ACM 国际大学生程序设计竞赛
试题与解析

（一）

吴文虎　主编
倪兆中　王　帆　编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

ACM 国际大学生程序设计竞赛是目前国际上历史最长、水平最高、影响最广泛的大学生计算机竞赛。本书精选了近年 ACM 总决赛中的 20 余道难题,对之加以分析,理出思路与解法,并给出作者编写的参考程序。这些试题具有实际背景,所考查的知识范围比较全面,题意新颖,给解题者留有广阔的思维空间和创新的余地,是高等学校大学生和研究生的很好的课外读物,从中可以学习如何用计算机编写程序解决难题的思路与算法。

图书在版编目(CIP)数据

ACM 国际大学生程序设计竞赛试题与解析 (1)/ 吴文虎主编;倪兆中,王帆编著 . - 北京:清华大学出版社,1998.12
ISBN 7-302-03212-2

Ⅰ.A… Ⅱ.①吴… ②倪… ③王… Ⅲ. 程序设计-解题 Ⅳ.TP311

中国版本图书馆 CIP 数据核字(98)第 33078 号

出版者: 清华大学出版社(北京清华大学校内,邮编 100084)
http:// www .tup .tsinghua .edu .cn
印刷者: 昌平环球印刷厂
发行者: 新华书店总店北京发行所
开 本: 787×1092 1/ 16 印张: 10 .5 字数: 245 千字
版 次: 1998 年 12 月第 1 版 1998 年 12 月第 1 次印刷
书 号: ISBN 7-302-03212-2/ TP·1716
印 数: 0001～5000
定 价: 12.00 元

序

ACM国际大学生程序设计竞赛是目前世界上规模最大的计算机学科赛事。该项赛事发起于1977年,要比国际信息学奥林匹克(IOI)的首届比赛早12年。ACM(Association of Computing Machinery,美国计算机协会)是国际计算机界的权威学术机构。由ACM发起并组织的这项赛事的宗旨是:为高等学校的大学生们提供一个展示自己在计算机编程解题方面才能的机会。通过竞赛使各国学生相互交流学习经验,为信息学科的发展不断注入新的活力,增进友谊,促进合作。因此,这项一年一届的赛事吸引了几乎所有的知名高等学府。譬如,1998年的第22届大赛就有1250支代表队参赛,其规模与影响之大可见一斑。

该项赛事的题目涉及计算机在各种应用领域中提炼出来的一些理论问题、方法问题,特别是有些问题没有固定算法,需要选手在比赛现场运用所学基础知识,经过认真分析研究归纳整理,才能得出解决问题的办法,此后还要经过编程调试、提交通过等严格的步骤加以检验,因此有相当难度,要想获奖是十分不易的。参加这样的竞赛要具备高等数学、图论集合论、组合数学、高级编程语言、人工智能和算法等方面的基本知识,要有熟练的编程解题功夫,特别是要有将实际问题抽象为数学模型的能力;在心态方面要有不怕困难、不畏强手、敢于拚搏的精神;特别强调要有集体协同作战的团队精神。比赛是三人一组解若干道难题,比谁做得快做得好,可以说是争分夺秒,讲究配合默契,强调协作攻关。

这种比赛形式有其独到的特点,在培养学生的创新能力方面能够起到促进作用。我们在准备参加ACM计算机竞赛的过程中收集了国内外有关的竞赛试题,是历年ACM世界大学生竞赛的题目,对这些题目进行了较详细的分析并做出解答,由参加过ACM大赛的两位选手将其汇集成册,正式出版,目的是将我们的经验体会还有一些技巧介绍给对竞赛有兴趣的同学。当然,也许你从来也没想去参加世界大赛,但我们相信,你看了这套试题解析也会有所收获,起码可以学到使用计算机编程分析问题到解决问题的全过程。因此,本书又可以作为学习高级编程语言和算法的参考书。

吴文虎

1998年9月

前　言

ACM 国际大学生程序设计竞赛(以下简称 ACM/ ICPC)是目前国际上历史最悠久、水平最高、影响最为广泛的大学生计算机竞赛,每年都吸引了成千上万来自世界各地的大学生参加。

ACM/ ICPC 的试题具有以下特点:

