ADA程序设计

〔美〕J.G.D.巴恩斯 著
娄 宾 译
张兆庆 校

高等教育出版社

ADA 程 序 设 计

[美] J.G.P.巴恩斯 著
娄 宾 译
张兆庆 校

高等教育出版社

内 容 提 要

本书详细地介绍了 Ada 语言及其应用.全书共 16 章,按教材形式编写,着重介绍了 Ada 的主要概念和特征,写得重点突出,层次分明,举例生动,叙述流畅并且富有幽默感.本书作者是 Ada 语言设计小组成员之一,对 Ada 进行过深入的研究,因此在本书中对 Ada 语言的历史、现状和发展前景做了较详尽的描述,并对语言各部分功能的优劣做出了客观的评价.

本书可作为大专院校软件专业高年级学生和研究生选修课教材或参考书,也可供从事计算机工作的有关专家、学者和工程技术人员阅读参考.

ADA 程序设计

[美] J.G.P.巴恩斯 著

娄 宾 译

张兆庆 校

*

高等教育出版社出版

新华书店北京发行所发行

高等教育出版社照排中心照排

国防工业出版社印刷厂印装

*

开本 787×1092 1/16 印张 20.75 字数 510 000

1990 年 8 月第 1 版 1990 年 8 月第 1 次印刷

印数 0001—2 000

ISBN7-04-002006-8/TP · 42

定价 5.35 元

序　言

1983 年 1 月，美国国家标准协会 (ANSI)的 Ada 标准的出版，加速了本书第二版的问世.

经过了ANSI对 Ada 的标准化，结果使 Ada 发生了许多变化.我对全书内容作了彻底的修订，以符合新标准的细节与精神.本书总的目的没有变化，只是在某些方面添加了一些细节和更多的实例.

我还趁此机会改正了第一版中的各种错误.在这方面，我从收到的许多读者的大量评论中获得很大的帮助.我特别对Randall Barron ,Theodore Chaplin , Donald Clarson , EdWard Colbert , Ian Mearns , Angel Rodriguex , Mike Tedd 和 Jo Whitfield 提出的广泛的建设性的批评意见表示感谢.

J.G.P.巴恩斯

于雷丁

1983 年 10 月

第一版序言

本书是讲述Ada语言的.Ada是一种强有力的新程序设计语言,最初是为美国国防部研制,用于嵌入系统.这种系统的典型例子有过程控制、导弹制导、甚至还有洗碟机的定序.过去这些系统用JOVIAL, CORAL66和RTL/2等语言编写程序.

Ada语言是在Pascal基础上,集许多重要特性诸如:数据抽象、多任务处理、异常处理、封装、类属等为一体的第一个实用语言.虽然Ada最初是为嵌入系统设计的,但它是一种通用语言,早晚要取代FORTRAN甚至COBOL.Ada背后的政治和技术力量也表明它将成为80年代的重要语言.

我写此书的目的在于给出一个对Ada的全面描述.我们假定读者具有一些关于程序设计原理的知识.读者如熟悉Pascal对理解本书将会有所帮助,但这并不是绝对必需的.本书按照教材的方式编写,内有大量例子和习题.我已尽力阐明许多语言特征的理论依据,因为这不但会使论述更有趣味,而且会使论据更易于记忆.在可能的情况下,我尽量使用没有具体应用背景的例子;这些例子大部分取自日常生活,或取自数学,数学毕竟是科学和工程的基础.希望读者不要对书中偶尔出现的幽默感到不妥,因为学习程序设计语言是非常枯燥的.任何能使之轻松的办法似乎都是可取的.

借此机会,谨向直接或间接帮助过我写作此书的各位致谢.首先,我要感谢美国国防部允许我使用Ada语言参考手册(LRM)中的材料.其次,我要感谢Jean Ichbiah和Robert Firth,我曾愉快地同他们在世界各地讲授Ada语言课程.这些课程不但有助于我对Ada语言整体有了有益的认识,而且还提供了某些例子的来源.再者,我还要感谢Andrew McGettrick,感谢他曾对本书的原稿提出了许多有益的意见,感谢他的许多有益的建议,使我能够很快完稿,我对英国Ada研究小组的同事们也表示感谢,他们曾提出了一些有价值的意见.最后,我要深切地感谢我的夫人,她为打印手稿作了不倦的努力,没有她的帮助,我的工作将会大大拖延.

