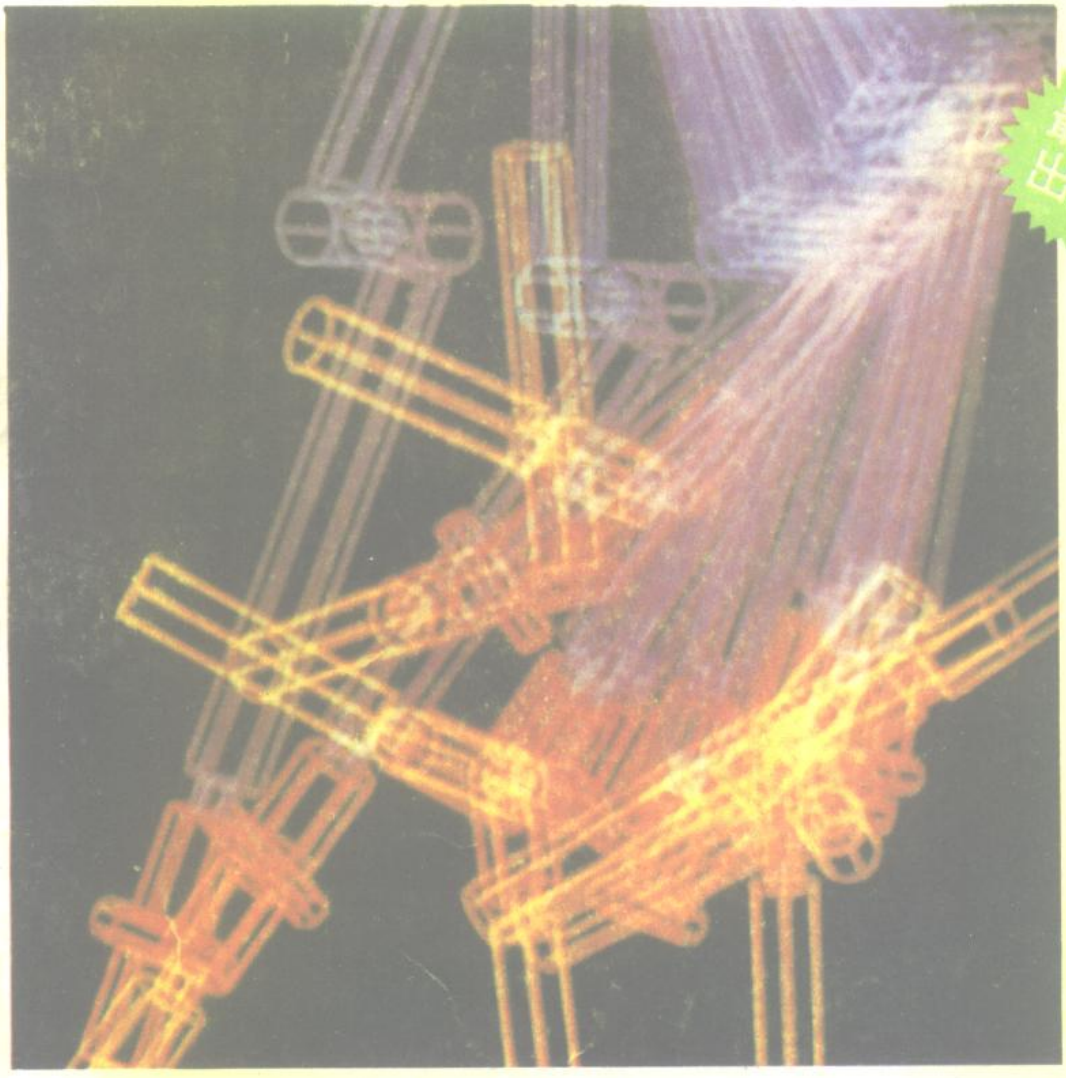
Ada

高级程序设计语言

陶辅周　纪希禹　李旭伟　编著



電子工業出版社

Ada高级程序设计语言

陶辅周　纪希禹　李旭伟　编著

电子工业出版社

京(新)登字 055 号

内 容 简 介

本书主要讲述 Ada 程序设计语言、如何利用程序包组织大型程序和利用程序包隐藏信息，以及如何定义抽象数据类型，最后还介绍了分别编译、异常和文件。书中所述为最新 Ada 语言版本。

