陈涵生　沈德琪等　编著

Ada語言实用教程

A PRACTICAL

ADA

COURSE

华东计算技术研究所
1988.5

序

Ada是一个十分通用的过程式程序语言，它不仅适用于实时系统，也适用于科学研究、工程、商业等各个领域。尽管它还有这样或那样的问题，美国国防部仍然坚持Ada的方向，毫不动摇。因为除了Ada之外，没有任何其它语言更能全面地满足嵌入系统的要求。然而，Ada的推广、应用，尤其是在嵌入系统的应用方面，却是一个世界范围的难题。我们认为，要使Ada语言在我国的计算机工业界和军事科学界逐步生根、开花、结果，必须在开展Ada研制的同时，不失时机地抓住Ada的推广、应用工作。为此，我们编写了这本《Ada语言实用教程》，希望能为专业工作者提供一个学习Ada的方便工具，为培训Ada程序员提供一份实用教材。

Ada语言是一个十分复杂、几乎无所不包的大语言。国际上不少软件开发者抱怨“Ada太年轻、太特殊、太难学”。我国的某些用户也有类似的意见。因此，我们在这本教程中试图在“简明、实用”上下功夫，这主要表现在以下四个方面：

第一，语言介绍中，在不损害其完整性的基础上，力求“少而精”，重点突出。我们只着重介绍了常用的数据类型、基本语句和基本说明、四类程序单位的结构与特点。

第二，通过大量的程序实例，深入浅出地介绍了Ada的一些重要概念及特性，不片面追求语言的形式化和叙述的严谨性；

第三，本教材附有《Ada—RTS编译系统使用说明》和可以在微机INTEL86／330,310上运行的程序实例，使读者能利用这个编译系统上机实习、边学边实践，以达到事半功倍、立竿见影的效果。

第四，每章都有一定数量的练习，並附以相应的参考答案，便于自学。

本书不是程序设计的启蒙教程，也不象Ada参考手册那样涉及到Ada语言的每一细节，而是集中地论述了多数程序员最常用到的一些特征。

本教程共有十二章。第一章介绍了Ada语言产生的历史背景、过程及其主要特点。同时扼要介绍了我们自行研制的Ada—RTS编译系统。第二、三、四、五、六章则分别介绍了Ada语言的基本成份。这和大多数现代过程语言没有什么实质性的区别，只不过在形式上和深度上有所不同。第七、八、九、十、十一章介绍了Ada的程序包、类属、异常、任务和分别编译等支持大型程序设计和体现现代软件工程原理的语言结构。第十二章涉及到Ada的输入、输出及其它依赖于机器的低级设施。为了便于参考和使用，本教程提供了Ada语法定义、Ada小字典、Ada—RTS使用说明、程序实例及练习答案等五个附录。

这本书是参与《Ada—RTS编译系统》研制的同志们集体努力的结果。陈涵生同志编写了第一、七、八、九、十章；沈德琪同志编写了第二、三、四、五、六章；施　礽同志编写了第十二章及Ada—RTS使用说明；徐　林同志编写了第十一章；陈双燕同志编写了四个附录。华东计算技术研究所情报室编辑组为本书的编辑、出版给予了大力支持和有效的合作，在此表示衷心的感谢！

由于水平有限，加上编写时间仓促，本书难免有错误及不当之处，诚恳欢迎读者批评、指正。

目　录

第一章　导　论

1.1　Ada的历史背景

Ada语言是由美国国防部（DOD）发起并主持设计的军用标准高级语言，它的产生是美军软件发展的结果。美国军用软件工作起步很早，从50年代中期算起已有三十余年的历史。到70年代美军武器系统中嵌入的计算机装备已相当普及。从低级的炮兵连、雷达站、飞行中队和舰队到高级战略指挥控制中心都配有大量的计算机，陆海空三军的主要战场系统和武器系统都配置了计算机软件，小到战术作战软件，大到全球性军事指控系统，用于战略进攻、战略防御和作战的指挥、控制、通信、情报系统，以及后勤支援和军事训练等各方面。

