1 对象类型和子类型	252
11.4 Smalltalk 的发展	254
11.5 Smalltalk 语言的特点	255
11.5.1 术语	255
11.5.2 类和对象	255
11.5.3 继承	258
11.5.4 Smalltalk 的抽象性	260
11.6 Smalltalk 的灵活性	260
11.6.1 动态查找和多态	260
11.6.2 布尔变量和块	261
11.6.3 self 和 super	262
11.6.4 系统扩充: Ingalls 测试	263
11.7 子类型与继承的重要性	264

11.7.1 对象类型作为接口	264
11.7.2 子类型	265
11.7.3 子类型和继承	265
11.8 本章小结	267
习题	268
第 12 章 C++对象与运行效率	277
12.1 设计目标和限制	277
12.1.1 与 C 的兼容性	277
12.1.2 C++的成功	278
12.2 C++概述	278
12.2.1 增加了 C 中没有的对象	279
12.2.2 面向对象的特点	282
12.2.3 好的决定和问题所在	282
12.3 类、继承和虚函数	284
12.3.1 C++类和对象	284
12.3.2 C++派生类（继承）	285
12.3.3 虚函数	287
12.3.4 为什么 C++的查找比 Smalltalk 的查找简单	288
12.4 子类型	292
12.4.1 子类型原理	292
12.4.2 公有基类	293
12.4.3 public 成员的特殊类型	294
12.4.4 抽象基类	294
12.5 多重继承	295
12.5.1 多重继承的实现	296
12.5.2 命名冲突、继承和虚拟基类	298
12.6 本章小结	301
习题	302
第 13 章 可移植性和安全性：Java 语言	319
13.1 Java 语言概述	320
13.1.1 Java 语言的目标	320
13.1.2 设计决策	320
13.2 Java 的类和继承	322
13.2.1 类和对象	322
13.2.2 包和可视性	325
13.2.3 继承	325
13.2.4 抽象类和接口	327

13.3	Java 的类型及子类型关系	328
13.3.1	类型的分类	328
13.3.2	类和接口的子类型关系	329
13.3.3	数组、协变和反协变	330
13.3.4	Java 异常类的层次关系	331
13.3.5	子类型多态和通用编程	333
13.4	Java 系统架构	336
13.4.1	Java 虚拟机	336
13.4.2	类加载器	337
13.4.3	Java 链接器、检验器及类型约束	337
13.4.4	字节码解释器和方法查询	338
13.5	安全特性	342
13.5.1	缓冲区泄漏攻击	343
13.5.2	Java 沙箱	344
13.5.3	安全和类型安全	346
13.6	本章小结	347
习题		349

第 4 部分 并发性与逻辑编程

第 14 章	并发和分布式编程	358
14.1	并发的基本概念	359
14.1.1	执行顺序和非确定性	359
14.1.2	通信、协调和原子性	361
14.1.3	互斥和封锁	361
14.1.4	信号量	364
14.1.5	管程	365
14.2	Actor 模型	366
14.3	并发 ML	369
14.3.1	线程和通道	369
14.3.2	选择式通信和保护命令	371
14.3.3	一流的同步操作：事件	373
14.4	Java 的并发性	377
14.4.1	线程、通信与同步	378
14.4.2	同步方法	380
14.4.3	虚拟机与存储模型	382
14.4.4	分布式程序设计与远程方法调用	386
14.5	本章小结	388
习题		390

第 15 章 逻辑编程范例和 Prolog	396
15.1 逻辑编程的历史	396
15.2 逻辑编程范例的简要概述	397
15.2.1 说明性编程	397
15.2.2 交互编程	397
15.3 作为原子动作统一解决的等式	398
15.3.1 项	398
15.3.2 置换	399
15.3.3 最通用的合一置换	399
15.3.4 合一算法	400
15.4 子句作为过程声明的一部分	402
15.4.1 简单子句	402
15.4.2 计算过程	402
15.4.3 子句	404
15.5 Prolog 编程	405
15.5.1 单个程序的多重使用	405
15.5.2 逻辑变量	406
15.6 Prolog 中的数学	409
15.6.1 数学运算符	410
15.6.2 数学比较关系	410
15.6.3 对算术表达式的赋值	412
15.7 控制、双性语法和元变量	414
15.7.1 剪切	414
15.7.2 双性语法和元变量	415
15.7.3 控制设备	416
15.7.4 失败的否定	418
15.7.5 高阶编程和 Prolog 中的元编程	419
15.8 Prolog 的评价	421
15.9 书目评价	423
15.10 本章小结	423
附录 A 程序实例补充	425
A.1 程序和面向对象机制	425
A.1.1 类型的程序: 典型案例版本	426
A.1.2 shape 程序: 面向对象版本	430
附录 B 术语表	433