本书语言生动，内容深入浅出，可读性强。其读者对象可以是初学程序设计的高校学生，也可以作为具有一定 PASCAL、FORTRAN 语言程序设计经验的读者的 Ada 语言参考书。

Ada 高级程序设计语言

陶辅周　纪希禹　李旭伟　编著

责任编辑　秦　梅

*

电子工业出版社出版(北京市万寿路)

电子工业出版社发行　各地新华书店经售

北京顺义李史山胶印厂印刷

*

开本：787×1092 毫米 1/16 印张：10.75 字数：272 千字

1993 年 3 月第 1 版　1993 年 3 月第 1 次印刷

印数：6000册　　定价：8.00元

ISBN7-5053-1952-3/TP・477

序

本书旨在作为从事计算机应用工作者和高等院校学生学习计算机程序设计的一本入门教材，并且假定读者以前没有任何程序设计的知识；同时也试图为那些有一定 PASCAL 或 FORTRAN 语言程序设计经验的读者提供一本有益的 Ada 语言参考书。

目前，PASCAL 语言是较流行的语言。事实上，它最初是为教学目的而设计的，如果不加以适当地扩充是不能用其编制大型软件的。可靠性是现代计算机语言的重要特征，Ada 语言中的程序包概念是研制可靠程序的有效工具，而 PASCAL 语言就没有程序包概念，可以说程序包的概念是 Ada 语言对程序设计语言的最大贡献。

软件的可靠性及研制费用的增长曾导致了一场“软件危机”，为此，美国政府在 1975 年起草了一份需求草案，并希望这些要求能体现在某一语言中。然而，在当时还没有一种语言能满足所提出的要求，于是决定研制一种新的语言。在这场国际招标中，由 J·I Chbiah 所领导的 C Ⅱ Honeyewll Bull 的研究小组中标获胜。将这种新语言命名为 Ada 是为了纪念 Augusta，她被公认为是最早的程序员。

最初的 Ada 语言参考手册于 1980 年问世，自那时起，Ada 语言已经过数次修改，本书所述的是最新的版本。国内也于最近完成了汉化 C-Ada 的研制，从而推动了 Ada 语言在国内的普及。

Ada 是一个相当大型的语言，本书不可能面面俱到，关于类型和对象的描述已被简化，但这并不影响 Ada 语言的体系。本书的后半部分介绍了如何利用程序包组织大型程序和利用程序包隐藏信息，以及如何定义抽象数据类型。最后还介绍了分别编译、异常和文件。

本书的写作过程中，始终得到中科院数学所徐泽同研究员的热心支持，在此表示衷心感谢！

编者

1991 年 2 月于四川大学

目　录

第一章　怎样用计算机解题

§1-1　引　　言

本书介绍怎样用计算机解题。主要描述怎样系统地设计算法以及怎样在计算机上实现这些算法。进行这一系列工作的最终目的是编制计算机程序，编制的程序应具有易理解、简洁、可靠、效率高等特点。

Ada 程序设计语言能够适应上面的要求，是一种比较好的语言。它广泛应用于国防部门，日益受到用户的喜爱。由于它的设计严谨、合理，弥补了其他语言的不足，因此越来越受到社会各个领域计算机工作者的欢迎。为了跟上时代的步伐，从读者的愿望出发，在本书中我们采用通俗易懂的语言，较系统地介绍 Ada 语言的程序设计方法。即使以前没学过任何程序设计语言的读者，也可顺利地阅读本书。