1. 试题具有丰富的实际背景,趣味性和实用性较强。许多试题都是从计算机软硬件的实际开发过程中演变而来的。

2. 试题所考查的知识范围较为全面。不仅要求解题者具备高级语言程序设计、数据结构和算法设计等计算机基础知识,还需要熟练掌握人工智能、计算几何等计算机科学的深层次内容,特别是要求具有较强的数学功底和广博的知识面。

3. 试题的层次性较好。不同水平的解题者都能找到适合于自己的试题。

4. 试题灵活、新颖。绝大部分试题没有定解,给解题者留出了广阔的思维空间,对创造性的培养十分有益。

参加 ACM/ ICPC 本身就是一个增长知识、培养能力的绝好机会,这也正是其魅力所在。而在 ACM 竞赛过程中所体现出的团队精神和合作精神,也正是当代大学生的必修课。

国内大学参加 ACM/ ICPC 的历史相对较短,但在各大学师生的努力下,近年来还是取得了长足的进步,一些起步较早的大学也先后达到了世界级的水平。然而必须看到的是,就整体而言,国内大学的水平与世界一流相比,还存在着很大的差距。同时,国内有关这方面的专著和资料也极为缺乏,这同大学生计算机竞赛正在国内蓬勃兴起的现状是不相适应的。本书正是基于这样一种考虑来组织编写的。

本书首先用两章的篇幅,分别向读者介绍了 Borland Pascal 高级程序设计技术和 ACM/ ICPC 的基本概况,作为读者阅读本书的基础。在之后的六章中,精选了近年来 ACM/ ICPC 总决赛的 20 余道试题,将其翻译为中文,并编写了相应的解题思路和参考程序,供读者参考。

本书的编写是在中国计算机学会普及工作委员会主任、清华大学计算机系吴文虎教授的指导下完成的。他亲自参与了本书的选题与规划,并仔细审阅和修改了本书的初稿,确保了本书编写任务高质量地完成。ACM 在国内的分支机构——Tsinghua University ACM Student Chapter——对本书的编写给予了极大的支持和帮助。

希望本书的出版能起到抛砖引玉的作用,促进全国范围内大学生计算机竞赛水平的提高。随着 ACM 国际大学生程序设计竞赛的发展,我们还将继续编写《ACM 国际大学生程序设计竞赛试题与解析(二)》。必须指出的是,虽然编著者在本书的编写过程中努力工作,力图奉献给读者高质量的精品,但由于水平有限,书中的不足和误值之处在所难免,恳请广大读者提出宝贵意见。

编著者

1998 年 5 月于清华园

目　　录

第 1 章　专题讲座——Borland Pascal 高级程序设计技术

1.1　引　　言

首先来明确一下程序设计竞赛对选手的要求:在有限的时间内,用自己所掌握的知识与具备的能力,凭借可能的工具,通过编写程序,去有效地解决问题。

解决程序设计竞赛试题的过程应当是顺着这样一条路线行进的:

理解题意并建立相应的数学模型
↓
根据数学模型的特点选用或构造适当的数据结构及算法
↓
将算法实现为程序并调试正确

在这样一个过程中,算法是首要的,占有战略性的地位。算法的好坏从根本上区分了选手的优劣。

但是,在实际竞赛中,往往会看到这样的情况:一个选手在比赛时找出或构造出了正确的算法,但由于时间不够,造成程序没有编完或没有编正确。测试之后,同不会做的选手一样,没有分数;两个选手,想到的算法大致一样,而编程实现所用的时间却有快有慢,测试时,程序的效率也相去甚远。以上都是非常普遍的现象。

为什么会出现这样的情况呢?一般地,这是由于选手没有熟练、扎实、深入的基本功所致。具体体现在平时对编程环境不够了解,运用不够熟练,忽视了将算法实现到程序这一环节的必要性和重要性。

我们说算法是首要的。但是,一个算法再好,如果不能在限定的时间内转化为正确的程序,那么对于竞赛来说,其效果就是零。这就是算法实现的必要性。与此同时,算法实现的快慢,决定了选手的时间利用率,决定了选手能否有时间去解决其它试题,因而也就影响了选手的成绩;算法实现的好坏,决定了算法的表现,对选手的成绩也有不小的影响,这主要体现在程序的运行时间上。在有多人想到了正确算法的情况下,算法实现的质量就尤其显得重要了。这就是算法实现的重要性。

