Ada 程序设计语言

殷月康
张志贤
编著

科学技术文献出版社

Ada程序设计语言

殷月康　张志贤　编著

科学技术文献出版社

内 容 简 介

全书共12章。内容包括：1.Ada语言概述；2.Ada语言基本成分；3.Ada语言的基本语句；4.过程和函数；5.程序包；6.自定义数据类型；7.构造型数据类型；8.专用数据类型；9.文件的输入/输出；10.异常处理；11.任务；12.类属程序单元。最后附有预定义程序包的源程序供参考。

本书特点：内容完整、资料翔实。文字叙述简明扼要、通俗易懂。既可自学，又可作为教材。

供具有高中以上文化程度的科技人员、计算机工作者及大专院校师生阅读。

Ada程序设计语言

殷月琨　张志贤　编著

科学技术文献出版社出版

北京昌平星城印刷厂印刷

新华书店北京发行所发行　　各地新华书店经售

*

787×1092毫米 32开本 9　　印张 194千字

1990年6月北京第一版第一次印刷

印数：1—3000册

科技新书目：　218—126

ISBN 7-5023-1215-3/TP·76

定　价：7.00元

前　言

Ada语言是美国国防部历时8年，耗资5亿美元研制成功的计算机程序设计语言。

在美国国防部（DOD）内部，自1984年7月以来已用Ada语言实现了几个重大系统（mission critical system）。在C^3I中的应用更是与日俱增，如世界范围的军事命令控制系统（WWMCCS）、信息系统（WIS）、MILSTAR和炮兵作战数据系统（AFATDS）。各种使命支撑系统也正在开发之中。其它领域诸如人工智能、石油地震分析、通讯、航空管理、数据库管理及计算机网络系统等方面的应用也在稳步发展。

在教育领域，目前美国已有200种左右不同的 Ada 语言教材，全国有26所大学开设了Ada语言课程。国际上，许多国家已宣布了军用系统的Ada语言标准化策略。

为适应这一新形势的需要，我院决定开设 Ada 语言课程，同时下达了编写书的任务。标准Ada语言是一个用于嵌入式计算机系统的大型程序设计语言系统，配备有比较完善的程序设计环境支撑软件包，国内目前还没有标准的Ada语言全集或子集。现在IBM PC/XT机上配置的JANUS/Ada语言是非标准的（仅相当于 Ada 语言的一个子集），其功能只能达到全集基本功能的四分之三左右（尚未实现的内容有多维数组、切片、聚集值及第十章以后的部分），但由于Ada

语言的功能可通过自身所携带的程序包扩充，因而，JANUS/Ada语言配置的功能已能满足一般需求。

鉴于上述原因，编写本书的基本指导思想是：内容尽量与Ada语言标准相吻合，以保证使用的稳定性；对JANUS/Ada语言已经实现部分叙述从详，尚未实现部分（第十章以后）叙述适当从简；结构安排力图循序渐进，严谨合理；文字叙述力求通俗易懂，简明扼要，以利自学；凡是完整的例题，都经过上机运行。每章后面，基本都列有练习题。

JANUS/Ada语言仍不失其资源丰富的特色，为了方便开发利用，在本书中公开了该语言预定义程序包的全部内容。包的声明部分，作为附录列于第五章后，供了解语言已设置的功能；包的体部分（完全用汇编设计的包体除外）作为附录一列于本书其后，供分析算法、学习语法规则及研究程序设计技巧时作样板参考。

本书由殷月康（1—8章及12章）、张志贤（9—11章）两同志编写。限于编者水平，加之时间紧迫，书中有不妥之处，敬请读者批评指正。

编　者

1986.9于宝化

目　　录

第一章　Ada语言概述

Ada语言是美国国防部耗资5亿美元，历时8年才研制成功的全军统一程序设计语言。它体现了现代软件工程的成功经验和最新成就，被誉为第三代计算机语言的顶峰，第四代计算机语言的成功代表。

本章将介绍Ada语言开发史、程序结构与在IBM PC/XT机上使用Ada语言的过程。

▲重点：要求掌握Ada语言程序的编辑、编译、连接及运行方法。

第一节　Ada语言开发史

Ada语言的研制经历了分析、需求定义、设计、测试、运行和维护五个阶段。

一、分析阶段

70年代初期，美国国防部就已注意到大型防务系统的软件费用有升高的趋势。在那个时间以前，这种系统的硬件费用远远超过软件费用；虽然，软件不可靠以及费用超支等问题才开始暴露，但大型软件的开发问题还未被人们充分认识。1973年，美国国防部用于计算机的支出总数为75亿美元，软件费用占46%（超过30亿美元）。另外，嵌入式计算

机系统的软件支出又占整个软件支出的56%，而数据处理占19%，科学计算应用仅占5%。剩余的20%包括其它的和非直接的软件费用。

从1968年到1973年，虽然硬件开支大幅度减少，但是美国国防部用于计算机系统的直接开支仍然增加了51%，实际上这些开支仅仅指出了美国国防部碰到的一部分问题。

此外，软件工具本身存在的一些先天性缺陷，也影响着这些问题的解决。这些缺陷表现在如下几个方面：

▲多种程序设计语言并存；

▲各种语言的应用水平低劣；

▲流行的语言不支持现代程序设计方法；

▲缺少有用的软件环境。

在此期间，美国国防部系统至少有450种通用语言，若考虑到这些语言的计算机系统的特点，实际上有500种到1500种不同的高级语言和汇编语言。在那时美国国防部并不坚持控制每种语言的用途，每个设计机构实际上都在自由地创造自己的语言或使用与现存语言不兼容的各种版本。于是导致了训练问题、不同课题之间的技术转移问题的脱节与混乱，以致造成资源的普遍浪费。此种现象被称为软件危机。

美国国防部拥有庞大的形式各异的嵌入式计算机系统（即计算机通常并不单机使用，而是将它加入到网络中或联到控制系统中，按这样的应用方式组合起来的系统被称为嵌入式计算机系统），它的应用也是多种多样的。归纳起来，这些软件系统的特征是：

▲ 规模庞大，代码达千百万行；

▲ 生存期长，10年到15年；

▲ 持续变化，要根据需求的变化进行不断的修改；

▲ 物理的约束，目标机的地址空间或运行速度方面的限制；

▲ 高可靠性，能容错（某一个任务出现故障不影响其它任务的执行）。

除此之外，嵌入式系统又有自己特定的程序设计方面的要求：

▲ 并发处理（Parallel processing）；

▲ 实时控制（Real_time control）；

▲ 异常处理（Exception handling）；

▲ 独特的输入/输出控制（Unique I/O control）。

鉴于美国国防部的大多数软件费用与嵌入式计算机系统相联系这一事实，迫使她将注意力转移到这个特定的领域，并开始理解从整个软件生存期的费用角度来考察使用一个高级语言的效益。此外，硬件速度变得更快更可靠，而且编译技术也已能使得高级语言的翻译足够有效，使得高级语言能够使用于实时系统。然而还没有一个能够满足嵌入式系统需求的合适的高级语言。于是，在1974年，美国陆军、海军和空军各自独立地建议开发一个高级语言，以便在所属部门中使用，并且各自为此设立了一个研究项目或一个语言设计计划。

1975年1月，国防研究和工程委员会（DDR&E）主任Malcolm Currie提出只使用一个通用高级语言的建议。根据此建议，建立了一个联合机构——高级语言工作组（HOLWG）。高级语言工作组以美国空军中校William Whitaker为主席，包括各军种代表、美国国防部其它机构

以及联合王国、联邦德国和法国的联络官员。高级语言工作组的任务是：

▲ 确定美国国防部高级语言的需求；

▲ 针对这些需求对现有语言作出评价；

▲ 推荐采用或实现这一程序设计语言的最小集合。

为了保证高级语言工作组的工作不受干扰，与此同时，国防研究和工程委员会中止了所有其它新语言的研究和开发工作。

二、需求定义阶段

确定美国国防部的需求是高级语言工作组的首要任务。

1975年4月，高级语言工作组向军队各部门、其它联邦机构、工业界和其它学术团体公布了"稻草人"（STRAWMAN）需求文件，并向欧洲计算机界的一些经过选择的专家征求了意见。

1975年8月，再次广泛地公布了经过修改了的"木头人"（WOODENMAN）文件。

1976年1月，根据评审的官方反应，产生并公布了"锡人"（TINMAN）需求文件。这是一整套完整的需求文件，它说明了美国国防部对高级语言所期望的性能。

所有这些文件都由代表85个美国国防部各组织、26个工业承包单位、16个大学及7个其它组织的二百余人进行了精心的修改。

大多数组织都期望着能产生一个支持现代软件方法论，并具有实时控制和异常处理结构的高级语言。

1976年4月，美国国防部发布了题为"重要国防系统中计

算机资源管理”的5000.29号命令。它要求在国防系统中使用美国国防部批准的高级语言，除非能证明其它语言对某种特殊应用更节省费用。此外命令中还规定只能由美国国防部来审核批准采用何种语言。该命令的执行结果是限制汇编语言用于国防系统，而鼓励使用已经证明是相当成功的高级语言。由于只能使用经过批准的语言，使得语言数目增加的趋势受到遏制。

1976年11月公布了一个过渡性的高级语言的清单。

不久，高级语言工作组在康奈尔大学主持了一个国际专题讨论会，公开讨论了对这种语言的技术需求。并以“锡人”为准绳，开始对现存语言进行了正式评审。

评审组正式评审了FORTRAN，COBOL，PL/1，HAL/S，TACPOL，CMS_2，CS_4，SPL/1，JOVIAL J3，JOVIAL J73，ALGOL60，ALGOL68，CORAL66，Pascal，SIMULA67，LIS，LTR，RTL/2，PDL2，EUCLID，PEARL，MORAL，EL/1等23种不同语言。此外，还研究了另外一些语言，以了解它们是否具有所期望的最后语言需要采纳的一些特点。

这些语言被区分为三种不同类型：

▲ 不适合的，过时或在问题域之外的语言；

▲ 并非不适合的，含有适合问题域或某些性质的问题；

▲ 可接受的。

评审组写出了长达2800页的评述报告。报告的结论是：

▲ 现存语言都不适合作为美国国防部嵌入式计算机系统的通用高级语言；

▲ 希望只使用一种语言；

▲ 寻求满足所有需求的一种新语言是可行的；

▲ 新语言应在适合的基础上开发。

尽管如此，评审组还是推荐Pascal，ALGOL和PL/1作为基础语言是合适的。

在完成评审工作的基础上，于1977年1月推出“铁人”（IRONMAN）文件。实际上“铁人”在内容上与“锡人”并无太大差异，仅仅是使用了另一种新的格式。

在嵌入式计算机资源管理委员会的要求下，以Barry Deroze为主席，从1977年1月到11月进行了两组独立的经济分析，以确定基于“锡人”需求开发一个新语种的可行性。两组分析的结果是肯定的，他们指出采用一种通用高级语言每年将为美国国防部节约几亿美元。

根据这些报告的结论，管理控制委员会命令高级语言工作组开始开发一种通用高级语言。工业界的刊物当时把这个语言称为DOD_1（DOD为美国国防部缩写）。国防高级研究课题管理局（DARPA）负责签订语言设计合同，William Carlson被任命为政府方面的课题负责人。

三、设计阶段

程序设计语言的设计总带有点艺术性：它以理论为基础，但又包含有一些主观的设计决策。美国国防部所关心的是：这种新语言的设计必须是高质量的，因为它将成为通用标准。而且若被社会接受，国防部也会从中得到巨大的效益。

基于以上原因，美国国防部决定进行国际范围的设计招

标，数支设计组将为评审提交设计方案，在这些方案中再挑选出少数几个设计组为最后评审完成详细设计方案。作为高级语言工作组政策的延续，国防高级研究课题管理局也将全部设计过程公开化，邀请美国国防部、美国工业界和各大学，欧洲工业界和各大学，以及联合王国、法国和联邦德国政府代表参加评审，同时把设计标准交给计算机界的专家进行严格的评审，以确保最终产品的质量。国防高级研究课题管理局使用这种战略，是希望促进这个体现了一致观点的语言的开发。

1977年4月，寻找申请投标的公告（RFP）在国际范围公布，征集对这种新的通用高级语言的设计方案。

1977年10月，高级研究课题管理局从收到的17份建议书中挑选出4个承包单位进行为期6个月的开发。同时公布了经过修订的“铁人”文件。由此，语言设计工作进入第一阶段。

每个方案接收后，用颜色作为标记，然后将每个建议书中任何带特征的标记全部去掉，以使评审者的注意力不致于从技术问题上转移开。该4个方案是：

- ▲ Softech　　(蓝)马萨诸塞州
- ▲ SRI International　(黄)加利福尼亚州
- ▲ Intermetrics　　(红)马萨诸塞州
- ▲ Cii Honeywell Bull　(绿)Honeywell在法国的子公司

值得注意的是，这4种方案均将 Pascal 作为基础语言。

第一阶段于1978年2月结束。2月至3月，来自世界范围近400名志愿者组成80个评审组，对这些方案进行评审，选中（红）、（绿）两个设计方案进行进一步细化。方案设计工

作由此进入第二阶段。

鉴于一个新语言必须与一个高质量的支持系统相联系，在第二阶段设计的同时，高级语言工作组组织了一部分力量研究新语言的设计环境。

1978年初，散发了"沙人"（SANDMAN）需求文件，提出了关于程序设计环境的技术要求和管理上的问题。并对环境问题进行公开讨论，形成"卵石人"（PEBBLEMAN）文件。

1978年6月，公布了"钢人"（STEELMAN）需求文件，弥补了第一阶段评审中发现的不足。

1979年3月，第二阶段设计工作完成。开始对红色语言、绿色语言进行评审，以确定语言设计的最后选择。评审工作由3月延续到4月，评审结果，高级语言工作组宣布绿色语言中选。

竞争的获胜者来自欧洲，主要设计者是法国人Jean D. Ichbiah。整个评审过程的基本标准是技术质量和非政治动机。

由于DOD_1过于军事化，非国防部门可能会轻视它，因此高级语言工作组从来没有接受过将它作为新语言的名称。在1979年的春天，海军后勤司令部的Jack Cooper提议，为纪念Augusta Ada Byron. Lovelace伯爵夫人，将新语言定名为Ada。

Ada Lovelace（1815—1851）是英格兰诗人Jord Byron（拜伦勋爵）的女儿，是一位数学家。她曾经和Charles Babbage一起研究过差分机和分析机，由于较早地研究了计算机的潜在能力而知名。特别是Ada提出了Babbage的机器

可以象Jacquard织布机一样编程．因而公认她是世界上第一位程序员。

经国防部副部长和Jovelace的继承人Lytton伯爵之间的正式信件来往，Lytton伯爵同意使用这个名字

1979年5月，Ada语言正式成为美国国防部通用高级语言的官方名字。

四、测试阶段

测试阶段的目的是检查软件的执行是否满足规格说明中的需求。为此，公开邀请自愿者把一个现有的系统用Ada语言实现。作为交换，这些志愿者可以使用一个Ada语言测试翻译器以及参加下列5个训练班中的任一个，海军研究生院学习班、空军研究院学习班、西点军校学习班、乔治亚技术学院训练班及英国台丁顿国家物理实验室训练班。

在此期间，军方又委派David Fisher代替 Whitaker 为高级语言工作组主席。

1979年秋，高级语言工作组又建立了一个Ada语言配置控制委员会，以控制语言的任何变化。

至1979年11月，有15个国家提交了五百多份语言报告。同月，高级语言工作组公布了"石人"（STONMAN)—Ada语言环境需求方案，以作为Ada语言程序设计支持环境（APSE）的基础。基于第三阶段评审工作中制定的各种报告，Ada语言设计组据此报告纠正了初步设计中的不足。

1980年7月，经过修订的Ada语言参考手册交到高级语言工作组，同年8月获得批准，与此同时，David Fisher辞去高级语言工作组职务，由Bill Carison 接替。这标志着语

言设计和测试工作的结束。

五、运行与维护阶段

1980年12月在波士顿召开的第一次ACM Ada语言讨论会上批准通过建立Ada语言工程联合办公室（AJPO），以管理有关Ada语言的所有活动，并撤销高级语言工作组。由Larry Druffel中校担任Ada语言工程联合办公室主任。同一天批准设立MIL_STD1815作为已批准的美国国防部Ada语言标准。

Ada语言工程联合办公室的工作使得Ada语言有次序地被推广到美国国防部的软件开发中。为防止派生出Ada语言子集和超集，于1981年1月申请将Ada语言作为美国国防部商标，以强迫执行禁用方言的政策。并规定各军事部门的责任是把Ada语言作为标准语言，在80年代中期所有嵌入式系统中止使用任何其它语言。

1983年2月17日，Ada语言标准获得美国国家标准协会（ANSI）批准。美国国防部相继宣布，自1984年2月以后，所有军用软件一律用Ada语言作为开发工具，否则不予承认。

绿色语言的主要作者是法国的Jean D.Ichbiah，设计组的其它成员包括法国的J.Heliard、O.Roubine和J.Abrial；美国的P.N.Hilfinger和H.F.Ledgard；联合王国的J.G.P.Barnes、B.A.Wichmann、M.Woodger和R.Firth；联邦德国的B.Krieg-Bruckner。他们为Ada语言的设计贡献了自己的全部心血和精力，他们的功迹是令人难以忘怀的。

Ada语言的开发过程体现了世界各国几千名计算机科学

家的辛勤劳动和智慧，以至所有参与者的名单看起来就如同计算机界的名人录。她不仅体现了现代软件的开发原理，而且将这些原理付诸实现。她的诞生使我们有了一个能够有效地建立软件系统的统一的工具。

我国计算机科学界和军事科学界一开始就对Ada语言的研制给予了十分的关注，并进行了广泛的评论和深入的研究。普遍认为Ada语言具有强类型、并行处理、异常处理、类属定义、信息隐藏等多种其它语言所不具备的特性，对改善软件系统的清晰性、可靠性、有效性及可维护性提供了强有力的开发工具。她既适用于大型、实时、嵌入式系统，也适用于其它应用系统；既可用她开发系统软件，又可用她开发应用软件。纷纷预测，在20世纪末叶，Ada语言将成为世界范围的通用语言。

目前，Ada语言在我国已经引起广泛的注意，而且已有几个实验性子集问世，现正酝酿着开发我国自己的Ada语言标准，一个学习、研究、应用Ada语言的热潮正在软件产业中兴起。

第二节　Ada语言程序结构及其特点

Ada语言程序由基本程序单元组成，基本程序单元有4种类型：

(1) 子程序subprogram

它是这样的单元：它的执行要由主程序调用引起。子程序有两种形式，过程和函数(procedure & function)，其差别在于，过程定义了要机器执行的一系列动作；而函数则定

义了一个带返回值的计算。

(2)程序包package

程序包定义了逻辑上相关的一组实体。如：它将一组相关的子程序从形式上封装在一个程序包中，需要时又可从程序包外面进行调用。这为程序单元的组织提供了很大的便利。

(3)类属程序单元generic

Ada语言是强类型的，程序中的每一个数据的类型必须预先说明。而Ada语言则提供了支持强类型特征的类属程序单元——类属子程序或类属程序包。

类属单元定义了一个类型未经具体说明、通用的工作模块。该模块不能直接使用，但将其类型具体化后，就可象其它单元一样被引用。这样，一个类属单元可根据不同的使用要求被实化为许多类型不同的程序单元，使大型程序的编制、修改变得更容易。

(4)任务task

任务定义了一个可以与其它程序并行运行的程序单元。

这种处理方式使得多个任务既可在多个处理器上同时运行，又可在单个处理器上交替运行。

程序包、类属单元、任务是Ada语言所特有的程序组织形式。

每一个Ada语言程序单元，其结构均可大致分为程序说明和程序体两个基本部分。如：Eg1.

```
1  Package Body EG1 Is          --程序单元名是EG1;
2          Length,Broad,Area:Integer Range 0..Inte-
           ger'Last;
         --定义长度、宽度、面积为整型变量;取值范围0至
```

整型数上界；

```
3     Begin                                    --程序体开始；
4     Put("Input Length,Broad:");              --题示赋初值；
5     Get(Length); Get(Broad);                 --由键盘赋值；
6     Area:=Length *Broad;                     --计算面积；
7     Put(Area); New_Line;                     --输出面积并换行；
8     End EG1;                                 --程序EG1结束；
```

Ada语言程序并无行号，为便于阐述问题才加上的。

▲ 1—2行为说明部分，3—8行为程序体部分。说明部分通常包含程序名说明、数据类型说明、程序单元的规格说明等；程序体从Begin开始，到End结束。

▲ 当一个语句结束时，需用分号做结束符；

▲ 一个语句行可写一个语句，也可写多个语句；

▲ 可在程序中插入注释，在注释前需加"—"符号；

▲ 程序主体中出现的变量名，必须在相应声明部分经过声明。

第三节　Ada语言程序的编译

本节主要介绍在IBM PC/XT机上进行程序编辑、编译、连接与运行的一般过程。

.PKG	为Ada 语言主程序源文件后缀或程序包体源程序后缀
.JRL	为 * .PKG 文件经 JANUS 编译后的中间文件的后缀
.LIB	为程序包声明部分源程序后缀

.SYM　　为＊.LIB文件经JANUS 编译后的中间文件后缀

.COM　　为经 JLINK 连接后生成的可执行文件的后缀

(一) 编辑

程序的编辑可通过EDLIN, WORD STAR, TURBO, dBASE编辑程序等多种途径实现。这里介绍用WORD STAR编辑的方法。

(1) 进入菜单调用

键入WS后回车

(2) 进入编辑状态

按下 N键，待屏幕出现文件名？提示后键入你要编辑的程序名(后缀为.PKG或.LIB)，并按序键入程序

(3) 退出编辑状态

按^K^D键，即先按下Ctrl+K键，尔后按下Ctrl+D键。或者按F1键则存盘后退出编辑状态。

按^K^Q键，或者按F2键则放弃已经编辑的程序，并退出编辑状态。

(4) 退出WS菜单

若在编辑状态，则按下 ^K^X 键

若在菜单调用状态，则按下 X 键

(5) 移动光标

利用小键盘上的控制键进行，按键上箭头标示的方向为光标移动的方向

↑　↓键分别使光标向上或向下移动一行

← →键分别使光标向左或向右移动一个字符位置

Pg Up 键使屏幕向前翻卷一页

Pg Dn 键使屏幕向后翻卷一页

(6) 删除字符

^G　　删除光标所在位置的字符

Del　　键删除光标左面的一个字符

^Y　　删除光标所在行的全部内容

(7) 插入

Ins　　该键被按下次数为奇数时，为插入状态；被按下次数为偶数时，为退出插入状态

(二) 编译

目前可在PC机上使用的Ada语言版本之一是Janus/Ada。编译程序为Janus

键入

JANUS 文件名.PKG 用于主程序体文件

或　JANUS 文件名.LIB 用于程序包声明文件

(三) 连接

用于连接的程序是JLINK

键入

JLINK 文件名 连接后生成*.COM文件

(四) 运行

键入　　文件名 回车

要点

(1) Ada语言的研制是为摆脱软件危机提出的，她的研制经历了分析、需求定义、设计、测试、运行和维护五个阶段。

Ada语言本身没有解决危机，但由于她为软件工程实践提供了一个良好的工具，利用她能使问题得到缓解。

（2）Ada语言体现了现代软件开发原理，是一个面向问题的语言。她具有强类型、并行处理、异常处理、类属定义、数据抽象、信息隐藏等多种其它语言所不具备的特性。所有这些强有力的功能使得她有可能代替现存的所有语言。因而有人预测：Ada语言将是自动程序设计语言到来前的最后一个语言，

Wegner 提出过一个公式：　成就＝设计＋实现＋权势＋需要

Ada语言的崛起，就在于她在各方面都获得了成功。

（3）Ada语言基本程序单元有子程序、程序包、类属单元、任务等四种基本类型。其中子程序具有过程和函数两种形式。

（4）Ada语言程序要经过编辑、编译、连接三个阶段后才能运行。

编辑的结果是生成可读的源程序，即Ada语言程序；

编译的结果是将源程序翻译成机器代码模块，即目标码模块；

连接是将目标码模块拼接成可执行程序，即命令程序。

习题

（1）为什么要研制Ada语言？她是根据什么研制的？

（2）Ada 语言有什么特点？我们为什么要推行她？美国军方经历了一个学习、推广、强制执行的过程，你说我们会吗？

（3）将你所熟悉的语言与Ada语言比较一下，她们之间有哪些区别和共同点？

（4）上机操作实习，对Eg1进行编辑、编译、连接处理，尔后进行运行。运行时请你分别键入正整数、负整数、小数试一试，根据运行结果你体会一下Ada语言的强类型要求表现在哪些方面？

第二章　Ada语言基本成分

第一节　基本符号

一　基本有效字符集

1.大写字母

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

2.数字

0 1 2 3 4 5 6 7 8 9

3.特殊字符与名称

" 引号　　# 井号　　& 与号

' 撇号　　(左括号　　) 右括号

* 星号、乘号　　+ 加号　　- 减号、负号、连字号　　/ 除号、斜杠　　: 冒号　　; 分号　　< 小于　　= 等于　　> 大于

_下划线　　| 竖杠　　, 逗号

. 小数点

4.空格符与控制符

空格符　　定义一个空格

控制符　　回车、换行、换页、制表符等

二、扩充字符集

1.小写字母

a b c d e f g h i j k l m n o p q r s t u v w x y z

2.其它特殊字符与名称

!	惊叹号	$	美元符	%	百分号
?	问号	@	单价符	^	音调号
〔	左方括号	〕	右方括号	\	左斜杠
{	左花括号	}	右花括号	⊢	抑音号
~	代字号				

三、分隔符

分隔符是用以区分程序中任意两个相邻对象（如：数据、单词、标识符等）的符号。分隔符可以是空格符、格式控制符或一行结束符。

- ▲ 除在注释、串直接量或空格字符直接量中以外，空格符均为分隔符；
- ▲ 除横向制表符外，其它格式控制符总是分隔符；
- ▲ 除在注释中使用外，纵向制表符是分隔符；
- ▲ 一行的结束总是分隔符；
- ▲ 在任意两个对象之间，至少要有一个分隔符。

四、定义符

定义符也被译为定界符，它们在语言中已被赋予确定的含义（如：代表某项运算、测试或定义某个关系属性等）。

1.由下列单字符确定的定义符

& ' ( ) * + — .

. , ; / < = > |

2.由下列组合字符确定的定义符及名称

>= 大于等于　　<= 小于等于　　/= 不等于

:= 赋值号　　.. 选择符　　=>箭头符

《 左标号符　　》 右标号符　　() 空合符

** 指数运算符

五、保留字

保留字是在语言中已被赋予确定含义的词汇，它们已经具有特殊的意义，用户不能再对其另行定义作为它用。这是用户定义的禁区，否则将会发生错误。

Ada语言保留字有63个

Abort Abs Accept Access All And Array
At
Begin Body
Case Constant
Declare Delay Delta Digits Do
Else Elsif End Entry Exception Exit
For Function
Generic Goto
If In Is
Limited Loop
Mod
New Not Null
Of Or Others Out
Package Pragma Private Procedure

Raise Range Record Rem Renames Return Reverse
Select Separate Subtype
Task Terminate Then Type
Use
When While With
Xor

六、字符替换约定

下列情况下对竖杠、井号、引号等基本字符进行替换是允许的：

▲ 用做定界符时，可用惊叹号！代替竖杠｜；

▲ 某一基直接量中的井号＃可用冒号：代替，但若＃成对使用时，必须都替换；

▲ 字串直接量两端的引号"可用百分号％代替．此时串中的原有的一个百分号％必须改为两个百分号（如％％），其原意才不改变。

注：

若键盘字符够用时，不提倡进行字符替换。

有的键盘竖杠符是间断的（如｜），也不需替换。

七、标识符

标识符是一个字符序列，是用来表示常量、变量、类型、过程、函数、程序包、任务等名称的标记。

标识符的构成必须符合以下规则：

(1) 标识符可由一个或多个字符组成，但必须以字母开

头；

（2）字母后面可以是字母、数字组成的字符序列；

（3）下划线可以出现在标识符中间，但不能用其开头或结尾；

（4）标识符长度（即场宽）未作限制，大小写不区分，具有相同含义。

注：

一个好的程序，必须恰当选择标识符，并赋予其实际意义，以增强可读性。一般，若程序较短，或使用的标识符数量不多，或某个标识符在程序中出现的次数很少，可用单个字母作标识符；否则最好根据其含义，用英文或汉语拼音作标识。

第二节　基本数据类型与运算操作

数据是计算机程序处理的主要对象。要实现对数据的处理，必须在程序中解决两个问题：(1) 对要处理的数据进行描述；(2) 对数据实施的操作进行描述。

Ada语言具有极丰富、复杂的数据类型。按其特点大致可分为四种不同类型，

（1）基本类型：整型、实型、字串型和布尔型；

（2）自定义类型：枚举型、子类型、派生类型；

（3）构造类型：数组型和记录类型；

（4）专用类型：私有类型和访问类型。

本节仅讨论基本类型，其它类型将在后面有关章节讨论。

一、数据直接量

数值常量直接说明了数值的内容，因此，该种形式的量被称为直接量。它有四种类型：

- 整型直接量
- 实型直接量
- 字串直接量
- 布尔直接量

1.整型直接量

整型直接量包括正整型数、负整型数和整数零。它不带小数部分，也不带小数点。

如：　-324　　0　　+324　　正确

　　-324.0　0.0　+324.78　错误

整数表达数据的范围一般因计算机的字长而异。例PC机字长为16位，表达范围为　　-32768～+32767。

2.实型直接量

实型直接量包括正实数、负实数和实数零。

如：　3.1415926　　0.0　　-21.10　　正确

　　3　　.0　　-21.　　不正确

▲ 整数与实数都可用指数形式表示，指数部分必须是整数

例：

整数　1900　可表示成　19E2　指数不能带符号

实数　6371.14　可表示成　6.37114E+3　或

637114.0E-2

▲ 当整型直接量或实型直接量的位数较长时，允许用下划线进行分隔，以提高可读性。

例：

1_978　　3.14_1592_6

▲ 数据直接量还可根据需要用有基直接量表示，其格式为

〈基数〉#〈有基整数〉[.〈有基整数〉] [#〈指数〉]

例：

2#1111_1111 16#FF# 016#0FF# —值为255的整型直接量

16#E#E1　　2#1110_0000 —值为224的整型直接量

16#F.FF#E+2　2#1.1111_1111_111#E11 —值为4095.0实型直接量

3.字串直接量

字串直接量被分为字符直接量与串直接量。

▲ 字符直接量由基本字符集中的任意一个，而且只能是一个字符确定。

字符直接量要用一对单撇号将其引起。如：

'A'　　'*'　　'/'

▲ 串直接量是一个由基本字符集中的字符组成的字符序列。

串直接量要用一对双引号将其引起。如：

"Mcssage of the day"

"　　" —一串空格

由于每一行的结束均有一个行结束符，因此，同一串直接量必须出现在同一程序行中。倘若串直接量较长，一个程序行的场宽不够用时，可通过连接符&对多个短串直接量进行连接的方法来获得。如：

"FIRST PART OF A SEQUENCE OF CH-

```
ARACTERS" &
"THAT CONTINUES ON  THE NEXT
LINE"
```

等价于

```
"FIRST PART OF SEQUENCE OF CHARACT-
ERS THAT CONTINUS ON THE NEXT LINE"
```

4.布尔直接量

布尔直接量取TRUE(真)与FALSE(假)两个值。布尔量的运算结果仍是布尔量。作用于布尔直接量的运算操作有：

(1)逻辑运算

AND	与运算	OR	或运算
XOR	异或运算	NOT	非运算

(2)比较操作

布尔量TRUE的机器值是1，FALSE的机器值是0，所以，两个布尔量可以进行比较。

所有的比较运算符均可用于布尔量运算。

二、变量说明

所谓变量是指其值在运算过程中允许被改变的量。Ada语言的强类型要求之一就是所有在程序中使用到的变量必须在程序的说明部分加以说明。这种说明过程有时也被称为变量设置。

变量的说明内容可以简单地分为基本说明与分量说明两个部分。

基本说明内容在说明中是不可缺少的，主要有：

(1)变量标识符.

(2)变量取值类型。其基本类型有：

integer	整数类型
float	浮点类型
boolean	布尔类型
character	字符类型
string	字串类型

分量说明内容在说明中可根据实际需要进行选择. 这类说明是在基本说明的基础上进行的. 其作用主要用以说明变量取值的数量关系.如初值、变量取值的范围(上、下界)等.