目前，计算机已深入社会生活的各个领域，以致普通的人对计算机也略知一二。为了使读者对计算机的内部结构和原理有一个较清晰的认识，我们首先给出一个简单的定义——计算机是一个能接受指令并执行指令序列的电子装置。

为了使计算机完成某一任务，必须首先给计算机输入明确的指令，这些指令必须是由某种计算机能理解的语句写成。这些指令集就称为计算机程序。

当用计算机解题时，不仅仅是坐下来写一个程序而已。首先必须明确题意，然后再设计相应的算法，最后才开始编制程序。因此，程序设计不仅仅是编制程序，明确题意和算法设计比编制程序更为复杂。

在介绍程序设计之前，首先简介计算机的结构。

§1-2　关于计算机的一般知识

从个人微机到巨型计算机，尽管它们在体积、性能、价格上的差异可能很大，但它们都是由相同的基本器件构成的。

1. 中央处理单元(CPU)

CPU 是实际执行指令的单元，每条指令是相当简单的。计算机的性能由执行指令的速度确定，一般每秒钟可完成几万次至几千万次的操作。对于计算机来说，用秒作时间单位已显得过大，一般都用毫秒(10^{-3} 秒)、微秒(10^{-6} 秒)、毫微秒(10^{-9} 秒)。运行速度快是计算机的一大特点，计算机迅速地进行一系列简单的操作就可以完成很复杂的任务。

2. 主存储器

主存储器用于存放程序指令和指令所需的数据。计算机在执行指令前，这些指令须事先装入计算机的主存储器中。当计算机执行这些指令时，就称作运行或执行程序。

3. 后备存储器

通常对不需立即调用的程序和数据可以存储在所谓的后备存储器中。后备存储器可以是磁带或磁盘。一般它所能存储的信息要比主存储器多得多，因此可用它长期保存程序和数据。后备存储器可被视为一个大型文件编排系统，它把程序之类的信息归档为单个文件。程序被运行前，必须事先把程序指令从后备存储器的文件中拷贝后调入主存储器中。

4. 输入、输出设备

输入设备用于把信息读入计算机，而输出设备则用于输出程序运行的结果。最常用的设备是交互式终端，它能同时用作输入、输出设备。交互式终端通常由一个显示器（相当于家用电视机的屏幕）和一个打字机键盘组成，一般把它称为直观显示部件（VDU）。信息由键盘敲入，键入的信息以及程序的运行结果都显示在屏幕上。用这样的方式，用户就可和计算机进行"对话"。打印机是另一种输出设备，它可以把程序或结果打印出来以供进一步研究。

主存储器、CPU、后备存储器、输入输出设备都被称为计算机硬件，而程序被称为计算机软件。如果只有硬件，则计算机尤如一个白痴，是干不了什么事情的；相反，没有硬件，则"皮之不存，毛将焉附"，软件的功能也就无从谈起。

§1-3 计算机软件

当使用计算机时，总会发现机器中事先已装入了一系列的程序，这些程序的功能是对用户提供帮助。这个程序集叫做计算机操作系统。用户所使用的计算机实际上是硬件和操作系统的结合体。我们可以简单地把这两部分视为一个整体而不必关心它们的具体分工。以后将称硬件与软件的组合体为计算机系统。

至此，尚有一重要的问题没有考虑，即每台计算机仅能理解一种语言——即它的机器语言。由于机器语言很难理解，用它来编写程序很不方便，故代之以特殊设计的、易于使用的语言来编写。这些语言被称为高级程序设计语言，如 FORTRN、COBOL、C 语言、PASCAL 和 Ada 等等。