算法与算法实现的关系,是战略与战术的关系。算法起决定性作用,从根本上决定了算法实现质量的数量级,但算法实现反过来也影响着算法,并且在一定条件下,这种影响有可能超过算法本身的作用。

在算法实现上,“时—空”这一对对立统一的矛盾起着很大作用。算法可以有多种不同的实现,这些实现有的侧重于时间,有的侧重于空间。一般地,程序设计竞赛对程序的速度要求较高,而空间占用只要不溢出,就对成绩没什么影响,因此选手应多注意程序的

时间效率。当然这里有一个前提,那就是空间占用不能超过允许的范围。

Borland Pascal 是标准 Pascal 的超集,它向程序员提供了数以千计的数据类型、语法结构、标准函数与子程序等语言成分,其中绝大多数是标准 Pascal 所没有的。这些语言扩展向程序员提供了强大有力的开发支持,运用得当的话,可以起到事半功倍的效果。正是基于此,无论国内或国际的程序设计竞赛,都提供 Borland Pascal 的最新版本 7.0 ,并建议选手采用。然而目前很少有资料详细、全面、系统地介绍了这些高级技巧及其应用。我们根据自己多年来运用 Borland Pascal 语言工具解决大量竞赛题的体会和经验,对Borland Pascal 7 .0 版本作了较深入的剖析,并从功能、效率、可读性等方面对其语言概念作了大致的总结、归纳与分析,从中选择出了“ 数据类型 ”、“ 常用函数 ”、“ 指针及堆 ”、“ 文件缓冲 ”、“ 快速操作 ”、“ 位操作 ”、“ 编译开关 ”、“ 保护模式 ”等一些对程序设计竞赛较为有用的语言扩展,供读者参考。

1.2 数 据 类 型

Borland Pascal 实现了标准 Pascal 所有的预定义数据类型,并扩充了不少十分有用的数据类型。在 Borland Pascal 中,这些数据类型得到了高效的实现。对数据类型的深入了解将有助于程序设计水平的提高。

1. 整数

标准 Pascal 里定义的整数类型是 Integer, Borland Pascal 还提供了 Shortint、Byte、Word、Longint 这四种整数类型。表 1.1 是它们的详细说明。

表 1.1

类型	大小	范围	存储格式	预定义最大值常量
Shortint	1 字节	- 128 . .127	带符号 8 位	无
Integer	2 字节	- 32768 . .32767	带符号 16 位	MaxInt
Longint	4 字节	- 2147483648 . .2147483647	带符号 32 位	MaxLongInt
Byte	1 字节	0 . .255	无符号 8 位	无
Word	2 字节	0 . .65535	无符号 16 位	无

2. 实数

标准 Pascal 里定义的实数类型是 Real, Borland Pascal 另外定义了 Single, Double, Extended, Comp 四种实数类型。表 1.2 是它们的详细说明。

Borland Pascal 7 .0 的集成环境有两个。turbo .exe 只能在实模式下编译,通常又称为 Turbo Pascal 7 .0; bp .exe 可以在三个目标模式下编译,是真正完备的 Borland Pascal 7 .0。两者的用户界面、语言环境都是一致的。

表 1.2

类型	范围	精度	大小	是否要求协处理器
Real	2.9e-39..1.7e38	1112 位	6 字节	否
Single	1.5e-45..3.4e38	78 位	4 字节	是
Double	5.0e-324..1.7e308	1516 位	8 字节	是
Extended	3.4e-4932..1.1e4932	1920 位	10 字节	是
Comp	-9.2e18..9.2e18	1920 位	8 字节	是

3. 字符

Borland Pascal 中字符的大小是一个字节,字节的值是该字符的 ASCII 码值。在程序中除了可以用' * '的形式表示可显示字符外,还可以用 # 号后跟上该字符的 ASCII 码值来表示任意一个字符。例如, # 0 表示第一个 ASCII 字符, # 9 表示制表符等。

4. 布尔类型

布尔类型在 Borland Pascal 中也用一个字节存储,但实际上只有字节的最低位被用来表示信息。最低位为 0 时表示 false,最低位为 1 时表示 true。

5. 集合

Borland Pascal 的集合类型最多允许有 256 个元素。在实现时,采用了一个字节表示 8 个元素的存储方式。故集合类型存储较布尔数组存储在空间上要节省一些。