J.G.P.巴恩斯

于雷丁

1981年4月

前　言

1979 年 5 月，热切盼望的时刻终于来到了——美国高级语言工作小组宣布“绿色方案为 Ada！”

我们小组的人员对此并不惊讶，因为大家一直在等待这一时刻的到来．尽管如此，这个消息仍无疑给我们每个人都带来了巨大的喜悦．当时，我们都认为 Ada 的这个最初版本是完善的．但看一看随后十五个月中对版本所作的修改的数量，以及经这些修改所产生的 Ada 的 1980 年 7 月的定义，你也许会感到惊讶．但要解释这点也很容易，为了设计一个东西，你就必须相信自己的设计，必须不断地检查所有特征的组合情况．当然，很多决定是在相互冲突的目标之间进行折衷的产物．但你在对产生这些折衷办法的论点作了反复斟酌之后，最终会把它们综合成你的设计的基本假设．因此，以为这样得到的结果应该是完善的答案是不奇怪的．

很清楚，只有重新估价某些折衷办法中隐含的假设，才能使设计进一步完善．而这正是任何设计工作中的主要矛盾之点．一方面，只有埋头研究现实存在的那一部分逻辑，你才能够使各种特征得以和谐地综合起来；只有通过这一途径，你才能够获得一种完美的结合．另一方面，这种对完美性的理解，以及对某些无意识假设的隐含认可，又会妨碍设计的进一步改进．

在 Ada 的设计中，John Barnes 确实显示了一种在认可和重新估价之间做出适当选择的非凡能力．1979 年 5 月，John 和设计小组的其他人一样，也为我们的成功而自豪．但在 6 月初，他就开始思考并对 Ada 的任务处理模型进行了尖锐甚至是严厉的审视，就好象他根本没有参加过设计似的．他写出了名为 《任务处理的问题 》的第一篇语言学习笔记．学习笔记共 200 篇．他在笔记中概括了在最初 Ada 的任务处理设备中发现的 10 个主要问题．在对 Ada 修改时，这些问题花去了我们小组几个月的时间，以使这些问题能在最后的 Ada 版本中得到满意的解决．而我自己无疑是花了更多的时间才对最初的 Ada 版本有了客观的认识．最初我确实对 John 的亵渎行为深感不安．但随着时间的推移，我已非常看重和 John 的这种相互影响了．

《Ada 程序设计》一书就是这样由 Ada 语言设计小组的这位主要成员写成，他对这一语言的设计的各个方面，无论从建设意义上还是从批评意义上说，都是十分了解的．象平时那样说句玩笑话，我相信 John 的热情和对于 Ada 语言精神的理解一定会传给本书的各位读者．

Jean D．Ichbiah

于凡尔赛

1981 年 5 月

目　录

第1章 引 言

Ada是一种高级程序设计语言，最初由美国国防部发起研制，用于嵌入系统这一应用领域(所谓嵌入系统是以计算机为其中主要部分的较大系统，如化工厂、导弹、洗碟机等)。在引言中，我们将简要概述Ada的研制过程，它在整个语言舞台中的位置以及本书其余部分的一般结构。

1.1 历史

Ada语言的历史大约要追溯到1974年，当时美国国防部认识到用于软件的开销实在太大。因此，他们对各个应用领域的开支分配情况进行了详尽的分析，结果发现一半以上的开销直接用于嵌入系统。

进一步的分析对象是各领域中所使用的程序设计语言，结果发现，COBOL是用于数据处理的通用标准，而FORTRAN是用于科学和工程计算的通用标准。虽然这两个语言不是新近开发的，但它们一直在各自领域得到应用这一事实表明，应该避免做那种不必要的、昂贵的复制工作。

但是嵌入系统的情况完全不同，正在使用的语种数目巨大。美国三军不但有各自偏爱的高级语言，而且还使用了许多汇编语言。此外，那些高级语言已衍生出各式各样的变种，三军所签订的一系列合同似乎也促进了用于不同应用领域的专用版本的研制。最后结果是大笔的钱花在一些并非完全必要的编译程序上。同时由于缺乏标准化，还要有用于训练和维护的各种额外开销。