然而，计算机是不能理解这些高级语言的。为使计算机能理解采用高级语言编制的程序，必须首先将其翻译成机器语言。因此，除了操作系统外，还需有特殊的翻译程序，即编译程序。如果想在计算机上运行一个用 FORTRAN 或 PASCAL 编制的程序，就需要有相应的 FORTRAN 或 PASCAL 编译程序，同样，若要运行一个用 Ada 编写的程序就要有 Ada 编译程序。总之，对于每一种高级语言都要有相应的编译程序。

由于需要编译程序，使用高级语言显得很麻烦，但事实上却有其很大的优越性。例如，在一台计算机中配置有 Ada 编译程序，就可以运行一个由 Ada 语言写成的程序。因此人们通常将 Ada 等这类高级语言称为独立于机器的语言。但在实际应用中，语言并不完全独立于机器。因为大部分程序设计语言存在不同的版本，不同的机器性能也不一样，这就可能造成为一类机器设计的程序在另一类机器上运行不了的问题。针对可能出现的这些问题，Ada 语言的设计者通过对语言的严格定义以及避免"方言"等手段做了处理，通过这种处理，Ada 语言真正地和机器相互独立开来了。

使用高级语言编写程序还有另一个优越之处：能够及早发现程序中的一些错误以供程序员修改。程序设计语言和自然语言一样，也有其语法规则，这些规则规定了什么是允许的以及什么是不允许的。如果在编写程序时违犯了语言规则，则在从高级语言转换成机器语言的过程中，编译程序会发现并指出这些错误并通知程序员修改。这样，程序员就可以纠正错误然后再进行编译调试。

由高级语言转换成机器语言的过程如图 1.1 所示。

注意，计算机实际执行的是 Ada 程序的目标程序（由机器语言写成），而不是源程序自身。运行

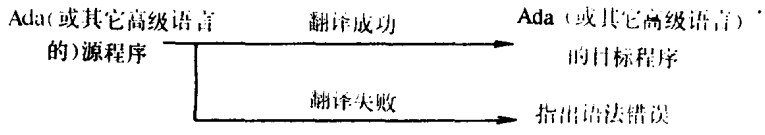


图 1.1

期间，可能会发现运行时的错误。当计算机系统发现这个错误后就会中断运行，然后显示提示信息以告之程序员。显示的信息是以 Ada 源程序的形式表示的，懂 Ada 的程序员就可以理解这些信息并对源程序进行适当的修改。

有时，某一个程序运行通过之后，并不能保证其运行结果的正确性。这是因为所设计的算法中可能会有逻辑错误，故精心选择测试数据来调试程序是相当重要的。用这种方法来检查和纠正错误称为调试(debugging)。

或许读者已看出，检查纠正编译时的错误比运行时的错误容易；而检查和纠正运行时的错误比发现和改正逻辑错误容易。Ada 语言的设计使得许多错误尽可能在其编译的过程中被检查出来。当然，逻辑错误仍将会引起运行错误，这就需要通过调试手段来测试纠正。

§1-4 逐步求精程序设计方法

下面主要介绍逐步求精的程序设计方法。在编制程序之前，必须事先进行几个重要的步骤。

一、明确题意

对于一个复杂的问题，明确题意是最关键的，并且也是整个过程中最难的步骤。这个步骤中所出现的任何错误将会导致付出昂贵的代价。因为这会导致解决一个不正确的问题而误入歧途。牢记这个步骤是非常重要的，特别是在处理商业、工业以及科学研究中的各类问题。

二、设计算法

如何设计算法是我们所感兴趣的问题。算法可以定义为：一个明确的指令序列，执行这个序列，可在有限的时间里得出所需的结果。

下面，我们通过一个简单的实例来看一下如何设计算法。

例 1-1 求 1 至 99 之间所有奇数的平方和。

首先考虑如何用笔和纸解这道题：依次取一个奇数并计算其平方；然后将其加到某个累加器中。于是产生计算机解题的雏形。这个解题步骤可以写成：

置累加器之值为 0

取 1 至 99 之间的奇数，将其平方值加到累计器中

写出累计器的终值

这个算法是相当粗的，没有涉及怎样生成奇数序列。实际应用时还需要对其精细化，用以指明如何依次产生奇数。我们可以从第一个奇数开始，在当前奇数上加 2 其值总是奇数。由此该算法被扩展成：