这个时期美军军用软件的发展有以下几个问题：

（1）软件规模大，研制周期长

一个大型军用软件从开始设计到成熟应用往往要几年甚至十几年时间。例如一艘DD963驱逐舰战术作战软件，有10万行代码，700多软件人员参加，研制周期为4年；地面防空系统100万行代码，花了10年时间才完成；战略空军司令部自动指挥控制系统70万行代码，工作量为1500人年；卫民反导系统软件214万行代码，工作量约5000人年，研制周期为7.5年。全球军事指挥系统软件2000万行代码，航天飞机软件约4000万行代码，这两个大型系统的软件工作量在数万人年以上。

（2）复杂程度高、可靠性差

许多军用软件要求实时并发控制、特殊的输入输出转换，通讯接口多，错综复杂，动态性和偶发性都很强，当出现意外时还要求有异常处理和自修改与自恢复的能力。70年代初期，军用计算机硬件平均无故障时间大约2000小时，而当时军用软件平均无故障时间只有8～48小时，可靠性很差。大型软件在使用过程中仍不断发现潜在性错误，约60%发生在设计阶段，40%发生在编码阶段。假设在设计时修改错误所需费用为1，到编码时为修改错误付出的费用要高1.5～3倍，联调时修改费用则高至3～5倍，鉴定验收时高至4～15倍，到运行时修改费用要高至11～90倍。因此，软件开发费用约有一半花在解决软件正确性问题上，70年代空军有个统计数字表明，有54%的错误在交付试验时仍未发现。

（3）投资大、费用高

如美军花在半自动地面防空系统软件的费用为2.5亿美元，全球军事指挥系统为7.2亿美元，国防部软件投资逐年增加很快，列表如下：

1973年	1980年	1984年	1986年	1990年
30亿	45亿	80亿	100亿	360亿

美军战斗机控制软件一条指令的成本为75美元，而修改一条已配置软件的指令成本为4000美元。软件费用超过了硬件，到70年代中，美军的软件费用占软、硬件总费用的8%左右，软件费用与硬件费用比例约4：1。

（4）各自为政、发展混乱

首先是军用语言不统一，陆海空三军各用其自己选定的语言，形成语种泛滥（约450余种），软件互不兼容。而许多软件产品功能类同，工作大量重复，给软件的使用、移植和交流造成极大的困难，浪费人力物力，而且影响实战使用。美国国防部为此付出高昂的代价，

背上沉重的包袱，迫使它不得不采取断然措施，统一三军语言，制定软件标准化计划，于是导致了Ada语言的诞生。

1.2 Ada的产生过程

70年代初，美各军部为改变高级语言繁杂的状态，纷纷设计和研制本部门的通用语言，但这种做法不能彻底解决问题。美国国防部从1974年起下决心开始执行一个新的语言计划，首先规定几种过渡语言，最终搞出三军统一的高级语言，这就是后来的Ada语言。

Ada语言是程序语言史上第一次按软件工程方法开发的语言。美国国防部提出语言要求，语言设计单位研制符合他们要求的程序语言，然后产生确认工具（Validation tools），最后实现这个语言。

1974年，成立了一个高级语言工作组（HOLWK），国防部颁布指令，宣布研制三军统一的语言，中止了所有其它新语言的研究和开发计划。1976年高级语言工作组批准了一个过渡性的高级语言名单，并以美国国防部5000.31号命令正式公布，它们是：

·美国国防部	FORTRAN	
	COBOL	
·陆军	TACPOL	
·海军	CMS－2	
	SPL／2	
·空军	JOVIAL	J3
	JOVIAL	J73

同时按照语言设计要求，先后发表了草人要求（STRAWMAN）和木人要求（WOODENMAN）。

1976年，经过反复讨论研究，将草人和木人要求修改成锡人要求（TINMAN），并按此计划对现有FORTRAN，COBOL，PASCAL，CMS－2，JOVIAL等23种语言进行评审，结果认为没有一种现成的语言能满足要求，但推荐PASCAL，PL／1与ALGOL68作为研制新语言的基础语言。