(一) 基本说明

*基本说明格式

〈变量标识符〉[,〈变量标识符〉]，〈类型说明关键字〉;

1.整型类型说明格式

〈变量标识符〉[,〈变量标识符〉]: INTEGER;

```
X: Integer;              --定义X为整数型变量
Y,Z: Integer;            --定义Y,Z为整数型变量
```

2.浮点类型说明格式

〈变量标识符〉[,〈变量标识符〉]: FLOAT;

```
CELSIUS_TEMP: Float;     --定义摄氏温度为浮
                           点类型变量
CELSIUS_TEMP, FAHRENHEIT_TEMP:Flo-
                                    at;
        --定义摄氏温度、华氏温度为浮点类型变量
```

3.布尔型变量说明格式

〈变量标识符〉[,〈变量标识符〉]: BOOLEAN;

A：Boolean；　　一定义A为布尔型变量

B,C：Boolean；　　一定义B，C为布尔型变量

注：布尔型变量仅取两个值TRUE（真）、FALSE（假）。

4.字符型变量说明格式

〈变量标识符〉[,〈变量标识符〉]：CHARACTER；

C1：Character；　　一定义C1为字符变量

C2，C3：Character；　　一定义C2，C3为字符变量

5.字串型变量说明格式

〈变量标识符〉[,〈变量标识符〉]：STRINT；

S1：Strint；　　一定义S1为字符变量

S2，S3：Strint；　　一定义S2，S3为字符变量

（二）分量说明

分量说明分为范围说明与初值说明，当两个说明并列时，应先说明范围后说明初值。

说明格式

〈基本说明〉[〈范围说明〉][〈初值说明〉]；

范围说明格式：[RANGE〈上界〉..〈下界〉]

初值说明格式：[:=〈表达式〉]

eg：

SIZE：Integer：=0；　　--SIZE初值取0

COUNT：Integer Range 0..1_000：=0；

--COUNT整型，取值范围下界为0，上界为1000，初值为0

SORTED：Boolean：=FALSE；

--SORTED布尔型，初值为FALSE（假）

三、常量说明

这里所讲常量系指用标识符命名，且其值在运算过程中不允许发生变化的量。凡是在程序执行部分要用到的常量，与变量一样必须在说明部分加以说明。

常量由常量名、常量类型和常量三个要素构成。常量说明亦即对该三个要素的说明。

常量说明格式

〈常量标识符〉[,〈常量标识符〉]：CONSTANT〈类型说明〉：=表达式；

eg：

PI：Constant float：=3.141_592_6；

——定义常量PI浮点型，值为3.1415926

N：Constant integer：=1；

SUM_N：Constant integer：=N+10；

——定义常量N整数型，值1；定义常量SUM_N整数型，值11（N+1）

——因SUM_N值与N值相关，这样定义的好处是，修改程序时仅需修改N即可

四、运算操作

Ada语言的强类型属性突出地表现在如下两个方面：

- 赋给变量的值必须与变量类型一致；
- 对任何数值、变量所施加的操作必须与其所属类型要求相吻合。

因此，变量类型一经定义，那么能对其施加的操作也就

相应地被隐含说明了。

Ada语言定义了六个不同运算优先级别的运算操作，按优先级由高到低的顺序为：

```
**  |  ABS | NOT          —— 最高级运算符
*   |  /   | MOD | REM    —— 乘法类运算符
+   |  -                  —— 单目加法类运算符
+   |  -   |  &           —— 双目加法类运算符
=   | /=   |  <  |        —— 关系运算符
<=  |  >   |  >=
AND |  OR  |  XOR         —— 逻辑运算符
```

所谓单目运算符是指该运算符仅作用于一个操作数。如：

```
+23                  —+23取正，结果不变
-23                  —-23取负，结果符号相反
```

所谓双目运算符是指该运算符作用于两个操作数。如：

```
130 * 10     —130乘10，130为左操作数，10为右操作数
```

(一)基本运算符的有效使用规则

1　单目运算符

运算符	运　算	操作数类型	结果类型
+	恒　同	任何数值型	同操作数
-	取　负	任何数值型	同操作数
ABS	绝对值	任何数值类型	同操作数
NOT	逻辑非	任何布尔类型	同操作数

2　操作数同型的双目运算符

运算符	运　算	左操作数类型	右操作数类型	结果类型
+	加	任何数值型	同左型	同左型

-	减	任何数值型	同左型	同左型
MOD	取模	整型	同左型	同左型
REM	求余	整型	同左型	同左型
=, <, <=	关系	任何型	同左型	布尔型
/=, >, >=	运算			
AND	与	布尔型	同左型	布尔型
OR	或	布尔型	同左型	布尔型

3 操作数不同型的双目运算符

运算符	运算	左操作数类型	右操作数类型	结果类型
*	乘	整型	整型	整型
		浮点型	浮点型	浮点型
		定点型	整型	定点型
		整型	定点型	定点型
		定点型	定点型	定点型
/	除	整型	整型	整型
		浮点型	浮点型	浮点型
		定点型	整型	定点型
		定点型	定点型	定点型
**	取指数	任何整数类型	INTEGER	同左
		任何浮点类型	INTEGER	同左

注：

a 浮点型数指数运算其指数可取负数；

b 整数型数指数运算其指数不可取负数，否则引发异常。

(二) 运算关系表

1.逻辑运算真值表

A	B	A and B	A or B	A xor B
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE

2.整数除、REM与MOD间的关系

A	B	A/B	A rem B	A mod B
10	5	2	0	0
11	5	2	1	1
12	5	2	2	2
13	5	2	3	3
14	5	2	4	4
10	-5	-2	0	0
11	-5	-2	1	-4
12	-5	-2	2	3
13	-5	-2	3	-2
14	-5	-2	4	-1
-10	5	-2	0	0
-11	5	-2	-1	4
-12	5	-2	-2	3
-13	5	-2	-3	2
-14	5	-2	-4	1
-10	-5	2	0	0
-11	-5	2	-1	-1
-12	-5	2	-2	-2
-13	-5	2	-3	-3
-14	-5	2	-4	-4

注：

. 对于A和B，A/B是商，Arem B是A被B除时的余数；

. REM满足下列关系： A rem(-B)=A rem B

(−A)rem B=−(A rem B);

. 对于任一整数K，MOD 满足下列关系 A mod B=(A+K*B)mod B。

（三）类型的转换

在数值之间允许进行类型转换，转换格式为

〈类型说明符〉(〈表达式〉)

. 整型转换为浮点型 Float (23)

. 浮点型转换为整型 Integer (12.4)

（四）表达式

表达式是由直接量、变量、常量、函数及运算符号组成的运算式。

直接量、变量、常量、函数也可作为表达式的特殊形式看待。

表达式的组成规则必须满足下述两个基本要求：

(1) 各操作对象的数据类型必须相同；

(2) 对操作对象实施的操作必须与其类型属性相吻合。

要点

(1) Ada语言数据量有三种表达形式

. 直接量

. 常量

. 变量

其中常量、变量是用标识符表示的量。标识符由字母、数字和下划线组成。标识符必须以字母开头，下划线不能在标识符开始或结束位置出现。

程序执行部分要用到的常量或变量，必须在说明部分进

行说明。应尽量赋予标识符以实际意义，以提高可读性。

（2）变量一经说明，则能够加于其上的操作也就被相应地说明。因此，在组成表达式时，不但各个操作数的类型要相同，而且所实施的运算操作也必须与操作数的属性相一致。

（3）程序中的错误可分为三种情况：

a　语法错误

这一类错误系指在程序的组织结构与语句组成方面存在不符合语法要求的地方。编译时发现的错误，大多属于这一类。

b　运行错误

编译、连接都能通过，但没有运行结果；运行时可能因出错而挂起；也可能因出错而中途中断。该类错误常因数据溢出、超出函数定义域引起。

c　逻辑错误

运行有结果，但结果有错。该类错误主要由赋值错误，运算关系混乱引起。

对于一个不太熟练的程序员来说，百分之七十的错误发生在语法错误上，因此，正确地理解、记忆一个语言的基本成分，对于提高编程质量，减少程序错误，是大有益处的。

（4）Ada语言子集本身虽未带函数库，但可通过调用C、Pascal等高级语言程序函数库的方法来使用函数。

PC/XT机Ada语言子集的函数使用是通过调用8087函数来实现的，与此有关的内容将在函数一章中再作具体介绍。

习题

（1）指出下列数哪些是合法的？哪些是非法的？为什么？

a．1_0.6　b．10_.60　c．.5023　d．e+3
e．89e　f．0．　g．'a'　h．+1

（2）判别下列表达式哪些是合法的？哪些是非法的？

a．3**2**3　b．(3**2)**3　c．A/B/C
d.(A/B)/C

第三章　Ada语言的基本语句

语句定义要执行的动作，基本语句包括赋值语句和程序执行流程控制语句。

第一节　赋值语句与输入/输出调用

赋值可用赋值语句或输入调用完成。

在Ada语言中，输入/输出是通过过程调用实现的。将在输入/输出一章中加以叙述。在这里仅进行简要的使用说明，以求得编程实践和教学内容的协调。

一、赋值语句

赋值语句用一个由表达式确定的新值代替变量的当前值，指名的变量和右边的表达式类型必须相同。

赋值语句语法格式

〈变量名〉：＝〈表达式〉；

若下列举例中的变量都已经正确说明，则这些赋值语句均会是有效的：

```
INDEX: =0;                          --赋初值
TODAY: =Tue;                        --赋星期几
DISCRIM: =(B**2-4.0*A*C);           --赋实数值
SYMBOL_FOUND: =False;               --赋布尔初值
```

```
SYMBOL_FOUND: =A And B;          --赋布尔值
SYMBOL_STRING: ="FAHRENHEIT TEMP";
                                 --赋串值
SYMBOL_LENGTH: =Size(TOKEN);     --赋函数值
```

赋值语句由变量名、赋值号、表达式与分号组成。变量名指明给什么对象赋值，表达式说明要赋的值是什么？

：＝称为赋值号或赋值运算符。它所定义的执行动作是将其右边表达式的运算结果赋给左边的变量。

表达式可以是常量、变量与函数，也可以是算术表达式、布尔表达式。

赋值语句中变量与表达式类型必须相同，否则在编译时将出现错误信息。

二、输入调用

输入调用是一种通过键盘来为变量赋值的语句。该种方式要求变量的类型在程序中说明；变量的内容由用户在程序运行过程中灵活地确定。

语法格式

Get（变量名）；

例：

```
Get(A); Get(B);
... ...
```

该程序执行的结果是，如果键盘没有输入数据，则处于等待状态；如已经输入数据，则把从键盘得到的数据赋给对应变量。

由键盘输入的数据的类型应与预定义的变量的类型一

致。

三、输出调用

为了得到中间运行数据或最后结果，在程序的相应位置应有输出数据的功能。此功能可通过输出调用实现。

输出调用需要说明三个问题：输出对象是什么，以什么格式输出，输出给谁？这里讲的输出已经约定是输出到监视器，所以，无需再说明输送到何处。

语法格式

Put（变量名[.宽度][.小数位数]）；

Put实施与Get相反的过程。

Put是过程名，变量名说明输出对象，Put过程将变量的当前值转换为字符序列输出。

宽度是可缺省参数，用以确定输出项的场宽，该参数为一整形数。若可见字符序列字符个数小于指定场宽，则用前导空格填补不足部分。

四、输出一空行

有时需要将输出内容在新的一行显示。这时可在Put调用前调用New_Line。

使用形式：

New_Line；

eg2：

```
... ...
    A,B,C:Integer;
Begin
  A:=10; B:=20; C:=30;
```

```
        Put(A,4);Put(B,4);Put(C,4);New_line;
        Put(B,4);New_line;
        Put(C,4);
    End ;
```

该程序先在一行输出A、B、C的值，每个数值场宽4位，而后再输出B、C的值，每个数值场宽还是4位，但B、C的值都在新的一行输出。

执行结果

```
10      20      30
20
30
```

五 分程序语句

分程序的作用是，可在程序内部将任一组连续语句行的集合定义为一个程序块。

程序被分块后可进一步改善可读性和变量使用的灵活性。

程序块可以是并列关系，也可以嵌套。

语法规则

```
〔〈程序块标识符〉:〕DECLARE
                    〔〈声明部分〉〕;
                BEGIN
                    〈语句_序列〉;
                END〔〈程序块标识符〉〕;
```

例：

```
1       Package Body T is
2       Begin
3           A1: Declare
```

```
4               I,J,K: Integer:=100;
5          Begin
6               A101: Declare
7                     J,M:Integer:=101;
8          Begin
9        Put(I,6); Put(J,6);Put(M,6):New_line;
10         End A101;
11         A102: Declare
12           K,N: Integer:=88;
13         Begin
14         Put(I,6);Put(K,6);Put(N,6);New_line;
15             Put(M,6);  —错误. 超出M作用域
16         End A102;
17         Put(I,6);Put(J,6);Put(K,6);New_line;
18       Put(M,6);Put(N,6);—错误,超出M、N作用域
19         End A1;
20       End T;
```

运行结果

```
100     101     101
100      88      88
100     100     100
```

说明

1.程序块标识符定义了程序块名，该名可缺省。若有，必须在块的开始和结束部分同时出现；

2.变量作用域由声明处开始，延伸到该块的结束处。在变量声明的作用域内，如变量在某处出现，则说明变量在该处是可见的；如处处可出现，则说明处处可见；

3.同一标识符如在另一部位被再次声明时，若前次声明作用域已经结束，则两次声明的作用域不相重叠，变量在各自作用域内处处可见；若前次声明作用域并未结束，则两次声明的作用域是嵌套的，而且总是后定义的作用域被嵌套在

先定义的作用域中。

在声明嵌套、变量名被重名情况下，同一变量名标识了不同变量。作用域是外层覆盖内层，而可见性是内层覆盖外层，即外层变量与内层变量重名时，在内层变量作用域内，外层变量不可见。

本节例程序由A1、A101、A102三程序块组成。

4行定义的I、J、K变量作用域延伸到19行。其中I变量在其作用域范围内处处可见，J变量在块A101域内不可见，K变量在A102域内不可见。

第二节　条件语句

计算机在执行程序时，一般是按照程序中语句出现的先后顺序逐句地执行。但对于许多较复杂的问题，常要求依据一定的条件执行不同的流程。

例：

西瓜重量3斤以下者，每斤价格0.13元；超过或等于3斤重者，每斤价格0.18元。现要求编一按重量计费的程序。

题意分析：

设西瓜重为WEIGHT斤，价钱为PRICE元。则要根据不同的重量分别计费。即

P＝0.13＊W　　—WEIGHT＜3……①式

P＝0.18＊W　　—WEIGHT＞＝3……②式

执行流程应是：

1. 从键盘输入重量赋给W；
2. 判断W值是否小于3；

若W＜3　　则按①式计费；

若W＞＝3　　则按②式计费；

3.将P值输出到监视器。

条件语句的基本思想是：

如果　条件表达式为真，则执行一组语句序列；否则，执行另一组语句序列或不执行任何语句序列。

条件语句语法格式

```
If  条件表达式  Then
    语句_序列;
{Elsif  条件表达式  Then
    语句_序列;}
〔Else
    语句_序列;〕
Ena if;
```

应用形式一

```
If  条件表达式  Then
    语句_序列;
End If;
```

例，

```
    ......
     T: Integer;                    定义成绩
     M: Integer  :=0;               定义总分,赋初值
   Begin
       ......
       Get (T) ;
   If T>=60 Then
```

```
            M: =M+T;
              Put(M);
        End if;
        ......
```

该例执行结果是：

若输入的成绩达到60分以上，则累计及格分数，并输出总分。否则，什么事也不做。

应用形式二

```
If  条件表达式  Then
      语句_序列;
      Else
      语句_序列;
End if;
```

eg:

```
    ......
      T: Integer;                    —分数
      NJ,J: Integer; :=0;  —NJ,J分别为不及格、及格人数
  Begin
      Get(T);
      If T <60 Then
            NJ: =NJ+1;
         Else
            J: =J+1;
      End If;
      ......
      Put(NJ, 8); Put(J. 8); New_line;
      ......
  End;
```

本例执行结果是：

若输入成绩低于60分，则不及格人数加1，否则及

格人数加1。

应用形式三

```
If  条件表达式一  Then
      语句_序列;
        Elsif  条件表达式二  Then
          语句_序列;
End if;
```

例：

```
        ……
          T: Integer·
          NJ, J: Integer=0;          —NJ, J分别为不
                                       及格、及格人数
Begin
          Get (T);
            If T <60 Then
                NJ: =NJ+1;
              Elsif T <80 Then
                  J: =J+1;
              End If;
          ……
          Put (NJ, 8): Put(J, 8): New_line;
          ……
End;
```

条件语句的执行结果是：

如分数低于60分，不及格人数加1，否则如果分数低于80分，良好以下人数加1。

应用形式四

```
If  条件表达式一  Then
    语句_序列;
    Elsif  条件表达式二  Then
```

```
            语句_序列;
    Else
            语句序列;
End if;
```

例：

```
        ......
          T: Integer;
          NJ,J,L: Integer; =0;        --NJ,J,L 分别为不及格、及格、良好以上人数
Begin
          Get(T);
          If T <60 Then
              NJ: =NJ+1;
          Elsif T <80 Then
              J: =J+1;
              Else
                  L: =L+1;
          End If;
          ......
          Put(NJ,8);Put(J,8); Put(L,8);New_line;
          ......
End;
```

该条件语句的执行结果是：

如果分数低于60分，则不及格人数加1，否则如果分数低于80分，及格人数加1，否则良好以上人数加1。

按照此思想，请读者自己选择一种合适的形式编出按西瓜重量计费的程序。

综上所述，条件语句的基本成分可分为三部分：

条件语句定义符、语句序列和条件。

▲ 条件语句定义符有以下几种基本组成形式：

if…then…end if;

if…then…else…end if;

if…then…elsif…end if;

if…then…elsif…else…end if;

▲ 语句序列则按问题要求的功能进行组织。

▲ 条件表达式的组成：

条件表达式可分为简单条件表达式与复合条件表达式。

1.简单条件表达式

简单条件表达式由算术表达式与关系运算符组成。

形式为：

〈算术表达式〉〈关系运算符〉〈算术表达式〉

例：

```
D>=0.0
A+B/=D*C
```

2.复合条件表达式由简单条件表达式与逻辑运算符组成。基本形式为：

〈简单条件表达式〉〈逻辑运算符〉〈简单条件表达式〉

例：

```
A>B AND C=E
C/=0 OR B**2>D
```

此外，在基本形式的基础上，还可根据需要，组合成更复杂的情况。

如 (COLD and SUNNY) or WARM等

Ada语言还提供了项测试操作和短路控制形式

(1) 项测试操作

项测试操作有IN与NOT IN两种。

它可以测试一个项目的内容是否被包含在另一个项目

的内容之中。

例：

I in 1..N　　—与 I>=1 And I<=N 等价

I Not in 1..N　　—与 I<1 or I>N 等价

(2) 短路控制形式

短路控制有 and then 和 or else 两种形式。它们是对同一布尔类型的两个操作数定义的，产生的结果也为同一类型，短路控制形式总是先计算左操作数。

如果控制形式and then的左操作数算得的结果为FALSE，那么该控制形式的结果即为FALSE，就不再对右操作数进行运算。

如果控制形式 or else 的左操作数算得的结果为TRUE，那么该控制形式的结果即为TRUE，就不再对右操作数进行运算。

在两操作数都进行计算的情况下，and then与and的作用相同，or else与or的作用相同。

条件语句允许嵌套使用，即条件语句内部允许其它条件语句存在。

第三节　情况语句

在条件语句中，根据对一个条件判断的两种不同结果来选择不同的执行路径。实际上，往往要在许多选择中选取一个执行路径。这样用条件语句就显得很不方便了。

情况语句则可根据选择表达式的值，选择所需执行的路径。因此它多用于多分支控制场合。

语法格式：

```
case〈选择表达式〉is
   〔when  选择值=〉语句_序列〕；
  〔when others=〉语句_序列〕；
end case;
```

例：

```
        ……
        N: integer;
Begin
        ……
          Put("Please input N");
              New_line;
              Get(N);
        case N is
              when 0=〉put("星期日");
              when 1=〉put("星期一");
              when 2=〉put("星期二");
              when 3=〉put("星期三");
              when 4=〉put("星期四");
              when 5=〉put("星期五");
              when 6=〉put("星期六");
         when others=〉put("输入数据错误");
      end case;
        ……
end;
```

在该例中，若输入值为0，则打印星期一；

若输入值为6，则打印星期六；

若为其它值，则提示输入错误。

本例可说明，在情况语句中

case…is间的表达式的值，为离散类型。

when…=〉间是选择值。如表达式值与该选择值相同，则该分支被选中。

when others提供的分支，在表达式为任何其它值时被选中。

若在说明部分定义了N的范围在0-6之间，如

N：integer range 0..6；

且选择值包括了N全部值，则可以省去when others选择部分。

情况语句也可根据需要嵌套使用。

第四节　转移语句

现代程序设计的观点总是排斥转移语句的使用，最主要的原因是：过多使用它，会破坏程序的清晰性，让人弄不清程序已经执行到何处，这给程序的判读和解析带来很大的困难，相应地还会给大型程序的检查、调试工作带来麻烦。

但是这并不等于转移语句一无是处，在某些场合，只要用得好，还是有益的。因此，并不奇怪Ada语言还保留有转移语句，但对它的使用确要谨慎。

语法格式

GOTO〈标号名〉；

这是一个无条件转移语句，GOTO定义了在此处要执行转移，至于转移到何处，则由标号名指定。因此，标号必须和GOTO语句结合使用。

Ada语言没有行号，它是通过在程序中设置标号来标记

转移目的地的。

标号格式：

《标识符》

标号由《 》符与标识符组成。《 》定义了标号，标识符定义了标号名。两者必须结合使用。

例：

```
……
X: Integer: =0;
……
《TKL1》              —标识符 TKL1 要用《 》符括起
   X: =X+1;
If X<10 Then
   Goto TKL1          —转移到 TKL1 处继续执行，
                        TKL1不用《 》括起
End If;
Put(X);
……
```

本例的执行过程是：

如X＜10成立，就转移到TKL1处执行加1运算，否则打印X值。

原则地说，标号如同其它高级语言的行号一样，在任一程序行前均可冠以标号，且只要需要，就可以直接转移到那里执行。

从逻辑上讲，是不能随便转移的。由于设计的程序通常是面向问题模块化了的，程序的执行就不能直接转移到循环结构、条件语句内部，也不能转到各个程序单元结构的内层中去，这在使用时要注意。

第五节　循　环

在很多实际问题中，常会遇到许多有规律性的重复运算，因此在程序中就需要重复执行某些语句序列。这种在一定条件下，重复执行某一语句序列的运行方式，被称为循环。这种循环流程控制机制，常用循环语句来实现。

循环语句的作用就是用以组成循环控制结构，来重复运行某一语句序列。

在循环结构中，一组被重复执行的语句序列——即循环对象，叫做循环体；每重复一次，都要决定是继续循环还是停止循环，决定是否要继续循环的条件叫做循环终止条件。在终止条件中，有一个用以控制循环次数的循环控制变量。一个得以正常运行的循环结构通常均包含确定循环开始与终止位置的定义符——循环定义符、循环体与循环终止条件三个部分。

Ada语言中，循环语句有LOOP、FOR LOOP与WHILE LOOP三种类型。

一、LOOP循环

LOOP循环称为基本循环。

语法格式：

```
LOOP
        〈语句_序列〉;
END LOOP;
```

LOOP定义了循环体的开始位置，它也是循环入口；END LOOP标明了循环体的结束位置；处于其间的语句序

列为被执行的循环体。

该种形式循环无脱离循环的终止条件，即无正常的出口。因而它的执行方式是进入循环后，不能结束循环，即所谓进入死循环。

例：

```
        ……
        T: Integer;
    Begin
        Loop
            Get(T);
                T: =T * T;
                Put(T);
        End Loop;
    End;
```

该例循环体是执行输入、自乘和输出。但无循环终止条件，因而无法正常退出循环控制。一经执行，就循环不止——进入死循环。此种循环方式仅用于一些经起动后就不能中止的特殊场合，如自动控制系统等。

死循环的应用场合是不多的，但Ada语言为LOOP循环提供了脱离循环出口，它也适用于其它两种类型的循环，这就给循环方式的应用带来了极大的灵活性。

出口语句语法格式：

EXIT〔〈循环名〉〕〔WHEN〈条件〉〕；

EXIT为循环出口语句定义符；

WHEN〈条件〉为可缺省参数，用以指定脱离循环的条件；

〈循环名〉也为可缺省参数，一般在循环嵌套情况下使用。在多重循环嵌套情况下，它用以具体说明要终止的循环

对象。

应用形式一

EXIT;

EXIT使得程序无条件地脱离循环控制。

例:

```
        ......
        N: Integer;
    Begin
        Loop
            Get(N);
                if n <0 then
                    exit;
                end if;
            Put(N);
        End loop;
    END;
```

该例执行的结果是如果输入一个负数，就结束循环，否则继续循环。

应用形式二

EXIT WHEN〈条件〉;

该形式的执行结果是：若条件成立就结束循环，否则继续循环。

例:

```
        ......
        N: Integer;
    Begin
        Loop
            Get(N) ;
                Exit When N<0;
            Put(N);
        End Loop;
    End;
```

该程序执行结果与上例相同。

```
    Exit When N<0;        语句与下列三行语句等价
If N<0 Then
    Exit;
End if;
```

即如N<0成立，就结束循环。

应用形式三

EXIT〈循环名〉WHEN〈条件〉;

该形式执行的结果是，条件成立就跳出被指定的循环，否则继续循环。

例：

```
……
X,Y,Z: Integer;
……
A1:
    Loop
        Put("输入X"); Get(X);

            Exit A1 When X<0;
        A2:
            Loop
                Put("输入Y");Get(Y);
                    Exit A2 When Y<0;
                Z:=X+Y;
                Put(Z);
            End Loop A2;
    End Loop A1;
……
```

本例题说明如下几个问题：

(1) 循环可以根据需要嵌套使用；

(2) 为了能够准确地说明从哪一个循环跳出，可根据需要给循环定义一个名称；

(3) 一个循环的名称由标识符后加：号组成。其书写位置：应在被命名循环结构的入口语句之前出现，并且在循环语句的末尾也应注明其对应的标号，以增加程序的可读性；

(4) 任何循环出口语句，均只能在被指定循环结构的内部出现，而不应在循环语句的外部出现，即出口仅遵循内转外方式。

(5) 只允许在有名循环中使用带循环名字的出口语句，此出口语句也仅作用于该循环；不带循环名的出口语句仅作用于包含它的最内层循环。

二、FOR循环

语法格式：

```
FOR〈循环控制变量〉IN〔REVERSE〕〈初值〉..
                              〈终值〉LOOP
        〈语句_序列〉;
    END LOOP;
```

其中循环变量名可为任一合法的变量名。它的类型派生于其值域的类型，作用域仅局限于该循环结构，因而该变量在循环结构外部不可被调用(称为不可见)。进入循环后，循环值域仅计算一次，因而循环控制变量值在循环结构内部不可被修改(不能被重新赋值)。

循环变量取值范围称为循环域，该域由初值和终值确定，但终值必须大于初值。若初值大于终值，则被视为空域，此种情况循环体将不被执行。因而通常情况下，控制变量总是以递增的方式变化的。

REVERSE使控制变量按递减的方式变化。这是一个可

缺省的关键字，缺省时为递增方式。

控制变量每变化一次的增量，称为步长。该循环语句的步长总是为1个单位。

初值与终值可以是已经被赋值的任一离散型变量，其值在循环体内部即使被改变，也不能影响循环次数。

例：

(一)

```
For I In 1..10 Loop
    Put(I); Put(" ");
End Loop;
```

运行结果　　1 2 3 4 5 6 7 8 9 10

该形式控制变量是按递增方式变化的。

(二)

```
For I In Reverse 1..10 Loop
    Put(I); Put(" ");
End Loop;
```

运行结果　　10 9 8 7 6 5 4 3 2 1

Reverse使控制变量按递减方式变化。

(三)

```
.....
I: Integer: =100;
For I in A..F Loop
    For I in 1..10 Loop
        Put(I);Put("  ");  —此处输出内层循环
                              循控变量值
    End Loop;
    Put(I);                —此处输出外层循环 循 控
                            变量值
End Loop;
    Put(I);                —此处输出整型变量I的值
.....
```

该例说明：

(1) 参变量的值域可为任一离散的类型；

(2) 循环参数的作用域，从循环参数说明延伸到循环结束语句，仅局限于本循环结构的内部，在循环体中才是可见的，循环结构外部则不可见。

(3) 程序开始处定义的变量，其作用域由定义处延伸到程序的结束处。当外部的变量与内部的循控变量同名时，外部变量在循环结构内部不可见。

(四)

```
   N: Integer: =10;
For I In 1..N Loop
    Put(I); Put(' ');
    N: =5;
End Loop;
```

运行结果 1 2 3 4 5 6 7 8 9 10

此例说明：循控变量的初值、终止值仅在入口时计算一次；在循环体中改变N的值，循环次数并不改变。

(五)

```
For I In 1..1000 Loop
    Null;
End Loop;
```

此例说明：任何循环不能没有循环体，循环体最简单的形式是空语句NULL。此时的运行结果仅是空等待，起时间延迟的作用。

三、WHILE循环

这种循环也被称为当循环。

语法格式：

WHILE〈条件表达式〉LOOP

〈语句_序列〉；

END LOOP；

此处的条件表达式是循环条件。每次执行循环时，首先对表达式进行测试，若结果为真时，执行循环，结果为假时，脱离循环。若条件总是成立，则总是执行循环，就成为死循环了。若一开始就不成立，则循环体一次也没被执行。

若条件中仅含一个变量，则往往需要在循环体中改变该变量的值，才有可能脱离循环，此时的循环控制变量就是一个。

如条件中含有多个变量，则任一个变量值的改变，都可能使循环继续或停止。因此，若条件中的变量有多个，且在循环体中又有多个变量被重新赋值，那么此时的循环控制变量就有多个。

另外，步长可根据需要任意确定。因此当循环给程序员个人编程技巧的使用，提供了极大的余地。

例：

```
      ……
   N: Float: =0.0;
While N<=10.0 Loop
      N: =N+0.1;
            Put(N,4,1);
               End Loop;
```

这个例题首先对条件表达式N<=10.0进行测试，若结果为真继续循环、否则终止循环。需要注意的是，在入口时为使循环条件成立，需先给循环参数赋一适当的初始值。参变量的初值在进入循环前给出，终值在条件表达式中给出。参变量每变化一次所改变的量，称为步长。为避免出现死循环，在程序设计时应正确处理初值、终值与步长三者关系。

第六节　应用举例

通过本章的学习，我们已经掌握了各种语句的功能与使用规则。现在我们进一步研究如何使用它们编制程序的问题。

eg3：

编一求解一元二次方程实数根的程序。

(1) 题意分析

设自变量为X，二次项、一次项系数、常数项分别为A、B、C。则该方程具有如下的一般形式：

$$AX^2+BX+C=0$$

该方程成立的条件是$A\neq 0$

设方程实数根为X1、X2

实数根存在的条件是$B^2-4AC\geqslant 0$

因此程序要实现的功能是输入A、B、C值，输出X1、X2值

(2) 确定算法

1. 设置变量

用到的变量有：A，B，C，X1，X2，D

D变量为中间变量：$D=B^2-4AC$

变量类型均为浮点型

2. 确定数学模型

通用形式解模型为

$X1，X2=(-B\pm\sqrt{D})/(2A)$

3. 程序功能设置

程序应能反复使用，由于A=0方程退化为一次方

程，在定义的问题域之外，因此可考虑将A=0作为终止运行的条件。

D<0时仅有复数解，也不在问题域之内，这时显示“无实根”，尔后回到初始状态，等待重新输入。

(3) 编制程序

程序是算法的另一种表达形式，它更直接地定义了要实现的功能。由于Ada语言源程序具有较好的可读性，因此它可替代框图。

程序的组织必须遵循语言本身已经定义了的各种规则，问题在于如何灵活应用它。

程序实例如下

```
  With Io, Floatio, Mathlib;
Package Body T1 is
  Use Mathlib;
  A, B, C, D, E1, E2, X1, X2: Float;
Begin
  Loop
    io.put("输入A,B,C的值");floatio.Get(A);
    Exit When A=0.0;
          floatio.Get(B);
          floatio.Get(C);
               D:=B*B-4.0*A*C;
    If D>=0.0 Then
       E1:=-B/(2.0*A);E2:=Sqrt(D)
       /(2.0*A);
       X1:=E1+E2; X2:=E1-E2;
       New-line; io.put("方程解");
       io.put("X1=");floatio.put(X1,10,2);
       io.put("X2=");floatio.put(X2,10,2);
             New_line;
       Else
        New_line; io.put("该方程无实数解");
        New_line;
```

```
            End If;
        End Loop;
                New_line; io.put("运行结束;");
    End T1;
```

(4) 运行测试

一个程序要使它没有错误是件很困难的事情，欲排除隐患更难，有些错误经过使用多少年后才被发现。

因此编好一个程序往往仅做完了一半工作，编制大型高质量程序更是如此，有时还需推倒重来。

一般情况下，只要带进一些数据运行程序，如能达到预期的结果，就可以说当初定义的功能已经被实现，就可交付使用。因此以上程序总是没有什么问题，总是能被接受的。

但存在一个漏洞，在开始分析问题确定需求时，没有对使用范围——输入值的变化范围进行说明。这就要通过一些特殊的测试手段来查明该程序的适用范围，看看这个范围是否包含了全部需求，以便最终验证程序的质量。

以上程序在某些缺少数据保护情况下，会得到一些意外的结果。

如运行以上程序时，我们让

A＝1.0，B＝－1_000_000_004.0，C＝4_000_000_000.0

得到如下运行结果

X1＝1.00E＋9，X2＝0.00E＋0

我们可通过另一途径推出另一结果

X1＝1.00E＋9，X2＝4.00E＋0

这两个结果哪个正确呢？应该是后者，而不是前者。因为实际上该方程具有如下形式

$$X^2-(10^9+4)X+4.10^9=0$$

很明显　　$X1=10^9$　　$X2=4$

为什么计算机会算出错误的结果来呢？这是因为计算机浮点数的有效位数仅保留8位，超过部分均被舍去的缘故。

为能在此情况下也能得到正确的结果，应利用如下形式的计算模型

X1=(−B−B/ABS(B)√D)/(2A)

X2=C/A/X1

这方面的知识属于计算数学研究的问题，已经越出我们要讨论的范畴，在这里仅作说明，不作要求。以后涉及到的所有问题空间，均在计算机的有效域内讨论。

eg4:

编一程序计算　S=1+2+3+…+N−1+N

分析：

1.N是大于1的正整数；

2.利用累加器　S=S+I的计算模式实现：

S既是累加和，又是被加数。I是加数。

如S初值为零，I由1变化到N；如S初值赋1，I则由2变化到N。

3.可用FOR循环实现计算N项累加和，控制变量I，取值范围可设1到N；

这种由计算加1的和，推广到加N的和——即由初始状态推算到所求问题终结状态的计算方法，通常称为递推法，也有的称其为迭代法。这是计算领域中经常要用到的一种方法。

FOR循环机制是实现该种计算方法的最方便的控制结

构，因而也有人将FOR循环称做迭代循环。

4.按照求前N项和的公式，程序的结果应与下列公式的结果相同。

$$S=N(N+1)/2$$

可用其进行检验

程序清单

```
With Io;
Package Body T2 is
  S: Integer: =0;
  N: Integer Range 2..Integer'Last;
Begin
  Put("输入N"): Get(N);
  For I In 1..N Loop
  S: =S+I;
  End Loop;
     New_line;Put("计算结果S=");Put(S,5);
  End T2;
```