第 1 部分

函数与基本原理

第1章 导 言

媒介就是消息。

——Marshall McLuhan

1.1 程序设计语言

在计算机编程中，程序设计语言是作为表达的媒介。一个理想的设计语言能让程序员轻松地编写出简洁、清晰的程序。由于程序在其生存期中必须易于理解、修改和维护，一个好的设计语言还要让其他人易于阅读程序并理解它们是怎样工作的。软件的设计和构造是一项复杂的工作。许多软件系统都由相互作用的多个部分组成。这些部分，或者说是软件的构件，它们之间的交互可能非常复杂。为了控制复杂性，必须仔细地定义这些构件之间的接口和消息传送。一个适于大规模编程的、好的设计语言将会帮助程序员高效地管理软件组件之间的交互。当我们评价设计语言的时候，必须考虑到设计、实现、测试和维护软件的工作，看看每种语言对软件生命周期中的各个部分是如何提供支持的。

在设计语言的设计过程中有很多时候需要折衷利弊，这是项困难的工作。有些语言的特色使得我们能够很快地编写出程序，但是在设计测试工具和方法的时候却要困难得多。而有些语言的构造方式使得编译器能够容易地对程序进行优化，但又会使得编程过程烦琐。由于不同的计算环境和应用需要不同的程序特性，于是不同的程序语言设计者选择了不同的折衷办法。实际上所有成功的程序语言最初都是为某一特殊用途而设计的。这并不是说每种语言都只适于一种目的。然而，把注意力集中在某一应用上能够帮助语言设计者作出一致的、有目的的决定。单一应用还有利于解决语言设计的最难对付的问题：遗漏好的想法。

即使你不使用本书中的许多程序设计语言，你仍能运用这些语言所体现的概念框架。在 20 世纪 70 年代中期，当我还是学生的时候，所有“严谨”的程序员（至少在我所在的大学）都使用 Fortran 语言。Fortran 语言不允许递归，而大家普遍认为递归的效率太低，对“实际的编程”不实用。然而，我选修的一门课程的讲师认为递归是一个非常重要的思想，并向我们解释在 Fortran 语言中可以怎样利用递归技术管理数组数据。我很高兴我选修了那门课程，从而认识到递归并不是一个不实际的思想。在 20 世纪 80 年代，许多人认为面向对象编程效率不高、太笨拙，不适合实际的编程。但是，在 20 世纪 90 年代，学生会为自己在 80 年代学习了面向对象编程这一“未来派”语言而感到高兴，因为面向对象编程已经被广泛接受和使用。

尽管这不是一本讲述程序设计语言历史的书，但整本书都对历史有一定关注。讨论历史上的各种语言的目的之一是，通过事实来理解程序设计语言中的各种折衷方案选择。例

如，当机器速度很慢并且内存资源很宝贵时的程序跟现在的程序是不同的。所以 20 世纪 60 年代的程序语言设计者所关心的问题与现在的程序语言设计者所关心的问题不同。通过想像过去年代中的实际情况，我们就能更深地理解为什么语言设计者要做出那些决定。这种思考语言和计算的方法可能会在将来对我们有所帮助。例如，最近掌上型计算设备和嵌入式处理器的兴起使得人们对有限内存和有限计算能力的设备编程又重新恢复了兴趣。

在本书中当我们讨论某种特定语言的时候，我们一般都指的是这一语言的原始形式，或者是其在历史上很重要的形式。比如，“Fortran”指的是 20 世纪 60 年代和 70 年代早期的 Fortran。这些早期语言被称为 Fortran I、Fortran II、Fortran III，等等。在最近几年，Fortran 逐步演化，包括了更多现代特征，Fortran 与其他语言的区别在某种程度上已经不太明显。同样，Lisp 一般都是指 60 年代的 Lisp，Smalltalk 一般指 70 年代后期和 80 年代的 Smalltalk，等等。

1.2 目标

在这本书中，我们所关心的是现代程序设计语言中出现的基本概念，它们之间的相互关系，以及程序设计语言与程序开发方式的关系。一个重复出现的话题是语言表现性与实现的简单性之间的折衷。对于我们考虑的每一个程序设计语言特征，都将研究在编程中能够使用它的各种方式，以及可以用来高效编译和执行它的各种实现技术。