1977年，在锡人要求的基础上修改成铁人要求（IRONMAN），并明确语言要求除了可移植性和效率外，关键因素是可靠性、可读性和可维护性。美国国防部决定对语言进行国际招标，公开征求新语言的设计方案，收到了17家公司的投标，从中选了4家公司的文本，它们是：CII HONEYWELL BULL公司的绿色文本、INTERMETRICS公司的红色文本、SOFTECH公司的兰色文本、以及SRI INTERNATTON公司的黄色文本。值得注意的是，这些语言文本都以PASCAL作为基础语言。

1978年，从4个文本中选取了法国人Jean Ichbiah为主设计的绿色文本和Boston的红色文本。由于铁人要求不完备，后修改成钢人要求（STEELMAN）。

1979年，经过一系列会议研究，最后选中CII HONEYWELL BULL公司的绿色文本，并正式命名为Ada语言，以纪念世界上第一个程序员Ada Lovelace伯爵夫人。同时对Ada语言安排了正式的试用和评审，对用“Ada初稿”编写的各种不同的程序进行测试，提出了900条补充修改意见。

1980年，对绿色文本进行修改，7月公布Ada参考手册，12月10日是Ada的生日，颁布了Ada程序设计语言军用标准MIL STD－1815－1980，以后又经过多次讨论和修改，最后于1983年2月作为双标准形式（ANSI／MIL—STD—1815A—1983）定稿，从而完成了新

语言的设计和标准化工作。研制Ada的主要经验是：只要目标明确，看来是极其复杂的工作也是可以圆满完成的；使用计算机网络这一先进的通讯技术，集中世界各地1000多名专家的智慧，共同开发空前复杂的Ada语言，国际性学术交流使语言不断地完善；一名出色的、坚强的设计领导人保证了语言总体结构的一致性。

美国国防部提出公共高级语言的原始目标，除了改善程序的可靠性，同时要求开发可移植软件和可移植软件工具，降低软件全生命期的费用。他们在研制Ada语言的早期就认识到，这些目标单靠Ada语言本身是不够的。在开发Ada软件时，需要有一个良好的、集成的程序设计支撑环境。因此，从1979年夏季开始，与程序设计工作平行，提出了关于Ada程序设计支撑环境的一系列要求文件，经过“沙人”，“卵石人”文件，最后于1980年2月发表了John Buxton教授起草的“石人要求”（STONEMAN）。但是，由于当时人们对程序设计支撑环境的认识还很粗浅，其技术水平还处于发展的初期阶段，后来的工作发现，“石人要求”是一个缺乏具体细节的指导性文件，到现在为止，世界上还没有一个令人满意的Ada程序设计支撑环境。

1.3 Ada的主要特点

Ada语言是在PASCAL语言基础上发展起来的，它继承了60年代以来语言设计中形成的传统构造，其中包括ALGOL、PASCAL中典型的控制结构、数据类型和分程序、过程等概念。因此，它属于AP语言族。70年代发展起来的，以大型系统的设计、组织、实现与维护为其主要研究课题的软件工程，深刻地影响了Ada语言的设计。大多数程序设计语言，就语言本身而言，并不提供支持软件工程技术的设施。但软件工程中规范和风格，如果没有具体的技术措施来保证，仅仅靠软件人员之间内部规定和约定，实践上往往是难以贯彻实施的。Ada语言中许多概念和构造，直接反映了过去十年来软件工程技术中产生和形成的主要概念和成果。Ada语言是经典的语言构造与近代软件工程技术结合的产物。模块性、并发性、实时性等既是近代软件工程中的重要概念，也是Ada语言的主要特征。

（1）模块性

“模块性”要求能使系统比较容易地分解成可管理的一个个模块。Ada提供了两种方式的模块化支持，即由程序单位构造提供的逻辑模块，和由可分别编译的编译单位提供的物理模块。逻辑模块除了传统的子程序构造外，更重要的是程序包（package）和任务（task），前者为顺序模块，后者为并发模块。程序包（简称为包块）由外部可见的包块式和内部的包块体组成。包块式也称为程序包规格说明，它提供了程序包的使用界面，它可以包含类型、常量、数据对象、子程序、任务以至内含的程序包等说明。包块体对用户是隐蔽的，用户不可存取包块体内的局部数据。

```
package    P    is
--可见的类型，数据对象，子程序等说明;
end    P;
package  body  P  is
--数据、子程序、程序包、任务等局部说明;
--可见子程序、程序包、任务等的具体实现;
begin
--对局部量赋初值;
end  P;
```