要点

语言中的语句是基本的执行机构，也是流程的控制机构。

赋值语句的基本作用是对表达式进行加工，并将运算得到的值赋给变量。

在组织赋值语句序列时要注意关照如下三个方面：

(1) 处理对象是什么？

(2) 对该对象如何实施操作？

(3) 得到的结果如何处置？

控制语句的作用是控制执行的流程路线及对某种对象实施处理的时间、条件等诸方面进行调度与控制。

在组织控制语句时，要严格掌握如下三个环节：

(1) 严格掌握控制的顺序；

(2) 严格掌握实施控制的时机、条件；

(3) 严格掌握解除控制的时机、条件。

条件语句、情况语句、循环语句都是控制语句。

条件语句的特点是可提供一个、两个或三个不同的执行分支；情况语句可提供多个执行分支。

循环语句有三种控制机制，一般情况下三者之间可以互相转换使用。

LOOP循环特点是进入循环是无条件的，终止循环需由循环体内灵活控制转出；

FOR循环特点是循控变量类型由循环值域的类型派生，值域可由任何一种离散类型的数值确定。步长取离散型数值的一个单位，程序员不能从外部干预。进入循环条件是循环值域不为空，循环执行次数由初值和终止值间的差值决定。

WHILE循环的特点是无论进入循环还是终止循环，均由表达式的布尔值决定；表达式为真时进入并执行循环，为假时则不进入或终止循环；进入循环后何时转出，由循环体内通过改变循控变量值来控制。

程序是科学与艺术相结合的产物，这里有发挥个人聪明才智的充分条件，问题是多进行编程实践，多研究分析一些高质量程序。因而对每一个语句的用途，我们均不作推荐说明，以免束缚思路。

习题

(1) 编一程序，先输入无大小顺序的三个数，经排序后

使之按先大后小顺序输出。

(2) 计算$M=1^1+2^2+3^3+\cdots\cdots$直到$M>10^{30}$。

(3) 计算$S=1+1/1+1/2+1/3+\cdots\cdots$直到最后一项小于0.000001。

(4) 计算$C=M!/(M-N)!/N!$

其中M，N为正整数，且$M>N$。

(5) 求整数M，N的最大公约数与最小公倍数。

(6) 某年如能被4除尽，但不能被100除尽，就是闰年。另外，能被400除尽的年也是闰年，否则是平年。问1900年至2000年间哪些是平年，哪些是闰年？

(7) 从1986年开始到2000年，我们将遇到多少个素数年号？

(8) 公元5世纪末，我国古代数学家张丘建在他的《算经》里提出一个不定方程问题，即有名的"百鸡"问题：

鸡翁一，值钱五；鸡母一，值钱三；鸡雏三，值钱一。

百钱买百鸡，问鸡翁、母、雏各几何？

(9) 编一程序，根据三角形三条边的值，判断其是等边三角形还是等腰三角形？要求对输入值（边长）进行检查。

第四章　过程和函数

过程与函数就是通常所说的子程序，它们都是Ada语言基本程序单元。

目前，还没有一种程序设计语言能直接地提供我们所需要的各种操作，通常仅提供一种由最基本的指令组成的机制，以便让我们在一个较高的层次上去定义算法。这种机制就是子程序。

在设计一个比较复杂的程序时，常按其要实现的功能，将具有相同功能的程序段单独分离出来，命名为过程或函数，以供随时调用。这样就可不直接通过一组程序，而是间接地通过一个过程或函数来描述客观世界的算法。

用这种面向问题的方法设计出来的程序，结构清晰、逻辑关系强、可读性好、易于调试与修改。这是一种较好的程序设计风格，它使得程序能反映问题解的高级设计结构上的操作，使语言扩充为能用问题解本身的术语来表达它的解。因此，子程序的使用是模块化程序设计的重要手段，也是扩充语言能力、缩短源程序代码、提高编程效率的重要途径。

子程序既然被分离为相对独立的模块，就有一个如何从外部接受信息与如何向外部传递信息的问题。过程与函数都以各不相同的方式同外部进行通讯，因而可以笼统地说，过程与函数的差别就在于对调用环境的影响方式与作用域的不同。

第一节　过程

现在先考察下面一个应用过程的简单例题（行号是为便于叙述才加上的）：

```
1     Package Body T4 is
2       Procedure SPACES (K:in Character;
                          COUNT:in Integer) is
3         Begin
4           For I in 1..COUNT Loop
5              Put(K);
6           End Loop;
7       End SPACES;
8   Begin
9         SPACES ('*',10);
10  End T4;
```

运行结果是打印出10个＊号

＊ ＊ ＊ ＊ ＊ ＊ ＊ ＊ ＊ ＊

该程序先在程序的声明部分定义一个过程，然后再在主程序中调用它。

过程的名字是SPACES，它的作用是打印COUNT个字符K。至于究竟打印什么字符，打印多少数量，在过程里还不能具体确定，要到调用时才能确定。因此这里的参数K与COUNT是未经实化（没有赋值）的形式参数，又由于它们的值需由外部提供，所以参数的传递方式是由外到内，对过程而言就是输入——in方式。

第2行是过程的声明部分

Procedure...is是过程声明定义符

SPACES是过程名

括号内是参数说明，声明参数K为输入方式，字符型；参数COUNT为输入方式，整数型。

该过程的声明部分说明了过程名、参数名、参数的工作方式及数据类型。

3—7行是程序体部分，第3行表明过程体由此开始，第7行表明过程体到此结束。4—6行是迭代循环打印COUNT个字符K，注意这里K是字符型的变量，而不是要打印的字符。

第9行是过程调用，该语句括号里的内容是实参部分，语句指明调用的是名为SPACES的过程，传送给K变量的字符是'*'号，传送给COUNT变量的整型值是10。该语句的调用引起名为SPACES过程的执行，即打印10个*号。

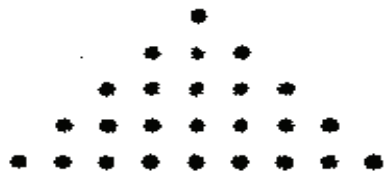
本程序略加改动，就可打印出一个由'*'号组成的三角形。

经改动后的程序如下：

eg5：

```
Package Body T4 is
   Procedure SPACES (K:Character; COUNT:
                                    Integer) is
    Begin
      For I in 1..COUNT Loop
          Put(K);
      End Loop;
  End SPACES;
Begin
   For I in 1..5 Loop
    SPACES(' ',10-I); SPACES('*',2*I-1);
    New-line;
   End Loop;
End T4;
```

执行结果



一、过程的定义

过程分为过程说明和过程体两部分，因此过程定义包括定义其调用约定的过程说明和定义其执行的过程体。

过程定义的一般格式

```
PROCEDURE〈过程名〉[(形参说明)]  IS
    [〈变量说明〉];
BEGIN
    〈语句_序列〉;
END〈过程名〉;
```

1.过程定义属于程序声明的一部分，因此过程定义只能在程序的声明部分出现；

2.PROCEDURE是系统的保留关键字，过程声明必须由此开始，过程名为一合法的标识符；

3.形参说明可以缺省。缺省时表明过程同外部仅是调用与被调用关系，除此没有其它的联系；

若有形参说明，则过程名与形参说明必须位于关键字PROCEDURE与IS之间，且其排列顺序不可颠倒；

4.变量说明定义了过程体内部使用的变量，其作用域由说明开始，延伸至该过程体结束语句，外部不可见。若过程体内不使用除形参以外的变量，此说明可缺省；

5.过程体由BEGIN开始，到END〈过程名〉处结束，介于其间的语句序列定义了过程要执行的动作；

6.过程内部可以嵌套其它子程序，另外过程也可以调用其它子程序，包括自己调用自己。

二、形参说明

形参说明由形参名、形参模式与形参类型组成，其说明格式如下：

(〈形参名〉{[,形参名]} :[in]|out|in out〈类型说明〉[:=〈表达式〉][;〈形参说明〉])

形参模式有三种：

1. in　输入方式，该模式参数仅用于从外部接收信息，参数为该模式时，关键字in可缺省；
2. out　输出方式，该模式参数仅用于向外部传送信息；
3. in out 输入/输出方式，该模式参数既用于从外部接受信息，又用于往外部传送信息。

过程说明的例子：

(1) Procedure SPACES (A,B:Integer:=0);

(2) Procedure SPACES (A:Integer;B:out Integer);

(3) Procedure SPACES (A:in Integer;B:out Integer;C:in out Float);

三、过程调用与参数匹配

过程执行由调用引起，而过程能否正确执行，则要看对

应的形参是否能被实化。

过程调用格式：

〈过程名〉[(〈实参部分〉)]；

其中过程名是指要引起执行的过程的名称；实参部分列出的是能使过程中in或in out模式形参实化的参数。

说明：

1.形参和实参的作用是在过程与主程序间建立某种联系，形参是过程的通讯接口，实参是主程序的通讯接口。

2.按强类型规则，形参说明部分不仅要确定通讯方式，还要确定其类型属性；调用时的实参类型和与之关联的形参类型必须兼容。

3.与in方式关联的实参可以是常量、变量和表达式。

4.与in out或out方式关联的实参只能是变量名。

5.一般说来，将实参值传递给形参的过程称为匹配，形参接受实参值的过程称为实化。按实参值给出方式的不同，实参与形参的匹配方式可分为三种：

（1） 按位匹配——实参未点名关联对象时，按实参、形参的排列顺序建立匹配关系；

（2） 按名匹配——实参点名关联对象时，按实参、形参标识符建立匹配关系；

（3） 混合匹配——实参中既有点名关联，又有未点名关联情况时，则优先按位匹配，尔后按名匹配。此种模式为混合匹配。

点名关联格式：

〈形参标识符〉=〉〈表达式〉

"=〉"符右端回答实参值是什么，左端回答与谁匹配？

过程说明举例：

```
Procedure ABC (A,B:in Integer;C:in out
                                Integer);
```

过程调用举例：

```
                X:Integer:=10;
                ...   ...
11        ABC(3,5,X);
12        ABC(C=>X,A=>3,B=>5);
13        ABC(C=>X,5,A=>3);
            ...     ...
```

以上11,12,13程序行的调用结果相同。

11行实参是按位关联，调用时按位匹配，即按参数的排列顺序进行实化，依次A取3，B取5，C取10。

12行实参是点名关联，调用时按名关联，与排列顺序无关。箭头=>在此处标明匹配关系，右端给出实参，指明要匹配的值是什么？左端给出形参名，回答该值与谁匹配问题。

13行实参是混合式关联，此时点名实参可不按位排列，但按位关联实参必须对位。

6.若形参在过程说明时已经实化，则调用时实参可缺省。

过程说明时形参赋初值的举例：

```
Procedure ACTIVATE(PROCESS: in PROC-
                          ESS-NAME;
                   AFTER: in PROCESS-
                   NAME:=NO-PROCESS;
                   WAIT: in DURATION:
                   =0.0;
                   PRIOR:in BOOLEAN =
```

```
                                        FALSE);
    Procedure PAIR (LEFT, RIGHT: PERSON-
                              NAME: =new PERSON);
```

调用以上过程时实参缺省举例：

```
ACTIVATE(X);
ACTIVATE(X,AFTER=>Y);
ACTIVATE(X,WAIT=>60.0,PRIOR=>TRUE);
ACTIVATE(X,Y,10.0,FALSE);

PAIR;
PAIR (LEFT=>new PERSON, RIGHT=>new
                                    PERSON);
```

7.只能以语句形式实施过程调用，过程调用不能出现在表达式中或赋值语句中。

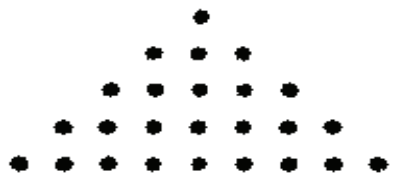
8.当形参取in out或out模式时，过程将返回一个值给实参变量，因此过程调用后，将引起与out模式形参相关联的实参变量值的改变。

例：

```
Package Body T4 is
    Procedure SPACES (K: in out String;
                      COUNT:in Integer) is
            C:String:=K;
      Begin
         For I in 1..COUNT-1 Loop
            C:=C&K;
         End Loop;
         K:=C;
      End SPACES;
     A,B:String;
Begin
    For I in 1..5 Loop
      A:=" ";B:="*";
       SPACES(A,10-I); SPACES(B,2*I-1);
```

```
            Put(A);Put(B);New_line;
        End Loop;
    End T4;
```

过程调用后，其值又经形参K分别反馈给A、B变量。运行结果为



此例仅说明in out方式使用方法。

第二节　函数

函数与过程形式上的差别在于：函数声明从关键字FUNCTION开始，并在语句序列中包含一个或数个RETURN语句；对环境影响作用的差别在于，过程通过OUT模式参数影响环境，而函数仅将返回值传递给调用函数，但它并不影响环境。

一、函数的定义

函数定义的一般格式：

```
FUNCTION〈函数名〉[(形参部分)]RETURN
                                〈返回值类型〉IS
    [〈变量说明〉];
BEGIN
    〈语句_序列〉;
```

{RETURN〈表达式〉;}

END〈函数名〉;

函数的形参可以缺省，若有形参，则必须是in模式。

二、函数调用

先看函数定义与调用的一个举例：

eg3：

```
1       Package Body T5 is
2            STRK:String;
3          Function SPACES (K:String;
           COUNT:Integer) Return String is
4               K1:String:=K;
5         Begin
6               For I in 2..COUNT Loop
7                  K1:=K1&K;
8               End Loop;
9            Return K1;
10         End SPACES;
11      Begin
12        For I in 1..5 Loop
13              STRK:=SPACES (" ", 10-I) &
                SPACES ("*", 2*I-1);
14              Put (STRK); New-line;
15        End Loop;
16      End T5;
```

运行结果：

```
          *
        * * *
      * * * * *
    * * * * * * *
  * * * * * * * * *
```

不难看出，该例与eg5运行结果相同，只不过该结果是通过函数调用实现的。

该例3—10行定义了一个名为SPACES的函数。

3—4行是函数规格说明部分，函数有一个字串型的形参K和一个整数型的形参COUNT，返回字串型的值，并且还定义了一个函数内部使用的字串型变量K1，初值取形参K实化后的串值。

5—10行是函数体部分，它定义的计算是通过迭代循环将K串连接到K1串中，并通过K1串变量将连接好的字符串返回给函数。

1—10行是主程序的声明部分，11—16行是主程序体部分。

13行两次调用函数：

调用SPACES（" "，10-I）得到10-I个空格串；

调用SPACES（"*"，2*I-1）得到2*I-1个*号；

然后将它们拼接到一起赋给串变量STRK。

14行执行打印输出。

12—15行利用迭代循环完成正三角形打印输出。

关于函数的几点说明：

1.凡是适用于表达式的场合，均适用于函数；

2.函数调用时，形参与实参的匹配规则和过程调用时的匹配规则相同；

3.过程既可不向外部传递值，也可通过形参传递多个值。如有值输出，退出过程控制时，该值被保留在与其关联的实参中；函数必须返回值，且只能返回一个值，退出控制时，返回值被传递给调用函数。

4.函数体内的语句序列中，至少包含一个指明返回值的返回语句，但每次调用仅有一个返回语句起作用。

5.不同的子程序共用一个名，或者说一个子程序名代表多个不同子程序的情况，被称为子程序重载（subprogram ovreloading）。重载时，若在参数的个数、基本类型及结果的基本类型方面可区分，则为无二义性的合法重载，否则可能引起二义性。

6.Ada语言允许将算术运算符用作函数名标识符，这种情况称为运算符重载。

例：

```
Function "+" (LEFT, RIGHT:Matrix) Retur n
Matrix;
Function"+" (LEFT, RIGHT:Vector) Return
Vector;
```

若A，B，C都是Vector类型，则下面三个语句等价：

```
A:=B+C;
A:="+" (B,C);
A:="+" (Left=>B, Right=>C);
```

第三节　递归调用举例

过程与函数在大型程序设计中有着极其广泛的应用价值，本节讨论递归调用问题。

例：

编制计算m，n组合的程序，若m=10，n=6，则有C_{10}^{6}形式。

下面分别用迭代与递归两种不同的方法来实现，从中对比一下它们之间的裨益。

用迭代法实现的思想如下：

1.设置C，M，N三个整型变量。其中C为组合值，M＝10，N＝6；

2.由于$C^6_{10}=10!/6!/(10-6)!=10!/6!/4!$
$=10*9*8*7*4!=10*9*8*7/4/3/2/1$

所以可不用求全阶乘的方法，以免产生数据溢出；其程序如下：

eg7：

```
Package Body T6 is
      M:Integer:=10;
      N:Integer:=6;
      C:Integer:=1;
Begin
    For I in N+1..M Loop
        C:=C*I;
    End Loop;
    For I in 1..M-N Loop
        C:=C/I;
    End Loop;
        Put ("组合值=");Put (C, 6);
End T6;
```

运行结果是

组合值＝210

用递归法处理的基本思想如下：

因为有

$$C^6_{10}=C^4_{10}=C^4_9+C^3_9$$

又有结束条件：

m取0的组合数＝1

m取1的组合数＝m

m取m的组合数＝1

所以，可以通过函数自己调用自己本身——递归的方法，来完成求组合的运算。

其程序如下：

eg8：

```
Package Body T7 is
      M:Integer:=10;
      N:Integer:=6;
   Function CNST (X.Y1:Integer)  Return
                                    Integer is
        Y:INTEGER:=Y1;
      Begin
        If X<2*Y1 Then
            Y:=X-Y1;

        End if;
    case y is
      when 0=>return(1);
      when 1=>return(x);
      when others=>return CNST (X-1, Y)
                        +CNST (X-1,Y-1);
    end case;
  End CNST;
Begin
  Put ("组合值=");Put (CNST(M,N), 6);
End T7;
```

运行结果是

组合值=210

eg8中函数体内的计算，调用了函数自身，这种子程序自己调用自己的算法规则，就是递归。

以上两例的运行结果是一致的，但如作进一步比较就可看出，解决此例递归调用要比迭代法优越。具体表现如下：

1.表达形式简练，使用灵活，可读性好；

2.不易产生数据溢出，有较宽的使用范围。

但在 eg 7中当M增至11时就会产生溢出；而在eg 8中当M增至19时才溢出。

因而递归调用在完成阶乘、级数运算、幂指数运算等方面有独特的功效。但决不可随意采用递归调用算法。一般地当经过调用后，函数或过程能够以明显的速度（即经过有限次递归）退化到其初始状态时，递归调用才是有效的，否则容易因堆栈溢出而无法到达初始状态。在递推与递归之间的选择，通常要根据其所需要的临时存储区域的空间大小来决定。

要点

Ada语言的基本程序单元有过程、函数、程序包和任务。

子程序的声明部分，定义了该程序单元与外部的接口；子程序体部分，定义了该程序单元的功能；而且程序单元内部定义的变量，其作用域仅局限于本单元内部，外部是不可见的。所以Ada语言程序具有较强的封闭性和模块性。对于控制语句，此特征也是很明显的。

此规则直接支持了面向问题、自顶向下的现代程序设计思想。因此，企图绕过接口，进入到程序单元内部；或者企图不通过入口，直接调用程序单元内部资源的做法，都可能会违背程序设计准则，为系统所拒绝。

只要不会出现二义性，子程序就可重载。例如过程

Put (X);

就是被重载了的。X为整数型，就输出一整型值；X为字串型时，则输出一字串值。

习题

1.按如下调用格式编制过程：

(1) INPUT ("〈字串〉",〈变量〉)；

(2) PRINT ("〈字串〉",〈变量〉)；

2.用递归与迭代两种方法计算下列各题：

(1) $E(N)=1+1/1!+1/2!+1/3!+\cdots\cdots+1/N!$

(2) $TOTAL(N)=1+2+3+\cdots\cdots+N$

(3) $$P_n(x)=\begin{cases}1 & n=0\\ x & n=1\\ ((2n-1)xP_{n-1}(x)-(n-1)\cdot P_{n-2}(x))/n & n>1\end{cases}$$

3.计算二元一次方程组的解

$$\begin{aligned}Ax+By&=C\\ Dx+Ey&=F\end{aligned}$$

4.计算函数$f(x)=x^2-2x+2$在a，b区间上的积分。

5.计算方程$3x^3-7x^2+x-5=0$的实数解。

6.用一元人民币兑换1分、2分、5分的硬币，有几种不同的兑换方法？

7.编一根据工资数统计毛票、分票的程序。

8.甲、乙、丙三人分西瓜，这堆西瓜中的一半又半个分给甲，剩下的西瓜再一半又半个分给乙，剩余的全给丙。但每次分时并没切开西瓜，问三人至少各得几个瓜？

9.大花猫吃老鼠时，先让老鼠排队，第一批吃掉报单数的，剩下的老鼠重新报数，第二批仍吃掉报单数的，以后各批也如此继续。最后剩下的一只老鼠可以不吃，与第二天抓来的老鼠一起重新排队报数。

大花猫发现，一连好几天，最后剩下的总是一只机灵的小白鼠。请问，小白鼠采用什么方法躲过了杀身之祸？

10.猎人带狼、羊、白菜各一摆渡过河，一次只能带一样。为了避免无人时，狼吃羊，或羊吃白菜的事情发生，问猎人怎样才能将它们安全运送到对岸？

11.汉诺塔（Tower of Hanoi）问题

约在19世纪末，在欧洲珍奇商店里出现了一种称为汉诺塔的游戏。这种游戏附有推销材料。材料解释说，古代印度布拉玛庙里的僧侣当时正在玩这种游戏，他们的游戏结束，就标志着世界末日的到来。

游戏的装置是在一块铜板上有三根杆，最左边杆自下而上，按由大到小顺序串着64个金盘叠成的塔。游戏的目的是把左边杆上的金盘全部移到最右边的杆上，规则是一次只能移动一个盘，并且任何时候都要保持杆上小的盘在上面，为保证以上规则的执行，允许借助中间杆移动盘。

可以推出，n个盘从左面杆移到右面杆需要的次数为

$2^{64}-1=18\_446\_744\_073\_709\_511\_615$

一台功能较强的计算机，如果一微秒处理一次，则需100万年。手工操作，若每秒钟移动一次，也需近5800亿年。从能源角度推算，太阳系的寿命只有150亿年，按此计算，则世界末日等不到该游戏结束就玩完了。

试编一程序，输出移动N（$0\leqslant n\leqslant 64$）个盘的过程。

12.设计一计算最大公约数及最小公倍数的函数

13.设计一过程，使A，B两变量互相交换数值

第五章　程序包

封装的概念

显然，依据原先知道的实体去定义更高级的实体，随后就用名字调用它，这对任何程序设计语言来说都将大大增强它的能力。这就在语言内部建立了大块结构。所有的程序于是无需按照指令系统显式地写出。程序员可以构造他自己的模块，每一块都有自己的名字且可用在程序内的任何地方，就象它是语言固有的特征一样。

D.R.Hofstadter
Godel,Escher,Bach:
永不退色的金穗带〔1〕

从第四章我们已能看出，子程序是封装着算法的可执行程序单元。有了这种封装技术，就可让低层模块提供“如何做”，高层模块仅描述“做什么”，这使得大型系统的主程序不涉及任何实现细节。

程序包推广了原始的封装思想，它允许将逻辑上相关联的一组实体及计算资源——数据、类型、算法及其组合——给封装起来，组成语言资源的一部分，而且使用者不需知道如何执行细节就可调用。在实用习惯上，可将过程、函数、任务等看作是问题求解的工具，将程序包看作是放置这些软件工具的专用工具包。

先看一个应用程序包的举例：

(一) 包程序规格说明部分

```
Package MOTOR is
      Procedure MOVETOWER(N1:Integer;X1,
                          Y1,Z:1Character);
End MOTOR;
```

(二) 程序包体部分

```
Package Body MOTOR is
      Procedure MOVETOWER (N1:Integer:X1,
                           Y1,Z1:Character)is
            T1:String:="--)";
    T2:String:="(--";
    TA:Constant String:="左";
    TB:Constant String:="中";
    TC:Constant String:="右";
Begin
  If N1>0 Then
    MOVETOWER (N1-1, X1,Z1, Y1, );
  CASE X1 is
    When'A'=>
      Case Y1 is
        When'B'=>Put(TA & T1& TB);Put("  ");
        When'C'=>Put(TA & T1 & TC);Put("  ");
      End Case;
    When'B'=>
      Case Y1 is

        When'A'=>Put(TA &  T2 &TB);Put("  ");

        When'C'=>Put(TB & T1 & TC);Put("  ");
      End case;
    When C'=>
      Case Y1 is
        When'A'=>Put(TA & T2& TC);Put(" ");
        When'B'=>Put(TB & T2 & TC);Put(" ");
      End Case;
End CASE;
    MOVETOWER(N1-1, Z1, Y1, X1);
```

```
    End If;
  End MOVETOWER;
End MOTOR;
```

(三) 调用程序包的主程序

```
    With MOTOR;
  Package Body T20 is
      Use MOTOR;
        MAX:Constant:=64;
        N: Integer Range 0..MAX;
  Begin
    Put("这是求解汉诺塔问题的例题");
  Loop
    New_line; Put("输入要移动的金盘数:");
    Get(N);
  Exit When N<=0;

      New_line;
    MOVETOWER(N, 'A','C','B');
      New_line; New_line;
    End Loop;
  End T20;
```

本例给出了一个求解汉诺塔问题的程序包，程序包的结构与其它基本程序单元一样，分为规格声明与包体两部分。

在主程序中打开程序包，并使用里面的资源。问题求解的具体细节蕴藏在包体内部，主程序中屏蔽掉有关实现的详细情况，仅通过MOVETOWER( )话句激活该过程，如同语言本身就设置有该语句一样。

下面分别叙述程序包规格、体的定义及程序包的使用和编译问题。

一、程序包规格说明

程序包说明格式：

Package〈程序包名〉is

〈规格说明〉；

End〈程序包名〉；

说明

1.程序包声明部分可以由单纯的对象、类型说明的集合组成，此时的程序包是没有包体的程序包，它用于表示一组公共常量或变量，或表示对象和类型的公共联合体。

描述一组公共变量的程序包的举例：

```
Package PLOTTING_DATA is
  PEN_UP:Boolean;
  CONVERSION_FACTOR, X_OFFSET, Y_OF-
  FSET: Real;
End PLOTTING_DATA;
```

描述对象和类型公共联合体的程序包的举例：

```
Package WORK_DATA is
  Type DAY is (MON, TUE, WED, THU,
                        FRI, SAT, SUN);
  Type HOURS_SPENT is Dalta 0.25 Range 0.0
                                    ..24.0;
  Type TIME_TABLE is Array(DAY) of HO-
                                URS_SPENT;
    WORK_HOURS: TIME_TABLE;
  NORMAL_HOURS: Constant TIME_TABLE:=
     (MON..THU=>8.25,FRI=>7.0,SAT | SUN
                                =>0.0);
End WORK_DATA;
```

2.如果程序包中的一个说明项是某个程序单元的式样说明，那么由该说明项所说明的程序单元的体，必须存在于该程序包的体中；

3.在声明部分可以加上保留字PRIVATE，这样在其前面的说明项叫该程序包的可见部分，在其后面的说明项叫做

该程序包的私有部分。

程序包的可见部分向外部提供了其它程序单元可以了解的全部信息：私有部分的实体在外部则是不可见的，使用它的目的在于禁止用户从外部访问私有信息并进行修改，以防包中数据被意外地破坏。

格式：

```
Package〈程序包名〉is
        {〈说明项〉} —该说明项为可见部分
  〔Private
        {〈说明项〉} 〕—该说明项为私有部分
End〈程序包名〉;
```

二、程序包体

程序包体说明格式

```
        Package Body〈程序包名〉is
                〔〈说明部分〉〕;
    Begin
      〈分程序单元语句_序列〉;
      End〈程序包名〉;
```

说明

1.包体是程序包的一个组成部分，是分程序单元的集合，它包含有程序包规格说明中指明的所有可访问实体的实现细节；

2.包体中说明的变量其作用域由说明开始延伸至包体结束处，因而只有在包体中才能改变它的值；

3.程序包体中可省去Begin；若有Begin，则Begin后面的语句序列为程序包初始化部分，即只要连用该程序包，这些语句序列就会自动被执行。

三、程序包的访问

例：

访问系统包FLOATIO中的浮点GET（）、PUT（）过程与MATHLIB中的SQRT（）函数

下面介绍使用包资源的两种方法

访问规则(一)

```
With Floatio,          --使FLOATIO直接可见
  Mathlib;             --使MATHLIB直接可见
Package Body T8 is
  Use Floatio,         --使GET(), PUT()直接可见
    Mathlib;           --使MATHLIB直接可见
  X:Float;

Begin
  Get(X);              --调用浮点输入过程
  If  X)=0.0 Then
X:=SQRT(X);            --调用浮点数开平方函数
    Put(X);            --调用浮点输出过程
  End If;
End T8;
```

说明

1.WITH称为带有子句，是上下文说明，它的位置必须在程序的第一行：WITH用以指明在编译单元中需要的程序包，亦即定义编译单元之间的链接工作关系。

语法格式：

With〈程序包名〉[{〈,程序包名〉}];

2.USE称为使用子句，其作用在于获得被点名程序包可见部分的直接可见性。USE子句可紧随在WITH子句之后出现，也可出现在某个程序段的说明部分。该子句作用域在使

用后即开始：如属前种情况，其作用域延伸到与该编译单元关联的终点；如属后种情况，其作用域延伸到与该程序段关联的终点。

语法格式：

Use(程序包名)〔{,(程序包名)}〕;

3.IBM PC/XT机上配置的JANUS/ADA语言所携带的程序包有15个。

(1) 程序包目录单见89页表。

(2) STANDARD是系统预定义的标准程序包，所有系统预定义的类型（例如类型BOOLEAN, INTEGER, CHARACTER)、预定义的运算等，均由本程序包说明。包的源程序见ADA标准参考手册353页《预定义的语言环境》，该包的作用域包含所有库单元；

(3) 涉及到浮点函数运算的包，均需8087支持；

(4) FLOATIO，MATHLIB仅是其中的两个包，所有包的可见部分作为附录列于本章其后，以供使用者参考。

访问规则(二)

```
  With Floatio, Mathlib;
Package Body T8 is
  X:Float;
Begin
  Floatio.Get(X);        --调用浮点包的输入过程
If X>=0.0 Then
X:=Mathlib.Sqrt(X);--调用数学库中的开平方函数
Floatio.Put (X) ; -- 调用浮点包的输出过程
End If;
End T8;
```

说明

访问规则(二)与(一)的差别在于没有使用USE子句。

序号	包　　名	后缀	字节	年　　月	时　分
1	ASCII	LIB	5632	6-30-83	4:40p
2	BIT	LIB	2048	7-01-83	3:48p
3	BLKIO	LIB	1024	7-01-83	3:48p
4	CHAINLIB	LIB	1024	7-01-83	3:47p
5	FLOATIO	LIB	2176	1-11-84	4:42p
6	IO	LIB	3584	7-01-83	3:47p
7	JLIB86	LIB	5120	7-01-83	3:48p
8	LONGIO	LIB	1024	7-01-83	3:46p
9	LONGOPS	LIB	2176	6-21-83	12:51p
10	MATHLIB	LIB	4096	7-01-83	3:49p
11	MATHLIB2	LIB	1536	7-01-83	3:45p
12	STANDARD	SYM	1024	7-12-83	5:40p
13	STRLIB	LIB	1536	7-01-83	3:47p
14	TIMELIB	LIB	1024	7-01-83	3:48p
15	UTIL	LIB	2048	7-01-83	3:47p
16	STACK	LIB	128	1-01-80	12:42a

此种情况下为能获得调用对象的可见性，需在调用的对象前冠以程序包名和圆点符。

语法格式：

〈程序包名〉.〈过程 | 函数〉；

四、程序包的编译

程序包声明与体部分既可一起提交编译，又可分别提交编译。本章开始例中包的声明与体部分是被分离开的，现对分别编译的步骤作一说明：

1.编译包声明

JANUS 〈包名〉.LIB

2.编译包体

JANUS 〈包名〉.PKG

3.编译主程序

JANUS 〈主程序名〉

4.连接主程序

JLINK 〈主程序名〉

5.运行主程序

〈主程序名〉

要点

1.程序包提供了一种封装机制，它为高层次上的数据抽象提供了强有力的工具，使得程序模块高度集成化；

2.程序包的应用主要表现在三个方面：

(1)将逻辑相关的声明(数据)归并为一个有名集合，以供其它单元引用。如其中数据、定义对象和类型需要变动，只要改动某一个程序包就行了，这就确保了整个程序系统内部的一致性，也提高了程序的可维护性；

(2)将功能相关的程序单元(过程、函数、任务等)归并为一个有名集合，使得其中任一单元的修改（例如为了提高精度、修改算法等），都不会对使用该单元的程序产生任何影响；