1.2.1 总体目标

本书有以下总体目标。

- 理解程序设计语言的设计空间。这包括过去的程序设计语言的概念和构造，以及将来可能会被广泛使用的概念和构造。我们还会试图理解语言特征之间的一些主要的冲突和折中，包括实现代价。
- 通过各种程序设计语言之间的比较，更好地理解我们目前所使用的程序设计语言。
- 理解与各种语言特征所关联的编程技术。学习程序设计语言的一个方面，就是要学习解决问题、软件构造和开发的概念框架。本书中的很多思想都是职业程序员的共识。本书所介绍的内容和思考方式将会对你以后的编程有所帮助，如果你在一个软件公司工作或者获得了一个工作面试机会，这也会对你有所帮助。当这门课程结束的时候，你将能够评价一个语言的各种特征、它们的代价以及它们是如何结合在一起的。

1.2.2 特殊主题

下面是本书中将反复强调的内容。

- **可计算性** 有些问题是计算机无法解决的。停机问题的不可判定性意味着程序设计语言的编译器和解释器并不能完成我们希望它们能够完成的所有工作。

- **静态分析** 编译时和运行时是有区别的。在编译时，程序是确定的但是输入是未知的。在运行时，程序和输入对于运行时系统来说都是确定的。尽管程序设计者或实现者希望能在编译时找出错误，但很多错误都是在运行时才能被发现的。在编译时检测程序错误的方法通常都是保守的，即当它们说某个程序不含有某种类型的错误，这句话肯定是正确的。然而，编译时错误检测方法会说程序含有某些错误，而实际上程序在运行的时候可能并不会出现错误。
- **可表达性与效率** 在很多情况下如果程序设计语言的实现工具能够自动地做一些工作则会带来很大的便利。其中一个示例就是在第3章中讨论的内存管理：Lisp运行时系统使用垃圾收集器来检测程序中无用的存储单元。如果系统能自动完成一些工作，则必然要付出代价。尽管一个自动化的方法能够让程序员不再考虑某些事情，但是语言的执行速度将会慢很多。在某些情况下，自动化方法会使得编写程序变得更容易并且编程不容易出错。在另一些情况下，程序执行速度的减慢是无法容忍的，这时自动化方法就不可行了。

1.3 程序设计语言的历史

在过去的50年中，人们已经设计并实现了上百种程序设计语言。这些程序设计语言中的50种都包含了新的概念、有用的改进，或者值得一提的创新。这些程序设计语言之多，在这里我们无法一一研究，但是我们将集中于6种程序设计语言：Lisp、ML、C、C++、Smalltalk 和 Java。这6种语言涵盖了自高级程序语言在1960年前后从最初的汇编语言中脱颖而出以来所发明的大多数的重要语言特征。

现代程序设计语言的历史始于大约1958—1960年Algol、Cobol、Fortran 和 Lisp 的发展。本书的大部分都讲的是Lisp，对于Algol以及随后出现的相关语言只有简短的讨论。鉴于有些读者对程序设计语言的史前史感到好奇，在这里我们简单的介绍一些早期的语言。

20世纪50年代，人们开发了一些语言来简化书写一系列计算机命令的工作。用现代的标准来看，那十年中的计算机是相当原始的。大多数的程序是用底层硬件的本地机器语言书写的。由于当时的程序很短小并且效率非常重要，人们还能接受这样的语言。在50年代开发的最重要的两种语言是Fortran 和 Cobol。

Fortran 是在大约1954—1956年由IBM的John Backus所率领的小组开发出来的。Fortran (formular translation 的缩写)主要的创新在于可以在表达式中使用普通的数学符号。例如，将*i*的值加上两倍*j*的值，Fortran表达式是 $i+2*j$ 。在Fortran开发出来以前，则需要把*i*的值存放在一个寄存器里面，把*j*的值存放在另一个寄存器里面，把*j*的值乘以2之后再与*i*相加。在Fortran中，通过对变量使用符号名，程序员能更自然地思考数字计算而把计算顺序等问题留给编译器处理。Fortran也有子程序（一种程序或函数的形式）、数组、格式化输入输出，以及声明。声明让程序员能够显式地控制变量和数组在内存中的存放位置。然而，这就是它的全部。关于Fortran的局限性，这里仅举几例。很多早期的Fortran编译器把数字1, 2, 3……存放在内存中，这样的话程序员就有可能由于疏忽大意而改变了这些数字的值。另外，Fortran子程序不能调用其本身，因为支持它的内存管理技术在当