包块体部分不必紧接在其可见部分之后。程序员可以先集中力量写好各模块的规格说明部分，以后再考虑具体的实现细节。包块式与包块体又是可分别编译的编译单位，即也是物理模块。这为清晰地规定系统中不同模块的接口提供了良好的手段，对于有许多程序员参加的大系统的研制者来说尤为重要，可使很多程序员平行地、彼此分离地进行模块的编制与测试工作。

（2）抽象性

所谓“抽象”是指在考察不同事物或对象时，我们强调其共性和本质的特征，而忽略其个性和次要的方面。抽象是一种认识工具，它使我们在不同时刻把注意力集中到对象的不同侧面。引入不同形式的抽象，减少了某一时刻必须处理的信息量，支持了系统的模块化。

Ada提供了一般程序语言所没有的数据抽象和类型抽象的机制。Ada的数据抽象按其强弱程度又可分为数据封装与结构抽象两种形式。数据封装把数据类型与该数据类型对象的一组基本操作结合在一起。例如：

```
package REAL-STACK is
    type STACK is array (1..533) of FLOAT;
    procedure PUSH (F : in FLOAT; S : in out STACK);
    procedure POP (F : out FLOAT; S : in out STACK);
end REAL-STACK;
```

结构抽象是数据的强化形式，它不仅包括数据封装，而且隐藏了被封装类型的数据结构。例如：

```
package REAL-STACK is
    type STACK is private;
    procedure PUSH (F : in FLOAT; S : in out STACK);
    procedure POP (F : out FLOAT; S : in out STACK);
end REAL-STACK;
```

这两个例子的区别是：前一个用户可直接对栈中元素进行操作，而后一个用户只能通过PUSH、POP操作间接地访问栈。人们常常把数据抽象作为衡量系统分解成内聚性好的模块的一条原则，系统的良好的模块化结构往往是应用数据抽象的结果。

类型抽象是编写可重复使用软件（或称为“软件组件”）的先决条件。Ada的类属单位（generic unit）是体现类型抽象的一种机制。例如，我们要建立关于栈的可重复使用包块它不依赖栈的元素类型，后可以按栈元素的具体要求衍生出相应的程序包。

```
generic type ELEMENT is private;
package STACK-MODULE is
    type STACK is private;
    procedure PUSH(E : in ELEMENT; S : in out STACK);
    procedure POP (E : out ELEMENT; S : in out STACK);
end STACK-MODULE;
```

类属单位是具体程序单位的样板或模式。类属单位与它衍生的程序单位的关系类似于数据类型与它相应的数据对象。我们现在可以衍生出种种不同的栈程序包。

```
package REAL-STACK is new STACK-MODULE (FLOAT);
package INT-STACK is new STACK-MODULE (INTEGER);
```

```
ackage ACT-STACK is new STACK-MODULE (ACCOUNT);
package SCORE-STACK is new STACK-MODULE (SCORE);
```

（3）并发性

并发性是现代程序语言的一个重要特征这是因为有不少问题只能用并发进程解决，如操作系统、飞机订票系统、银行系统、专家系统等；还有不少问题可将它自然地分解成并发进程，如分类算法、Dijkstra废料回收算法等。即使某些传统的顺序计算问题，也能找到并发的解决办法。需要并发处理的系统很难用不支持并发性的语言编写，因为这涉及到对共享数据与共享资源的存取问题。Ada是为数甚少的，语言自身提供并发进程支持的一种语言。任务（task）是支持Ada并发特性的主要语言构造，它在结构上类似于程序包，由可见部分和实现体组成。与程序包的情形一样，用户界面说明与任务的实现明显地分离。但任务与程序包有两个基本的区别：它与程序包的情形相反，任务体定义了独立的活动；任务界面（即可见部分）定义了任务的入口点。例如，考察管理航班座位的任务：

```
task FLIGHT-LIST is
    entry RESERVE (S : out SEAT);
    entry CANCEL (S : in SEAT);
end FLIGHT-LIST;
```