因而，人们认识到要控制花费，就必须在嵌入系统领域建立一个标准，最终目标当然是创立一个单一的语言。在短期内，初步入选的语言名单开列出来了，在CMS-2Y，CMS-2M，SPL/1，TACPOL，JOVIAL J3，JOVIAL J73，当然还有用于别的领域的COBOL和FORTRAN。

着手建立单一标准的第一步是写出一个概括各种需求的文件，第一个版本叫做稻草人，于1975年初公布。在收到来自各方面的意见后，对它进行了改善，使之成为木头人。进一步照此办理，于1976年6月产生了锡人。锡人完全是一个专用文件，它标识了该语言所需的功能。

在这一阶段，对照锡人对许多现存语言做了评价，看看其中是否有能作为最后标准的语言，同时也对需求本身进行详尽的评价。正如所料，现存语言中没有一个能令人满意；另一方面，人们得到一个总的印象，即可以研制一个基于现有技术水平观念的单一语言，以满足各种需要。

通过评价，把现存语言分成了三类，说明如下：

“不合适的”：这些语言或是已经过时，或是使用领域不适当，对它们不再做进一步的考虑。这一类语言包括FORTRAN，CORAL 66。

“并非不合适的”：这类语言就其现状来说也不令人满意，但却有一些饶有趣味并且可以从中得到启示的特色，这一类语言包括RTL/2和LIS。

"推荐作为基础的":这一类包括三种语言:Pascal,PL/1,Algol 68.它们被认为是设计最终语言时可以接受的起点.

这时,又对需求文件做了修订和整理,产生了铁人.然后请承包者提出方案,从"推存作为基础的"中的一个语言出发,设计出新的语言,总共收到17份方案,选中其中的4个,让他们同时继续搞下去,相互竞争,这四个承包者用颜色作代码: CII Honeywell Bull (绿), Intermetrics (红),Softech (蓝),SRI International (黄),采用颜色代码是为了能对所得到的初始设计方案进行匿名比较,以期排除偏见.

1978年初送来了初始设计方案,世界各地的许多小组仔细地估量了这些设计方案的相对长处,美国国防部断定:绿色和红色的设计方案比蓝色和黄色的更有前途.这样,后两者就被排除掉了.

于是研制工作进入了第二阶段.又给剩下的两个承包者一年的时间来改进设计方案.根据初始设计的反馈信息,对需求也做了修订,使之成为最后的文件——钢人[1].

1979年5月2日,对语言做出了最后选择,宣布由Jean Ichbiah领导的国际小组在CII Honeywell Bull研制的绿色语言获胜.

随后美国国防部(DOD)宣告,新语言将命名为Ada,以纪念Lovelace伯爵夫人Augusta Ada Byron (1815—1852),Ada是Byron勋爵的女儿,她作为英国数学家Charles Babbage的助手和赞助者,曾在他的分析机上工作过*.因此在实际意义上说来,她是世界上第一个程序员.

Ada语言的研制工作于是进入了第三阶段.这一阶段的目的是让语言和那些重要的、有代表性的最终用户见面,让他们评价Ada对其需要的适应程序,于是在美国和欧洲,开始讲授关于Ada语言的各种各样的课程,同时,许多小组着手进行评价工作.他们写出了大约80篇综合报告,提交给了1979年10月的波士顿会议.总的结论是:Ada是好的,但有些方面需要进一步改进.另外,还收到近1000份关于语言的技术问题的短篇报告.在对所有这些报告加以考虑之后,又对Ada语言的最初设计做了修订,于是在1980年7月,Ada语言的第一个定义性版本出版了.然后它被作为一个标准提交给美国国家标准局(ANSI).

ANSI用了两年多的时间使Ada语言标准化,其结果对Ada作了一定数量的更改.大部分更改都是细小的,但却有相当的重要性,这对编译程序的编写者来说,尤其如此,最后ANSI标准语言参考手册(LRM)于1983年1月出版了,而正是这一版本构成了本书的主题[2].

但是,不应该认为Ada仅仅是另外一种程序设计语言.还应该认识到它只是每个程序员应该能使用的工具箱的一部分,虽然是一个重要的部分.因此,如果还能建立一个统一的程序设计环境,就会领略到另外的益处,为此,与语言设计并行,为Ada程序设计支撑环境(APSE)研制了一系列需求文件,这些文件取名为沙人,卵石人,最后版本称为石头人[3].这些文件没有相应的语言文件那么详细,因为在此领域中目前技术水平还不成熟.尽管如此,还是希望所做的这些工作将会防止不必要的多样化,并能研制出一个或多个高质量的APSE.但那是本书范围以外的事了.