置累计器之值为 0

置奇数为 1

循环，当奇数小于或等于 99 时

　　将当前奇数之平方加至累计器

　　当前奇数加 2

结束循环

写出累计器之终值

对于 1 至 99 中的每个奇数，都要执行介于“循环”与“结束循环”之间的步骤。

这种解题方法称为逐步求精算法设计方法。首先产生一个粗略的算法，它不涉及一些内容细节。然后用逐步求精的方法将其进行精细，直到可以用某种程序设计语言编程为止。算法以及它的精细是用文字表述的，它与某一特定的程序设计语言毫无关系。

由于本题是相当简单的，现在就可以编制计算机程序了。当算法设计完成后，用 Ada 或者其它语言编制程序就比较容易了。所以脑力劳动主要花在算法设计而不是用在编制程序上。

下面给出了上述算法相应的 Ada 程序，读者可以看到一个完整 Ada 程序。这个 Ada 程序与用文字描述的算法是很相似的，甚至对 Ada 一无所知的读者也会理解。

程序1-1

```
WITH student_io; USE student_io;
PROCEDURE odd_square IS
    sum_odd: integer := 0;
    odd_number: integer := 1;
BEGIN
    -- 求1到99范围内奇数的平方和
WHILE odd_number <= 99 LOOP
    sum_odd := sum_odd + odd_number * odd_number;
    odd_number := odd_number + 2;
    --sum_odd中存入了奇数平方的和
END LOOP;
    put("The sum of the odd number is ");
    put(sum_odd); new_line;
END odd_square;
```

至此，还有一个重要问题：如何将已编制好的 Ada 程序输入计算机呢？答案是依赖于你所使用的计算机系统。

要想把程序输入计算机需要借助于操作系统。操作系统中有一个编辑程序，利用它可以把新的信息（如程序）输入计算机，也可以修改或编辑已输入计算机的信息。在编辑程序的控制下，可在终端键入 Ada 源程序。当 Ada 源程序被输入计算机后，编辑程序就可以将它拷贝至后备存储器的一个文件中。

如何具体使用编辑程序与所使用的计算机系统有关，这已超出本书的范围，在此不再赘述。

§1-5 大型程序的组织

计算机已广泛应用于社会经济各个领域，如控制原子能电站、管理银行账务等。由于计算机在这些应用中所承担工作的复杂性，使得为这些系统所编制的程序也相当复杂、庞大。因此，程序是否可靠、运行是否正确是至关重要的。

六十年代末期国际上出现了“软件危机”，其主要表现是：软件质量差，可靠性难以保证，软件成本增长难于控制，极少有在预定的成本预算内完成的；软件开发进度难于控制，周期拖得很长；软件的维护很困难，以使维护人员和费用不断增加。有的软件耗费了大量的人力财力，结果半途而废。

考虑到研制一个大型而复杂的软件系统同研制一台机器或建造一座楼房这样的工程有许多相

似之处，因此，人们就试图用“工程化”的思想作指导来解决软件研制中面临的困难和混乱，从根本上解决软件危机。这样就产生一个新的学科——软件工程。软件工程就是研究如何生产高质量、低成本、可靠性好的软件产品的一门学科。

在学习程序设计时，总是通过编写小型程序来提高编程技巧。一旦程序运行正确后，就将其搁置一边，需要时就来调用。在研制大型软件时，庞大的程序往往是由程序员小组共同协作完成，而且所编制的程序可能要使用许多年。在使用期间，根据系统的要求修改程序也是经常的事，这些修改通常也由原来编制程序的不同程序员分别进行。