6. 字符串

Borland Pascal 定义了字符串类型 string。它的存储结构如图 1.1 所示。

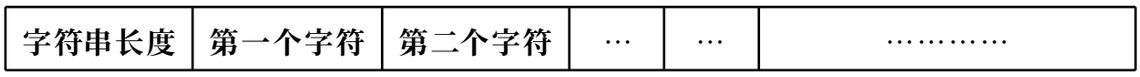

图 1.1

string 类型实际上是用 array[0..255] of char 来定义的。数组的第 0 位表示字符串的长度。因此,字符串的长度最大为 255。

1.3 常用函数

1. ReadKey 和 KeyPressed

Crt 单元提供了函数 ReadKey,其作用是从键盘读入一个字符。ReadKey 没有参数,返回值是一个键盘输入的字符。在读入字符的同时,ReadKey 不像 Read/ ReadLn 那样在屏幕上显示读入的字符。

Crt 单元还提供了函数 KeyPressed,其作用是确定键盘上有无按键动作。KeyPressed

也没有参数。返回值是一个布尔量,返回值为真时表示有按键,反之则无。KeyPressed 不会像 Read/ ReadLn/ ReadKey 那样等待键盘的输入,所以在程序中可以随时用 KeyPressed 来判断有无键盘输入。

结合使用 ReadKey 和 KeyPressed,可以做到在键盘输入的同时"并行"其它操作。

【例程】

下面的程序不断地向屏幕输出"Press any key . . .'信息,直到用户按下任一键后输出用户所按的键。

```
program Demo_of_ReadKey_and_KeyPressed;

  uses
    Crt;

  begin
    repeat
      writeln('Press any key . . .');
    until keypressed;

    writeln('You have pressed the key of', readkey);
  end.
```

2. SizeOf

SizeOf 是一个函数,由其可以得到类型的一个实例的大小。SizeOf 有一个参数,可以是任何类型、变量。SizeOf 返回参数所占内存的字节数。

【例程】

下面的程序输出了 People 类型的大小以及其实例 P 的大小。

```
program Demo_of_SizeOf;

  type
    People =
      record
        Name:string[20];
        Age:byte;
      end;

  var
    P:People;

  begin
    writeln('Size of People = ',sizeof(People));
```

```
  writeln('Size of P = ',sizeof(P));
end.
```

1.4 程序结构

在这一节里将介绍 Borland Pascal 相对于标准 Pascal 在程序结构控制上的几个主要扩展,通过合理使用这些扩展,能够使程序结构简洁、高效、易读。

1. Break 和 Continue

系统提供了 Break 过程,其作用是跳出当前循环。Break 没有参数,调用它以后将立即跳出当前的 for,while,repeat 三种循环结构,转而执行紧接着此循环体的下一个语句。如果对 Break 语句的调用不在一个循环体内,编译器将报告错误。

系统还提供了 Continue 过程,其作用是进入下一次循环。Continue 也没有参数。调用它以后将直接跳转至当前的 for,while,repeat 三种循环结构的入口,进行下一次循环。如果对 Continue 语句的调用不在一个循环体内,编译器将报告错误。

读者可以看出,Break 和 Continue 这两个标准过程,都是同循环结构的控制有关的。Break 是跳至循环结束后的下一个语句;Continue 是跳至循环入口的第一个语句。用 goto 语句也可以实现 Break 和 Continue 的功能。但读者如果试一下,就会发现用 goto 语句实现 Break/Continue 功能编程繁琐,极易引起混乱,至于可读性就更无法同 Break 和 Continue相比了。

【例程】

下面的程序从键盘输入一个结点范围在 1 到 100 之间的有向图的弧。用户每次输入弧的始、末结点。输入一个 0 表示输入结束,其它情况将被忽略。之后程序将输出图中所有的弧。

```
program Demo_of_Break_and_Continue;

  var
    G: array[1..100,1..100] of boolean;
    A, B:byte;

  begin
    fillchar(G,sizeof(G),0);

    repeat
      write('Please input a number: ');
      read(A);

      case A of
        0:
```

```
            begin
              readln;
              break;
            end;

          1..100:
            begin
              readln(B);
              if not(B in [1..100]) then
                continue;

              G[A,B]:= true;
            end;

          else
            begin
              readln;
              continue;
            end;
        end;
      until false;

      for A:= 1 to 100 do
        for B:= 1 to 100 do
          if G[A,B] then
            writeln(A,'-',B);
    end.
```