---

*英国数学家C.Babbage于1833年研制成的分析机体现了现在数字计算机的某些基本原理,是通用数字计算机的早期设想.——译注.

1.2 技术背景

程序设计语言的发展显然是以相当特别的方式进行的，但事后看来，现在有可能看到三个主要的进展，每一个进展似乎都与引进一级抽象相联系。抽象把不必要的、有害的细节从程序中排除掉。

第一个进展发生在50年代早期，象FORTRAN和Autocode这样的高级语言引进了"表达式抽象"。这样就有可能把语句写成

$$X = A + B \quad (I)$$

从而把如何使用机器寄存器来计算表达式的细节对程序员完全隐蔽起来。在这些早期语言中，表达式抽象还不完善，因为对表达式的复杂性有一些比较随意的限制，例如，下标必须取特别简单的形式，晚些时候的语言如Algol 60就去掉了这种限制，完善了表达式抽象。

第二个进展是有关"控制抽象"的。Algol 60就是最好的例子，该语言向前边进了值得注意的一步。从那以后，还没有哪一种语言能对后来的发展产生这样的影响。控制抽象的要点是控制流结构化，对每个控制点不必再命名或编号，这样，我们只要写出：

if X = Y **then** P := Q **else** A := B

编译程序就能生成那些转移和标号，但在象FORTRAN这样的语言中它们必须显式地写出来。表达式抽象早期的那种不完善现象在控制抽象中再现了。在控制抽象中明显的缺陷是Algol 60中令人生厌的开关语句，它现已被如Pascal等语言中的CASE语句所替代（早些时候的Algol 68中的CASE子句有其自己的问题）。

第三个进展是正在出现的"数据抽象"，即把数据表示法的细节与在该数据上定义的抽象运算分离开。

现有的大多数语言对数据类型的看法非常简单，数据在任何情况下都直接用数值项来表示。因此，如果要处理的数据不是真正的数值（可能是交通灯的颜色），那么就要由程序员来完成从抽象类型到一种数值类型（通常是整型）的某种变换，除了可能作为注释外，这个变换并不出现在程序中，而是完全在程序员的头脑中。这很可能是由于除了数值分析外其他软件库还没有出现的缘故，许多数值算法，如求矩阵特征值算法，是直接与运算的数相联系，因而，以数作为数据值的这些语言，已表明是适用的。主要之点在于这些语言只对这种情况才提供了正确的抽象值，对于其他情况，由于所需的变换未必一致，软件库就不成功了。事实上不同情况用不同的变换才可能处理得最好，而这些变换遍及整个程序，改变变换往往要对整个程序重写一遍。

Pascal引进了一定数量的数据抽象。枚举类型就是例证。枚举类型允许我们在谈论交通灯颜色时用其本身的术语，而不必了解它们在计算机中是怎样表示的。另外，枚举类型还防止我们产生一类严重的程序设计错误——偶然把交通灯和其他抽象类型如鱼的名字混淆了。如果在程序中所有这些类型都描述为数值类型，就可能产生这类错误。

数据抽象的另一种形式与可见性有关。人们早已认识到，传统的Algol 60分程序结构是不够用的。比如：要写出对公用数据进行操作的两个过程，使得这两个过程可存取这些数据而又不是直接可存取，这在Algol 60中是不可能的。很多语言通过分别编译的方法提供了对可见性的控制，这种技术对中型系统是足够的了，但由于分别编译功能往往依赖于某个外部系统，因

而不能实现对可见性的全部控制.Modula 语言的模块是一个合适结构的例子.

另一种语言 Simula 67 以其类程的概念而对数据抽象的发展做出了重要贡献.还有许多其他的实验语言,数目之多难以一一列举,它们也都做出了各自的贡献.

Ada 似乎是第一个把各个种类的数据抽象汇集到一起的实用语言.现在我们可能是"不识庐山真面目,只缘身在此山中".毫无疑问,正如 FORTRAN 的表达式抽象和 Algol 60 的控制抽象不够完善一样,Ada 的数据抽象也会有其不完善之处,但 Ada 确是一个重大进展,它为编写用于数值分析以外各领域的可重用的重要软件库提供了可能性,因而它将会促进软件行业的产生.