（4）实时性

一个用于实时程序设计的语言应提供以下四方面设施。第一，程序员可对任务调度进行显式控制。尽管Ada任务的入口项调用，其缺省方式是按先进先出的次序进行，但程序可用入口项族实现所要求的调度方案，也可利用优先级实行对任务的隐式调度。第二，任务通信的超时控制。Ada提供了关于任务的定时入口项调用（timed entry call）与选择等待语句中延迟（delay）候选等控制机制。第三，可与硬件设备的直接通信。Ada提供了与硬件设备接口的高级设施。利用Ada的表示子句（representation clause），程序变量可映射到设备的缓冲寄存器与状态寄存器；把任务的入口项与中断地址联系起来，可将中断作为入口项调用处理。第四，异常处理，实时程序应能对执行中产生的异常和出错进行实时修复，必要时要终止出错任务的继续执行。Ada提供了处理异常的机制，每个程序单位可分为正常实现部分和出错处理部分，以分层地进行出错处理。

（5）可靠性与可维护性

Ada语言的可靠性主要表现在它是一个典型的强类型语言，即语言中的每一对象都有唯一的类型，且其类型可编译时完全确定，这样，避免了逻辑上不同概念的混淆，可在编译时查出大多数语法错误。

美国国防部的调查指出，他们的软件总开支中，90%花在维护阶段。因此，语言设计者十分强调语言的易读性、易于理解、易于修改。语言的易读性考虑贯穿于Ada语言设计的各个侧面，包括词法结构，语句选择，程序单位构造等问题。Ada中广泛采用线性阅读原则，读一个Ada程序就象读一段普通的文章那样，一行接着一行，读到哪里就能理解到哪里，不象有的语言要前后反复对照才能理解这段程序的含义。

语言中提供的数据封装、分别编译、库管理等机制，既是编写大型软件所必需的，也是提高软件的可靠性、可维护性的重要措施。

1.4 一个简单的Ada程序

一个完整的Ada程序由一系列程序单位组成。最外层应包含一个主程序和若干个附带的

库单位，这些库单位为主程序提供必要的服务机制。主程序以过程的形式出现。

我们分析一个简单的Ada程序。该程序对一串数求平方根。如果输入的X值小于0，则印出“not calculable”，否则印出它的平方根值；如果输入的X值为0，则此程序计算结束。

```
with SQRT, SIMPLE-IO;
procedure PRINT-ROOTS is
use SIMPLE-IO;
   X: FLOAT;
begin
   PUT ( "Roots of various numbers" );
   NEW-LINE(2);
   loop
     GET(X);
     exit when X=0.0;
     PUT(X);
     PUT( "is" );
     if X <0.0 then
       PUT ( "not calculable" );
     else
       PUT (SQRT(X));
     end if;
     NEW-LINE;
   end loop;
   NEW-LINE;
end PRINT-ROOTS;
```

程序第一行的with子句，表示主程序要用到两个库单位，其中一个为SQRT函数，另一个为SIMPLE-IO程序包，它提供了若干个简单的输入／输出功能，这包括读入数、印出数、印出字符串等。

主程序为名叫PRINT-ROOTS的过程，过程体的语句用begin，end语句括号括起来。PUT(SQRT(X))为调用SIMPLE-IO程序包内PUT过程的过程语句，X浮点变量为调用SQRT函数的变元。use SIMPLE-IO子句使程序包SIMPLE—IO内各种功能可直接在过程体内被引用，如果省略了这个use子句，则必须写成

```
SIMPLE-IO.PUT(SQRT(X));
```

Ada中每个语句与说明均以“;”号作为终结符号，而在Algol或Pascal中，分号是分隔符而不是终结符号。Procedure，is等指示程序结构的标识符，不能作为一般标识符使用，我们称之为保留字。为程序易读起见，本书用小写字母表示保留字，用大写字母表示一般的标识符，如PUT，X等等。