2. Exit 和 Halt

Exit 是一个过程,其作用是退出当前模块。Exit 没有参数,调用它以后将退出当前子程序的运行。如果当前正在主程序里运行,则将终止整个程序的运行。

Halt 也是一个过程,其作用是终止整个程序的运行。Halt 的格式声明如下:

```
procedure Halt([ExitCode:Word]);
```

这里 ExitCode 表示程序退出至操作系统的返回代码,是可选的,其缺省值为 0(正常退出)。在程序中的任何地方调用 Halt 都将引起整个程序的终止并返回操作系统。

Exit 和 Halt 都具有中断运行的功能,但有很大差异。在子程序里使用时,Exit 会引起返回上一级程序,而 Halt 则会引起整个程序的终止。在主程序里使用时,虽然两者都会引起整个程序的终止,但是用 Exit 终止程序运行只能返回正常退出代码,而使用 Halt 则可以返回其它情况下的退出代码。

【例程】

下面的程序演示了 Exit 和 Halt 的作用。

```
program Demo_of_Exit_and_Halt;

  procedure Proc;

    begin
      writeln('Now in Proc.');
      exit;
      writeln('You will not see this.');
    end;

  begin
    writeln('Now in Main.');
    Proc;
    writeln('Now in Main Again.');
    halt(100);
    writeln('You will not see this.');
  end.
```

1.5 指针及堆

1. Borland Pascal 程序的存储机制

首先来介绍一下 Borland Pascal 程序运行时的存储机制。一个用 Borland Pascal 编译运行的程序在运行时将可用内存划分为以下几个部分:

代码段——存放程序代码;

堆栈段——存放程序运行过程中子程序的参数、局部变量;

常量段——存放程序中出现的常量;

数据段——存放在程序中已声明的变量;

堆——存放程序运行过程中动态声明的变量(通过指针引用)。

堆栈段、常量段、数据段的大小都不得超过 64K,因此静态变量的大小受到很大限制。而动态数据操作灵活,并且通过堆可以用到所有的可用内存,因而有着广泛的应用。程序运行时,通过系统所提供的过程,可以在堆中动态地分配和释放内存块。

必须指出的是,动态数据的操作速度比静态数据大约要慢三分之一左右。

2. Pointer

Borland Pascal 的预定义类型中有一个叫做无类型指针的类型,名称是 Pointer。定义这样一种类型的原因是因为在标准 Pascal 中,指针是依附于它所指向的具体类型的,不同

类型的指针之间是不兼容的。无类型指针则可以指向任何变量,它可以同任何类型指针互相赋值。

3. GetMem 和 FreeMem

系统提供了过程 GetMem 和 FreeMem,它们的作用是分配和释放动态存储空间(堆空间)。GetMem 和 FreeMem 的格式声明如下:

```
procedure GetMem(var P:Pointer;Size:Word);
procedure FreeMem(var P:Pointer;Size:Word);
```

这里 P 是要进行操作的指针,Size 是要分配或释放的存储空间的字节数。GetMem 为指针 P 分配一块大小为 Size 的堆空间,而 FreeMem 正好是 GetMem 的逆操作:它将指针 P 所指向的大小为 Size 的堆空间释放以供其它指针使用。

GetMem 和 FreeMem 的例程见下一节。

4. Mark 和 Release

系统还提供了另一对关于动态存储空间分配的过程 Mark 和 Release,它们的作用是记录和恢复堆的使用情况。Mark 和 Release 的格式声明如下:

```
procedure Mark(var p:pointer);
procedure Release(var p:pointer);
```

指针 p 被用来保存堆使用情况。Mark 将堆顶指针保存到 p 中,从而记录堆空间的使用情况。Release 将堆顶指针设置为 p 的值,从而使堆空间的使用情况恢复到上一次的 Mark(p)。结合使用 Mark 和 Release,可以方便、彻底、安全地释放堆空间。