1.3 本书的结构及其目的

学习程序设计语言有点象学开汽车.要取得实际进步必须先学某些关键的东西,虽然我们不必了解怎样使用雨刷,但至少必须要会启动汽车发动机、挂档、驾驶和刹车.对学习程序设计语言来说也是如此,要能编出可用的程序,我们不必先了解 Ada 的全部内容,但必须要学习相当多的内容.况且只有当编写大型程序时,Ada 的许多优点才会明显地显示出来,就象如果我们开 Rolls Royce 汽车只到附近的商店去,它的许多优点就不会显出来一样.

本书不是程序设计入门,而是对 Ada 程序设计的全面描述.我们假定读者对使用某种高级语言进行程序设计具有相当的知识.了解 Pascal 将是有益的,但这不是绝对必需的.

本书第二章对 Ada 的一些概念作了简要概述,为的是让读者感受 Ada 的风格,了解它的目标.其余部分按教材方式编写,对各个题目的介绍是按照非常简单易懂的顺序进行的.截止到第七章,所讲的内容就已包括了小型语言 (如 Pascal)的传统功能.其余各章包括了与数据抽象有关的、令人兴奋的新材料:大型并行处理中的程序设计.

本书大部分章节都有习题.即使不全做也要做大部分习题,这点对读者是很重要的,因为这些习题是论述的必要组成部分,后面的章节常常用到前面习题的结果.全部习题的答案附在本书最后.虽然在编写时并未将全部答案提交给一个 Ada 编译程序,但我们希望这些答案都是正确的.

大部分章节的末尾有一张需要记住的要点的简短的检验表.虽然这些检验表内容不完整,但还是会有助于读者巩固理解所学内容.另外,我们鼓励读者参阅附录 4 中的语法,语法的顺序与所介绍的题目顺序相对应.

本书虽然包括了 Ada 的各个方面,但并没有考察各种变态情况.我们的目的是告诉读者 Ada 特点的效果和对这些特点的应用.有些方面论述得并不完全;这包括定点算术运算、与机器有关的程序设计和输入输出.定点算术运算是一个非常专门的题目,对大部分程序员来说不是必需的.与机器有关的程序设计,顾名思义是非常依赖于具体实现的,故仅对它简单介绍一下似乎就可以了.输入输出虽然重要,但除了一大堆细节外并没有引进新概念,故也只做了简单介绍.上述题目的更多的细节可在经常参阅的语言参考手册 (LRM)中找到.

为了使本书合理地自成一体,我们提供了各种附录,其中大部分是基于从语言参考手册中抽出来的材料.建议读者查阅 Ada 的正式定义——语言参考手册,但这并不是绝对必需的.

1.4 参考文献

[1] *Department of Defense Requirements for High Order Computer Programming Languages* – "STEELMAN", Defense Advanced Research Projects Agency, Arlington, Virginia, June 1978.

[2] *Reference Manual for the Ada Programming Language*, (ANSI/MIL – STD – 1815A) United States Department of Defense, Washington D.C., January 1983.

[3] *Department of Defense Requirements for Ada Programming Support Environments* – "STONEMAN", Defense Advanced Research Projects Agency, Arlington, Virginia, February 1980.

第2章 Ada的概念

在这一章中，我们对Ada的一些目标，概念和特点做一简要概述.

2.1 主要目标

Ada是一个大型语言，因为它涉及到了现实世界中实际系统的程序设计的许多重要问题.例如，它比Pascal庞大得多，Pascal若不做某种扩充，实际上则只适用于教学目的(它也正是为此而设计的)和个人小型程序.Ada的一些重要结果是：

· 可读性——人们认识到阅读专业程序的次数远比书写这些程序要多.因此，避免象APL中的那种过于简洁的表示法是重要的.虽然这种表示法能很快写出程序，但要读懂它，除了原作者在刚刚写成时还可以外，则几乎是不可能的.

· 严格类型化——这点保证了每个实体都有一个定义明确的值的集合，避免了混淆逻辑上不同的概念.结果许多错误就被编译程序发现了，而在其他语言中会产生一个虽可以执行但不正确的程序.