在最后的end与分号之间，可任选地重复过程名PRINT-ROOTS，以便使程序的总体结构更加清楚。

要注意Ada的括号结构。loop一定要与end loop匹配，if一定要以end if结尾。Ada的所有控制结构都是这样闭括号结构形式。

我们现在概括地考察库单位SQRT函数与SIMPLE_IO的一般形式。

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

函数与过程在结构上十分类似，所不同的是：函数返回一个结果值，作为表达式的一部分被调用；而过程作为语句被调用，没有返回值。

Ada的注解以"--"符号开始，这里注解表示省略了在R中获得F平方根的求值步骤。

```
package SIMPLE_IO is
    procedure GET(F : out FLOAT);
    procedure PUT (F : in FLOAT);
    procedure PUT(S : in STRING);
    procedure NEW_LINE (N : in INTEGER := 1);
end SIMPLE_IO;
```

此包块式内含有四个过程说明。GET过程有一个out输出参数，它的作用把过程内求得的值传送到GET(X)的实在参数X。这里有两个PUT过程规格说明，一个的参数为FLOAT类型，另一个为STRING类型。最后要指出的，NEW_LINE过程的参数有一个缺省值1，即如果相应实参没有给出，(例如，NEW_LINE;)，则就取缺省值1，即换一行。

Ada中有一个特殊的程序包STANDARD。对每一个实现，我们认为它总是存在的。该包块内含有如FLOAT这样的预定义标识符。所谓预定义标识符，即不必另加说明就被赋以明确的含义，但这种预定义标识符可在程序中重新定义。我们可以假定STANDARD程序包自动地存在，且程序内不必用with子句提及它的名字。

1.5 Ada_RTS编译系统

Ada_RTS系统是一个在微型机Intel 86/330(iRMX86操作系统)上实现的Ada真子集编译系统。它的软、硬件环境的最低要求是：

(1)硬件

提供以下配置的Intel 86/330A系统(或Intel 86/310系统，或国产M-0572全加固微机系统)

- 键盘与CRT
- 盘驱动器
- 不小于640K字节的内存
- 8087浮点部件

此外，考虑到程序调试的效率，系统应具有硬拷贝设施。

(2)软件

- iRMX86实时操作系统(4/5/6版本)
- PASCAL 86程序设计语言
- PL/M 86程序设计语言
- ASM 86宏汇编语言
- LIB 86库管理程序
- LINK86连接程序

· LOADER 86系统装入程序　　· CEL 87常用基本函数库

· iRMX扩充I／O系统　　· iRMX基本I／O系统

Ada-RTS语言为Ada的一个实时子集，它按以下原则和条件确定的：

①它应是一个Ada的真子集，以保证用Ada-RTS写的程序能与Ada程序向上兼容。

②它应能在实时系统的某些领域内得到应用。该子集的选舍，主要考虑到空军武器和防卫系统对程序设计语言的要求，即要满足作战飞行软件、指挥与控制系统、通讯、区域支援、自动检测设备及仿真和训练等方面的要求。

③要求该语言编译能在一般的微型机上实现，且其实现效率是用户单位可接受的。因此，我们不得不删去了一些不大影响实际使用的Ada数据类型和某些语言功能。

与Ada语言文本相比，Ada-RTS在以下方面作了不同程度的删减：

①词法元素上基本不动，但有三点小的区别：格式控制符删去了横向列表、纵向列表与换页；带底数面值中，允许的底是：0，16，2。编译信息(pragma)只有 LIST，PRIORITY及SUPPRESS三种。

②说明和类型中删去了以下内容：

· 基本说明中没有类属说明、异常说明、类属衍生和换名说明。

· 没有一般的实数类型定义，代之以预定义浮点类型FLOAT，LONG FLOAT及预定义定点类型DURATION。整数类型定义增加了预定义类型LONG INTEGER。