(3)利用私有部分对类型封装，使得每个程序包仅输出相关类型的最小集合。

习题

1.建立一个标识符为COMPLEX的程序包，使它具有如下功能：

加、减、乘、除、不同角度单位间的转换及求三角形的边长和角度。

2.结合本身专业，编一能处理某一专业范畴内问题的应用软件包。

3.阅读下列各程序包的声明部分，分析其中之可用资源。

附：系统包声明部分源程序

(1) ASCII_PACKAGE

```
Package Ascii Is
    --Copyright 1982,1983 RR  Software,Inc.,P.
      O. Box 1512,Madison WI 53701
    --Permission is hereby  given  to  distribute
Object Code produced from these libraries.
    All Other rights reserved.
--Last Modified 6/30/83
```

```
    --Note This Package Includes a full definition
  of the Character Set unlike
    --Full Ada which does only some.
    --Control Characters
NUL:Constant Character:=Character'Val(0);
SOH:Constant Character:=Character'Val(1);
STX:Constant Character:=Character'Val(2);
ETX:Constant Character:=Character'Val(3);
EOT:Constant Character:=Character'Val(4);
ENQ:Constant Character:=Character'Val(5);
ACK:Constant Character:=Character'Val(6);
BEL:Constant Character:=Character'Val(7);
BS :Constant Character:=Character'Val(8);
HT :Constant Character:=Character'Val(9);
LF :Constant Character:=Character'Val(10);
VT :Constant Character:=Character'Val(11);
FF :Constant Character:=Character'Val(12);
CR :Constant Character:=Character'Val(13);
SO :Constant Character:=Character'Val(14);
SI :Constant Character:=Character'Val(15);
DLE:Constant Character:=Character'Val(16);
DC1:Constant Character:=Character'Val(17);
DC2:Constant Character:=Character'Val(18);
DC3:Constant Character:=Character'Val(19);
DC4:Constant Character:=Character'Val(20);
NAK:Constant Character:=Character'Val(21);
```

```
SYN:Constant Character:=Character'Val(22);
ETB:Constant Character:=Character'Val(23);
CAN:Constant Character:=Character'Val(24);
EM :Constant Character:=Character'Val(25);
SUB:Constant Character:=Character'Val(26);
                                    --End of File
ESC:Constant Character:=Character'Val(27);
FS :Constant Character:=Character'Val(28);
GS :Constant Character:=Character'Val(29);
RS :Constant Character:=Character'Val(30);
US :Constant Character:=Character'Val(31);
SP :Constant Character:=Character'Val(32);
DEL:Constant Character:=Character'Val(127);
                                    --Rub out
--Other Characters
EXCLAM      :Constant Character:='!';
QUOTATION   :Constant Character:='"';
SHARP       :Constant Character:='#';
DOLLAR      :Constant Character:='$';
PERCENT     :Constant Character:='%';
AMPERSAND   :Constant Character:='&';
S_QUOTE     :Constant Character:=''';
L_PAREN     :Constant Character:='(';
R_PAREN     :Constant Character:=')';
STAR        :Constant Character:='*';
PLUS        :Constant Character:='+';
```

```
COMMA                 :Constant Character:='.';
MINUS                 :Constant Character:='-';
PERIOD                :Constant Character:='.';
FORE_SLASH            :Constant Character:='/';
COLON                 :Constant Character:=':';
SEMI_COLON            :Constant Character:=';';
LESS_THAN             :Constant Character:='<';
EQUALS                :Constant Character:='=';
GREATER_THAN          :Constant Character:='>';
QUERY                 :Constant Character:='?';
AT_SIGN               :Constant Character:='@';
L_BRACKET             :Constant Character:='[';
BACK_SLASH            :Constant Character:='\';
R_BRACKET             :Constant Character:=']';
CIRCUMFLEX            :Constant Character:='^';
UNDERLINE             :Constant Character:='_';
GRAVE                 :Constant Character:='`';
L_BRACE               :Constant Character:='{';
BAR                   :Constant Character:='|';
R_BRACE               :Constant Character:='}';
TILDE                 :Constant Character:='~';
--Lower Case Letters
LC_A:Constant Character:='a';
LC_B:Constant Character:='b';
LC_C:Constant Character:='c';
LC_D:Constant Character:='d';
```

```
LC_E:Constant Character:='e';
LC_F:Constant Character:='f';
LC_G:Constant Character:='g';
LC-H: Constant Character: ='h';
LC_I :Constant Character:='i' ;
LC_J :Constant Character:='j' ;
LC_K:Constant Character:='k';
LC_L:Constant Character:='l' ;
LC_M:Constant Character:='m';
LC_N:Constant Character:='n' ;
LC_O:Constant Character:='o' ;
LC_P:Constant Character:='p' ;
LC_Q:Constant Character:='q' ;
LC_R:Constant Character:='r' ;
LC_S:Constant Character: ='s' ;
LC_T:Constant Character: ='t' ;
LC_U:Constant Character:='u' ,
LC_V:Constant Character:='v' ;
LC_W:Constant Character:='w';
LC_X:Constant Character:='x';
LC_Y:Constant Character:='y';
LC_Z:Constant Character:='z' ;
--Upper Case Letters
UC_A:Constant Character:='A';
UC_B:Constant Character:='B' ;
UC_C:Constant Character:='C' ;
```

```
UC_D:Constant Character:='D';
UC_E:Constant Character:='E';
UC_F:Constant Character:='F';
UC_G:Constant Character:='G';
UC_H:Constant Character:='H';
UC_I :Constant Character:='I';
UC_J :Constant Character:='J';
UC_K:Constant Character:='K';
UC_L:Constant Character:='L';
UC_M:Constant Character:='M';
UC_N:Constant Character:='N';
UC_O:Constant Character:='O';
UC_P:Constant Character:='P';
UC_Q:Constant Character:='Q';
UC_R:Constant Character:='R';
UC_S:Constant Character:='S';
UC_T:Constant Character:='T';
UC_U:Constant Character:='U';
UC_V:Constant Character:='V';
UC_W:Constant Character:='W';
UC_X: Constant Character:='X';
UC_Y:Constant Character:='Y';
UC_Z :Constant Character:='Z';
ONE  :Constant Character:='1';
TWO  :Constant Character:='2';
THREE:Constant Character:='3';
```

```
FOUR :Constant Character:='4';
FIVE  :Constant Character:='5';
SIX   :Constant Character:='6';
SEVEN:Constant Character:='7';
EIGHT :Constant Character:='8';
NINE  :Constant Character:='9';
ZERO: Constant Character: ='0';
End Ascii;
```

(2) BIT_PACKAGE

```
Package Bit Is
    --Bit operations library
    --Last modified 3/27/83
    --Copyright 1982,1983 RR Software.P.O.
      Box 1512, Madison WI53701
    --Permission is hereby given to distribute
      Object Code produced from
    --these libraries.
    Subtype bit_num Is Integer Range 0..15;
    Procedure Setbit(Val:In Out Integer;Bit:In
                                        Bit_num);
        --Set bit number bit
    Procedure Clrbit(Val:In Out Integer;Bit:In
                                        Bit_num);
        --Clear bit number bit
    Function Tstbit(Val:In Integer;Bit:In Bit_
                        num) Return Boolean;
```

```
    —Return the value of bit number bit (
      =True)
Function Land(Val1,Val2:In Integer) Return
                                    Integer;
    —Logical And
Function Lor(Val1,Val2:In Integer) Return
                                    Integer;
    —Logical Or
Function Lxor(Val1,Val2:In Integer) Return
                                    Integer;
    —Logical Xor
Function Lnot(Val:In Integer) Return Integer;
    —Logical Not
Function Peek(Addr:In Integer) Return Byte;
    —Returns the byte at address in the data
      segment
Procedure Poke(Addr:In Integer; Val:In By_
                                    te);
    —Changes the byte at address in the data
      segment to Val
Function LPeek (Segment,Offset:In Integer)
                                Return Byte;
    —Returns the byte at address Segment:
      Offset
    —Ought to be called Peek, but no over—
      loading in the assembler
```

```
    Procedure LPoke(Segment,Offset:In Integer;
                                    Val:In Byte);
        --Changes the byte at address Segment:
          Offset to Val
        --Ought to be called Poke, but no over-
          loading in the assembler
    Function Code_Seg Return Integer;
        --Returns the Code Segment Value(Will
          not be usable in future
        --versions of Janus/Ada)
    Function Data_Seg Return Integer;
        --Returns the Data Segment Value(Will
          not be usable in future
        --versions of Janus/Ada)
    Procedure InPort(Portnum:In Integer;Value:
                                    Out Byte);
        --Reads a byte from the port portnum,
          returns it in Value
    Procedure Outport(Portnum:In Integer;Value:
                                    In Byte);
        --Writes a byte(Value) to the port port-
          num
End Bit;
```

(3) BLKIO_PACKAGE

```
Package Blkio Is
```

```
------------------------------------------------
---This library implements random access
  sector I/O via                              ---
---the procedures read_blk and  write_blk.
                                              ---
---This is the specification.                 ---
---Last Modified 6/29/82                      ---
------------------------------------------------
--Copyright 1982 RR Software,P.O. Box
  1512, Madison WI 53701
--Permission is hereby given to distribute
  Object Code produced from
--these libraries.
Type Sector Is Array(0..127) Of Byte;
        --Generally used in a variant record
Procedure write_blk(Fyle:In File; sec:In Se-
                          ctor; blk: In Integer);
    --Write the block number blk into the
      file fyle
Procedure read_blk(Fyle:In File:sec: Out
                       Sector; blk; In Integer);
  --Read the block number blk from the
    file fyle
Procedure rclose(Fyle:In Out File);
  --Close a random access file
End Blkio;
```

(4) CHAINLIB_PACKAGE

```
Package Chainlib Is
    --The program chaining and calling library
    --Last modified 8/2/82
    --Copyright 1982 RR Software, P.O. Box
      1512, Madison WI 53701
    --Permission is hereby given to distribute
      Object Code produced from
    --these libraries.
    Procedure Chain(Str: In String);
      --Chains a program, destroying the data
        segment
      --Note: The Jlib86 library must be the sa-
        me for both the chaining
      --and chained programs for this routine to
        work
    Procedure Prog_Call(Str:In String);
      --Calls a program
    Procedure Prog_Return;
      --Returns from a called program
End Chainlib;
```

(5) FLOATIO_PACKAGE

```
Package Floatio Is
    --Floating Point I/O Package
    --Written March 1983
    --Last Modified 1/11/84
    --Copyright 1983,1984 RR Software, P.O.
```

```
   Box 1512, Madison WI 53701
--Permission is hereby given to distribute
   Object Code produced from
--these libraries.
Pragma Arithcheck(Off);Pragma Rangecheck
   (Off);
@ Pragma Arithcheck(On);Pragma Rangech-
   eck(On);
Procedure Get(Fyle:In File; Value:Out Flo-
   at);
Procedure Get(Value:Out Float);
Procedure Put(Fyle:In File; Value:In Flo-
   at);
Procedure Put(Value:In Float);
Procedure Put(Fyle:In File; Value:In Float;
   Fore:In Integer);
Procedure Put(Value:In Float; Fore:In inte-
   ger);
Procedure Put(Fyle:In File; Value:In Float;
   Fore,Aft:In Integer);
Procedure Put (Value:In Float;Fore, Aft:In
   Integer);
Procedure Put(Value:In Float;Fore,Aft,Exp;
   In Integer);
Procedure Put(Fyle:In File; Value:In Float;
                  Fore,Aft,Exp:In Integer);
```

```
   —Puts formatted value to file
Function  Float_to_String  (Value:In  Float)
                                  Return String;
Function Float_to_String (Value:  In  Float;
        fore, aft,exp:Integer) Return String;
   —Formats value into a string
Procedure Get(Fyle:In File;Value:Out Long_
                                      Float);
Procedure Get(Value:Out Long_Float);
Procedure Put(Fyle:In File;Value:In  Long_
                                      Float);
Procedure Put(Value:In Long_Float);
Procedure Put(Fyle:In File; Value:In Long_
                    Float; Fore:In Integer);
Procedure Put(Value:In  Long_Float;Fore:In
                                    Integer);
Procedure Put(Fyle:In File;Value:In  Long_
                  Float; Fore,Aft:In Integer);
Procedure  Put(Value:In   Long_Float:Fore,
                                  Aft:In Integer);
Procedure Put(Value:In Long_Float;Fore,A—
                                ft,Exp:In Integer);
Procedure Put(Fyle:In File;Value:In  Long_
            Float; Fore,Aft,Exp:In Integer);
   —Puts formatted value into file
Function Float_to_String(Value:In Long_Fl—
```

```
      oat) Return String;
    Function Float_to_String(Value:In Long_Fl-
      oat; fore, aft, exp: Integer) Return
                                        String;
      --Formats value into a string
End Floatlo;
```

(6) IO_PACKAGE

```
Package IO Is
--The I/O package for JANUS V. 1.4.6-MS-DOS
  8086 Version
--Last Modified 5/5/83
--Keypress & Purge Added 2/13/83, with bug
  correction to Get_line
--Mode, Putw, and Second Get_Line added 3/27
  /83
--Put_Line(s) added 5/5/83
    --Copyright 1982,1983 RR Software, Inc.,
      P.O. Box 1512, Madison WI 53701
    --Permission is hereby given to distribute
      Object Code produced from
    --these libraries. All Other rights reserved.
Type File_Mode Is(No_Access,Read_Only,Write_
                                  Only,Read_Write);
Subtype LString Is String(255);
IOresult:Integer;--The result of the IO operation
Procedure Open(Fyle:In Out File; Name:In Str-
```

```
    ing; Mode:In File_Mode);
      --Open the file name and give it the mode
        mode
  Procedure Create(Fyle:In Out File;Name:In Stri-
                                ng;Mode:In File_Mode);
      --Create the file name and give it the mode
        mode
  Procedure Delete(Name:In String);
      --Delete the file name
  Procedure Close(Fyle:In Out File);
      --Close the file fyle
  Function Name(Fyle:In File) Return String;
      --Return the name of the Open file
  Function Mode(Fyle:In File) Return File_Mode;
      --Return the file mode of the Open file
  Function Is_open(Fyle:In File) Return Boolean;
      --Is the file fyle open?
  Function Get_Line Return LString;
      --Get a line from Current_Input
  Function Get_Line(Fyle: In File) Return LS-
    tring;
      --Get a line from the file fyle
  Procedure Put_Line (Fyle: In File: Str: In LS-
                                tring);
      --Put a line to the file with a New_line
  Procedure Put-Line(Str:In LString);
```

```
    --Put a line to current_output with a new_
      line
Procedure Put_Hex(Fyle:In File;val:In Integer);
    --Write the integer in hexidecimal (no
      special format)
Procedure Putw(str:In LString;width:In Integer);
    --Write the string to the default file,with
      blank padding to fill width
Procedure Putw(Fyle:In File; Str: In LString;
                                width: In Integer);
    --Write the string to fyle, with blank padd-
      ing to fill width.
Function End_of_file(fyle:In File) Return Boole-
                                              an;
    --End of File Reached(in a text file)?
Function EOF(fyle: In File) Return Boolean;
    --End of File Reached(in a binary file)?
Function Disk_full(fyle:In File) Return Boolean;
    --Is the Disk full?
Function End_of_Line(fyle: In File) Return
                                        Boolean;
    --End of Line Reached?
Function Keypress Return Boolean;
    --Returns true if a key has been pressed sin-
      ce last read
Procedure Purge(name:In String);
```

```
    --Deletes file name without an error if it
      does not exist
End IO;
```

(7) JLIB86_PACKAGE

```
Package Jlib86 Is
        --These are the External declarations
          for JLib86
        --Copyright 1982,1983 RR Software,
        --P.O. Box 1512, Madison WI 53701
        --Permission is hereby given to distribute
          Object Code Produced from
        --these libraries.
        --The specifications below cannot be ch-
          anged or deleted.
        --The user may add entries to the end
          of the table

        --The Procedures below cannot be cal-
          led from a JANUS program (they
        --can be used in a Assembly language
          program) unless they are marked
        --by an asterisk( * )
        --The Entry Point Table -     Idnum

        Procedure EHalt                 -- 1*
        Procedure PrgRet;               -- 2
```

```
Procedure PChain;                    — 3
Procedure NotUsed4;                  — 4
Procedure Exp2;                      — 5
Procedure Mod2;                      — 6
Procedure NotUsed7;                  — 7
Procedure NotUsed8;                  — 8
Procedure NotUsed9;                  — 9
Procedure NotUsed10;                 —10
Procedure NotUsed11;                 —11
Procedure NotUsed12;                 —12
Procedure NotUsed13;                 —13
Procedure NotUsed14;                 —14
Procedure NotUsed15;                 —15
Procedure NotUsed16;                 —16
Procedure NotUsed17;                 —17
Procedure NotUsed18;                 —18
Procedure Member1;                   —19
Procedure Member2;                   —20
Procedure SMember1;                  —21
Procedure SMember2;                  —22
Procedure ECWrite;                   —23
                          Code Seg write
Procedure CSAssign;                  —24
                  Code Seg String Assignment
Procedure Copy_Str;                  —25
Procedure NotUsed26;                 —26
```

```
Procedure NotUsed27;        —27
Procedure NotUsed28;        —28
Procedure NotUsed29;        —29
Procedure NotUsed30;        —30
Procedure Sour_Err;         —31*
Procedure Rangel;           —32
Procedure Range2;           —33
Procedure SRangel;          —34
Procedure SRange2;          —35
Procedure EErr_Exit;        —36*
Procedure Null_Ptr;         —37
Procedure Str_Bound;        —38
Procedure CaseErr;          —39
Procedure GetInt;           —40
Procedure EPut_Str;         —41
Procedure Put_CStr;         —42
Procedure NotUsed43;        —43
Procedure NotUsed44;        —44
Procedure EPutInt;          —45
Procedure PutHex;           —46
Procedure EPutIntw;         —47
Procedure EPutEnum;         —48
Procedure EPutEnumW;        —49
Procedure NotUsed50;        —50
Procedure ProcInit;         —51
Procedure ProcFin;          —52
```

Procedure EClose; —53
Procedure RPlcByte; —54
Procedure SLt; —55
Procedure SLe; —56
Procedure SEq; —57
Procedure SNeq; —58
Procedure SGe; —59
Procedure SGt; —60
Procedure Sassign; —61
Procedure Concat; —62
Procedure EVWrite; —63
Procedure ERead; —64
Procedure EWrite; —65
Procedure ENew_Line; —66
Procedure ESkip_Line; —67
Procedure Func_Release; —68
Procedure Func_Ret; —69
Procedure EFile_Name; —70
Procedure New_Ptr; —71
Procedure EMemAvail; —72
Procedure EMaxAvail; —73
Procedure EDispose; —74
Procedure FError_Chk; —75
Procedure Bool_Tab; —This
—is not a Procedure at all,
—rather.

```
--it is the enumeration table for Boolean
 --File Type definitions
 BUFFER_SIZE:Constant:=512;--Must
                    --be a multiple of 128
 Type Disk_fcb Is Record
  Disk_num      : Byte;          --Offset 0
  File_name     : Array(1..11) Of Char-
                                      acter
                                  --Offset 1
  Extent        : Integer;       --Offset 12
  Rec_Size      : Integer;       --Offset 14
  File_Size1    : Integer;       --Offset 16
  File_Size2    : Integer;       --Offset 18
  Date          : Integer;       --Offset 20
  Time          : Integer;       --Offset 22
  Reserved      : Array(24..31) Of Byte;
                                  --Offset 24
  Rec_num       : Byte;          --Offset 32
  Random_rec1   : Integer;       --Offset 33
  Random_rec2   : Integer;       --Offset 35
End Record;
Type File-mask;
Type File_ptr Is Access File_Mask;
Type File_mask Is Record
 ftype:Byte;    --0-Disk_File;1-Con;2-
                  Aux; 4-Lst;5-Kbd
```

```
                    --Offset 0
  fmode:Byte;   --Offset 1(Really of type
                    file_mode,see IO.LIB)
  fcb:disk_fcb;  --Offset 2
  buff:Array (0..BUFFER_SIZE-1) Of
  Byte;  --Offset 39
  buf_ptr:Integer;--Pointer into buff-Of-
                    fset 167
  eof_flag:Boolean;--Offset 169
  link:File_ptr;  --Offset 170
                    --Chain link so JANUS
                     can keep track of all
                    --open files.
End Record;
--JLib86 Data Area
DispStart:Integer;--Display 'Registers'
                     (0 and 1 unused)
Display1 :Integer;
Display2 :Integer;
Display3 :Integer;
Display4 :Integer;
Display5 :Integer;
Display6 :Integer;
Display7 :Integer;
Display8 :Integer;
Display9 :Integer;
```

```
        Display10:Integer;
        LineNo:Integer;      --Current Line Number being executed
        Input_File:Integer;  --Standard Input (Cannot be used directly)
        Output_File:Integer; --Standard Output (ditto)
        File_Chain:File_Ptr; --Chain of files(so they can be closed if
                              --the user forgets)
        Have_8087:Boolean;   --Set at initialization time.
        --Other user defined routines may be added here
Pragma sybdump(On);
End Jlib86;
```

(8) LONGIO_PACKAGE

```
With Longops;
Package Longio Is
    --Copyright 1983 RR Software. P.O. Box 1512.Madison WI 53701
    --Permission is hereby given to distribute Object Code produced from
    --these libraries.
    --Long Integer I/O
```

```
    --Do not USE (use clause) this package un-
      til the built-in procedure
    --bug is fixed.
    --Last Modified 3/27/83
    Use Longops;
    Procedure Get(Item:Out Long_Integer);
    Procedure Get(Fyle:In File;Item:Out  Long_
      Integer);
    Procedure Put(Item:In Long_Integer);
    Procedure Put(Fyle:In  File;Item:In  Long_
      Integer);
    Procedure Put(Item:In  Long_Integer;Width:
      In Integer);
    Procedure Put(Fyle:In  File;Item:In  Long_
      Integer;Width:In Integer);
End Longio;
```

(9) LONGOPS_PACKAGE

```
Package Longops Is
    --Copyright 1983 RR  Software.Inc.,  P.O.
      Box 1512, Madison WI 53701
    --Permission is hereby  given  to  distribute
      Object Code produced from
    --these libraries. All Other right reserved.
    --Long Integer Operations
    --Last Modified 6/21/83
    --The names of these routines will be chang-
```

```
    ed when operator overloading
  —is implemented.
  Type Long_Integer Is Private;
  —Normal Assignment and Equality operators
    are used
  Function Labs(Item:In Long_Integer) Ret-
    urn Long_Integer;
    —Operator"ABS"-Take the absolute value
      of operand
  Function Lneg(Item:In Long_Integer) Return
    Long_Integer;
    —Operator"-" -Negate the Operand
  Function Lint(Item:In Integer) Return Lo-
    ng_Integer;
    —Type Conversion Integer=>Long_Integer
  Function L_to_int(Item:In Long_Integer) Re-
    turn Integer;
    --Type Conversion Long_Integer=>Integer
  Function Ladd(Item,Item2:In Long_Integer)
    Return Long_Integer;
    —Operator"+" -Long Integer addition
  Function Lsub(Item,Item2:In Long_Integer)
    Return Long_Integer;
    --Operator"-"-Long Integer subtraction
  Function Lmul(Item,Item2:In Long_Integer)
    Return Long_Integer;
```

```
    --Operator"*"-Long Integer multiply
  Function Ldiv(Item,Item2:In  Long_Integer)
    Return Long_Integer;
    --Operator "/" -Long Integer division
  Function Lrem(Item,Item2:In  Long_Integer)
    Return Long_Integer;
    --Operator "REM" -Long Integer remainder
  Function Lmod(Item,Item2:In Long_Integer)
    Return Long_Integer;
    --Operator "-" -Long Integer modulus
  Function  Lgt(Item,Item2:In  Long_Integer)
    Return Boolean;
    --Operator ">" -Long Integer greater than
  Function  Lge(Item,Item2:In  Long_Integer)
    Return Boolean;
    --Operator ">=" -Long Integer greater
      than or equals
  Function  Llt(Item,Item2:In  Long_Integer)
    Return Boolean;
    --Operator "<" -Long Integer less than
  Function  Lle(Item,Item2:In  Long_Integer)
    Return Boolean;
    --Operator "<=" -Long Integer less than
      or equals
Private
  Type Long_Integer is Record
```

```
                a,b:Integer;
    End Record;
End Longops;
```

(10) **MATHLIB_PACKAGE**

```
Package Mathlib Is
    --Mathematic functions library
    --Last modified 4/25/83
    --Copyright 1983 RR Software. Inc.,P.O.
      Box 1512, Madison WI 53701
    --Permission is hereby given to distribute
      Object Code produced from
    --these libraries. All Other rights reserved.
    PI:Constant:=3.14159_26535_89793_23846;
    E:Constant:=2.71828_18284_59045_23536;
    Function Sqrt(Val:In Float) Return Float;
      --Returns the sqrt of val
    Function Round(Val:In Float) Return Float;
      --Rounds val to an integer value,and ret-
        urns it as a float
    Function Trunc(Val:In Float) Return Float;
      --Truncate val to its integer part,and re-
        turns it as a float
    Function Exp(Val:In Float) Return Float;
      --Returns e ** Val
    Function Log(Val:In Float) Return Float;
      --Returns the natural logarithm of Val.
```

```
    Val must be )0.
Function  Power(Val,Exp:In Float)  Return
  Float;
  --Returns Val ** Exp
--All angles are in radians!
Function Sin(Angle:In Float) Return Float;
  --Returns the Sine of the angle
Function Cos(Angle:In Float) Return Float;
  --Returns the Cosine of the angle
Function Tan(Angle:In Float) Return Float;
  --Returns the Tangent of the Angle
Function ArcTan(Val:In Float) Return Float;
  --Returns the ArcTangent of the Value
Function ArcCos(Val:In Float) Return Float;
  --Returns the ArcCosine of the Value
Function ArcSin(Val:In Float) Return Float;
  --Returns the ArcSine of the Value
Function ArcTan2(X,Y:In Float)  Return
  Float;
  --Returns the ArcTangent of X/Y
Function Deg_to_Rad(Angle:In Float) Return
  Float;
  --Converts the Angle in  Degrees  to  the
    same angle in Radians
Function Rad_to_Deg(Angle:In Float) Return
    Float;
```

```
    --Converts the Angle in Radians to the
      same angle in Degrees
  Function Sqrt(Val:In Long_Float) Return Lo-
    ng_Float;
    --Returns the sqrt of val
  Function Round(Val:In Long_Float) Return
    Long_Float;
    --Rounds val to an integer value,and retu-
      rns it as a Long_Float
  Function Trunc(Val:In Long_Float) Return
    Long_Float;
    --Truncate val to its integer part, and
      returns it as a Long_Float
  Function Exp(Val:In Long_Float) Return Lo-
    ng_Float;
    --Returns e ** Val
  Function Log(Val:In Long_Float) Return Lo-
    ng_Float;
    --Returns the natural logarithm of Val.
      Val must be )0.
  Function Power(Val,Exp:In Long_Float)
    Return Long_Float;
    --Returns Val ** Exp
  --All angles are in radians!
  Function Sin(Angle:In Long_Float) Return
    Long_Float;
```

```
    --Returns the Sine of the angle
  Function Cos(Angle:In  Long_Float) Return
    Long_Float;
    --Returns the Cosine of the angle
  Function Tan(Angle:In  Long_Float) Return
    Long_Float;
    --Returns the Tangent of the Angle
  Function ArcTan(Val:In Long_Float) Return
    Long_Float;
    --Returns the ArcTangent of the Value
    Function ArcCos(Val:In Long_Float) Ret-
      urn Long_Float;
    --Returns the ArcCosine of the Value
  Function ArcSin(Val:In Long_Float) Return
    Long_Float;
    --Returns the ArcSine of the Value
  Function ArcTan2(X,Y:In Long_Float)  Re-
    turn Long_Float;
    --Returns the ArcTangent of X/Y
  Function Deg_to_Rad(Angle:In  Long_Float)
    Return Long_Float;
    --Converts the Angle in Degrees to the sa-
      me angle in Radians
  Function Rad_to_Deg(Angle:In  Long_Float)
    Return Long_Float;
    --Converts the Angle in  Radians  to  the
```

```
        same angle in Degrees
End Mathlib;
```

(11) **MATHLIB2_PACKAGE**

```
Package Mathlib2 Is
    --Mathematic functions library
    --These routines do no checking,have  weird
      use restrictions,and
    --should not be used from user programs.Use
      Mathlib instead.
    --Copyright  1983  RR  Software,Inc.,P.O.
      Box 1512,Madison WI 53701
    --Permission is hereby  given  to  distribute
      Object Code produced from
    --these libraries. All Other rights  reserved.
    Function Two_Exp(Val:In Long_Float) Ret-
      urn Long_Float;
        --Returns 2 ** Val
    Function Exp(Val:In Long_Float)  Return
      Long_Float;
        --Returns E ** Val
    Function Power(X,Y:In Long_Float) Return
      Long_Float;
        --Returns X ** Y
    Function  Sin(Angle:In Long_Float) Return
      Long_Float;
        --Returns the Sine of the angle
```

```
    Function  Cos(Angle:In Long_Float) Return
      Long_Float;
        --Returns the Cosine of the angle
    Function  Tan(Angle:In Long_Float) Return
      Long_Float;
        --Returns the Tangent of the Angle
End Mathlib2;
```

(12) STRLIB_PACKAGE

```
    Package Strlib Is
      --String Handling Package Specification
      --Last Modified 6/3/82
      --Copyright 1982 RR Software, P.O. Box
        1512, Madison WI 53701
      --Permission is hereby given to distribute
        Object Code produced from
      -- these libraries.
      Subtype Mstring Is String(255);--Maximum
                                    --string length
      Subtype StrIndex Is Integer Range 0..255;
        --Maximum string indices
Function Length(str:In Mstring) Return Integer;
      --Return the length of the string
Function Remove(str:In Mstring;pos,size:In Str-
  Index) Return Mstring;
      --Remove size characters from str at pos
Function Insert(source,dest:In MString;pos:In
```

```
  StrIndex)  Return MString;
    --Insert source into dest at pos
Function Extract(str:In Mstring;pos,size:In Str-
  Index) Return Mstring;
    --Extract size characters from str at pos
Function Position(Pattern,str:Mstring) Return In-
  teger;
    --Return the position of the first  occurance
      of pattern in str,
    --or 0 if there is none
Function char_to_str(char:character)  Return Str-
  ing;
    --Convert a character into a string of length
      1
Function str_to_int(str:Mstring) Return Integer;
    --Convert a string into an integer
Function int_to_str(int:Integer) Return Mstring;
    --Convert an integer into a string
End Strlib;
```

(13) **TIMELIB_PACKAGE**

```
Package Timelib Is
    --Time Library -Contains procedures for ge-
      tting the time and date
    --from MS-DOS
    --Copyright 1982 RR Software,  P.O.  Box
      1512, Madison WI 53701
```

```
    --Permission is hereby given to distribute
      Object Code produced from
    --these libraries.
Type time Is Record
      hours  :Integer;
      Minutes:Integer;
      seconds:Integer;
      fract  :Integer;
End Record;
Type date Is Record
     year  :Integer;
     month:Integer;
     day   :Integer;
End Record;
Function get_time Return Time; --Get and retu-
                               --rn the curren ttime
Function get_date Return Date; --Get and  retu-
                               --rn the current date
Procedure put_date(fyle:In file;day:date);
                                  --Put the date to
                                    the file
Procedure put_time(fyle:In file;clk:time);
                                  --Put the time to
                                    the file
Function elapsed_time(start,finish:Time) Return
  Time;
```

```
        --Figure the elapsed time between start
          and finish
End Timelib;
```

(14) UTIL_PACKAGE

```
With Jlib86;
Package Util Is
    --Spec for the package util
    --Last modifed 3/1/83
    --Contains the utility routines,and the basic
      file handling routines
    --Copyright 1982,1983 RR Software,P.O.
      Box 1512,Madison WI 53701
    --Permission is hereby given to distribute
      Object Code produced from
    --these libraries.
    Use JLib86; --So the file definitions are av-
                                ailable
    Procedure Err_Exit;
    Procedure Halt;
    Function Hi(val:Integer) Return Byte;
    Function Lo(val:Integer) Return Byte;
    Function Memavail Return Integer;
    Function Maxavail Return Integer;
    Function Command_Line Return String;
      --Returns the command line
      --Default File Procedures
```

```
    Function FConvert(Fyle:In File)  Return Fi_
      le_ptr;
    Procedure FFConvert(Fyle_ptr:In  File_Ptr;
      Fyle:Out File);
      --Convert to and from the type file to the
        type file_ptr
      --For system use only,Not to be  used  in
        user programs.
    Function Standard_Input Return File;
      --Returns the initial default system  input
        file
    Function Standard_Output Return File;
      --Returns the initial default system output
        file
    Function Current_Input Return File;
      --Returns the current default input file
    Function Current_Output Return File;
      --Returns the current default output file
    Procedure Set_Input(Fyle:In File);
      --Set the current default input file to fyle
    Procedure Set_Output(Fyle:In File);
      --Set the current default output file to fyle
    Function Has_8087 Return Boolean;
      --Returns true if the system has  an  8087
        chip
End Util;
```

第六章　自定义数据类型

前几章我们重点讨论了程序的流程控制和程序组织方面的问题，意在追求良好的程序结构和设计风格。从本章开始的大部分章节，将讨论有关数据类型和数据结构方面的问题。

人类自然语言的基本语汇由名词、动词两种结构展开，其它的语言特征用作进一步的描述，以起某种说明和限制作用。就是这些元素的有机结合，给了我们通讯的工具和思维的手段。在思维表达中，说明事物名称的是名词——Ada语言称之为对象，说明实施加工动作的是动词——Ada语言称之为操作，每个对象都有它自身的属性集合——Ada语言称之为类型。

简单回忆一下就可发现，已经熟知的变量类型定义格式也可用如下方式表达：

〈对象〉：〈属性〉〈值域〉：＝〈初值〉；

通过对数据类型及其结构的描述，把客观世界的对象、属性和操作都映射到计算机的解中，这对任何程序语言来说都是重要的。

就类型而言，我们已经接触到整型、浮点型、字符型与布尔型数据类型，这些类型都是系统预定义的标准类型。本章将讨论离散类型、实数类型、子类型与派生类型，它们是由程序员根据问题空间及Ada语言语法规则自己定义的简单数据类型，从这个广义角度上将它们统称为自定义类型，以

便于教材内容的组织和实施教学。

提倡使用自定义类型的基本理由有两条：

1.可以使得数据类型能够更直接表达问题空间的特性，从语言的底层提供面向问题编程的工具，从而改善程序的表达能力，一个复杂的情报信息处理系统更需如此；

2.自定义类型的运用与语言本身强类型的要求相结合，能使隐藏在程序中的某些错误信息，在编译阶段就被排除，从而提高了程序的质量。

第一节　离散类型

根据Ada语言标准，离散类型包括枚举类型和整数类型。字符类型CHARACTER与布尔类型BOOLEAN都是预定义枚举类型。

一、枚举类型

在程序设计的过程中，常会感到有些数据很难用标准类型来描述，经过一段时间后，甚至连程序员自己也搞不清变量在当初定义时的物理含义。

例如：季节有春、夏、秋、冬的变化；月份有一至十二月；方位可分为东、南、西、北四个方向或东、东南等八个方向。若用整数来表示很容易混淆，而采用枚举类型则能较好地解决如上问题。以描述季节为例，枚举类型处理的基本思想是将季节直接定义成一个季节类型，将春、夏、秋、冬定义为季节类型的全部值域，并按照其自然状态的固有顺序