· 大型程序设计——封装技巧、分别编译和库的管理对于书写可移植的、可维护的任何规模的程序都是必要的.

· 异常处理——重要的程序很少是正确的，现实生活就是这样.因此有必要提供一种能按分层和分块方法构造程序的手段，以便限制住某一部分中的错误所造成的后果.

· 数据抽象——如前所述，如果把数据表示的细节与对数据的逻辑操作的说明分离开来，会得到特佳的可移植性和可维护性.

· 任务处理——对许多应用来说，把程序表示成一系列并行活动而不是单一系列的活动，是重要的.在语言中建立适当的功能，而不是通过调用把这些功能添到操作系统上，这会产生更好的可移植性和可靠性.

· 类属单位——在很多情况下，一部分程序的逻辑独立于要处理的值的类型.因此有必要设一个从单一模式中产生程序有关部分的机构.这对建立库特别有用.

2.2 概述

程序设计的最重要的目标之一是要重复使用已有的程序段，从而使琐碎的编写新代码工作减少到最低限度.程序库的概念应运而生，因此，程序设计语言的一个重要方面就是说明如何使用库中东西的能力.

Ada语言认识到了这种形势，并引入了库单位的概念，一个完整的Ada程序被表示成一个可以调用其他库单位为其服务的主程序(它本身就是一个库单位).可以认为这些库单位构成了整个程序的最外层词法部分.

主程序的形式是一个具有适当名字的过程.服务库单位可以是子程序(过程或函数)，但是更可能是程序包.程序包是一组相关项目，这些项目可以是其他实体及子程序.

假设我们想写一个程序来打印某个数，如 2.5 的平方根．有各种计算平方根并产生输出结果的库单位提供我们使用．我们的工作不过是要写一个按我们的意愿使用这些服务程序的主程序．

为了讨论方便起见，我们假设通过调用库中名为 SQRT 的函数就能得到平方根．另外还假设我们的库中含有一个叫做 SIMPLE_IO 的程序包，它包括了各种简单输入输出功能．这些功能包括读数、打印数、打印字符串等过程．

我们的主程序可以是这样：

```
with SQRT,SIMPLE-IO;
procedure PRINT_ROOT is
   use SIMPLE_IO;
begin
   PUT (SQRT (2.5));
end PRINT_ROOT;
```

这程序写成一个名为 PRINT_ROOT 的过程，它前面有一个 with 子句，给出了要使用的库单位的名字．过程体只含一个语句

```
PUT (SQRT (2.5));
```

它把用参数 2.5 调用函数 SQRT 所产生的结果作为参数，去调用程序包 SIMPLE_IO 中的过程 PUT．写出

```
use SIMPLE_IO;
```

使我们能直接使用程序包 SIMPLE_IO 中的各种功能．如果我们省掉这个使用子句，那么为了指出到哪去找 PUT，我们必须要写成

```
SIMPLE_IO.PUT (SQRT (2.5));
```

让程序读入我们所要求的平方根的数，能使我们的程序更有用．这样程序变成了：

```
with SQRT,SIMPLE_IO;
procedure PRINT_ROOT is
   use SIMPLE_IO;
   X:FLOAT;
begin
   GET (X);
   PUT (SQRT (X));
end PRINT_ROOT;
```

现在，过程的整个结构更清楚了．我们可以在 **is** 和 **begin** 之间写说明，在 **begin** 和 **end** 之间写语句．粗略地说，说明引入那些我们要处理的实体，语句指示要执行的顺序动作．

现在我们引入一个 FLOAT 类型的变量 X，FLOAT 是语言的预定义类型．这种类型的值是确定的浮点数集合，X 的说明指出 X 只能从这个集合取值．在我们的例子中通过调用程序包 SIMPLE_IO 中的过程 GET 将值赋给 X．

需要注意某些微小细节．各种语句和说明均以分号结尾．这点不同于其他语言，如 Algol 和 Pascal，分号在那里用做分隔符而不是结束符．程序中包含各种标识符如 **procedure**，PUT 和 X．

它们分成两类.有些(实际上是63个)标识符如**procedure**和**is**用来指出程序的结构;它们被保留,不做它用.其它所有标识符如PUT和X可任凭我们随意使用.其中有些,特别是我们例子中的FLOAT,具有预先定义好的含义,尽管如此,若我们愿意的话,也可重新使用它们,但是这样做可能造成混乱.为清楚起见,在本书中我们用小写黑体字母表示保留标识符,其他标识符用大写字母.这纯粹是为了表示方便.语言规则不区分大小写,但处理字符本身时例外.另外,请注意如何使用下横线把一个长标识符分成若干有意义的部分.