· 没有存取类型和派生类型定义

· 记录类型中没有判别式和变体部分

③名字和表达式中删去了以下内容：

· 名字中没有数组片

· 结构常量（aggregate）中没有按名的结构常量

· 逻辑运算符中没有短路控制形式

· 受限表达式只有枚举类型名'（表达式)这种形式

④语句中删去了以下内容：

· 简单语句中没有转语句，引发语句和代码语句，也没有标号部分

· 复合语句中没有分程序（block）语句

⑤子程序中删去了以下内容：

· 函数指向符（designator）中没有运算符号

· 参数指明中没有缺省表达式

· 参数关联中没有按名关联

· 函数运算符与枚举值都不允许一名多用（overloading）

⑥程序包部分删去了限定类型（limited type）且不允许嵌套。

⑦几乎全部保留了Ada的任务部分。

⑧程序结构中，二级单位的定义中没有子单位（Subunit）。

⑨异常处理中只支持五种预定义异常。

⑩输入／输出部分中没有直接存取文件。

Ada-RTS编译系统的总体结构如图1—1所示

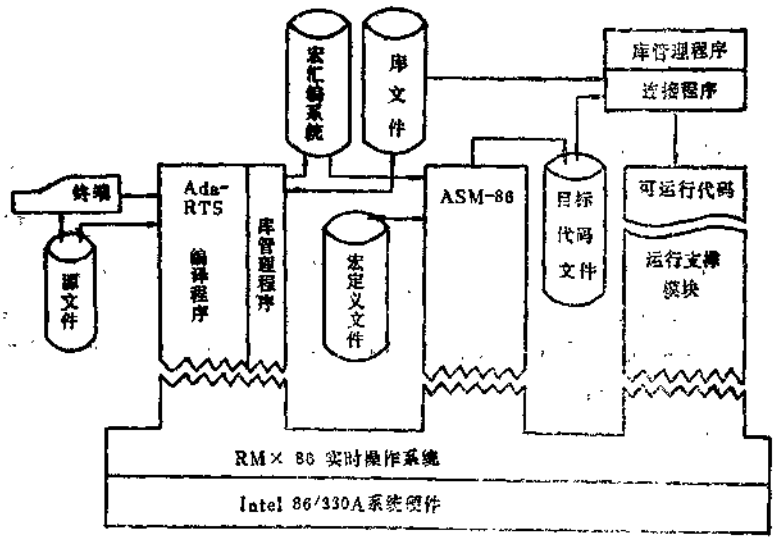

图 1—1 Ada_RTS编译系统的总体结构

从图1—1中可以看出，整个编译系统由五个部分组成：编译程序，宏汇编系统，运行支撑模块，以及支持分别编译设施的库管理程序与连接程序。

Ada-RTS编译程序的输入为Ada-RTS源程序，它执行词法分析，语法分析，语义分析和代码生成等工作，其输出为ASM86宏汇编程序。ASM86宏汇编系统接受的编译程序生成的宏汇编程序及宏汇编程序中要引用的宏定义文件，它的输出为Intel目标模块形式的目标代码文件。

库管理程序与连接程序是支持Ada-RTS系统分别编译设施的两个实用程序。在分别编译的情况下，每个Ada程序都将与一个库文件对应。库文件里含有组成Ada程序所有编译单位的有关信息，它包括子程序规格说明，程序包规格说明，任务规格说明中有关对象名和类型信息。这些信息由编译程序产生，库管理程序则通过对库文件的统一管理，负责在库文件中记录、删改、读取这些信息。分别编译时，编译程序通过库管理程序读入关于正在编译的编译单位的环境信息。编译完成后，记录新的编译单位信息，更新相应的环境信息。连接程序通过库管理程序读入库文件，并根据库文件中有关的环境信息，从目标代码文件取出相应的目标模块，完成连接工作，从而得到一个可运行的Intel 86程序。

运行支撑模块主要用于支持Ada-RTS程序的实时运行和支持Ada的I／O操作，它是实现Ada并发处理和I／O操作中动态语义的主要程序部件。这主要包括任务的实现、调度与任务间协调和Ada I／O例行程序。它与编译程序的控制接口为一个可供调用的过程集合。任务方面包括任务对象的制作、激活、夭折、延迟执行、终止、入口项调用、入口项接受、任务选择等几个过程。为提高执行效率，这些过程的过程体全部用汇编语言编写。在编译时，每遇到TASK类型定义时，系统要构造一个称为 TSIB(Task-Static-Information-Block)的信息块，它记录了创立TASK类型对象所要求的一些信息，以供TASK对象制作时使用，因此TSIB为运行支撑模块与编译程序的数据接口。I／O方面包括create，open，reset，close，Read，write，getchar，getstring，put string，get line及长、短整数，实数的翻译等16个例行子程序，它们建立在系统提供的基本I／O系统（BIOS）基础上。