排列，以使春、夏、秋、冬分别与第一、二、三、四季度顺序关系相对应。

枚举类型定义格式：

TYPE〈类型标识符〉IS(枚举_直接量〔，枚举_直接量〉)；

例：

```
1    Type SEASON is(SPRING, SUMMER,
     AUTUMN, WINTER);
2    SEASON_F: SEASON: =AUTUMN;
```

该例分别定义了一个季节类型和一个季节类型的变量。

第1行定义了一个称为SEASON——季节的枚举类型，春、夏、秋、冬四个枚举直接量组成季节类型值的集合。

第2行定义了一个称为SEASON_F的变量，该变量为SEASON类型，初值为秋。该类型变量的值只能是季节类型值域中的某一个。

说明

1.枚举类型的值为离散量的有序集合，它通过枚举一系列枚举直接量来定义；

2.定义中的枚举直接量是该类型值集合的全部。所有直接量都是有序的，序数按其排列位置由左至右增大，依次为整数0，1，2……。因而，枚举直接量按其排列位置序号有大小可分。即SPRING＜SUMMER＜AUTUMN＜WINTER，在它们之间可进行大小测试比较，其中最大值是SPRING，最小值是WINTER。

3.枚举直接量只允许由标识符或字符组成，它们分别代表某个被枚举的事物或者某个字符，因此字符串不能被列为枚举直接量。

例：

```
Type HEXA is('A', 'B','C','D');        —正确
Type HEXA is("A","B","C","D");
                                       —错误
```

二、整数类型

某些场合，仅用到整型数值域中一个很小的一部分，这时允许自定义一个整数类型。

整数类型定义格式：

TYPE〈类型标识符〉IS RANGE〈左界〉..〈右界〉；

例：

```
1        Type YEAR is Range 1980..2000;
2        Type MONTH is Range 1..12;
3        Type DAYS is Range 1..366;
```

1行定义了一个整数年类型，取值范围1980—2000；

2行定义了整数月份类型，值域1—12；

3行定义了日类型，值域1—366（含闰年）。

说明

1.左右界也即上下界，值域可以取负数，但必须遵循右界大于左界—即上界大于下界的规则；

2.一旦进行了类型定义，就引进了一个不同于其它类型的新类型，但仍遵循强类型规则。

三、离散类型属性函数

Ada语言提供了一组函数用以计算离散类型的某些属性。

函数调用格式：

〈类型标识符〉' 〈函数名〉

常用测试函数有如下几种（设类型为T）：

T' FIRST	产生T的下界，属性值与T同类型；
T' LAST	产生T的上界，属性值与T同类型；
T' POS（直接量）	产生直接量位置值，类型为整型；
T' PRED（直接量）	产生直接量前导位值，类型为整型；
T' SUCC(直接量)	产生直接量后续位值，类型为整型；
T' VAL（位置）	产生指定位的直接量，类型同T。

注：

除以上列举的函数外，还有其它一些函数，详见Ada语言标准参考手册中的有关章节。

第二节　子类型与派生类型

一、子类型

子类型定义格式：

SUBTYPE〈子类型标识符〉IS〈基类型标识符〉
〔〈RANGE〉〈下界〉..〈上界〉〕;

例：

(1) SUBTYPE F_integer is Integer Range
0..1000;
M_integer: F_integer;

(2) M_integer: Integer Range 0..1000;

例(1)与(2)是等价的。

说明

1.本例中的F-INTEGER是子类型，而INTEGER是基类型或父本类型；

2.子类型带有基类型全部特性，差别仅在于子类型的值域范围比其父辈受到进一步的限制；

3.子类型定义并未生成一个新类型，因而父类型对子类型是向上兼容的，可以相互赋值，只是在对子类型赋值时不能超出其约束范围。

二、派生类型

派生类型定义格式：

TYPE〈派生类型标识符〉IS NEW〈基类型标识符〉〔RANGE〈下界〉..〈上界〉〕；

例：

```
Type LOCAL_COORDINATE is NEW Coordinate;
Type F_INTEGER is New Integer Range 0..1000;
Type  F_SEASON is Season Range SPRING..
AUTUMN;
```

说明

1.派生类型，定义生成了一个新类型，它的特征由其父辈类型派生而来，因此称作派生类型；

2.按Ada语言规则派生类型与其父类型是互不兼容的不同类型（不能互相赋值），但属于同一类型类，派生类型与其父类型可通过显式转换；

3.派生类型的概念可扩大到派生子程序、派生数组、派生记录等。

第三节　实数类型

实数类型包括浮点类型和定点类型，它们均用于提供实数的近似值。

一、浮点类型

浮点数由尾数与指数两部分组成，其尾数部分的有效位数是固定的。浮点类型具有相对精度，其误差界由最少有效位数指定。

浮点类型定义格式：

```
Type〈浮点类型标识符〉is Digits〈尾数有效位数〉
        〔Range〈左界〉..〈右界〉〕;
```

浮点类型定义的举例：

```
1    Type SMALL_FLOAT is Digits 5;
2    Type BIG_FLOAT is Digits 10 Range 0.0..
     1.0E6;
```

1行定义了浮点类型SMALL_FLOAT，有效位数五位，值域同预定义浮点类型；

2行定义了浮点类型BIG_FLOAT，有效位数十位，值域范围下界0.0，上界1.0E6。

说明

1.子类型、派生类型的概念均适用于浮点类型，例如：

```
Type MID_FLOAT is Digits 7 Range 1.0..
1.0E6;
```

等价于下列说明：

```
Type N_FLOAT is Digits 8;
```

```
Type M_FLOAT is New N_FLOAT;
Subtype MID_FLOAT is M_Float Digits 7
Range 1.0..1.0E6;
```

2.部分离散型属性测试函数适用于浮点子类型，除此还增加了一些适用于浮点子类型属性的测试函数，主要函数如下：

T' DIGITS	给出类型T的有效位数，值类型为整型；
T' SMALL	给出类型T的最小正数，值类型为实型；
T' LARGE	给出类型T的最大正数，值类型为实型。

二、定点类型

定点数小数点位置是固定的。

定点类型具有绝对误差，其误差界由绝对值(DELTA)指定。

定点类型定义的举例：

```
Type X is Delta 0.01 Range 0.0..
100_000.0;
```

该例定义了一个定点类型X，精度达0.01，值域范围下界0.0，上界100_000.0。

定点类型定义格式

TYPE〈定点类型标识符〉IS DELTA〈精度〉[〈下界〉..〈上界〉]。

说明

1.定点类型与浮点类型主要差别在于精度约束方式不同；

2.定点类型新增加了一组定点属性测试函数，主要有：

T' DELTA	给出子类型T定点精度值，值类型为实型；

T' FORE　　给出子类型T任一值整数部分的字符长度，包括符号位。

要点

本章涉及到的数据类型，都是由用户自行定义的简单数据类型。

（一）枚举类型小结

值集合 结　构	离散值的有序结合 (E0, E1, …En)	Ei是有序枚举直接量
操作集合	:=	赋值
	in　not in	成员关系测试
	= /= < <= > >=	关系比较
属性测试	ADDRESS　BASE　FIRST　IMAGE LAST　POS　PRED　SIZE SUCC　VAL　VALUE　WIDTH	
预定义类型	BOOLEAN	CHARACTER

注：　(1) 枚举类型变量不能由键盘直接赋值；

(2) 枚举类型可用于循环控制，例如：

```
Package Body T10 is
   Type DIRECTION is (NORTH,EAST,SOUTH,
   WEST);
   LA:Direction:=Direction'FIRST;
Begin
   Put(LA);Put('  ');LA:=Direction'SUCC(East);
   Put(LA);New- line;
For I in Direction Loop
      Put(I);Put('  ');
```

```
        End Loop;
            New-line;
        For I in 0..3 Loop
            Put(Direction'val(I));Put("  ");
        End Loop;
    End T10;
```

运行结果

```
NORTH    SOUTH
NORTH    EAST    SOUTH    WEST
NORTH    EAST    SOUTH    WEST
```

（二）整数类型小结

值结合 结　构	连续的整数集合 Rang(L)..(R)	(L),(R)为整数值域
操　作 结　合	+ − ABS + − := • • in not in • / MOD REN = /= 〈 〈= 〉 〉=	单目类 加法类 赋值 指数 成员关系测试 乘法类 关系比较测试
属性 测试	ADDRESS BASE FIRST IMAGE LAST POS PRED SIZE SUCC WIDTH VAL VALUE	
预定义 类　型	INTEGER LONG-INTEGER SHORT-INTEGER NATURAL POSITIVE	

（三）实数类型小结

值集合	实数近似值
结　构	Digits N Range (L)..(R)　N静态整数型，指定

	(浮点)	有效位数 (L),(R)指定浮点值域
	Delta D Range (L)..(R) (定点)	D静态实型,指定绝对精度 (L),(R)指定定点值域
操作集合	(同整数型)	
属性测试	ADDRESS AFT BASE DELTA FIRST FORE LARGE LAST MACHINE_OVER_FLOWS MACHINE_ROUNDS MANTISSA SIZE SAFE-LARGE SAFE-SMALL SMALL (定点)	
	ADDRESS BASE DIGITS EMAX EPSILON FIRST LARGE LAST MACHINE-EMAX MACHINE-EMIN MACHINE_MAN_TISSA MACHINE_OVERFLOWS MACHINE-RADIX MACHINE-ROUNDS MANTISSA SAFE-SAFE-LARGE SAFE-EMAX SMALL SIZE SMALL (浮点)	
预定义类型	FLOAT LONG-FLOAT SHORT-FLOAT	

习题

1.一个类型表征是什么？

2.为以下标量类型编写声明：

(1) 一个仅有负值的COUNTER（计数器）；

(2) 一个有12位有效数字的COEFFICENT(系数)；

(3) 一个精度分别为0.1米，0.1米，1.0米的坐标X、Y、H；

(4) 虹的色彩COLOR_OF_RAINBOW。

3.为下列子类型和派生类型编写声明：

(1) 只有七位精度的COEFFICIENT的子类型；

(2) 限于整数范围(1)的第二代子类型；

(3)只有七位精度的COEFFICIENT的派生类型。

第七章　构造型数据类型

以程序为中心的观点侧重于建立程序，只是当数据成为程序加工对象时，才考虑到数据。这种观点适用于数值计算问题，这类问题属于在简单数据结构上进行复杂函数的变换问题。以数据为中心的观点是把数据结构作为问题的中心部分（如数据库），而把程序看成是围绕着数据结构缓慢爬行的小虫，它时而询问，时而修改或扩充当前驻留在内存中的数据。这种观点适合于航空订票系统、信息管理系统、情报检索系统等非数值问题的解决，它们都要求采用复杂的数据结构描述系统的状态。某些科学家曾断言，程序设计以数据结构为中心的观点，将对未来程序设计语言的设计产生重大影响。

数据结构〔1.2〕

简单数据类型的值域由一系列元素组成，这些元素可通过变量名访问。

复杂数据类型研究的是一些逻辑上由多个分量组合而成的对象。组成对象的组合规则体现为数据对象的构造方法，因此也称为构造型数据类型。数组类型及记录类型都是这一种类型。

第一节　数组类型

一组类型相同并可按一定顺序排列的数均可用数组表示，构成数组的基本成分有数组名和下标，数组名标识了一组数的集合，数组中的不同元素用序标的不同值区分。数组的维数等价于序标的个数，如果数组用一个序标标识元素，则说该数组是一维的；有两个序标则为二维数组…。Ada语言对数组的最大维数未加限制。

一个数组中所有元素的类型必须相同，元素的数据类型也即数组的类型。

除了整型数外，Ada语言还允许序标为其它离散类型。

综上所述，数组声明应包括数组标识符、序标数量、类型、值域、元素类型、类型等。

Ada语言提供了数组类型，使得数组可以面向问题空间进行定义，这大大地提高了运用的灵活性和解题能力。

一、数组定义

数组定义的基本格式：

〈数组标识符〉：ARRAY(〈序标说明〉)OF〈数组的类型〉
〔RANGE〈L〉..〈R〉〕；

序标说明格式：

(1)　〔〈类型〉〕〈L〉..〈R〉｛，〈序标说明〉｝

(2)　〈离散类型〉｛，〈序标说明〉｝

例：

```
DATA: Array(Integer 1..100)of Integer;
```

定义了一个名为DATA的数组，序标为整数型，范围1到100，所有元素类型为整数型。

```
COUNT: Array(1..100)of Integer Range
0..500;
```

数组COUNT序标为整数型(整型说明可缺省)，范围1到100，元素值为整数型，范围0到500。

```
G: Array(1..100, 1..3)of Integer;
```

数组G为二维数组，序标值为整型，第一序标范围1至100，第二序标范围1至3，元素为整型。

```
Type DAY_OF_WEEK is(SUN, MON, TUR
                    WED, THU, FRU, SAT);
HOURS_WORKDE: Array(DAY_OF_
                    WEEK)of Integer Range 0..24;
```

数组HOURS_WORKDE序标DAY_OF_WEEK类型，范围从星期日到星期六共七个分量，该例表明可用一个离散类型来说明序标。

二、动态数组

上面接触到的数组其下标范围在编译阶段就已被确定，因而该数组的规格是静态的，称其为静态数组。

在数组定义中允许下标上下限只给出参变量，而不赋值，在运行实现时再具体赋值，这种下标范围在编译阶段不确定，在运行阶段才确定的数组被称为动态数组。

例如：过程

```
Procedure P_array(N: Integer) is
      A,B,C: Array(1..N) of Integer;
Begin
      For I in 1..N Loop
```

```
            A(I)：=0；
        End Loop；
            B：=A；C：=A；
End P_array；
```

该过程调用后即建立A，B，C三个数组，并将其初始化为零。

三、数组赋值

数组赋值有几种情况，一是给数组中某一分量赋值；二是给数组中某一组分量赋值；三是将一个数组值赋给另一数组。下面结合具体实例来说明数组赋值过程。

赋值形式（一）：

〈数组名〉(〈序标〉[..〈序标〉])：=〈表达式〉；

例：

```
1       COUNT(1)：=100；
2       COUNT(2..100)：=80；        --下标2至100间元素赋80
3       DATA：=COUNT；              --下标2至100间元素与COUNT同
4       DATA(51..100)：=COUNT(1..50)
```

说明

1．一次可为一个分量赋值，也可同时为多个分量赋值；

2．2行、4行属分片(slices)赋值形式。

3．可将一个数组值赋给另一个数组，但有一个附加条件，就是互相赋值的两个数组必须兼容。按照规则，在同一个语句中声明的数组才是兼容的，否则不兼容。

例如：

相兼容数组说明的举例

```
COUNT,DATA: Array(1..100)of Integer;
```

不兼容数组说明的例

```
COUNT: Array(1..100)of Integer;
DATA: Array(1..100)of Integer;
```

赋值形式(二)：

例如：

(1) 对位关联

```
COUNT(1..50): =(0,0,0,others=>10);
```

(2) 点名关联

```
COUNT: =(1=>0,2=>0,3=>0,others=>10);
COUNT: =(i  | 2 | 3=>0,others=>10);
COUNT: =(1..3=>0,others=>10);
```

(3) 混合关联

赋值形式(二)是将元素值的集合组合成对应数组分量值的集合. 由赋值号右端确定的运算关系表达式称为聚集值或数组聚集值(ARRAY AGGREGATES)。

数组聚集值格式：

〈聚集值〉::=(〈分量关联〉{,〈分量关联〉});

〈分量关联〉::=[〈选择〉{|〈选择〉}=>]〈表达式〉

四、数组类型定义

关于数组类型的基本概念：

在类型声明中，下标取值范围已确定的称为受约束数组类型，否则为未受约束数组类型。

受限数组类型定义格式：

TYPE〈数组类型标识符〉IS ARRAY(〈下标说明〉)OF〈分量类型说明〉;

例:

```
Type MY_VEC is Array(1..100)of Integer;
A_1, A_2: My_vec;
```

等价于下列声明

```
A_1, A_2: Array(1..100)of Integer;
```

非限制型数组类型定义与受限数组类型定义格式差别在于下标说明不同。

非限制型数组类型下标说明格式:

(〈序标类型标记〉RANGE〈〉{ {, 〈序标类型标记〉RANGE〈〉} })

例:

```
Type SM11 is Array(Integer Range<>)of Integer;
Subtype SM22 is S 11(1..10);

Type S1 is Integer Range 1..Integer' Last;
Type SM11 is Array(S1 Range<>, Character Range<>)of Integer;
Subtype SM22 is SM11(1..10,'A'..'Z');
```

五、数组类型属性运算

设数组为A, 则:

A' FIRST	给出第1个序标的下界值, 类型同序标;
A' FIRST(N)	给出第N个序标的下界值, 类型同序标;
A' LAST	给出第1个序标的上界值, 类型同序标,
A' LAST(N)	给出第N个序标的上界值, 类型同序标;
A' LENGTH	给出第1个序标范围值的个数, 类型为整型;

A' LENGTH(N)给出第N个序标范围值的个数。类型为整型。

六、应用举例

(1) 数组元素的类型还可以是数组类型

```
Package Body T14 is
    Type ar is Array(1..5)of Integer;
         A: Array(1..3)of ar;
Begin
         For I in 1..3 Loop
              For J in 1..5 Loop
                   A(i)(j):=j;
                   Put(A(i)(j), 9);
              End Loop;
              New_line;
         End Loop;
    End T14
```

运行结果

```
1        2        3        4        5
1        2        3        4        5
1        2        3        4        5
```

实际上该例定义了一个二维数组。

(2) 利用数组设置一个按后进先出方式工作的堆栈

压入过程定义为PUSH

弹出函数定义为POP

```
Package Body T15 is                    --主程序T15
      MAX: Constant:=100;
      S: Array(1..max)of Integer;
      TOP: Integer Range 0..max:=0;
      Procedure PUSH(X, Integer) is
                                       --压入过程为
           Begin                        PUSH
```

```
                TOP: =TOP+1;
                S(TOP): =X;
            End PUSH;
        Function POP Return Integer is
                                    --弹出函数为POP
            Begin
                TOP: =TOP-1;
                Return S(TOP+1);
            End POP;
    Begin
        For I in 10..15 Loop        --压入
            PUSH(I);
        End Loop;
        For I in 1..5 Loop;         --弹出
            Put(POP(), 6);
        End Loop;
End T15;
```

运行结果

```
    15    14    13    12    11
```

数组数据类型小结

值集合 结　构	可索引的相同类型集合 ARRAY(序标 {,序标}) OF 类型 (无约束数组)	序标是无约束离散类型
	ARRAY(序标约束) OF类型 (约束数组)	序标约束是离散类型列表
操　作 集　合	& := AND OR XOR NOT IN　NOT IN = /= 〈 〈= 〉 〉= NOT	加法类(一维) 赋值 逻辑 成员关系 关系比较 单目 聚集

属　性	ADDRESS　BASE　FIRST FIRST(N)　LAST　LAST(N) LENGTH　LENGTH(N)　RANGE RANGE(N)　SIZE
预定义类型	STRING

第二节　记录类型

数组标识了类型相同的一组数的集合；记录标识的是类型不同的一组数的集合。由于集合元素类型的不同，使得记录与数组在数据结构的构造方法上形成了各自的特色。

例如：

```
Package Body TI6 is
    Type DATE is          --定义记录类型DATE
        Record
          YEAR: Integer Range 1986..2000;
          MONTH: Integer Range 1..12;
          DAY: Integer Range 1..31;
          FESTIVAL: STRING(6);
        End Record;
      Today: Date;          --定义记录TODAY
    Begin
        TODAY.YEAR: =1986;
                          --记录分量赋值
        TODAY.MONTH: =10;
        TODAY.DAY: =1;
        TODAY.FESTIVAL: ="国庆节";
            Put(TODAY.YEAR,4);
            Put("年");      --输出记录分量值
            Put(TODAY.MONTH,2);
```

```
        Put('月');
    Put(TODAY.DAY.2);
        Put('日');
    Put(TODAY.FESTIVAL);
End T16;
```

运行结果

1986年10月1日国庆节

说明

（1）该例体现了记录类型的定义、给分量赋值与输出分量的三个过程；

（2）DATE为日期记录类型，TODAY为日期类型记录，它包含四个分量：年、月、日和节日；

（3）记录由记录名和分量标识符组成，分量说明包括分量标识符、类型与值域、初值等。

一个记录如果其分量值域都被约束，则该记录是受限制的，否则是不受限制的。另外，一个记录如果其分量是不可选的，则该记录属非变体型(记录体不变化)记录，否则为变体型(记录体可变)记录。据此将受约束、非变体型记录归结为基本类型，余者归类为非限制型和变体型记录。对它们分别进行讨论。

一、基本记录类型定义

记录类型定义格式：

```
TYPE〈记录类型标识符〉IS RECORD
                {〈分量说明〉；}
            END RECORD;
```

分量说明格式：

〈分量标识符〉：〈类型〉[〈值域〉][：=〈初值〉]；

记录类型中的分量说明具有与变量说明相类似的形式；同样也类似于数组名不能与下标分开，分量标识符必须与记录名一起使用。

分量表示格式：

〈记录名〉.〈分量标识符〉

说明

（1）数组元素可以定义为记录类型，记录分量也可以定义成数组类型，因而可依据问题域组合成很复杂的数据结构；

（2）按照参数匹配的基本规则，记录分量赋值表达式允许以聚集值形式出现。

例：

```
TODAY: =(1986, 10,1,'国庆节');
```

或

```
TODAY: =(Year=>1986, Month=>10, Day=>
          1, Festival=>'国庆节');
```

与T16例中对应的记录分量赋值语句等价。

二、非限制型记录类型定义

允许记录类型定义中的分量是不受约束的，而在记录类型生成具体记录对象时或者在运行时，再对其加以约束。分量空间一经限制，就不可再变。

例：

—未受限制记录类型定义举例：

```
Type DATE(N: Integer: =6) is
    Record
```

```
        FESTIVAL: STRING(N);
    End Record;
```

—生成受限制记录举例：

```
        X_1: DATE(3);
        X_2: DATE(N=>5);
```

—生成未受限制记录举例：

```
        Y_1. Y_2: DATE;
    Begin
      Y_1: =(4, "元旦");          —Y_1记录中节日分量长度4个字符
      Y_2: =(6, "国庆节");        —Y_2记录中节日分量长度6个字符
    End;
```

在记录类型定义中，项(N: Integer: =6)称为判别项，因为分量的空间长度与记录形式需通过该项来识别。所以非限制型记录定义差别就在于在定义中增加了判别项。

判别项格式如下：

〈参数标识符〉:〈类型说明〉[(〈值域〉)][: =〈初值〉];

三、变体记录类型定义

将判别项与情况语句结合起来使用，可使得记录中的分量成为可选择的成分。

例：

—定义变体记录类型举例：

```
        Type DATE(UNIT: Integer Range 0..1;
                                    =0) is
        Record
            YEAR: Integer Range 1986..2000;
            MONTH: Integer Range 1..12;
```

```
        DAY: Integer Range 1..31;
          Case UNIT is
              When 0 =>Null
              When 1 =>FESTIVAL: STRIN-
              G(10);
          End case;
    End Record;
```

--定义记录子类型举例：

```
    Subtype DATE_0 is DATE(0);
                    --该子类型没有节日分量
    Subtype DATE_1 is DATE(1);
                    --该子类型带有节日分量
```

--定义记录

```
    DATE_N0: DATE(UNIT=>0);
```

或

```
    DATE_N0: DATE_0;
                    --该记录没有节日分量
    DATE_N1: DATE(UNIT=>1);
```

或

```
    DATE_N1: DATE_1;
                    --该记录带有节日分量
```

关于判别项及情况语句的使用格式参阅以前章节的有关部分。

四、记录数据类型小结

值集合 结构	不同类型分量的结合 RECORD 〈分量表〉 END RECORD		分量表由分量说明组成
操作集合	:= IN =	NOT IN /=	赋值 成员测试 关系测试

属　性	ADDRESS BASE CONSTRAINED SIZE (记录类型)
	FIRST-BIT LAST_BIT POSITION (记录分量)

要点

(1) 数组与记录都是构造型数据类型，都是用一个名称标识一组数据的集合；

(2) 数组描述的对象是类型相同的一组数据集合，记录表述的对象是类型不同的一组数据集合；

(3) 数组中的分量用不同的下标区分，记录中的分量用不同的标识符区分。

习题

1.对N个无序的数进行排序，并按先大后小顺序输出。

2.对N个无序的数进行排序，打印出第k（k＜N）个最小的数。

3.编一程序打印输出N以内的素数。

4.建立一个一维整型数组，该数组元素可大于M＋N。在不借用其它变量情况下，请设计一个程序，以最快的速度将M个元素与N个元素的位置进行交换，但各自内部的排列关系不能打乱。

例如：交换前

M	N
1 2 3 4 5 6 7 8 9	21 22 23 24

交换后

N	M
21 22 23 24	1 2 3 4 5 6 7 8 9

5.设有N个学员，每位学员有学号、姓名、三门功课（数学、物理、语文）成绩及平均成绩等项。试编一程序记录每一位学员的学号、姓名、三门功课成绩，算出每人的平均成绩，最后按平均成绩排序，以先高后低顺序输出。

6.输入两个复数，计算并输出它们的和数与乘积。

7.建立矩阵A和B，矩阵A和B相乘得C．输出矩阵C。注：要求矩阵A的列数和矩阵B的行数相等．否则无法计算。例如假定矩阵A为四行三列，矩阵B为三行四列，按矩阵乘法规则，矩阵C应该为四行四列。

8.建立一按先进先出方式工作的队列。

第八章　专用数据类型

本章介绍访问类型（ACCESS TYPE）和私有类型（PRIVATE TYPE），这两种数据类型在某些专门领域中具有特殊用途，因此将它们作为专用数据类型或特殊数据类型进行讨论。

第一节　访问类型

先分析一个应用访问类型的简单举例：

```
Package Body T23 is
Type G_Integer is Access Integer;
        --将G_Integer定义为访问类型，访问对象
          类型是Integer;
X,Y:G_Integer;
        --将X,Y定义为G_Integer类型的变量；此
          时X,Y不指向任何对象，初值为空NULL
A:Integer;
Begin
X:=New Integer; Y:=New Integer;
        --分别生成新的Integer类型对象，并将各个
          对象的地址分别返回给X,Y，此时X,Y分
          别指向两个不同的对象。
A:=1;X.all:=5;Y:=X;
        --将5赋给X，并将X复制给Y.
Put(A,8);Put(X.all,8);Put(Y.all,8);
        --输出X,Y访问对象的值
```

End T23;

运行结果

1 5 5

该例说明了如下几个问题：

（一）访问类型定义格式：

TYPE〈访问类型标识符〉IS ACCESS〈访问对象类型标识符〉；

要定义的访问类型用类型标识符指明，访问对象的类型由访问对象类型标识符指明，对象类型可为任何子类型.

（二）访问类型变量定义与其它类型变量定义格式相同.

（三）分配算符应用格式：

〈访问型变量名〉：=NEW〈对象类型〉| NEW〈限定表达式〉；

NEW在此被称为分配算符，它有两个作用，一是生成一个新的对象；二是将新的对象的地址回送给访问变量。

（四）被访对象如果是简单数据类型，那么该类型的域仅由一个唯一的量组成，如被访对象是一个构造型的复杂数据类型，则该类型的全部分量组成其域。

域的说明格式：

〈访问变量〉.〈域名〉

若为全域则用关键字ALL.

访问对象为复杂类型的举例

```
Package Body T24 is
      Type Int_R is Record
            A,B:Integer;
      End Record;
```

```
        Type Acc_Int_R is Access Int_R;
        P,Q:Acc_Int_R;
Begin
        P:=New Int-r;
        P.A:=5;P.B:=10;Q:=P;
        Put(P.A,8);Put(P.B,8);Put(Q.A,8);
        Put(Q.B,8);
End T24;
```

运行结果

```
    5       10       5       10
```

注，按Ada语言规则，允许以聚集形式赋值，如

```
        P, =New Int-R (A =) 5,B=) 10);
```

一、不完整类型说明

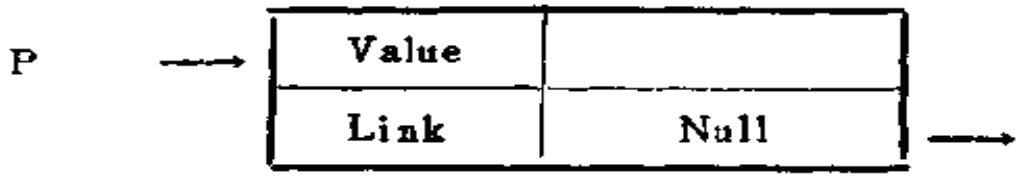
复杂数据类型的分量可以是一个访问类型，而且允许访问对象就是该数据类型自身，这就是递归访问。这种类型的说明要求对一个或多个类型有个领先的不完整类型说明。例如：

```
Type Int_R;
                --不完整类型说明
Type Acc_Int_R is Access Int_R;
Type Int_R is Record
      A:Integer;
      B:Acc_Int_R;
End Record;
                        --定义了一个具有两个分量的
                        记录，其中一个分量为访问类
                        型，访问对象是该记录本身
P,Q:Acc_Int_R;
                        --定义P,Q为访问类型变量
```

二、应用举例

访问类型在组织动态数据结构领域中（如链表、树、图等)有着很重要的应用价值．这里以最简单的向前链表为例，说明访问类型运用的一些基本方法。

向前链表是以记录作为其基本元素的一种数据结构．每个记录构成链中的一个结点．在每个记录内部有一个指针，用于指向链表中的下一个记录．若指针值为 NULL．则为空．表示链表结束。向前链表中元素的物理表示如下图：



其中P是指针．它指出该元素的起始地址；Value是该记录的值域，Link是该记录的链域，用以指出其后续元素的起始地址。

链表处理属于非数值算法。链表处理基本技术有链接、插入、删除、检索、排序。

链接——将一个新元素加到链表的末尾；

插入——将一个新元素加到链表的中间；

删除——从链表中取消一个指定的元素；

检索——从链表中找出符合给定条件的元素；

排序——使链表中元素排列顺序按某种要求重新排列。

链表所拥有的元素数量即链表长度，是不确定的。链表在执行阶段动态地形成．它所占有的内存空间不一定是连续的，链中元素一旦被删除(没有指针指向它时)，就成为"垃

取”，其所占空间自动地从链中释放并被系统回收。

例如：

一个学生记录值域有学号NUM，各门功课平均分数RES两个分量。现将N个学生记录组成一个具有N个结的向前链表，并对其进行各种操作。

```
Package Body T26 is
--类型定义
    Type RESULT;
    Type Acc_RES is Access RESULT;
    Type RESULT is Record
        NUM,RES:Integer Range 1..100;
        LINK:Acc_RES;
    End Record;
--定义指针，P_Head指向头部，P指向尾部，Q移动指针
    P_HEAD,P,Q:Acc_RES;
    N,P_G:Integer Range 1..100;
Begin
    Put("键入学生人数:");Get(N);
--生成记录
      Put("键入成绩:");
For I in 1..N Loop
    Q:=New Result;
     Get(P_G);
    Q.Num:=i;          Q.Res:=P_G;
--生成头部
    If I=1 Then
       P_Head:=Q;        P:=Q;
    Else
--链接
    P.Link:=Q;         P:=Q;
    End if;
End Loop;
--插入
```

```
    Q:=New Result;  Q.Num:=50;      Q.Res:=95;
   P:=P_Head.Link;P_Head.Link:=Q;Q.Link:=P;
--删除
    P_Head.Link:=P;
--检索，找出成绩95分以上的学生
Q:=P_Head;
  For I in 1..N Loop
       if Q.Res)=90 Then
            Put(Q.Num,8);Put(Q.Res,8);
       End if;
            Q:=Q.Link;
 End Loop;
End T26;
```

附带指出，该例仅用于说明访问类型的一般应用方法，而未讨论有关单向链表处理的具体算法。有关单向链表、双向链表及树形结构等的链接、检索、排序算法，需要专门讨论，有兴趣的读者可参阅有关数据结构与算法设计方面的资料。

第二节　私有类型

在程序包一章中，曾提及关键字PRIVATE的使用。Ada语言还提供了私有类型。与其它类型相同的是：私有类型也定义了一组值的集合和操作集合；所不同的是：私有类型提供了另一种形式的封装机制，使得除经程序员特意说明的某些操作功能以外的所有操作权限，均被屏蔽。本节将对私有类型的使用作一简要说明。

(1) 私有数据类型分私有类型和受限私有类型两种，分别由关键字PRIVATE和关键字LIMITED PRIVATE定

义：

例如：

```
Type BUFFER is Private;
                    --定义私有类型 BUFFER

Type COMPLEX is Limited Private;
                    --定义受限私有类型COMPLEX
```

（2）私有类型与受限私有类型的差别在于，

私有类型允许对类型标识符、用以赋值的预定义函数进行访问，以及进行相等或不相等测试操作；

受限私有类型则对赋值、相等或不相等测试功能做了进一步屏蔽，使得程序员得以最大限度地控制外部对信息使用的权限.