最后,注意过程名PRINT_ROOT在最后的**end**和结束符号之间又重复出现了.这是可以省略的,虽然象上面例子那样的小程序的结构亦很清楚,但我们仍建议使用,以便使整个程序结构清晰.

我们的程序仍然很简单.如果让它能处理一组数并把每个结果分行打印出来,这程序可能会更有用.我们可以给它一个零值使程序随时停下来.

```
with SQRT,SIMPLE_IO;
procedure PRINT_ROOTS is
    use SIMPLE_IO;
    X:FLOAT;
begin
    PUT ("Roots of various numbers");
    NEW_LINE (2);
    loop
        GET (X);
        exit when X=0.0;
        PUT ("Root of");
        PUT (X);
        PUT ("is");
        if X<0.0 then
            PUT ("not calculable");
        else
            PUT (SQRT (X));
        end if;
        NEW-LINE;
    end loop;
    NEW-LINE;
    PUT ("Program finished");
    NEW_LINE;
end PRINT_ROOTS;
```

调用程序包SIMPLE_IO中的过程NEW_LINE和PUT,使输出功能得到加强,调用NEW_LINE将输出由参数指定的那么多个新行;过程NEW_LINE以这样的方式写出,

即在不给它提供参数时，则省缺值为 1．对 PUT 的调用还有用字符串做参数的．事实上，它是一个与打印数 X 不相同的过程．由于参数的类型不同，因而编译程序能分清它们．多个过程使用同一名字称为重载．还要注意字符串形式；在这种情况下，字符的大小写要起作用．

还引进了各种新的控制结构．**loop** 和 **end loop** 之间的语句被重复执行，直到发现 **exit** 语句中的条件 X＝0．0 为真时终止；这时循环结束，立即从 **end loop** 之后接着执行．我们还要检查 X 是否非负；若为负则输出信息"不可计算"，而不去调用 SQRT．这由条件语句完成；若 **if** 和 **then** 之间的条件成立，则执行 **then** 和 **else** 之间的语句，否则执行 **else** 和 **end if** 之间的语句．

应遵守一般的括号结构规则．**loop** 和 **end loop**，**if** 和 **end if** 相配对．Ada 的所有控制结构都是这种封闭式的，而不是 Pascal 的那种会产生不良结构和不正确程序的开启式．

我们或许会问，如果没有检验 X 的负值而用负变量调用 SQRT，那将会发生什么情况？假设 SQRT 本身编写得当，那很清楚，它决不会给出一个用作 PUT 参数的值，而是引发一个异常．异常的引发表示发生了不寻常的事情，正常的执行顺序被打断．在我们这种情况下，异常可能是 NUMERIC_ERROR．如果我们不对这种可能性做任何考虑，那程序就会终止，Ada 程序设计支撑环境 (APSE)必定会给出一个不客气的信息，告诉我们程序失败了，以及为什么失败．但是我们可以提防异常，若异常发生了，就采取补救措施．实际上我们可以用下边语句代替条件语句．

```
begin
    PUT (SQRT (X));
exception
    when NUMERIC_ERROR=>
        PUT ("not calculable");
end;
```

现在我们来大体上考虑一下函数 SQRT 和正在使用的程序包 SIMPLE_IO 的可能的一般形式．

函数 SQRT 要有个类似我们主程序的结构；主要不同在于有参数存在．

```
function SQRT (F:FLOAT) return FLOAT is
    R:FLOAT;
begin
    --compute value of SQRT (F) in R
    return R;
end SQRT;
```

这里我们看到了对形参 (此处只有一个)和结果类型的描述．我们用以两个横线开头的注释来表示计算的细节．返回语句是指出函数结果的一种手段．注意函数与过程的不同：函数返回一个结果并作为表达式的一部分被调用，过程不带结果并作为单个语句被调用．

程序包 SIMPLE_IO 要有两部分，描述与外界接口的说明部分和包含实现细节的体．若它只包含我们已用过的过程，那么其说明可以是