第二章　基　础

2.1 词法元素（lexical unit）

Ada程序是由一次或多次编译正文所组成。编译正文是由一列互相分隔的词法元素所组成。每个词法元素可以是标识符（包括保留字）、定界符、字面值或注解。程序的效果只依赖于构成它的各次编译的特定词法元素序列，但不包括其中可能有的注解。

2.1.1 字符集（character set）

对于一个特定的Ada编译系统，其允许使用的字符集要取决于实现，不过语言定义要求所有的实现至少都要包含下列字符：

- 26个大写字母

  A B C … X Y Z
- 10个十进制数字

  0 1 2 … 7 8 9
- 专用字符

  " # & ' ( ) * + , - . / : ; < = > _ |
- 空格字符

对于竖杠（|），井号（#）和引号（"）可允许用替换符来代替它们。

基本字符集还可扩充，ASCII字符集的其他字符也可用，它可以包括：

- 26个小写字母

  a b c … x y z
- 其他专用字符

  ! $ % ? @ [\ ] ^ { } ~
- 格式控制符，如横向制表符、竖向制表符，回车，换行。

定界符是下列特殊字符之一：

& ' ( ) * + , - . / ; < = > |

特别上述特殊字符可组合成下列的组合定界符之一。

=>　箭头，用在结构常量，情况语句中。

..　双点，用在范围。

**　双星号，用在指数。

:=　赋值号

/=　不等号

>=　大于或等于

<=　小于或等于

<<　左标号括号

>>　右标号括号

<>　盒号，用在数组说明

2.1.2 标识符（identifier）

标识符是一个字符序列，用以表示程序中所有不同实体的名字，如类型名、对象名、子

程序名等。

标识符按如下规则生成：

（1）标识符必须从字母开始。

（2）后续字符可以是字母、十进制数字或下划线符。

（3）标识符可由任意多个字符组成。

（4）若允许用小写字母，则大小字母没有什么区别，作为同一字符对待。例如PI，Pi，pi，pI都看成是同一标识符。

我们用稍有变动的巴科斯范式来描述这些规则，如下所示：

〈标识符〉::=〈字母〉｛〔下划线〕〈字母或数字〉｝

〈字母或数字〉::=〈字母〉|〈数字〉

〈字母〉::=〈大写字母〉|〈小写字母〉

花括号｛ ｝表示括在其内的部分可以重复任意次，包括零次。方括号〔 〕表示括在其内的实体是可选的。用竖杠隔开表示任选项。

用下面的例子来说明这些规则：

Ada86　　——正确，

Ad_a86　　——正确，

Ada-86　　——不正确，‘—’不是下划线‘_’。

86_Ada　　——不正确，不是字母开头。

Ada语言中有一些标识符有特定的语义，它被“保留”给编译程序自用，因此用户不能把它们当作标识符用，以后在各章中将会陆续碰到。如：if、procedure和end等，我们把它们称为保留字（reserved）並用小写字母来表示，这里共有六十二个保留字。

练习 2.1.2

（1）下面哪些不是标识符？为什么？

a）Ada

b）fish &chips

c）Ada-LIB

d）HDS 8209

e）Ada__86

f）77E2

g）X_

h）AdaLIB

i）begin

2.1.3 字面值（literal）

字面值又可称为直接量，它分为数值字面值，字符字面值和字符串字面值。下面，我们将详细介绍。

2.1.3.1 数值字面值（numeric literal）

数值字面值有两类：整数字面值与实数字面值，它们之间最主要的区别在于是否有小数点存在，有小数点的，意味着是实数字面值，反之就是整数字面值。

例如：130　　——整数字面值

1.3　　——实数字面值