（3）私有类型仅用于程序包内. 程序包声明部分的关键字PRIVATE用于将声明部分分离为可见部分和私有部分，私有部分的信息对外是不可见的。

一般要求私有类型在声明的可见部分作定义说明，在私有部分必须按序作完全定义说明。

例如：

```
Package EX1_01 is
      Type NUMBER is Private;
      Function MAKE Return Number;
                        --定义私有类型NUMBER

Private
                        --尔后部分为不可见部分
      Type NUMBER is New Integer;
                        --对私有类型NUMBER
                          作完全说明

End EX1_01;
```

要点

(一)访问数据类型小结

值集合 结构	访问值指明对象的值集合 ACCESS（子类型）	子类型指明被访对象类型
操作集合	:= IN　NOT IN =　/=	赋值 成员测试 关系测试
属性	ADDRESS　BASE SIZE　STORAGE_SIZE	

(二)私有数据类型

值集合 结构	对外部屏蔽 对外部屏蔽	
操作集合	:= IN　NOT IN =　/= (私有类型PRIVATE)	赋值 成员测试 关系测试
	IN　NOT IN 说明中定义的操作 （函数或过程等） (受限私有类型LIMITED PRIVATE)	成员测试
属性	ADDRESS　BASE　SIZE 带判别式的私有类型： CONSTRAINED	

习题

1.输入一整数数列，遇到0时停止，建立偶数链表和按从小到大排列的奇数链表。

2.建立一双向链表(每个元素中有两个指针，一个指向其前一元素，一个指向其后一元素)，并对其进行插入、删除与排序操作。

第九章　文件的输入/输出

这里讨论的文件是存放于外部设备，由用一文件名进行标识的数据文件。数据文件的输入/输出功能由预定义程序包提供。因此讨论输入/输出实际上就是研究如何利用预定义程序包资源来达到预期目的。由于不同机型配置了不同规格的预定义程序包，这就形成各自的输入/输出环境，自然Janus/Ada与Ada语言标准也就存有较大差异。因而本章讨论的内容是非标准的。

Janus/Ada的输入/输出环境分别由BLKIO，FLOAT-IO，LONGIO，IO三个程序包定义(见第五章附录)。Janus/Ada的IO程序包提供了文件输入/输出的基本操作，因此本章仅对该包的情况作一简要说明，其它情况均可通过分析有关程序包获得，就不再赘述。

一、文件标识符

Ada语言文件分两种，一种是存贮在外部设备上的文件，称为外部文件；一种是驻留于内存缓冲区中的文件，称为内部文件。内部文件是程序与外部文件交换信息的数据通道，数据经内部文件传送到外部文件的过程称为输出，反之称为输入。

外部文件与内部文件均要用不同的标识符来标志，其中外部文件按操作系统的标准文件格式标识，

外部文件标识格式：

[〈驱动器号〉:] [\路径\〉] 〈文件名〉. [〈后缀〉]

例如：

C:\ADA\G_xyh.Txt
—C驱动器，子目录ADA，文件名G_XYH，后缀TXT（文本）文件

\ADA\G-XYH
—当前驱动器，子目录ADA，文件名G-XYH

G_XYH
—当前驱动器，当前目录，文件名G_XYH

内部文件标识符定义格式：

〈文件名〉:FILE;

例如：

SOURCE:File;
—SOURCE被定义为文件类型的内部文件名

二、文件操作模式

文件操作模式是指允许对其实施操作的方式。操作模式规定了可以对文件实施操作的权限。基本模式有三种：

(1)	Read_Only	只读方式(只允许实施读操作)
(2)	Write_Only	只写方式(只允许实施写操作)
(3)	Read_Write	读写方式(既允许读又允许

写操作）

三、打开文件

通过缓冲区（被命名为内部文件）将程序与外部文件相连接，叫做打开外部文件。打开文件是通过调用过程OPEN实现的。要对文件实施读写操作，必须先打开该文件。

被打开的外部文件必须是在被指定的外部设备上已经存在的文件；内部文件标识符必须是在程序的声明部分已经被声明为文件类型的标识符。

打开文件过程调用格式：

```
OPEN(Fyle: In Out File;Name:In String;Mode:
                                    In File_Mode);
```

其中

FYLE	内部文件名
NAME	外部文件名
MODE	操作模式

操作模式被定义为枚举类型FILE_MODE.

例如：

```
Open(Source,"AA",Read_Write);
          --通过缓冲区Source,打开一个名为AA
            的外部文件
          --操作模式为可读写
```

四、生成文件

生成文件是指生成一个外部设备上还不存在的新文件，该文件生成的同时也就被打开，因而无须再去打开。生成文件也是通过过程调用来实现的。

生成文件过程调用格式：

```
CREATE(Fyle:In Out File; Name:In String; Mo-
                                de:In File_Mode)
```

例如：

```
Create(FG_name,'C:\ADA\G_name', Read_
Write);
  --通过缓冲区FG_name. 在C盘的ADA子目录上生
    成一个名为G_name的文件
  --操作方式为可读写
```

五、基本写操作

写操作是将指定数据输出至外部文件中。

写操作过程调用格式：

```
PUTW(Fyle:In Out File; Str: In LString;
                          Width:In Integer);
```

(1) STR是字符串，类型为长串型 LSTRING，其定义

```
Subtype LString is String(255);
```

(2) WIDTH是指写入文件中的串STR所占的场宽，当场宽超过串实际长度时，串靠左排列，剩余空间用空格填充。

例：

```
Putw(destination,'assshdd',10);
Putw(destination,stemp,16);   --stemp是串变量
```

六、基本读操作

读操作是将外部文件中的数据读出来，它是通过读函数GET_LINE实现的。

读调用格式：

```
GET_LINE(Fyle:In File);
```

它的作用是通过指定的缓冲区读入一数据，返回的数据是字串型。

例：

```
G_X:=Get_line(destination);
        —将读出的数据赋给字串型变量G_X
```

七、关闭文件

一般缓冲区填满后才将其中的数据写入文件；缓冲区中最后剩存的内容，需通过关闭文件方式写入。因此，对文件操作完毕应及时将其关闭，否则可能使最后滞留在缓冲区中的数据被丢失。

关闭文件格式：

```
CLOSE(Fyle:In Out File);
```

关闭由Fyle指定的文件。

八、删除文件

删除文件是从外部存贮设备上删除一个指定的文件。

删除文件格式：

```
DELETE(Name:In String);
```

NAME是外部设备上的文件标识符，要删除的文件应是已经存在的文件。

九、文件结束状态测试函数

有时读文件时不知文件是否已经结束，此时可用测试函数进行判别，返回函数值是布尔型，结果若为真则表明文件已经结束，否则未结束。

测试函数调用格式：

(1) END_OF_FILE(Fyle:In File);
--用于对文本文件(ASCII码文件)进行测试

(2) EOF(Fyle:In File);
--用于对压缩文件(如16进制压缩码文件)进行测试

例如：

```
If End_of_File(destination)=True Tren
        Close(destination);
Enf If;
```

要点

1．写入文件的顺序是：

(1) 如果文件已存在，就打开文件；否则生成一个新文件；

(2) 对文件进行写操作；

(3) 写毕关闭文件；

2．读文件的顺序是：

(1) 打开要进行读操作的文件；

(2) 测试文件是否已经结束，如未结束，则实施读操作；否则关闭该文件；

3．只要内存容量充足，允许同时打开多个文件(或通过修改根目录中系统文件CONFIG.SYS设置的文件数来实现同时打开多个文件)，每个文件均需使用不同的标识符．

4．基本操作仅提供顺序文件的存取方法，且每个文件中的数据必须是同一类型(字符串型)；当存取对象是其它数据类型时，可用BLKIO，FLOATIO，LONGIO包中有关

的输入/输出资源进行转换.

习题

将一整型数组的数据写到一外部文件中(用10位宽度写入). 然后将该文件中的数据读入一浮点型数组. 再将浮点型数组的数据写入另一文件中(用20位宽度写入)。

第十章　异常处理

异常(EXCEPTION)是指在运行过程中出现的意外事件。例如：分母出现零、数据超出函数定义域、数据产生溢出等。这些情况的出现会使得程序不能按预定要求正常运行，被迫强行中断或进入死锁状态。而在许多情况下(如自动化系统中)，则要求即使出现异常问题，也不能中断正常运行。因而就提出了在异常出现时，如何对其进行处理的问题。

Ada语言提供了这种对异常进行处理的功能。使用异常处理的手段，可使错误的影响范围缩小到最低限度，以确保整个系统的正常运转。

程序中异常处理的过程分为：声明、引发和处理三个部分。

一、声明异常

Ada语言预定义了如下异常：

CONSTRAINT_ERROR	范围、序标或判别式约束违例
NUMERIC_ERROR	计算结果超出给定对象的值域
PROGRAM_ERROR	一个选择语句不包括在所有备选之中 或检测出一个错误条件
STORAGE_ERROR	动态存贮区分配过大

TASKING_ERROR　　任务通讯期间发生异常

Ada语言还允许程序员用声明的方式定义自己的异常，其作用域与对象声明的作用域相同。

异常声明格式：

〈异常标识符〉［，〈异常标识符〉］：EXCEPTION；

其中EXCEPTION为异常保留字。

例如：

```
ZERO_ERROE: Exception;
PARITY_ERROR,ABOVE_LIMITS,BELOW_
                    LIMITS:Exception;
```

二、引发异常

当发生异常的条件已经具备时，将该异常提交系统处理，这个过程称为引发异常。

引发异常格式：

RAISE［〈异常标识符〉］；

RAISE为异常引发保留字。

说明

1．引发异常是一个显式语句，如同其它语句一样，它可以出现在程序的任何地方。

2．按照规则，某一时刻只能有一个异常被激活；若在异常处理过程中又有另一个异常发生，则后者将使前者挂起。

3．引发语句的作用是捕获异常，并暂时中断正常的处理次序，把控制转移到相应的异常处理段。

4．引发语句中的异常名可以省略，这时它只能出现在

异常处理段中，其作用是再次引发异常，并将控制重新转移到该异常处理段的初始位置。

三、处理异常

异常处理段提供了当异常被引发时该如何处理的全部操作，它可由程序员在程序中设置。

异常处理段格式：

```
EXCEPTION
        When 〈异常名〉 =〉 处理语句_序列;
          [ {〈异常名〉 =〉 处理语句_序列} ;]
            [ OTHERS=〉 处理语句_序列;]
```

EXCEPTION为异常处理段标志。

说明

1. 处理段中列出了每个可见的异常名以及为响应该异常需要执行的语句序列。others子句处理本处理段中没有被列名的所有其它异常。

2. 通常，一旦异常处理段处理结束，控制并不返回到引发异常时的中断点，而是返回到包含异常处理段的程序单元或程序块的底部。

3. 在实用中，引发异常后的处理方式有如下几种，

(1) 抛弃该单元的其余可执行部分。

(2) 试图重新操作。

(3) 使用备选路径处理。

(4) 纠正错误部分。

例如：

```
A,B,C:Integer;
```

```
   --声明异常
    ZERO_ERROR:exception;
Begin
              Put("输入A, B: ");
              Get(A);Get(B);
    --引发异常
    If B=0 Then
          Raise Zero_Error;
    End if;
    C:=A/B;
    Put(C,8);
    --异常处理
EXCEPTION
    When ZERO_Error=>
          Put("除数 B=0");
```

要点

1.按照发现的情况可将错误分为两级。一、编程级，这类错误可以通过显式的编程测试来处理。二、运行级，这类错误是在运行时动态发现的。

2.Ada语言提供的异常处理机制，不仅易于理解和使用，而且对应于错误处理的程序并不影响对程序正常的无故障操作的理解，从而保证了系统性能在异常情况下不受影响。

第十一章　任务

Ada语言提供的任务是另一种形式的基本程序单元。它是一种能与其它程序单元并行执行的程序实体。

所谓并行执行，在多个CPU情况下，是指多个任务同时在各自的CPU上运行，它在逻辑上与物理上都是并行的；在单CPU情况下，则是以交替的方式被运行，它仅在逻辑上是并行的。鉴于它们之间本质上的一致性，通常都不加区分地统称为并行处理。

任务的基本状态有四种：

（1）运行态

指已取得CPU的使用权，处于运行过程中的任务。处于运行态的任务的数目总是不超过CPU的数目。

（2）冻结态

指因等待某种条件（如延迟时间未到，或等待外部设备准备好的信号，或等待其它任务的呼叫等），而不能运行的任务（即便是此时CPU空闲着）。

（3）就绪态

指已具备其它运行条件，仅在等待使用CPU的任务。就绪态的任务一旦取得对CPU的控制，即可运行。

如果处于就绪态的任务多于CPU的数量，就需

要排队等待，每次总是优先级别最高的先执行；有时也可根据需要给较紧迫的任务预先赋给较高级别的优先权。

（4）终止态

是指再也不被激活运行的任务。

在具体应用中，任务同子程序有某些相似之处，即其它程序单元可以包含多个任务，一个任务体中也可包含有其它程序；但一个任务不能单独存在，也不能单独编译，它只能出现在其父单元中。

第一节　任务的定义

任务可以有多个入口，以用于被其它任务呼叫，与入口说明相对应，在任务体中应有一个相应的接受语句，而且允许一个特定的入口语句有多个与之对应的接受语句。当入口被呼叫时所采取的行动由对应的接受语句指明。允许入口带参数，参数是任务间交流信息的重要通道。

在时间上有依从关系的任务，且能协调一致地工作称为任务间的同步。要达到同步的要求，早到达的任务就要等待晚到达的任务，以便进行任务间的交会（RENDEZVOUS）。入口语句提供了响应任务间会合呼叫的接口；接受语句的执行则标志任务间交会的实现。

任务由声明和体两部分组成；任务的定义包括规格说明和体的定义。

任务规格说明格式：

TASK [TYPE]〈任务名〉[IS

{ENTRY〈入口名〉[(〈参数说明〉)] ;}
{〈表示子句〉;}
END〈任务名〉;]

任务体定义格式：

TASK BODY [任务名〉IS
[〈说明部分〉;]
BEGIN
〈语句-序列〉;
{[〈接受语句〉] ;}
END〈任务名〉;

接受语句格式：

ACCEPT〈入口名〉[形参说明] ;[DO
〈语句-序列;〉
END [〈入口名〉] ;]

会合后的动作用DO…END子句限定。若无动作，则可省去本子句，此时它的作用仅是使任务同步。

任务规格说明的举例：

(1)
```
Task CONSUMER is
  Entry RECEIVE_MESSAGE(M:In String);
End CONSUMER;
```

任务CONSUMER有一个入口，入口名RECEIVE_MESSAGE，M是入口参数，为输入方式，字符串型。

(2)
```
Task PROTECTED_STACK is
   Pragma PRIORITY(7);
     Entry POP (ELEMENT; Out Integer);
     Entry PUSH(ELEMENT; In  Integer);
End PROTECTED_STACK;
```

该任务优先级别为七级，有两个调用入口。

(3)
```
Task PRODUCER;
```

该任务无入口，象这种可以调用其它任务，而本身不为

其它任务提供服务的任务，被称之为原动任务(ACTOR TASK)；相反，本身提供入口，但不调用其它任务的这类任务，被称之为侍从任务(SERVER TASK).

(4)
```
Task Type PRINTER_DRIVER is
     Entry Put_Data(A: Itiem);
     Entry Reset;
End PRINTER_DRIVER;
```

定义任务类型 PRINTER_DRIVER，并声明其有两个入口.

任务声明与体定义完整的举例

```
Task SEQUENCER is
     Entry PHASE_1;
     Entry PHASE_2;
     Entry PHASE_3;
End SEQUENCER;
Task Body SEQUENCER is
     Begin
     Accept PHASE_1;
     Accept PHASE_2;
     Accept PHASE_3 Do
          INTIATE_LAUNCH;
     End PHASE_3;
End SEQUENCER;
```

第二节　任务控制语句

Ada语言提供了一组用于任务调度、交会控制的语句，为便于叙述，将它们放在一节中作一简要说明。

一、入口呼叫语句

具有入口标识的任务，需经入口呼叫调用才可能引起执

行。

入口呼叫语句格式：

〈任务名〉.〈入口名〉［〈实参〉］；

说明

①入口呼叫格式及其参数关联规则与过程调用相同。但过程可以调用自身以实现递归；而任务呼叫自身则会导致死锁。

②当一个任务发出入口调用后，它自身被冻结进入等待状态，直到与ACCEPT相结合的语句执行完毕，然后调用任务和被调用任务就重又各自并行执行。这一段过程就是任务的交会。

③原动任务一般从被激活时起就进入就绪态等待执行。

二、延迟语句

延迟语句用于使任务挂起，挂起的时间不少于由表达式给出的时间。

延迟语句格式：

DELAY〈表达式〉；

例：

```
Delay 2.0;
```

该语句使正在执行的任务至少挂起2秒钟，延迟到期时，继续执行被挂起的任务。

说明

表达式的值必须是预定义定点类型DURATION；它的值以秒为单位，系统提供的最大值不小于正负86400秒（一天），最小值不大于20毫秒；若表达式结果为负，则等同于

取零值.

三、选择语句

入口调用要求调用任务挂起等待一个不确定的时间，以期得到被调任务的响应。这对不许可延迟的任务是不利的。选择（SELECT）语句提供了一个途径，使得程序员可以去控制任务交会的执行。

选择语句有条件入口调用、限时入口调用和选择等待三种形式.

（1）条件入口呼叫

条件入口呼叫格式：

```
SELECT
        〈入口呼叫语句〉;
        [〈语句_序列〉;]
ELSE
        〈语句_序列〉;
END SELECT;
```

如果调用任务与被调用任务能立即会合，则执行与调用入口对应的语句序列，否则不等待，继续往下执行ELSE后面的语句.

例：

```
Select
    Single_Teller.Balance (My_Name, Current_Balance);
Else
    Null;
End Select;
```

（2）限时入口呼叫

限时入口呼叫格式：

```
Select
    〈入口呼叫语句〉;
    [〈语句_序列〉;]
OR
    〈入口呼叫语句〉;
    [〈语句_序列〉;]
 { [OR
    〈延迟备选〉;] }
END SELECT;
```

允许在延迟备选中使用延时语句，发出的限时入口呼叫在给定的延时期限内得不到被呼叫任务的响应，则取消该呼叫。

（3）选择等待语句

选择等待语句提供了对任务体中接受语句的控制。

选择等待语句格式：

```
Select
    〈选择备选〉;
{OR
    〈选择备选〉;}
[ELSE
    〈语句_序列〉;]
END SELECT;
```

选择等待必须至少包含一个接受语句的备选，此外允许包含多个延迟备选语句和一个终止语句，选择等待语句只允许出现在任务体中。

选择备选允许以选择（When〈条件〉=〉）开始，下面选择等待语句的举例均是正确的。

选择语句举例：

```
Select
    Accept DRIVER_AWAKE_SIGNAL;
Or
```

```
        Delay 30.0 * SECONDS;
        STOP_THE_TRAIN;
    End Select;
```

--本例所列备选是开放的

带有选择语句任务体的举例：

```
Task Body RESOURCE is
    BUSY:Boolean:=FALSE;
Begin
    Loop
        Select
            When Not BUSY=>
                Accept SEIZE Do
                    BUSY:=TRUE;
                End;
            Or
                Accept RELEASE Do
                    BUSE:=FALSE;
                End;
            Or
                Terminate;  --任务中止语句
            End Select;
    End Loop;
End RESOURCE;
```

使用WHEN的备选语句，如果其条件为真，则该 备选是开放的、否则是封闭的。

对于某一发出入口呼叫的任务，可能有几个开放备选在等待被其接受，此时仅选择其中一个。如果不能立即交会，且又无ELSE部分，则任务的等待一直延续到有一个开放的备选被选中。

四、优先权（PRIORITY）

优先权表示任务的紧迫性，可根据需要为每个任务设置

一个优先权．数值越高表明使用权越高．优先数的范围由不同的语言实现决定。

设置优先权格式：

PRAGMA PRIORITY（（表达式））；

五、夭折语句

夭折语句用以阻止任务的交会，使得任务非正常地终止。

夭折语句格式：

ABORT（任务名）｛，（任务名）｝；

本语句只能在要求无条件终止的情况下使用。

第三节　任务属性

对于任务T定义了如下属性：

T'CALLABLE	其值为布尔型。若为假表明该任务已完成，或已被终止，或已出现不正常情况。
T'TERMINATED	其值为布尔型，若任务已经终止，值为真，否则为假。
T.E'COUNT	其值为整型，给出当前对任务T的入口E发出呼叫尚未被接受数量（等待队列中的队员数）。

要点

1．任务是Ada支持并发特性的主要语言构造。它在结构上类似于程序包，由规格说明和体两部分组成。其中任务界

面定义了任务的入口点，任务体则定义了逻辑上可与其它程序单元并发执行的动作。

2.Ada任务通讯的主要类型可归纳如下：

（1）简单通讯；　（2）由待从任务选定的会合；

（3）由调用任务选定的会合；

3.Ada任务的应用领域如下：

（1）并发的动作；　（2）信息路径选择；

（3）共享资源管理；　（4）中断处理；

习题

1．由发件人、送件人、收件人执行邮包发送任务，为该三种服务人员写一规格说明。要求将发件人表述为原动任务；送件人有一标识为RECEIVE_PARCEL的入口调用及对其它任务的入口调用；收件人表述为待从任务，有一标识为ACCEPT_PARCEL的入口调用.

2．用简单通讯方式写出上题的三个任务体（假定邮包类型是PARCEL_KIND）.

第十二章　类属程序单元

在编制大型程序时，常会碰到这样的情况：有些程序单元之间的差别，仅是操作对象的数据类型不同，其它诸如内部处理的逻辑关系等，都是相同的。

如一个对整型数组排序操作的子程序，与对浮点型数组、字串型数组及其它类型数组排序操作的子程序，在逻辑关系处理上都是一样的。这种情况下，如果事先能预先设计出一个对数组排序操作的模式化子程序（类属子程序），在具体实现时再将其具体实化为对某种数据类型数组进行排序操作的子程序（建立类属子程序实例），这就可以省去许多重复工作，这不但可以使程序格调统一、修改方便，而且还大大地减少了程序出错机会，提高了编程效率。

具有如上特征的模式化程序单元，有的称其为式样单元或公用单元，但通常译为类属单元(GENERIC UNITS)。类属程序单元是一个不能执行的模块，所以不能直接使用。必须建立类属单元的实例后才能使用。建立可执行的类属程序单元实例的过程称为设置。

Ada语言类属单元有类属子程序和类属程序包两类。

第一节　类属程序单元的形式

一个类属子程序的举例：

① 程序声明部分

```
X,Y:Integer;
Generic                                    --类属声明
        Type ELEM is Private;              --定义类属参数
                                           --定义类属子程序
Procedure EXCHANGE (U,V:in out ELEM)
Procedure EXCHANGE (U,V:in out ELEM) is
    T:ELEM;
    Begin
        T:=U; U:=V; V:=T;
    End EXCHANGE;
    Procedure SWAP is New EXCHANGE(Inte-
    ger)                                   --设置
```

② 程序体部分

```
        X:=86; Y:=90;
                --调用过程SWAP, 交换X, Y值
        SWAP (X,Y);
        Put(X,8);                  --输出90
        Put(Y,8);                  --输出86
```

本例表明类属子程序与非类属子程序形式上的不同点，就在于定义部分。后者直接定义一个子程序单元；而前者还要定义一个类属参数及将类属子程序具体实化成指定类型的新程序单元，这才和后者一致起来。实化后的程序单元的调用和一般程序单元的调用就没有什么两样了。

类属说明声明一个类属单元（类属子程序或类属程序包）。类属单元可以不带形参，也可以带形参。

一、类属子程序

类属子程序定义格式：

```
GENERIC
    { [〈类属参数说明〉; ] }
  | PROCEDURE〈类属过程规范说明〉
  | FUNCTION  〈类属函数规范说明〉
    〈类属子程序体说明〉;
```

类属子程序设置格式：

```
| PROCEDURE 〈过程名〉 IS NEW 〈类属过程名〉
[〈实参说明〉]；
| FUNCTION   〈函数名〉 IS NEW 〈类属函数名〉
[〈实参说明〉]；
```

类属子程序形实参匹配规则与子程序形实参匹配规则相同。

二、类属程序包

类属程序包的形式与类属子程序的形式相似。

类属程序包定义格式：

```
GENERIC
    { [〈类属参数说明〉; ] }
      PACKAGE〈类属程序包规范说明〉
          〈类属程序包体说明〉;
```

类属程序包设置格式，

PACKAGE〈程序包名〉 IS NEW 〈类属程序包名〉 [〈实参说明〉 >]；

第二节　类属参数

类属参数可以是类属形式类型；也可以是类属形式对象（变量）或类属形式子程序。

一、类属形式类型说明格式

GENERIC
TYPE〈标识符〉IS〈类属形式类型说明〉;

类属形式类型说明有多种形式，对应于不同的说明形式，有不同的匹配规则。

类属形式类型说明的举例及与之兼容的类型说明如下：

(1) Type ITEM is Private;

可与任何类型匹配，允许赋值、测试相等或不相等操作

(2) Type BUFFER is Limited Private;

可与任何类型匹配

(3) Type ENUM is (<>);

可与任何离散类型匹配

(4) Type INT is Range<>;

可与任何整型类型匹配

(5) Type ANGLE is Delta<>;

可与任何定点类型匹配

(6) Type MASS is Digits<>;

可与任何浮点类型匹配

(7) Type TABLE is Array (ENUM) of ITEM;

可与任何维数、下标类型及分量类型相同的约束数组匹配

(8) Type TABLE is Array(ENUM Range<>) of ITEM;

可与任何维数、下标类型及分量类型相同的非约束数组匹配

(9) Type LINK is Access SOME_OBJECT;
可与任何指向同一对象的访问类型匹配

二、类属形式对象的说明格式

GENERIC
〈标识符表〉：[IN，[IN OUT]]〈类型标识符〉[：=〈表达式〉]；

例：

```
Generic
     SIZE: Natural;
Generic
     LENGTH: Integer; =200;
     AREA  : Integer; =LENGTH*LENGTH;
```

使用类属形式对象时，必须将该类属参数与同类型的变量或常量相匹配。Ada语言把该类属值当作其余单元的常量使用。该类属参数模式仅允许IN方式或IN OUT方式。

三、类属形式子程序说明格式

〈WITH〈类属形式子程序规范说明〉[IS〈〉|〈名字〉]；〉

Ada语言允许将子程序作为类属参数使用。在设置时，子程序实例与形式子程序必须完全兼容(子程序参数的数量、排列顺序、模式、约束及函数的返回值均相同)。

例如：

类属声明

```
    Generic
        ROWS    : in Integer: =24;
        COLUMNS: in Integer: =80;
With Procedure SEND(VALUE: in CHARACTER);
With Procedure RECEIVE(VALUE:out CHARACT-
```

```
ER);
        Package TERMINAL is
              ... ...
```

类属设置

```
  Procedure MICRO_SEND  (V:in CHARACTER) is
        ... ...
  Procedure MICRO_RECEIVE(V:out CHARACTER)
                                          is
        ... ...
Package MICRO_TERMINAL is New TERMINAL
(ROWS=>24, COLUMNS=>40,SEND=>MICRO_SE-
              ND,RECEIVE=>MICRO_RECEIVE);
```

要点

类属程序单元的主要作用是提供模式化的程序单元，以满足各种特定的需要。另外，还可通过类属参数将类型与子程序作为参数传递到其它程序单元中去。利用类属程序单元还可建立通用性很强的程序单元，达到加快软件开发进程，提高程序清晰度和易维护性的功效。

习题

1. 设计一程序，其中包含两个类属子程序，一个用于数值交换，一个用于对数组进行排序。

2. 按上题要求，设计一个类属程序包，然后再打开使用它。

附录：预定义程序包体源程序

(1) BLKIO.PKG

```
With Jlib86,Util,IO;
Package Body Blkio Is
    ------------------------------------------------
    ----This library implements random access se-
    ctor I/O  via.
  —the procedures read_blk and write_blk.MS-DOS
  version.
  —Last Modified 7/19/82
    ------------------------------------------------
    —Copyright 1982  RR  Software, P.O.  Box
      1512, Madison  WI 53701
    —Permission  is hereby  given  to  distribute
      Object Code produced from these libraries.
Use Jlib86, Util, IO;
    qqaddr: Integer;  -Temporaries used below
    result: integer;
    fmask: file_ptr;
    Procedure write_blk(Fyle: In  File;  sec: In
                        Sector; blk: In Integer) Is
    —Write the block number blk into the file fyle
Begin
    Put("Write Block-"); Put(blk); New_Line;
```

```
    fmask:=FConvert(Fyle);
Pragma Sybdump(On);
    fmask.fcb.random_rec1:=blk; —Set the block
                                  number
    fmask. fcb. random_rec2:=0; —Zero the
                                  high bytes
    qqaddr: =sec' address; —Get the sector ad-
                                  dress
    Asm 16#8B#, 2#00_010_110#, qqaddr' address;
                          —Mov  DX,[qqaddr]
    Asm 16#B4#,26;        —Mov AH,26
    Asm 16#CD#,33;        —Int 33
                          —Call MS-DOS to
                            set transfer addr
    qqaddr:=fmask.fcb' address;—Fyle FCB ad-
                                  dress
    Asm 16#8B#,2#00_010_110#,qqaddr'address;
                          —Mov DX,[qqaddr]
    Asm 16#B4#,40         —Mov AH,40
    Asm 16#B9#,1,0;       —Mov CX,1 (No. of
                            records to write)
    Asm 16#CD#,33;        —Int 33
          —Call MS-DOS to write the sector
    Asm 16#B4#,0;         —Mov AH,0
    Asm 16#A3#,result' address; —Mov [result],
                                  AX(Save result)
```

```
  If result /=0 Then
    IOresult:=255;
  Else
    IOresult:=0;
  End If;

  End write_blk;
  Procedure read_blk(Fyle:In File; sec:Out
                     Sector; blk: In Integer) Is
--Read the block number blk from the file fyle
  Begin
    Put("Read Block-");Put(blk);New_Line;
    fmask:=FConvert(Fyle);
    fmask.fcd.random_rec1:=blk; --Set the bl-
                                ock number
    fmask.fcb.random_rec2:=0;--Zero the high
                              bytes
    qqaddr:=sec'address;--Get the sector add-
                         ress
    Asm 16#8B#, 2#00_010_110#, qqaddr'address;
                          --Mov DX,(qqaddr)
    Asm 16#B4#,26;        --Mov AH,26
    Asm 16#CD#,33         --Int 33
          --Call MS-DOS to set transfer addr
    qqaddr:=fmask.fcb'address;  --Fyle FCB
address
```

```
    Asm 16#8B#,2#00_010_110#,qqaddr'address;
                                --Mov DX,[qqaddr]
    Asm 16#B9#,1,0; --Mov CX,1(Read One
                          Sector)
    Asm 16#B4#,39; --Mov AH, 39
    Asm 16#CD#,33; --Int 33
              --Call MS-DOS to read the sector
    Asm 16#B4#,0; --Mov AH,0
    Asm 16#A3#,result' address;--Mov[result],
                                --AX (Save result)
    If result/=0 Then
      IOresult:=255;
    Else
      IOresult:=0;
    End If;
End read_blk;

Procedure RClose(fyle:In Out File) Is
  --Close a file for random access
Begin
  fmask:=FConvert(fyle);
  fmask.fmode:=Byte(1);--Set mode to read_
                                        only
  Close(fyle);--Close the file,without dumpi_
                                        ng the
--(unused)buffer
```

```
    End RClose;
End Blkio;
```