解决设计大型系统的最好办法是将其分解为若干相对独立的子问题，这些子问题可以独立解决。然后进一步将每个子问题再分解成若干较简单的子问题；依照这种办法继续下去，直到每个子问题足够简单。这也就是上节所说的逐步求精方法，只是从不同角度来看而已。本书的前半部分将介绍如何组织小型程序，只包括 Ada 语言的基本特征，其它部分留待后面介绍。

在介绍 Ada 语言的基本特征之后，引入了 Ada 程序包的概念。程序包概念是 Ada 对程序设计语言研制所做的最大贡献，是支持大型软件系统的有力设施。大型系统允许对子问题设计程序包，程序已可与程序的其余部分分开来单独实现与测试。当所有的子问题都得以解决并运行正确之后，就可将它们连接起来形成最终程序。对于最终程序，也可以对程序的一部分进行修改和调试而不影响整个程序。

当某一问题分成若干子问题后，或许会发现某个子问题已有现成的程序包。这样，只要直接利用这些程序包即可。大部分 Ada 程序的实现都有这样一些有用的程序包供用户调用，这就使我们研制大型软件系统时不必从头开始，省时省力。

第二章　简单的 Ada 源程序

§2-1　一个完整的 Ada 程序

首先仔细考虑如下一个 Ada 程序的结构，该程序将计算并输出数 94 和 127 的和。

程序 2-1

```
WITH student_io; USE student_io;
PROCEDURE add IS
    first, second, sum: integer;
BEGIN
    --program to print the sum of two integers
    (打印两整数之和)
    first := 94;
    second := 127;
    sum := first + second;
    put("sum =");
    put(sum);
    new_line;
END add;
```

下面对程序 2-1 逐行解释。

第一行：

```
WITH student_io; USE student_io;
```

就是所谓的上下文子句(context clause)，这是一个非常有用的语句，我们暂时承认它，其意义将在以后说明。

程序中的大写单词，如 WITH、USE、IS、PROCEDURE、BEGIN 和 END，这些词称为保留字(reserved word)。保留字是 Ada 语言的组成部分。为了使程序结构清晰，保留字采用大写形式，但对 Ada 而言，任何一种字型都是一样的。其它的保留字将在以后介绍，附录一给出了 Ada 语言的所有保留字。

一个简单 Ada 程序包含一个过程。程序 2-1 的第二行：

```
PROCEDURE add IS
```

引入名为 add 的过程。过程名是由程序员确定的，但必需养成好的习惯，就是名称的选择有助于了解程序的内容。在程序中，名字由标识符表示。

第三行：

```
first, second ,sum: integer;
```

称为过程的说明部分。它说明了三个变量对象并分别取名为 first、second、sum，以后就可以调用这三个变量。

在本程序中，变量对象或变量是用来存放值的，程序 2-1 中的三个变量说明为整数类型，也就

是说，变量对象 first、second、sum 只能用来存放整数。之所以把它们称为变量，是因在程序的执行过程中，它们所存储的值可以变化多次。

一个变量对象限制只能存放一特殊类型的值。程序 2-1 中的三个变量说明为整数类型，也就是说，变量 first、second、sum 只用来存放整数，我们称之为整数变量。

一个变量的类型确定了其所存放值的范围以及可以实施的运算。Ada 有一些内部类型，例如 integer 和 float(整数类型和浮点类型)。整数变量可进行加、减、乘、除等算术运算。一个变量必须有一个名字、一个值和类型。一旦一个变量被说明后，其名字和类型就被固定了，但其值在程序的执行过程中可以改变。

下面看一下程序的执行情况。运行程序时，依次执行 BEGIN 与 END 之间的指令语句。

BEGIN 之后的第一条语句为解释语句：

```
--program to print the sum of two integers
```

注释以两个连续的连字号作为开始，并以该行的结尾作为结束。注释仅仅是为了帮助阅读程序，执行程序时，计算机将不作任何处理。

注释语句之后是两个赋值语句：

```
first : =94;
second : =127;
```

即把右边的数值储存在左边的变量中。复合符号" : ="应读成"取值"。执行这二个语句的结果是，整数 94 和 127 分别被放在变量 first 与 second 中，我们说变量 first 和 second 分别被赋值为 94 和 127。

接着再执行下一语句：

```
sum : =first+second;
```

首先把变量 first 和 second 的值相加，所得结果放在变量 sum 中。

语句"put("sum=");"为输出语句，输出"sum="字样。在程序中，用引号括起来的字符序列称为串(string)。

另一个输出语句：

```
put(sum);
```

输出 sum 的值。注意，这两个输出语句是不相同的。在第一个输出语句中，因为我们希望字符序列按原样输出，所以用双引号括起来；而在第二个输出语句中，则输出变量的值。执行这二个输出语句的结果是：

```
sum=221
```

连续的输出语句将所有信息显示在同一行上，如果需要另起一行则可用语句："new _ line;"。上述语句已是语句序列的最后一个语句，程序运行到此为止。

在程序的最后一行中，过程名 add 又在 END 之后出现，这在 Ada 程序中并非是必要的。但我们提倡这样做，因为这样可以帮助人们阅读理解程序。

由上述分析可知，一个简单的 Ada 程序由两部分构成。在保留词 IS 与 BEGIN 之间是说明部分，它对程序中所使用的标识符进行说明；BEGIN 与 END 之间是语句序列，每个语句以分号";"作为分隔符，执行程序时，按语句的先后次序执行。稍加分析可以发现，程序 2-1 只能把数 94 与 127 相加，如果要计算另外两个整数的和就要修改源程序，因此不具有灵活性。我们将对程序 2-1 加以修改，增加两个输入语句，这样就可以把任何两个整数相加。

修改程序 2-1，它将读入两个整数并输出两数之和，程序如下。

程序 2-2

```
WITH student_io; USE student_io;
PROCEDURE add_any IS
    first,second, sum: integer;
BEGIN
    --program to read two integers and to print their sum
    (读入两个整数并输出其和)
    put("two integer numbers please");
    get (first); get (second);
sum := first+second;
    put("sum=");
    put(sum);
    new_line;
END add_any;
```

程序 2-2 使用户与程序的执行联系在一起，用户必须输入信息计算机才能计算并输出结果，以后我们总是在交互式终端上运行程序。

考察程序 2-2，三个整数变量 first、second、sum 再次被说明。第一个要执行的语句是：

```
put("two integer numbers please");
```

执行结果是打印出信息：

```
two integer number please
```

其作用是提示用户输入两个整数，因为紧接着的是两个输入语句，它要求输入两个整数作为数据。当键入两个整数后，这两个整数就被程序读入并分别存放在变量 first 和 second 中。接下来的语句与程序 2-1 相同，就不再描述了。

§2-2 注释与源程序的书写格式

在编程序时必须牢记：程序不仅仅被计算机读入和理解，而更重要的是被人阅读和理解。所以，我们总是选择富有意义的标识符，并且精心编排程序以使其结构清晰。例如程序 2-1、2-2，介于 BEGIN 与 END 之间的语句序列是采用分层缩进对齐的写法，这样有助于了解程序的结构。本书的示例可以作为编写 Ada 程序的示范。

程序的格式是不影响其意义的。标识符、保留字及数之间至少要用一个空格或空行将它们彼此分开。在允许使用一个空格的地方，总可以留几个空格或另起一行。

注释是用来帮助理解程序的，它以两个连续的连字号为开始并以该行末尾作为结束。尽管注释并不影响程序的内容，但对于阅读程序的人来说则带来极大的方便。

一般地，在过程开始处加一注释以描述该过程的内容或目的，程序 2-1 中的注释就是这样的。其它的注释可以放在程序的难点或关键处（本书中的注释都采用英文，适当用中文解释）。