(2) **CHAINLIB.PKG**

```
With IO,Util,Jlib86;
Package Body Chainlib Is
--The Program chaining package for JANUS V.
  1.4.5-8086 Version
--Last Modified 4/17/83
    --Copyright 1982 RR Software, P.O.Box
      1512, Madison WI 53701
    --Permission is hereby given to distribute
      Object Code produced from
    --these libraries.
Use IO,Util,JLib86;
Pragma Rangecheck(Off); Pragma Arithcheck
                                        (Off);
mask:File_ptr; --Temporarys used by chain and
                  prog_call
address,value,loc,topseg: In teger;
Procedure chain(str:In string) Is
  --Run the program in the file str now,junki-
                          ng the current program
    fil:File;sizel,size2:Integer;
    Begin
      Open(fil,str,Read_Only);
      If IOresult=255 Then
```

```
    Put(Str);Put ("File  missing");
    New_Line;
    Halt;
  End If;
  mask:=FConvert(fil);
  address:=mask.fcb'address;
  size1:=mask.fcb.file_size1;
  size2:=mask.fcb.fil e_size2;—get  the  file
                                  size
  —Size 1 is the low word.Size 2 is the high
    word
  value:=Integer( hi(size1))*2+(Integer( lo
          (size1))+127)/128;
    —This mess since the high  bit  of  size2
      can be set, making
    —it negative
  value:=size2*512+value;—size1  cannot be
                                  negative
  —Move the FCB into the default FCB (So
    the new program does not over write it)
  Asm 16#8B#,2#00_110_110#,address'address;
    —Mov SI,[address]
  Asm 16#BF#,16#5C#,0; —Default FCB add-
                          ress (Mov DI,addr)
  Asm 16#B9#,37,0;  —Move  37  bytes(Mov
                          CX37);
```

```
        Asm 16#0E#, 16#07#; —Push CS; Pop ES
                            (Set ES to Code segment)
        Asm 16#F3#,16#A4#;  —Rep Movsb (Block
                            Move)
        —Set CX to the number of sectors ot read
        Asm 16#8B#, 2#00_001_110#, value'address;
          —Mov CX,(value)
        JLIB86. PChain; —Call the PChain entry
                        point
        —Won't return here
End Chain;
Procedure Prog_call(str:In String) Is
    —Run the program in the file str now. If
      the called program
    —executes a'Prog_return',control will retu_
      rn here;
    fil: File; byt:Byte; size1, size2, tos:Integer;
    Begin
      Open(fil,str,Read_Only);
      If IOresult=255 Then
        Put(Str);Put('File missing');New_Line;
        Halt;
      End If;
      mask:=FConvert(fil);—Get pointer
      address:=mask.fcb'address;
      size1:=mask.fcb.file_size1;
```

```
size2:=mask.fcb.file_size2; --get the file
                                size
value:=Integer(hi(size1))*2+(Integer(lo
        (size1))+127)/128;
  --This mess since the high bit of size2
    can be set,making
  --it negative
value:=size2*512+value; --size1 cannot be
                                negative
 Asm 16#06#; 16#1E# --Pust ES: Push DS-
                            Save these segments
                  --for restoration after the new
                    program
                  --runs.SS is saved by called
                    program.
                  --and CS is saved by the call.
--Determine the program load paragraph
Asm 16#8C#,2#11_011_000#; --Mov AX,DS
Asm 16#05#,16#1000#; --Add AX,1000h
Asm 16#A3#,loc'address;--Save the segment
                                address
Asm 16#2E#; --Seg CS (top of mem is inCS)
Asm 16#A1#,16#02#,0; --Mov AX,[2]--Top
                                of mem address
Asm 16#A3#,topseg'address;--Mov [topseg],
                                AX
```

```
tos:=loc+value*8;
--Loc=load address,value*8=lenth of
  segment
If (Hi(topseg)<Hi(tos))Or Else
(Hi(topseg)=Hi(tos) And Then Lo(topseg)
  <=Lo(tos))Then
    Asm 16#1F#,16#07#;--Pop DS: Pop ES
    Put("** Not enough room for Program
      call of"):Put(str):
    New_Line: Err_Exit:
End If;
--Set the new terminate address
Asm 16#1E#,16#0E#,16#1F#;--Set DS to CS
Asm 16#B4#,37;--Mor AH, 37--Set interupt
Asm 16#B0#,34;--Mov AL,34 -Set terminate
Asm 16#BA#,chainlib'location+58;
                    --Termination address
Asm 16#CD#,33; --Call MS-DOS
Asm 16#1F#; --Restore DS
--Create the program segment
Asm 16#A1#,loc'address;--Mov AX,[loc]
                --Load the segment address
Asm 16#92#;--Xchg AX,DX
Asm 16#B4#,16#26#;--Mov AH, 26h-Create
                                  Prog Seg
Asm 16#CD#,33;--Int 33-Call MS-DOS
```

```
—Set the load point
Asm 16#8E#,2#11_011_010#; —Mov  DS,DX
Asm 16#BA#,16#100#;—Mov DX,100h—Load
                                    address
Asm 16#B4#,26;—Mov AH,26—Load address
                                    opcode
Asm 16#CD#,33;—Int 33—Call MS-DOS
Asm 16#1F#,16#1E#;—Pop  DS! Push  DS—
                    Restore seg.to old value
—Load the file into that area
Asm 16#8B#,2#00_010—110#,address'address;
  —Mov DX,[address]
—Set CX to the number of sectors to  read
Asm 16#8B#, 2#00_001_110#, value'address;
  —Mov CX,[value]
Asm 16#B4#,39;—Mov  AH,39—Read  value
                                    blocks
                —into the new code segment
Asm 16#CD#,33;—Call MS-DOS
—Set the segment registers DS and ES.
Asm 16#A1#,loc'address; —Mov AX,[loc]
          —Load the segment address
Asm 16#8E#,2#11_011_000#;—Mov DS,AX
Asm 16#8E#,2#11-000_000#;—Mov ES,AX
—Set up registers and call block
Asm   16#C7#,,2#00_000_110#,16#FC#,0,16#
```

```
                                100 ;--Mov [FCh],100h
                              --(PC of called program)
Asm 16#89#,2#00_000_110#,16#FE#,0;
                                        --Mov [FEh],AX
                              -(Get CS into call block)
Asm 16#BB#,16#FC#,0;--Mov BX,0FCh
Asm 16#FF#,2#00_101_111#;--Jmpf[BX]-Ju-
                                        mp to the program
                          --indirect thru BX
--Program will return here unless a <Ctrl>-C
  interrupt occurs
--%%%%Following line is needed only by
  MS-DOS version 1.xx
Asm 16#59#,16#59#,16#59#; --Remove garb-
                                        age from stack put
                          --there by the Halt interrupt.
Asm 16#5B#;--Pop BX-Pop return code
Asm 16#1F#,16#07#;--Pop DS! Pop ES
                          --Restore the segment registers
Asm 16#81#,2#11_111_011#,145,0;
                          --Cmp BX,145-Test for magic
                          --number.If it occurs,call halt
                          --immediately
  Asm 16#74#,3;--Jne OK
GoTo OK;
  Halt;
```

```
        《OK》
          Close(fil);--Toss the file
        End Prog_Call;
      Procedure Prog_Return Is
        --Return to the calling program
      Begin
        Jlib86.PrgRet;--Call the Prog_Ret entry
                      --point
                              --Won't return here
      End Prog_Return;
End Chainlib;
```

(3) **FLOATIO.PKG**

```
With IO,Util,Jlib86.Ascii;
Package Body Floatio Is
    Ues IO,Util;
    --Floating Point I/OPackage
    --Written March 1983
    --Last Modified 6/30/83
    --Copyright 1983 RR Software,Inc.,P.O.Box
      1512,Madison WI 53701
    --Permission is hereby given to distribute
      Object Code produced from
    --these libraries. All Other rights reserved.
    Pragma Arithcheck(Off);Pragma Rangecheck
      (Off);
    Pragma Arithcheck(On);Pragma Rangecheck
```

```
        (On);
    Itemp:Long_Float;
    control_word,save_control_word:Integer;
    Address:Integer;
Mask:Jlib86.File_ptr;
Char2:Character;
Procedure format_float(val:In  Long_Float; Aft.
  Exp:In Integer; str:Out String;fore:Out Intege-
  r) Is
    --Aft-Places after decimal point
    --Exp-Characters in exponent field(including
      E and sign)
    --Str-Result string
    --Fore-Characters before decimal point
    temp_val:Long-Float:=val;--Working  Value
    cnt:Integer:=0;--Buffer count
    exponent:Integer:=0;--Exponent
    texp:Integer;--Temporary Exponent Field
    cval:Integer;--Integer used for truncation
    rnd:Long_Float;--Rounding Value
    Function PTrunc(Val:In Long_Float) Return
      Long_Float Is
      --Truncate val to its integer part,and retu-
        rns it as a long_float
    Begin
      Return Long_Float(Integer(Val-(Long_Float
```

```
        (1)/Long_Float(2)));
            --Truncates number.
    End PTrunc;
    Procedure Ins_char(ch:In Character) Is
      --Insert Characters in string(Faster(?  )than
        concat)
    Begin
      cnt:=cnt+1;
      str(cnt):=ch;
      Standard.Put("Output Character=");
        Standard.Put(ch);New_Line;
        If cnt>125 Then
        Standard.Put("Floating  Point  Format
          Error-Fields");
        Standard.Put("too long");
        Err_Exit;
      End If;
    End Ins_Char;
Begin
    str(0):=Character'Val(127);--Full String
    Exponent:=0;
    Standard.Put("Format_Float-Int(val)=");
    Standard.Put(Integer(temp_val));New_Line;
    If temp_val(Long-Float(0) Then--negative
      Standard.Put("Sign is  Negative");New_Li-
        ne;
```

```
    Ins_Char('-');
    temp_val:=Abs_temp_val;
  Else
    Standard.Put ("Sign  is  Positive") ;
    New_Line;
  End If;
  --Reduce Exponent
  --Ought to reduce  by 10** 10 first,as  this
    way can take up to
  --0.1 sec on the 8087,and 100 sec(! )if  done
    in software
  If temp_val/=Long_Float(0) Then
    If temp-val )Long_Float(1) Then
      Standard.Put("Value is greater than  1");
        New-Line;
      While temp_val)=Long_Float(10) Loop
        Exponent:=Exponent+1;
        temp-val:=temp_val/Long-Float(10);
      End Loop;
     Else
      Standard.Put("Value is Less than Zero");
        New-Line;
      While temp_val(Long_Float(1) Loop
        Exponent:=Exponent-1;
        temp_val:=temp_val*Long_Float(10);
      End Loop;
```

```
    End If;
    Else
      Standard.Put("Value=0.0");New_Line;
  End If;
    Standard.Put("Exponent Value is");
    Standard.Put(exponent);New_Line;
    Srandard.Put("Int of Reduced Value is=");
    Standard.Put(Integer(temp_val));New_line;
--Now have exponent as integer  &  mantissa
--in the form x.yyyyy...
--Figure number of characters before the de-
--cimal point
If exp/=0 Then--Have exponent
  fore:=1;--Only one character before decim
           al point
Elsif exponent) =0  Then--Value  is  bigger
                                 than One
  fore:=exponent+1;
Else--Less than 1,must have leading zeros
  fore:=1;temp_val:=Abs_val;--Toss scaling,
                                 so we will
                            --have leading zeros
                             -(i.e.0.001)
End If;
  Standard.Put("Fore=");
  Standard.Put ("fo-re);
```

```
   Standard.Put('Aft=');Standard.Put(aft);
   Standard.Put('Exp=');Standard.Put(exp);
     New_Line;
--Round the mantissa value
rnd:=Long_Float(5);
For i In 1..(fore+aft) Loop
  rnd:=rnd/Long_Float(10);
End Loop;
   Standard.Put('Int of rnd  ought  to  be 0
                                        =');
  Standard.Put(Integer(rnd));New_Line;
  temp_val:=temp_val+rnd;--Round mantissa
                                    up
  Standard.Put ('Mantissa Rounded');
    New_Line;
-Rounding may have caused  scaling  to fail
  (Patch added 6/30/83)
If temp_val)Long_Float(10) Then
    temp_val:=temp_val/Long_Float(10);
    --Now must adjust exponent
    exponent:=exponent+1;
    --Now have exponent as integer & mant-
      issa in the form x. yyyyy...
    --(like we thought before).Adjust numb-
      er of characters before
    --the decimal point,if necessary.
```

```
        If exp=0 Then--Have  exponent, zero or
                              greater,no exponent to
                                --be printed...
           fore:=exponent+1;
        End If;
        Standard.Put("Fore=");Standard.Put (fo-
          re);
        Standard.Put("Aft=");Standard .Put(aft);
        Standard.Put("Exp=");Standard.Put(exp);
          New_Line;
      End If;
      --Now make up the string!
      For i In 1.. fore Loop
        --Determine and remove the first digit from
          the temp_val,
        --and Ins_char with the result
        --Temp_val is always between 0..10
        --Digit is removed,and the result is multip-
        --lied by 10,so that
        --the next digit is now the integer part
        cval:=Integer(PTrunc(temp_val));
        Ins_char(Character'Val(cval+48));
        temp_val:= (temp_val-Long_Float(cval))*
          Long_Float(10);
        Standard.Put("Cval=");Standard.Put(cva-
          l);New_Line;
```

```
End Loop;
Ins_Char('.');
For i In 1..aft Loop
   --Determine and remove the first digit from
     the temp-val,
   --and Ins-char with the result
   --Temp_val is always between 0..10
   --Digit is removed,and the result is multi-
   --plied by 10,so that
   --the next digit is now the integer part
   cval:=Integer(PTrunc(temp_val));
   Ins-char(Character'Val(cval+48));
   temp-val:=(temp_val- Long_Float(cval))*
     Long_Float(10);
   Standard.Put("Cval=");
     Standard.Put(cval);New_Line;
End Loop;
If exp/=0 Then--Format exponent field
   Ins_Char('E');
   If exponent>=0 Then
     Ins_Char('+');
   Else
     Ins_Char('-');
     exponent:=Abs exponent;
   End If;
   texp:=exp-2;--Count two characters alrea-
```

```
                                        dy generated
  --Expand field if necessary
  If exponent In  -9..9  Then
    If texp< 1 Then texp:=1;End If;
  Elsif exponent In  -99..99 Then
    If texp < 2 Then texp:=2;End If;
  Else              --exponent In-999 .999 Then
    If texp < 3 Then texp:=3;End If;
  End If;
  While texp > 5 Loop
    Ins_Char('0');
    texp:=texp-1;--Texp=characters remain-
                        ing in field
  End Loop;
  For i In Reverse 0..texp-1  Loop
    cval:=exponent/(10**i);
    Ins_char(character'val(cval+48));
    exponent:=exponent-(cval*(10**i));
  End Loop;
  --exponent (the var.) ought to be zero here
End If;
str(0) :=Character'Val(cnt);--Set string le-
                                        ngth
If val < Long_Float(0) Then
  fore:=fore+1; --Count sign
End If;
```

```
    --That's all
End Format_Float;
Procedure Replace_byte(Fyle:In File;Ch:In Char-
                                              acter) Is
--Replace byt in the file (Local)
Begin
    mask:=Util.FConvert(Fyle);
    address:=Mask.All'address;
    Char2 :=Ch;
    Asm 16#8B#,2#00_011_110#,address'address;
                --Mov BX,[address]
                --Load the file address
    Asm 16#A0#,Char2'Address;
                --Mov AL,[Char2]
    Jlib86.Rplcbyte; --Call Rplcbyte
End Replace_byte;
Procedure  Get_Float(Fyle:  In  File; Value:Out
                                    Long_Float) Is
    --Get a floating point value from the file
    lookahead:Character;--Lookahead character
    negative:Boolean:=False;
    Temp_float.bias:Long_Float;
    exponent : Integer;
  Begin
    Standard.Get(Fyle,Lookahead);
    While(Lookahead In Ascii.Nul..Ascii.SP)
```

```
  Loop Standard.Get.(Fyle,Lookahead);
              --Toss all control characters
End Loop;
If lookahead='-'  Then
  negative:=True;
  Standard.Get(fyle,lookahead);
Elsif lookahead = '+'  Then
  Standard.Get(fyle,lookahead);
End If;
Temp_Float:=Long_Float(0);
If Lookahead Not In'0'..'9' Then
                --No number at all!
      Raise Layout_Error;
    End If;
While Lookahead In'0'..'9'Loop
  Temp_Float :=Temp_Float*Long_Float(10)
    +Long_Float(Character'Pos(Lookahead)-
      Character'Pos('0'));
  Standard.Get(Fyle,Lookahead);
End Loop;
If Lookahead='.' Then
  Standard.Get(Fyle,Lookahead);
              --Toss decimal point
Else --Error,decimal point missing
  - Raise Layout_Error;
End If;
```

```
  Bias:=Long_Float(1);
  If Lookahead Not In'0'..'9' Then
--No  number after decimal point! !
--Raise Layout_Error;
      End If;
  While Lookahead In '0'..'9'Loop
    Bias:=Bias/Long_Float(10);
    Temp_Float:=Temp_Float+Bias*Long_Flo-
      at(Character'Pos (Lookahead)-Character'
                                    Pos('0'));
      Standard.Get(Fyle,Lookahead);
  End Loop;
  --Exponent
  If Lookahead='E'Or Lookahead='e'Then
    --Exponent
    Standard.Get(Fyle,Lookahead);
     --Here Bias is the multiplier that each ex_
       ponent count is worth
    If lookahead='-'Then
       Standard.Get(fyle,lookahead);
       Bias:=Long_Float(1)/Long_Float(10);
     Elsif lookahead='+'  Then
       Standard.Get(fyle,lookahead);
       Bias:=Long_Float(10);
     Else
       Bias:=Long_Float(10);
```

```
      End If;
      Exponent:=0;
      If Lookahead Not In  '0'..'9'  Then
                          --No exponent at allt t
      --Raise Layout_Error;
      End If;
      While Lookahead In '0'..'9'  Loop
        Exponent:=Exponent*10+(Character'
          Pos(Lookahead)-Character'Pos('0'));
        Standard.Get(Fyle,Lookahead);
      End Loop;
      For i In 1..Exponent Loop
        --Give the exponent it' s value
        Temp_float:=Temp_float * Bias;
      End Loop;
    End If;--Exponent
    Replace_byte(Fyle,Lookahead);
    If Negative Then
      Value:=-Temp_Float;
    Else
      Value:=Temp_Float;
      End If;
  End Get_Float;
  Procedure Get(Fyle: In File; Value:Out Float)
    Is
    temp:Long_Float;
```

```
Begin
  Get_Float(Fyle,temp);
  Value:=Float(Temp);
End Get;
Procedure Get(Value:Out Float) Is
  temp: Long_Float;
Begin
  Get_Float(Cur;ent_Input(),temp);
  Value:=Float(Temp);
End Get;
Procedure Put(Fyle:In File;Value:In Float)  Is
Begin
  Put(fyle,value,2,5,3);
  --Default values
End Put;
Procedure Put(Value:In Float)  Is
Begin
  Put(Current_Output(),value,2,5,3);
End Put;
Procedure Put(Fyle:In File:Value:In Float:Fo
  re:In Integer) Is
Begin
  Put(fyle,value,fore,5,3);
End Put;
Procedure Put(Value:InFloat:Fore:In Integer) Is
Begin
```

```
    Put(Current_Output(),value,fore,5,3);
End Put;
Procedure Put(Fyle:In File;Value:In Float:Fo-
                                re,Aft:In Integer) Is
Begin
  Put(fyle,value,fore,aft,3);
End Put;
Procedure  Put(Value : In  Float:Fore,Aft:In
                                    Integer) Is
Begin
  Put(Current_Output(),value,fore,aft,3);
End Put;
Procedure  Put(Value:In  Float:Fore,Aft,Exp:
                                  In Integer) Is
Begin
  Put(Current_Output(),value,fore,aft,exp);
End Put;
  Procedure Put(Fyle:In File;Value: In
      Float:Fore,Aft,Exp:In Integer)  Is
  --Put float into file
  str: String;
  tfore:Integer;
Begin
  format_float(Long_Float(Value), aft, exp,str,
    tfore);
  --Add enough leading blanks so that  tfore+
```

```
    --num_blanks=fore
  While tfore < fore Loop
    Standard.Put(fyle,' ');
    tfore:=tfore+1;
  End Loop;
  Standard.Put(fyle,str);--Output value
End Put;
Function Float_to_String(Value: In Float) Ret-
                                    urn String Is
  --Formats value into a string
  str : String; tfore : Integer;
Begin
  format_float(Long_Float(Value),5,3,str, tfo-
                                        re);
  Return str;
End Float_to_String;
Function Float_to_String(Value: In Float;fore,
  aft,exp : Integer)          Return String Is
  --Formats value into a string
  str : String;tfore : Integer;
Begin
  format_float(Long_Float(Value),aft, exp,str,
    tfore);
  While tfore < fore Loop
    str:=" " & str;
    tfore:=tfore+1;
```

```
    End Loop;
    Return str;
End Float_to_String;
Procedure Get(Fyle : In File;Value:Out Long_
                                        Float) Is
Begin
    Get_Float(Fyle,Value);
End Get;
Procedure Get(Value: Out Long_Float) Is
Begin
    Get_Float(Current_Input(),Value);
End Get;
Procedure Put(Fyle : In File;Value : In Long_
                                        Float) Is
Begin
    Put(fyle,value,2,14,3);
    --Default values for long float
End Put;
Procedure Put(Value: In Long_Float)  Is
Begin
    Put(Current_Output(),value,2,14,3);
End Put;
Procedure Put(Fyle : In File; Value: In Long_
                    Float;Fore: In Integer)  Is
Begin
    Put(fyle,value,fore,14,3);
```

```
End Put;
Procedure Put(Value : In  Long_Float;Fore :
                                    In Integer)  Is
Begin
  Put(Current_Output(),value,fore,14,3);
End Put;
Procedure Put(Fyle:In  File;Value:  In  Long_
                Float; Fore,Aft:In Integer)  Is
Begin
  Put(fyle,value,fore,aft,3);
End Put;
Procedure Put(Value : In Long_Float;Fore,Aft;
                                    in Integer)  Is
Begin
  Put(Current_Output(),value,fore,aft,3);
End Put;
Procedure Put(Value : In Long_Float;Fore, Aft
                              Exp : In Integer) Is
Begin
  Put(Current_Output(),value,fore,aft,exp);
End Put;
Procedure  Put(Fyle:In  File  : Value:In Long_
            Float; Fore,Aft,Exp: In Integer) Is
  --Put Long_Float into file
  str : String;
  tfore : Integer;
```

```
Begin
  format_float(Value,aft,exp,str,tfore);
  --Add enough leading blanks so that  tfore+
    --num_blanks=fore
  While tfore < fore Loop
    Standard.Put(fyle,' ');
    tfore := tfore+1;
  End Loop;
  Standard.Put(fyle,str); --Output value
End Put;
Function Float_to_String(Value : In Long_Flo_
                                  at) Return String is
  --Formats value into a string
  str : String; tfore : Integer;
Begin
  format_float(Value,14,3,str,tfore);
  Return str;
End Float_to_String;
Function Float_to-String(Value : In Long_Flo-
  at;fore,aft,exp : Integer) Return String is
  --Formats value into a string
  str : String; tfore : Integer;
Begin
  format_float(Value,aft exp,str,tfore);
  While tfore < fore Loop
      str := ' '& str;
```

```
            tfore := tfore+1;
        End Loop;
        Return str;
    End Float_to_String;
End FloatIo;
```

(4) IO.PKG

```
With Util,JLib86;
Package Body IO Is
--The I/O package for JANUS V.1.4.6 - MS-
  DOS Version
--Last Modified 5/ 5/83
--Includes open, close,create,delete,name,is_open,
--get_line, put_hex,disk_full,end_of_file,and end_
  of_line
--Added Keypress & Purge 2/13/83,and fixed
  bug in Get_Line
--Added Putw,Mode, and Second Get_Line 3/27/
  83
--Put_Line(s) 5/ 5/83
    --Copyright 1982,1983 RR Software,Inc., P.
      O. Box 1512, Madison WI 53701
    --Permission is hereby given to distribute Ob-
      ject Code produced from
    --these libraries. All Other rights reserved.
Use Util, JLib86;
Pragma Rangecheck(Off);
```

```
Pragma Arithcheck (Off);
Address : Integer; --Used to hold address of fcb
                        for routines below
Value : Integer; --Static used to hold the value
                        for Put_Hex
Loc : Integer; --Used to hold the address of the
                        routine to be called
    --The following temporary variables are
      declared Static for speed
mask: File_ptr; --Temporary used by many of
                        the procedures below
byt : Byte; --Temporary used by many of the
                        procedures below
Pragma sybdump(On);
Procedure set_fcb(Fyle : In Out File; str : In
                        String) Is
    --Set up an fcb in fyle with the name str
    temp_str : String;
    len : Integer; --Length of the input string
                        (after modification)
    Procedure Name_error Is
      --Compact the code
      Begin
        Put("File name error-"); Put(str);
        --Dispose(mask);--When raise an excep-
          tion
```

```
    --FFConvert(Null,fyle);--File not  open
Err_exit;
End Name_error;
Begin
  mask:=FConvert(fyle);--Get file into mask
  temp_str :=str;
  Declare--Remove Blanks from and capitali-
          ze temp_str
    i : Integer;
  Begin
    i :=0;len:=Character'Pos(temp_str(0)) ;
      --length of temp_str
    Loop
      i := i + 1;
      Exit When i > len;
      If temp_str(i)=' ' Then
        For j In i..len-1 Loop
          temp_str(j) := temp_str(j+1);
        End Loop;--Move the  string  down
                          to remove the blank
        len :=len -1;
        i := i -1;--Back up and look at
                          the character in the
          --i'th position again'cause its diff-
            erent
      Elsif temp_str(i) In 'a'..'z' Then
```

```
            temp_str(i) :=Character'Val(Chara-
                                cter'Pos(temp_
                    str(i))-32);
          End If;
        End Loop;
        temp_str(0)  :=Character'Val(len);
                                --Set the length
    End; --Remove_blanks;
    Put("Set_fcb-File  name = ");
      Put (tempstr);New_line;
    If temp_str ="CON:" Then
      mask.ftype :=Byte(1);
    Elsif temp_str = "AUX:" Then
      mask.ftype:=Byte(2);
    Elsif temp_str = "LST:" Then
      mask.ftype:=Byte(4);
    Elsif temp_str ="KBD:" Then
      mask.ftype:=Byte(5);
    Else--Disk File
      Temp_str :=temp_str & " ";
         --Add trailing space so MS-DOS does
                            --not go too far
      mask.ftype :=Byte(0);
      --Use  MS-DOS Parse File  Name Entry
        Point
      address :=mask.fcb'address;
```

```
        value :=temp_str'address+1;
        Asm 16#8B#,2#00_110_110#,value'address;
          --Mov SI,[value]
        Asm 16#8B#,2#00_111_110#,address'addre-
          ss; --Mov DI,[address]
        Asm  16#B8#,0,41;
          --Mov  AL,0; Mov AH,41
        Asm 16#CD#,33;--Call MS-DOS to parse
                          command line
        Asm 16#80#,2#01_111_101#,1,16#20#;
                    --Cmp [DI+1], ' '
        Asm 16#74#,3; -- Jz Error
      Goto Done;
        Name_Error;--Name error if name points
                                    to a blank
      《Done》
        mask.fcb.extent;=0;
        mask.fcb.rec_size :=128;
        mask.fcb.rec_num :=Byte(0);
        mask.fcb.random_rec1 :=0;
        mask.fcb.random_rec2 :=0;
      End If;
      --File is modified since it is a pointer (no
        back assignment req'd)
      End set_fcb;
Procedure Open (Fyle  :In  Out File; Name : in
```

```
                String: Mode : In File_Mode) Is
--Open the file name and give it the mode
  mode
Begin
  IOresult :=0; --At least until an error
                  occurs
  mask :=New File_mask;
    --Allocate a block for the file
  FFConvert(mask,fyle);
  set_fcb(fyle, name);
  mask.fmode: = Byte(File_Mode'Pos(mod-
    e)); --Set the mode
  mask.buf_ptr :=0; --Buffer empty
  If mask.ftype =Byte(0) Then --Open Disk
    --File
    address := mask.fcb'address;
    --Open the file (MS-DOS call)
    Asm 16#8B#,2#00_010_110#,address'addr-
      ess; --Mov DX,[address]
           --Get the address of the fcb
    Asm 16#B4#,15;
        --Mov AH,15 (Open File Op code)
    Asm 16#CD#,33; --Call MS-DOS
    Asm 16#B4#,0; --Mov AH,0 (Clear Up-
                                per Bits)
    Asm 16#A3#,IOresult' address; --Mov[I-
```

```
                                        Oresult),AX
    If IOresult /=255 Then -- No error
      IOresult :=0;
      If mode = Read_Only Then
        mask.buf_ptr :=BUFFER_SIZE - 1;
          --Make the file
          --empty for read
      Else its OK at 0
      End If;
    Else
      FFConvert(Null,fyle);
            --File is  not open on an error
      Dispose(mask);
    End If;
  Elsif mask.ftype = Byte(4) and mode /=
    Write_Only Then
      Put("Cannot Read from the List Dev-
        ice");
    --Dispose(mask); --When  raise  excep-
      tion
    --FFconvert(Null,fyle); --File not  open
      Err_Exit;
  Elsif mask.ftype = Byte(5) and mode /=
    Read_Only Then
  Put("Cannot Write to the Keyboard");
  --Dispose(mask); --When raise exception
```

```
        --FFConvert(Null,fyle); --File not open
        Err_Exit;
      End If;
      If mask /=Null Then  --Link it into the
                                File_Chain
        mask.link :=File_Chain;
        File_Chain :=mask;
        mask.Eof_Flag:=False;  --Not End of
                                File Yet
      End If;
    End Open;
Procedure Create (Fyle: In Out File;Name : In
              String; Mode :In File_Mode) Is
  --Create the file name and give it the mode
    mode
  Begin
    IOresult :=0; --At least until an error oc-
                                        curs
    mask := New File_mask; --Allocate a bl-
                            ock for the file
    FFConvert(mask,fyle);
    set_fcb(fyle,name);
    mask.fmode := Byte (File_Mode'Pos(mod-
                        e)); --Set the mode
    If mask.ftype =Byte(0) Then
                        --Create Disk File
```

```
address :=mask.fcb'address;
--Open the file (MS-DOS call)
Asm 16#8B#,2#00_010_110#,address'addre-
  ss; --Mov DX,[address]
      --Get the address of the fcb
Asm 16#B4#,15; --Mov AH, 15 (Open
                  File Op code)
Asm 16#CD#,33; --Call MS-DOS
Asm 16#3C#,255; --Cmp AL,255 (OK
                        if not found)
Asm 16#74#,3; --Jz OK (Skip error)
                  Goto Error;
--Create the file (MS-DOS call)
Asm 16#8B#,2#00_010_110#,address'addr-
  ess; --Mov DX,[address]
       --Get the address of the fcb
Asm 16#B4#,22; --Mov AH,22 (Open
                        File Op code)
Asm 16#CD#,33; --Call MS-DOS
Asm 16#B4#,0; --Mov AH,0 (Clear Up-
                              per Bits)
Asm 16#A3#,IOresult'address;
                  --Mov [IOresult],AX
If IOresult /=255 Then
  IOresult := 0;
  mask.buf_ptr :=0; --Start at begining
```

```
                                                    of sector
        --Link it into the file chain
        mask.link :=File_Chain;
        File_Chain :=mask;
        Mask.Eof_Flag :=False;
                    --Haven't reached EOF yet
      Else --Directory Full
        FFConvert (Null,fyle);
                --File  is not open on an error
        Dispose(mask);
      End If;
      Return;
    《Error》 IOresult :=254; --File exists
      FFConvert(Null,fyle);  -File is not open
      Dispose(mask);
      Return;
    Else
      Put("Cannot Create a device - ");
        Put(name);
      -- FFConvert (Null,fyle);
      --When an exception is raised
      --Dispose(mask);
      Err_Exit;
    End If;
  End Create;
Procedure Delete(Name : In String) Is
```

```
  --Delete the file name
    temp : File;
  Begin
    mask := New File_mask;
    FFConvert(mask,temp);
    set_fcb(temp.name);
  --Use the mask variable for a temporary FCB
    If mask.ftype =Byte(0) Then
                          --Delete Disk File
      address :=mask.fcb'address;
      --Delete the file (MS-DOS call)
      Asm 16#8B#,2#00_010_110#,address'addr-
        ess; --Mov DX,[address]
             --Get the address of the fcb
      Asm 16#B4#,19;
      --Mov AH,19 (Delete File Op code)
      Asm 16#CD#,33; --Call MS-DOS
      Asm 16#B4#,0;
             --Mov AH,0 (Clear Upper Bits)
      Asm 16#A3#, IOresult'address;
                          --Mov [IOresult],AX
      --IOresult = 255 if nothing deleted
      Dispose(mask); --Toss temporary block
    Else
      Put("Cannot  Delete  a  Device - ");
      Put(name);
```

```
            Dispose(mask);
            Err_Exit;
            End If;
        End Delete;
    Procedure Close(Fyle : In Out File) Is
        --Close the file fyle
        Begin
          mask :=FConvert(Fyle);
          loc :=JLIB86.EClose'Address;
          Asm 16#A1#,mask'address;
                    --Load the file pointer into AX
          Asm  16#8B#, 2#00_110_110#, Loc'address;
            --Mov SI,[Loc]
                    --Load the routine address
          Asm 16#FF#,2#11_010_110#; --Call SI
                          --CALL indirect to Close
          FFConvert(Null,Fyle);
                               --Set the file to closed
        End Close;
    Function name(Fyle : In File) Return String Is
        --Return the name of the Open file
        str : string(21);
        Begin
          mask :=FConvert(Fyle);
          address :=Mask.All'address;
          loc :=JLIB86.EFile_name'Address;
```

```
      Asm 16#A1#,address'address; --Mov AX,
                                    (address)
                  --Load the file address
      Asm 16#8B#, 2#00_110_110#, Loc'address;
                                  --Mov SI,(Loc)
              --Load the routine address
      Asm 16#FF#,2#11_010_110#; --Call SI
              --CALL indirect to File_name
      --Now have a pointer to the resull string
      Asm 16#50#; --Push AX (Save the string
                                    address)
      address :=str'address;
      Asm 16#8B#,2#00_111_110#,address'addre-
                        ss;-- Mov DI,(adderss)
              --Get the address of the string
      Asm 16#5E#; --Pop SI
      Asm 16#B9#,20,0; --Load a counter of tw-
                                    enty bytes
      --Copy String
      Asm 16#F3#,16#A4#; --Rep MovsB
      --Copy Done
      Return str;
   End name;
Function Mode(Fyle : In File) Return File_Mode
                                    Is
   --Return the file mode of the open file Fyle
```

```
                                        Begin
        mask :=FConvert(Fyle);
        Return File_Mode'Val(Integer(mask.fmo-
          de));
        --Will die with a pointer error if file is
          not open!
      End Mode;
Function Is_open(Fyle : In File) Return Boolean
      Is
      --Is the file fyle open?
      Begin
        IOresult :=0; --No errors possible
        mask :=FConvert(fyle);
        If mask =Null Then
          Return False;
        Else
          Return File_Mode'Val(Integer(mask.fmo-
            de)) in
            Read_Only .. Read_Write;
        End If;
      End Is_open;
Procedure Replace_byte(Fyle : In File;By : In
    Byte) Is
      --Replace byt in the file
      Begin
          mask:=FConvert(Fyle);
```

```
        address:=Mask.All'address;
        loc:=JLIB86.RplcByte'Address;
        byt:=By;
        Asm 16#8B#,2#00_011_110#,address'addre-
                              ss;--Mov AX,[address]
                              --Load the file address
        Asm  16#8B#, 2#00_110_110#,Loc'address;
                                --Mov SI,[Loc]
                           --Load the routine address-
        Asm 16#A0#,Byt'Address;
                              --Mov AL, [Byt]
        Asm 16#FF#,2#11_010_110#;  --Call SI
          --CALL indirect to RplcByte
    End Replace_byte;
Function Get_Line (Fyle:In File) Return LString
  Is
    --Get a line from the file fyle
    ch:Character;cnt :Integer ;ftype :Integer;str:-
  LString;
    Begin
        cnt:=0;
        str(0):=Character'Val(255);
          --Set string to maximum length
          --to avoid string bounds error
        Loop
            Exit When End_of_Line(Fyle);
```

```
                Read(Fyle,ch);
                cnt:=cnttl;
                str(cnt):=ch;
            End Loop;
            str(0):=Character'Val(cnt);
            If End_of_File(Fyle) Then
                Return Str;
            End If;
            Skip_Line(Fyle);
                --Remove End of Line Marker
            Return Str;
        End get_line;
    Function Get_Line Return LString Is
        --Get a line from the default file
        Begin
            Return Get_Line(Current_Input());
        End Get_Line;
    Procedure Put_Line (Fyle:In File;Str:In LString)
      Is
        --Put the line to the file,with a new_line
        Begin
            Put(Fyle,str);New_line(Fyle);
        End Put_Line;
    Procedure Put_Line (Str:In LString) Is
        --Put the line to current_output file,with a
          new_line
```

```
    Begin
        Put(str);New_line;
    End Put_Line;
Procedure Put_Hex(Fyle:In File;val:In Integer)
  Is
    --Write the integer in hexidecimal (no special
      format)
    Begin
        mask:=FConvert(fyle);
        address:=mask.All'address;
        value:=val;          --Put it into a static
        loc:=Jlib86.Puthex'Address;
        Asm 16#8B#,2#00_011_110#,
        address'address;  --Mov BX,[address]
                          --Load file address
        Asm  16#A1#,value'address;
    --Mov AX,[value] (Get the value to write)
        Asm 16#8B#,2#00_110_110#,Loc'address;
                    --Mov SI,[loc]
                    --Load the routine address
        Asm 16#FF#,2#11_010_110#;  --Call SI
                  --CALL indirect to Put Hex
    End Put_Hex;
Procedure Putw (fyle:In File;str:In LString;
  width:In Integer) Is
    --Put str to file with blank padding to width
```

```
        len:Integer:=Character'Pos(Str(0));
    Begin
        Put(fyle,str);
        For i In len+1..width Loop
            Put(fyle,' ');
        End Loop;  --Blank padding
    End Putw;
Procedure Putw (str:In Lstring;width:In Integer)
  Is
    --Put str to default file with blank  padding
      to width
     Copied rather than called other to increase
      performance
        len:Integer:=Character'Pos(Str(0));
    Begin
        Put(Current_Output();str);
        For i In len+1..width Loop
         --This loop could be sped up  by filling
            Put(Current_Output(),' ');
         --in larger units than a single space
        End Loop;
    End Putw;
Function End_of_file(fyle:In File)Return Boolean
  Is
    --End of File Reached(in a text file)?
    Begin
```

```
        mask:=FConvert(fyle);
        If mask.ftype /=Byte(0) Then Return
          FALSE;--Devices cannot have EOF
           Elsif mask.eof_flag Then Return TRUE;
                --Hardware EOF found
           Else
           read(fyle.byt);--Get a look_ahead
                                    character
           Replace_byte(fyle,byt);
           Return byt = Byte(26);
                --(Ctrl)-Z marks EOF in files.
        End If
    End End_of_file;
Function EOF(fyle:In File) Return Boolean Is
    --End of File Reached(in a binary file)?
    Begin
        mask:=FConvert(fyle);
        If mask.ftype /=Byte(0) Then Return
          FALSE;--Devices cannot have EOF
        Else Return mask.eof_flag;
                        --Return Hardware EOF
        End If;
    End EOF;
Function Disk_full(fyle:In File) Return Boolean
  Is
    --Is the Disk full?
```

```
    --Works because End_of_file flag gets set on
    --a write error
    Begin
        mask:=FConvert(Fyle);
        If mask.ftype /=Byte(0) Then
            Return FALSE; --Devices cannot
                                    have EOF
        End If;
        Return mask.eof_flag;
    End Disk_full;
Function End_of_Line(fyle:In File) Return Boolean
  Is
    --End of Line Reached?
    Begin
        mask:=FConvert(fyle);
        If mask.eof_flag Then Return TRUE;
                        --At Hardware EOF
        Else
            read(fyle,byt);
                --Get a look_ahead character
            Replace_Byte(fyle,byt);
          --Return(byt=Byte(13)) or (byt=Byte
          --(10)) or
          --(byt=Byte(26)) or (byt=Byte(12));
          --(CR) or (LF) marks EOLN in
                files
```

```
            Asm 16#A0#,byt'address;
                                    —MovAL,[byt]
            Asm 16#3C#,26;   —Cmp AL,26(EOF
                                    character? )
            Asm 16#74#,10:   —Return True
            Asm 16#04#,16#F6#;
        —Add AL,F6(Make Lf thru Cr=0..3)
            Asm 16#24#,16#FC#;
        —And AL,FC(Mask 2 lower bits)
            Asm 16#74#,4;    —Return True
            Asm 16#B0#,0;
            —Mov AL,0(Return False)
            Asm 16#EB#,2;    —Jmp Return
            —Return True
            Asm 16#B0#,1;    —Mov AL,1(Return
                                    True)
            —Fall off bottom,returns value in AL.
            End If;
        —Does not modify the orginal file
    End End_of_Line;
Function Keypress Return Boolean Is
    —Return True if a key has been pressed since
      last input
    Begin
        Asm 16#B4#,11;   —Mov AH,11 (Console
                                    Ready Op code)
```

```
        Asm 16#CD#,33;          --Call MS-DOS
        Asm 16#24#,16#01#;  --And AL,01(Mask
                                so only last bit left)
        --Falling off end of function returns
          value in AL
    End Keypress;
Procedure Purge(Name:In String) Is
    --Delete the file name without an error if it
      does not exist
        temp:File;
    Begin
        mask:=New File_mask;
        FFConvert(mask,temp);
        set_fcb(temp,name);         --Use the mask
                --variable for a temporary FCB
        If mask.ftype=Byte(0) Then
                                --Delete Disk File
            address:=mask.fcb'address;
            --Delete the file(MS-DOS call)
            Asm 16#8B#,2#00_010_110#,address'
              address;  --Mov DX,[address]
                --Get the address of the fcb
            Asm 1#B4#,19;           --Mov AH,19
                        --(Delete File Op code)
            Asm 16#CD#,33;      --Call MS-DOS
        End If;
```

```
        Dispose(mask);   --Toss temporary block
    End Purge;
End IO;
```