§2-3 语法图

在详细介绍 Ada 语句之前，我们先用语法图这种简单的概念来描述 Ada 语言。语法图对于学习 Ada 语言是极有帮助的。

和自然语言一样，Ada 语言也有其自身的语法规则。语法规则规定了什么是合法的、什么是非

法的程序，这些规则可以用文字来描述。

例如，对于已提及的标识符尚未给出严格的定义，可以用如下的一段文字来描述标识符的语法规则：

标识符由字母开头，其后可跟或不跟其它字母或数字，一个下划线也可以出现在任何两个字母或数字之间。下面是一些合法标识符的例子：

x，first _ Case，Julia，Chapter _ 2

用文字来描述语法规则显得很长而且可能语义不清，采用语法图描述语法规则是较好的方法。标识符可用如下的语法图描述(图 2.1)。

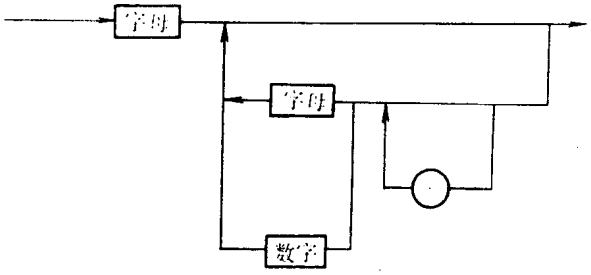


图 2.1

图 2.1 中所有可能的路径都构成合法的 Ada 标识符。语法图是一个等价的语法规则定义，它具有简洁明了的特点。

在语法图中，圈或弧矩形中的内容(如下划线)表示出现在 Ada 程序中的实际字符；而矩形框中描述了出现在 Ada 程序中的条目。矩形中的项必须用其它语法图加以定义。例如数字可定义为图 2.2 的形式。

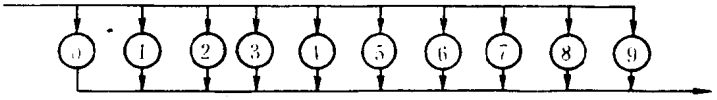


图 2.2

这表明数字是数 0～9 中的一个。

如果两个标识符仅仅是字母大小写的区别，则被视为同一标识符。因此 Same、same、SAME、sAME 是同一标识符 same。

最后，看一下简单 Ada 程序的结构语法图(如图 2.3 所示)。

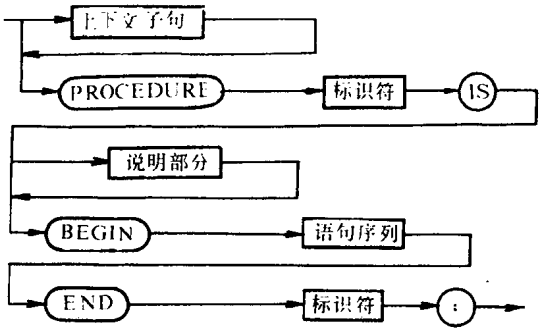


图 2.3

款项“上下文子句”、“标识符”、说明部分”及“语句序列”都在矩形框中，它们需要用语法图进一步定义；而 procedure、is、begin、end 都在圆弧框中，它们将直接出现在程序中。注意，保留字不能作为通常的标识符使用。

练习

1. 指出下列哪些是合法的 Ada 标识符。

   total Total sum _ 1 sum 1 Summ _ 1 sum _ 1. 5 SECOND begin average “fonal”

   Total count 2nd BEGIN END

2. 习题 1 中哪些是已学过的保留字？

3. 在程序 2-2 中，用户在提示：

   two integer numbers please

   下输入：

   15 28

   请写出运行结果。

4. 为什么说程序的格式以及在程序中使用注释是很重要的？

5. 编一程序，要求读入两个整数，输出第一个数减第二个数的结果。