(5) LONGIO.PKG

```
With Longops, Util, Jlib86, Ascii;
Package Body Longio Is
    --Copyright 1983 RR Software, P.O.Box 1512,
      Madison WI 53701
    --Permission is hereby given to distribute
      Object Code produced from
    --these libraries.
    --Long Integer I/O
    --Do not USE(use clause) this package until
      --the built_in procedure
    --bug is fixed.
    --Last Modified 4/ 5/83
    Use Longops;
    Pragma Arithcheck(Off);
    Pragma  Rangecheck (Off);
    Pragma Arithcheck(On);
    Pragma Rangecheck (On);
    --Variables used by Put. Get,and Rplcbyte.
      Static to increase speed.
    Temp_Output:String;
    Temp_Long:Long_Integer;
    Temp_width:Integer;
```

```
Sign:Boolean;
Char_str:String(1):=" ";
--Used for conversion from char to str.
Temp,Address:Integer;
Mask:Jlib86.File_ptr;
Char,Char2:Character;
Procedure Replace_byte (Fyle:In  File;Ch:In
                                    Character) Is
--Replace byt in the file(Local)
Begin
    mask:=Util.FConvert(Fyle);
    address:=Mask.All'address;
    Char2:=Ch;
    Asm 16#8B#,2#00_011_110#,address'addr-
      ess;--Mov BX,[address]
        --Load the file address
    Asm 16#A0#,Char2'Address;
                          --Mov AL,[Char2]
    Jlib86.Rplcbyte;      --Call Rplcbyte
End Replace_byte;
Procedure Get(Item:Out Long_Integer) Is
Begin
    Get(Util.Current_Input(),Item);
End Get;
Procedure Get (Fyle:In File;Item:Out Long_
                                    Integer) Is
```

```
Begin
    Standard.Get(Fyle,Char);
    While (Char In Ascii.Nul..Ascii.Blank)
      Loop
        Standard.Get(Fyle,Char);
                --Toss all control characters
    End Loop;
    --Get the sign
    If Char='+' Then
        Sign:=False;
        Standard.Get(Fyle,Char);
    Elsif Char='-' Then
        Sign:=True;
        Standard.Get(Fyle,Char);
    Else
        Sign:=False;
    End If;
    Temp_Long:=Lint(0);
    While Char In'0'..'9' Loop
      Temp_Long:=Ladd(Lmul(Temp_Long,
          Lint(10)),Lint (Character'Pos(Ch-
          ar) -Character'  Pos('0')));
        Standard.Get(Fyle,Char);
    End Loop;
    Replace_Byte(Fyle,Char);--Put back last
                            --character read
```

```
    Item := Temp_Long;
  --Full Ada would allow underscores, and
    based numbers!
End Get;
Procedure Put(Item:In Long_Integer) Is
Begin
    Put(Util.Current_Output(), Item, 0);
End Put;
Procedure Put (Fyle:In File;Item:In Long_
                                    Integer) Is
Begin
    Put(Fyle, Item, 0);
End Put;
Procedure Put (Item:In Long_Integer;Width:
                                  In Integer) Is
Begin
    Put(Util.Current_Output(), Item.Width);
End Put;
Procedure Put (Fyle:In File;Item:In Long_
                  Integer;Width:In Integer) Is
    --Note:this routine cannot print the lar-
      gest negative number.
    --which is disallowed by this implement_
      ation.
Begin
    Temp_Long := Item;
```

```
        Temp_width:=Width;
        --Reduce width if amazingly large
        While Temp_width)75 Loop
            Standard.Put(fyle,"          ");
            Temp_width:=Temp_width-10;
        End Loop;
        --Find sign,and save for later
        If Lge(Temp_Long,Lint(0)) Then
            Sign:=False;
            Standard.Put("Sign=Positive");
              New_Line;
        Else
            Sign:=True;
            Temp_Long:=Labs(Temp_Long);
            Standard.Put("Sign=Negative");
              New_Line;
        End If;
        If Temp_Long=Lint(0) Then
            Temp_Output:="0";
        Else
            Temp_Output:="";--No String yet
    While Temp_Long/=Lint(0) Loop
                temp :=L_to_Int (Lmod (Temp_
                        long,Lint(10)));
          Standard.Put ("Last character=");
                Standard.Put(temp);
```

```
                New_Line;
                char_str(1):=Character'Val(temp
                        +Character'Pos
                        ('0'));
                Temp_Output:=char_str & Temp
                        _Output;
                Temp_long:=Ldiv (Temp_Long,
                        Lint(10));
            End Loop;
        End If;
        --Add sign,if any
        If sign Then
            Temp_Output:="-" & Temp_Output;
        End if;
        --Add width,if necessary
        While temp_width>Character'Pos(Temp_
                        Output(0)) Loop
            Temp_Output:=" " & Temp_Output;
        End Loop;
        --Put result
        Standard.Put(fyle,Temp_Output);
    End Put;
End Longio;
```

(6) MATHLIB.PKG

```
With Util,Mathlib2;
Package Body Mathlib Is
```

```
--Mathematic functions library(8087 version)
--Copyright 1983 RR Software,Inc..P.O.Box
  1512,Madison WI 53701
--Permission is hereby given to distribute
  Object Code produced from
--these libraries.All Other rights reserved.
--Written by Randall Brukardt-March 1983
--Last Modified 4/27/83
Pragma Rangecheck(Off);
Pragma Arithcheck (Off);
--These must ALWAYS be --turned off.
WAIT:Constant:=16#9B#;          --Opcode
temp:Float;
ltemp:Long_Float
temp_x,temp_y:Long_Float;
Function Sqrt (Val:In Float) Return Float Is
    --Returns the sqrt of val
Begin
    Temp:=Val;
    Asm 16#BE#,temp'address;--Mov SI,temp
    Asm WAIT;
    Asm 16#D9#,2#00_000_100#;  --Fld Float
                                    ([SI])
    Asm WAIT;
    Asm 16#D9#,16#FA#;              --Fsqrt
    Asm WAIT;
```

```
        Asm 16#D9#,2#00_011_100#; --Fstp Float
                                          ([SI])
        Asm WAIT;
        Return temp;
    End Sqrt;
    Function Round(Val:In Float) Return Float Is
        --Rounds val to an integer value,and
          returns it as a float
    Begin
        Return Float(Integer(Val));
    End Round;
    Function Trunc(Val:In Float) Return Float Is
        --Truncate val to its integer part, and
          returns it as a float
    Begin
        Return Float(Integer(Val-0.5));
    End Trunc;
    Function Exp(Val:In Float) Return Float Is
        --Returns e** Val
    Begin
        Return Float(Exp(Long_Float(Val)));
    End Exp;
    Function Log(Val:In Float) Return Float Is
        --Returns the natural logarithm of Val.
          Val must be>0.
    Begin
```

```
        Temp:=Val;
        If Temp<=0.0 Then
            Put("Cannot take the logarithm of a
                negative number");
            Util.Err_Exit;
        End If;
        Asm 16#BE#,temp'address;—Mov SI temp
        Asm WAIT;
        Asm 16#D9#,16#ED#;      —Fldln2(Ln(2))
        Asm WAIT;
        Asm 16#D9#,2#00_000_100#;  —Fld Float
                                        ([SI])
        Asm WAIT;
        Asm 16#D9#,16#F1#;          —Fyl2x
        Asm WAIT;
        Asm 16#D9#,2#00_011_100#;
                            —Fstp Float ([SI])
        Asm WAIT;
        Return temp;
    End Log;
    Function Power (Val,Exp : In Float) Return
                                        Float Is
        --Returns Val ** Exp
    Begin
        If Val<0.0 Then
            Put ("Cannot raise a negative value
```

```
            to a power');
        Util.Err_Exit;
    Else
    Return Float (Mathlib2.Power(Long_
            Float(Val),Long_Float(Exp)));
    End if;
End Power;
--All angles are in radians!
Function Sin(Angle:In Float) Return Float Is
    --Returns the Sine of the angle
Begin
    Temp:=Angle;
    While Temp<0.0 Loop
                    --Make the value positive
        Temp:=Temp+10.0*PI;
    End Loop;
    Return Float (mathlib2.Sin (Long_Float
                (Temp)));
End Sin;
Function Cos(Angle:In Float) Return Float Is
    --Returns the Cosine of the angle
Begin
    Temp:=Angle;
    While Temp<0.0 Loop --Make the value
                                positive
        Temp:=Temp+10.0*PI;
```

```
        End Loop;
        Return Float (mathlib2.Cos (Long_Float
                        (Temp)));
    End Cos;
    Function Tan(Angle:In Float) Return Float Is
        --Returns the Tangent of the Angle
    Begin
        Temp:=Angle;
        While Temp<0.0 Loop
                        --Make the value positive
            Temp:=Temp+10.0*PI;
        End Loop;
        Return Float (mathlib2.Tan (Long_Float
                        (Temp)));
    End Tan;
    Function ArcTan(Val:In Float) Return Float Is
        --Returns the ArcTangent of the Value
    Begin
        Retu n Float(ArcTan(long_Float(Val)));
    End ArcTan;
    Function ArcCos(Val:In Float)Return Float Is
        --Returns the ArcCosine of the Value
    Begin
        Return Float(ArcCos(Long_Float(Val)));
    End ArcCos;
    Function ArcSin(Val:In Float) Return Float Is
```

```
    --Returns the ArcSine of the Value
Begin
    Return Float(ArcSin(Long_Float(Val)));
End ArcSin;
Function  Arc Tan2(X,Y:in Float) Return
                                        Float Is
    --Returns the ArcTangen of X/Y
Begin
    Return  Float(ArcTan2 (long_Float  (X),
                Long_Float(Y)));
End ArcTan2;
Function Deg_to_Rad (Angle:in Float) Return
                                        Float Is
     --Converts the Angle in Degrees to the
       same angle in Radians
Begin
    Return Angle *(PI / 180.0);
End Deg_to_Rad;
Function Rad_to_Deg(Angle: in Float) Return
                                        Float Is
     --Converts the Angle in Radians to the
       same angle in Degrees
Begin
    Return Angle *(180.0 / PI);
End Rad_to_Deg;
Function Sqrt(Val:in Long_Float) Return Long
```

```
                                        _Float Is
    --Returns the sqrt of val
Begin
    LTemp:=Val;
    Asm 16#BE#,ltemp'address;    --Mov SI,
                                    --ltemp
    Asm WAIT;
    Asm 16#DD#,2#00_000_100#; --Fld Long_
                                  Float([SI])
    Asm WAIT;
    Asm 16#D9#,16#FA#;          --Fsqrt
    Asm WAIT;
    Asm 16#DD#,2#00_011_100#;--Fstp Long_
                                  Float([SI])
    Asm WAIT;
    Return ltemp;
End Sqrt;
Function Round (Val:In Long _Float) Return
                                  Long_Float Is
    --Rounds val to an integer value,and
      returns it as a long_float
Begin
    Return Long_Float (Integer(Val));
      --Conversion Rounds
End Round;
Function Trunc (Val:In  Long_Float) Return
```

```
                                        Long_Float Is
    --Truncate val to its integer part, and
      returns it as a long_float
Begin
    Return Long_Float(Integer(Val-0.5));
End Trunc;
Function Exp(Val:In Long_Float) Return Long
                                        _Float Is
    --Returns e * * Val
Begin
    Return Mathlib2.Exp(Val);
End Exp;
Function Log(Val:In Long_Float) Return Long_
                                        Float Is
    --Returns the natural logarithm of Val.
      Val must be>0.
Begin
    LTemp:=Val;
    If LTemp<=0.0 Then
        Put("Cannot take the logarithm of a
              negative number");
        Util.Err_Exit;
    End If;
    Asm 16#BE#,ltemp'address;
                                --Mov SI, ltemp
    Asm WAIT;
```

```
        Asm 16#D9#,16#ED#;    --Fldln2(Ln(2))
        Asm WAIT;
        Asm 16#DD#,2#00_000_100#;
                              --Fld Long Float(CSI)
        Asm WAIT;
        Asm 16#D9#,16#F1#;    --Fyl2x
        Asm WAIT;
        Asm 16#DD#,2#00_011_100#; --Fstp Long
                                    _Float(CSI)
        Asm WAIT;
        Return Itemp;
    End Log;
    Function Power(Val,Exp:In Long_Float) Re-
                                urn Long_Float Is
        --Returns Val ** Exp
    Begin
        If Val<0.0 Then
            Put(" Cannot raise a negative value
                to a power");
            Util.Err_Exit;
        Else
            Return Mathlib2.Power(Val,Exp);
        End if;
    End Power;
    --All angles are in radians!
    Function Sin (Angle: In Long_Float) Return
```

```
                                        Long_Float Is
    --Returns the Sine of the angle
Begin
    LTemp:=Angle;
    While LTemp<0.0 Loop        --Make the
                                        value positive
        LTemp:=LTemp+10.0*PI;
    End Loop;
    Return Mathlib2.Sin(LTemp);
End Sin;
Function Cos (Angle: In Long_Float) Return
                                        Long_Float Is
    --Returns the Cosine of the angle
Begin
    LTemp:=Angle;
    While LTemp<0.0 Loop
                    --Make the value positive
        LTemp:=LTemp+10.0*PI;
    End Loop;
    Return mathlib2.Cos(LTemp);
End Cos;
Function Tan (Angle:In Long_Float) Return
                                        Long_Float Is
    --Returns the Tangent of the Angle
Begin
    LTemp:=Angle;
```

```
    While LTemp<0.0 Loop      --Make the
                                  value positive
        LTemp:=LTemp+10.0 * PI;
    End Loop;
    Return Mathlib2.Tan(LTemp);
End Tan;
Function ArcTan (Val:in Long_Float) Return
                                  Long_Float Is
    --Returns the ArcTangent of the Value
    Sign:Boolean:=False;
Begin
    If Val=0.0 Then
        Return 0.0;
    Elsif Val=1.0 Then
        Return (PI / 4.0);
    Elsif Val= -1.0 Then
        Return - (PI / 4.0);
    Else
        Temp_X :=Val;
        Temp_Y :=1.0;
        If Temp_X<0.0 Then
            Sign :=True;
            Temp_X:= -Temp_x;
        End If;
        If Temp_X>Temp_Y Then    --Invert
                --the operand,adjust answer
```

```
        Asm 16#BE#,temp_x'address;
          --Mov SI,temp_x
        Asm 16#BF#,temp_y'address;
          --Mov DI,temp_y
        Asm WAIT;
        Asm 16#DD#,2#00_000_101#;
          --Fld Long_Float([DI])
        Asm WAIT;
        Asm 16#DD#,2#00_000_100#;
          --Fld Long_Float([SI])
        Asm WAIT;
        Asm 16#D9#,16#F3#;  --FPATan
        Asm WAIT;
        Asm 16#DD#,2#00_011_100#;
          --Fstp Long_Float([SI])
        Asm WAIT;
        --Adjust answer for quadrant
        Temp_x:=(PI/2.0)-Temp_x;
    Else
        Asm 16#BE#,temp_x'address;
          --Mov SI,temp_x
        Asm 16#BF#,temp_y'address;
          --Mov DI,temp_y
        Asm WAIT;
        Asm 16#DD#,2#00_000_100#;
          --Fld Long_Float([SI])
```

```
            Asm WAIT;
            Asm 16#DD#,2#00_000_101#;
              --Fld Long_Float([DI])
            Asm WAIT;
            Asm 16#D9#,16#F3#;  --FPATan
            Asm WAIT;
            Asm 16#DD#,2#00_011_100#;
              --Fstp Long_Float([SI])
            Asm WAIT;
        End If;
        If sign Then
            Return -Temp_x;
        Else
            Return Temp_x;
        End If;
    End If;
End ArcTan;
Function ArcCos (Val: In Long_Float) Return
                                Long_Float Is
    --Returns the ArcCosine of the Value
Begin
    If(Val<-1.0)Or (Val>1.0)Then
        Put ( "Cannot take the ArcCos of
              Number Not In -1.0 to 1.0");
        Util.Err_Exit;
    Elsif Val=0.0 Then
```

```
        Return (PI / 2.0);
    Elsif Val =1.0 Then
        Return 0.0;
    Elsif Val= -1.0 Then
        Return PI;
    Else
        Return -ArcTan(Val / (Sqrt( (-Val)
                * Val+1.0)))+(PI/ 2.0);
    End If;
End ArcCos;
Function ArcSin (Val : In Long_Float) Return
                                Long_Float Is
    --Returns the ArcSine of the Value
Begin
    If (Val<-1.0)Or(Val>1.0) Then
        Put ( " Cannot take the ArcSin of
              Number Not In -1.0 to 1.0");
        Util.Err_Exit;
    Elsif Val=0.0 Then
        Return 0.0;
    Elsif Val=1.0 Then
        Return (PI / 2.0);
    Elsif Val= -1.0 Then
        Return -(PI / 2.0);
    Else
        Return ArcTan (Val/ (Sqrt((-Val) *
```

```
                    Val+1.0)));
        End If;
    End ArcSin;
    Function ArcTan2 (X,Y:In Long_Float) Return
                                        Long_Float Is
        --Returns the ArcTangent of X/Y
        --Uses Quadrant Recognition to return an
          Angle between 0 and 2 * PI
    Begin
        If Y=0.0 Then
            If X > 0.0 Then
                Return (PI / 2.0);
            Elsif X = 0.0 Then
                Return 0.0;
            Else
                Return 3.0 *(PI / 2.0);
            End If;
        Else
            Temp_X:=ArcTan(X / Y);
            If X>=0.0 Then
                Return Temp_X;
            Else
                Return Temp_X+PI;
            End If;
        End If;
    End ArcTan2;
```

```
    Function Deg_to_Rad (Angle: In Long_Float)
                                Return Long_Float is
        --Converts the Angle in Degrees to the
          same angle in Radians
    Begin
        Return Angle * (PI / 180.0);
    End Deg_to_Rad;
    Function Rad_to_Deg (Angle: In Long_Float)
                                Return Long_Float is
        --Converts the Angle in Radians to the
          same angle in Degrees
    Begin
        Return Angle * (180.0 / PI);
    End Rad_to_Deg;
End Mathlib;
```

(7) **STRLIB.PKG**

```
With Util;
Package Body Strlib Is
    --String Handling Package
    --Last Modified 10/ 5/82
    --Copyright 1982 RR Software,P.O.Box 1512,
      Madison WI 53701
    --Permission is hereby given to distribute
      Object Code produced from
    --these libraries.
Pragma Rangecheck(Off);      --Improve the code
```

performance

```
Pragma Arithcheck(Off);
Function Length (str:In Mstring) Return  integ-
  er Is
    --Return the length of the string
Begin
    Return Character'Pos(Str(0));
End Length;
Function Remove(str:In Mstring;pos,size: In Str-
                        Index) Return Mstring Is
    --Remove size characters from str at pos
    str2:Mstring;
    len:Integer:=length(str);
Begin
    str2:=str;
    If pos+size>len + 1 Then
        Put ("Attempt to Remove Characters not
              in string - pos=");Put(pos);
        Put(" size=");Put(size);Put("length=");
          Put(len); New_Line;
        Util.Err_Exit;
    End If;
    For i In pos+size..len Loop
        str2(i - size):=str2(i);
    End Loop;
    str2(0):=Character'Val(len-size);
```

```
    Return str2;
End Remove;
Function Insert (source.dest :In MString; pos:In
                    StrIndex) Return MString is
    --Insert source into dest at pos
    str :Mstring;
    len1 :Integer := Length(source);
    len2 :Integer := Length(dest);
Begin
    If len1 + len2 > 255 Then
        Put ("Strings too big to be inserted");
          New_Line;
        Util.Err_Exit;
    Elsif pos Not In 1..len2 + 1 Then
        Put ("Insert position not in string _
              pos=");Put(pos);
        Put('length=');Put(len2);New_Line;
        Util.Err_Exit;
    End If;
    str := dest;
    str(0) := Character'Val(len1 + len2);
                    --Set the new longer length
    For i In Reverse pos..len2 Loop
        str(i + len1) := str(i);
    End Loop;
    For i In 1..len1 Loop
```

```
        str(i + pos -1):=source(i);
    End Loop;
    Return str;
End Insert;
Function Extract (str : In Mstring;pos. size: In
                    StrIndex) Return Mstring Is
    --Extract size characters from str at pos
    str2:Mstring;
    len:Integer:=length(str);
Begin
    If pos+size-1>len Then
        Put("Attempt to Extract Characters not
            in string-pos=");Put(pos);
        Put(" size="); Put (size) ;
          Put(" length=");Put(len);New_Line;
        Util.Err_Exit;
    End If;
    str2(0):=Character'Val(size);
                              --Set the length byte
    For i In pos..pos+size-1 Loop
        str2(i - pos+1):=str(i);
    End Loop;
    Return str2;
End Extract;
Function Position (pattern,str : Mstring) Return
  Integer Is
```

```
    --Return the position of the first occurance
      of pattern in str,
    --or 0 if there is none
    len:Integer:=Length(str);
    len2:Integer:=Length(pattern);
    j: Integer;
Begin
    For i In 0..len-len2 Loop   --Zero trip loop
                                --if pattern is longer than
                                --string being matched
        j :=1;
    While j <=len2 And pattern(j)=str(i + j)
          Loop
            j :=j+1;
    End Loop;
        If j>len2 Then --Pattern found
            Return i+1;--Return the position
        End If;
        --Otherwise,go around again
    End Loop;
    Return 0;--Pattern is not found
End Position;
Function char_to_str(char:character) Return String
  Is
  --Convert a character into a string of length1
    ch:string(2);
```

```
Begin
    ch:=" ";
    ch(1):=char;
    Return ch;
End char_to_str;
Function str_to_int(str:Mstring) Return Integer Is
    --Convert a string into an integer
    pos,temp:Integer;
    len:Integer:=Length(str);
    sign:Boolean :=False;          --Positive until
                                   --otherwise determined
Begin
    temp:=0;--Value is zero so far
    pos:=1;        --Start at the beginning of the
                   --string
    If len =0 Then
        Return 0;
    End If;
    While (str(pos)=' ') or (str(pos)
          ='-') or (str(pos)='+')or
        (str(pos)=Character'Val(9))Loop
                   --Strip Blanks, Signs,and Tabs
            If str(pos)='-'Then --Flip the sign
                sign:=Not sign;
        End If;
        If pos=len Then
```

```
            Return 0;
        End If;
        pos :=pos+1;
    End Loop;
    While (str(pos) In '0'..'9') Loop
        If temp>3276 Then
            Put("Numeric Overflow");New_Line;
            Util.Err_Exit;
        End If;
        temp:=temp*10+(Character'Pos(str(pos))
                -Character'Pos('0'));
        If pos=len Then
            If sign Then Return temp;
            Else
            Return temp;
            End If;
        End If;
        pos:=pos + 1;
    End Loop;
    If sign Then
        Return - temp;
    Else
        Return temp;
    End If;
End str_to_int;
Function int_to_str(int:Integer)Return Mstring Is
```

```
    --Convert an Integer into a string
    str:String;
    sign : Boolean;
    temp : Integer;
Begin
    If int = 0 Then
        Return "0";
    Elsif int= -32768 Then
        Return "-32768";
    Else
        tr :=" ";
        sign :=int<0;
        temp :=Abs(int);
        While temp /=0 Loop
            str : =Char_to_str (Character ' Val
                ((temp Mod 10)+
                    Character'Pos('0'))) & str;
            temp :=temp / 10;
        End Loop;
        If sign Then
            str :="-" & str;
        End If;
        Return str;
    End If;
End int_to_str;
Begin
```

```
    Null; --No Initialization Code
End Strlib;
```

(8) **TIMELIB.PKG**

```
With Util;          --Hi and Lo are used in here
Package Body Timelib Is
    --Time Library - Get the current time and
      date from MS-DOS--
    --Last modified 6/22/82
    --Copyright 1982 RR Software, P. O. Box
      1512, Madison WI 53701
    --Permission is hereby given to distribute
      Object Code produced from
    --these libraries.
Use Util;
time1, time2 : Integer;--Local temporary variables
Function get_time Return Time Is
    --Get and return the current time
    temp : time;
Begin
    Asm 16#B4#, 44;      --Get time from OS
    Asm 16#CD#, 33;      --Call OS
    --Time is now in CX and DX
    Asm 16#89#, 2#00_001_110#, time1'address;
    --Mov [time1], CX
    Asm 16#89#, 2#00_010_110#, time2'address;
      --Mov [time2], DX
```

```
    --Now Hours in hi (time) ,minutes in lo
    --(time1).
    --seconds in hi (time2). 1/100 seconds in lo
      --(time2).
    temp.hours :=Integer(hi(time1));
    temp.minutes :=Integer(lo(time1));
    temp.seconds :=Integer(hi(time2));
    temp.fract :=Integer(lo(time2));
    Return Temp;
End get_time;
Function get_date Return Date Is
    --Get and return the current date
    temp: date;
Begin
    Asm 16#B4#,42;     --Get date from OS
    Asm 16#CD#,33;       --Call OS
    --Date is now in CX and DX
    Asm 16#89#,2#00_001_110#,time1'address;
      --Mov [time1],CX
    Asm 16#89#,2#00_010_110#,time2'address;
      --Mov [time2],DX
    --Now Year in time1,month in hi(time2),day
      --in lo(time2).
    temp.year :=time1;
    temp.month :=Integer(hi(time2));
    temp.day :=Integer(lo(time2));
```

```
    Return Temp;
End get_date;
Procedure put_date(fyle:In file; day : date) Is
    --Put the date to the file
Begin
    Case day.month Is
        When 1=>Put(fyle,"January");
        When 2=>Put(fyle,"February");
        When 3=>Put(fyle,'March');
        When 4=>Put(fyle,"April");
        When 5=>Put(fyle,"May");
        When 6=>Put(fyle,"June");
        When 7=>Put(fyle,"July");
        When 8=>Put(fyle,'August');
        When 9=>Put(fyle,'September");
        When 10=>Put(fyle,"October");
        When 11=>Put(fyle,"November");
        When 12=>Put(fyle,"December");
        When Others =>Put(fyle," * * Bad Date
                          * * ");
    End Case;
    Put(fyle,day.day);Put(fyle,",");
    Put(fyle,day.year);
End put_date;
Procedure Put_time(fyle:In file;clk:time) Is
    --Put the time to the file
```

```
Begin
    Put(fyle,clk.hours);Put(fyle,":");
    If clk.minutes In 0..9 Then  Put (fyle,'0');
      End If;
    Put(fyle,clk.minutes);Put(fyle,":");
    If clk.seconds In 0..9 Then  Put (fyle,'0');
      End If;
    Put(fyle,clk.seconds);Put(fyle,".");
    If clk.fract In 0..9 Then Put(fyle,'0') ;
      End If;
    Put(fyle,clk.fract);
End put_time;
Function elapsed_time(start,finish : Time) Return
                                          Time Is
    --Figure the elapsed time between  start and
      finish
    temp : time;
Begin
    temp.hours :=finish.hours - start.hours;
    temp.minutes :=finish.minutes-start.minutes;
    temp.seconds :=finish .seconds  start.seco-
                  nds;
    temp.fract :=finish.fract - start.fract;
    If temp.hours>=0 Then        --adjust so all
                                 --fields are positive
       If temp.fract<0 Then
```

```
            temp.fract := temp.fract + 100;
            temp.seconds := temp.seconds - 1;
        End If;
        If temp.seconds < 0 Then
            temp.seconds := temp.seconds + 60;
            temp.minutes := temp.minutes - 1;
        End If;
        If temp.minutes < 0 Then
            temp.minutes := temp.minutes + 60;
            temp.hours := temp.hours - 1;
        End If;
        If temp.hours < 0 Then
          Put(" * Error * Negative time elapsed");
          New_Line;
        End If;
    Else
        Put (" * Error *  Negative time elapsed" );
        New_Line;
    End If;
    Return temp;
End elapsed_time;
End Timelib;
```

参 考 资 料

(1) Ada导引与程序设计语言Ada参考手册（美国国家标准/军用标准）

〔美〕H.莱德加 著 袁崇义 徐泽同 译

科学出版社出版 1986年1月第一版

(2) Ada语言简明教程

〔美〕M.J.STRATFORD-COLLINS 著

麦中凡 译

北京航空学院出版社出版 1985年11月第一版

(3) Ada软件工程

〔美〕G.布奇 著 麦中凡 梁南元 译

科学普及出版社出版 1986年2月第一版

(4) AN INTRODUCTION TO ADA

〔英〕S.J.YOUNG

(5) PASCAL程序设计及其应用

王 诚 苏云清 赵毓陛 编著

清华大学出版社出版 1983年9月第一版

(6) C语言程序设计教程

孙玉方 孟庆昌 编著

北京大学第二分校出版 1984年2月

(7) 操作系统 —UNIX操作系统结构分析

刘日升 孙玉方 编著

北京大学第二分校出版 1983年2月