【例程】

下面的两个程序是等价的,后一个程序演示了 Mark 和 Release 的用法。

```
program Prog_for_Comparation;

  var
    p1,p2,p3:^string;

  begin
    new(p1);
    new(p2);
    new(p3);
    dispose(p2);
    dispose(p3);
  end.

program Demo_of_Mark_and_Release;
```

```
var
  p:pointer;
  p1,p2,p3:^string;

begin
  new(p1);
  mark(p);
  new(p2);
  new(p3);
  release(p);
end.
```

1.6 文件缓冲

1. 缓冲的原理

程序对磁盘的读写过程是:

程序\ 操作系统\ 磁盘

磁盘尤其是硬磁盘的数据传输速度是很快的,但磁头寻道的时间则相对要慢得多,因此,对磁盘的读写时间主要是花在磁头定位上,真正用来传输所需要的数据的时间不到百分之一。尤其是在读入文本文件时,往往是以几个字节大小的数字或字符为单位进行读写,一个几十 K 的文件,磁头往往要做上万次的寻道动作,所以时间浪费极大。

读者也许知道 DOS 里的 Smartdrv 这个应用程序。它的作用是提高磁盘读写的速度。其原理是在操作系统和磁盘之间开辟一个数据缓冲区。每次读磁盘时先查看数据是否在缓冲区中,如果在就直接将其复制至指定内存区域,仅当要读入的数据不在缓冲区里时才去读磁盘,并且一次要读入尽可能多的数据,以便提高以后的命中率。每次写磁盘时并不是将数据直接写入磁盘,而是先送往缓冲区累积起来,当缓冲区满后才一次将缓冲区信息全部写入磁盘。通过缓冲技术,Smartdrv 大大减少了磁盘的读写次数,从而节省了大量的空间。

有系统缓冲的磁盘读写过程是:

程序\ 操作系统\ 缓冲区\ 磁盘

2. SetTextBuf

在竞赛中,有一些试题需要对磁盘进行大量的读写,这时如果能有高速缓存,可以极大地提高程序的运行效率。然而,选手无权要求系统安装 Smartdrv,那么应怎么办呢?Borland Pascal 向程序员提供了设置文本文件自动内部高速缓存的过程——SetTextBuf。

有文本文件缓冲的磁盘读写过程是:

程序\ 缓冲区\ 操作系统\ 磁盘

SetTextBuf 的格式声明如下:

SetTextBuf(var F: Text; var Buf; Size: Word);

其中 F 是要被高速缓存的文件,Buf 是缓冲区变量,Size 是缓冲区的大小。

【例程】

下面的程序利用文本文件缓冲读入文本文件 Input .Txt 到屏幕。

```
program Demo_of_SetTextBuf;

  const
    BufSize = 16384;

  var
    P: pointer;
    F: text;
    S: string;

  begin
    assign(F, 'Input .Txt');
    getmem(P,BufSize);
    settextbuf(F,P^,BufSize);
    reset(F);

    while not eof(F) do
      begin
        readln(F,S);
        writeln(S);
      end;

    close(F);
    freemem(P,BufSize);
  end.
```

1.7 快速操作

为了提高程序的效率,Borland Pascal 提供了一些快速操作的过程。由于这些过程在实现中充分利用了汇编语言的特性,因此效率较普通的实现方法要高出不少。合理使用这些过程可以使程序的效率提高不少,编程也有一定的简化。

1. Fillchar

系统提供了过程 Fillchar,其作用是将一个给定的值写入给定长度的连续内存区域中。FillChar 的格式声明如下:

```
procedure FillChar(var X;Count: Word; value);
```

这里 X 是要写入的内存区域;Count 是要写入的字节数(如果 X 是变量,建议在这里使用 SizeOf(X));value 是写入的值,可以是 byte 类型或 Char 类型。FillChar 在实际应用中主要用来为数组或结构变量初始化清零或统一赋值等。

【例程】

下面的程序从键盘读入一个数值,然后将数组全部初始化为该值。

```
program Demo_of_FillChar;

  var
    X:array[1..1024] of byte;
    Value:byte;

  begin
    readln(Value);
    fillchar(X,sizeof(X), Value);
  end.
```

2. Inc 和 Dec

系统提供了过程 Inc 和 Dec。它们作用是对一个整型或有序类型变量作增量和减量操作。Inc 和 Dec 的格式声明如下:

```
procedure Inc(var X[;N:Longint]);
procedure Dec(var X[;N:Longint]);
```

Inc 是将变量 X 加上 N;Dec 是将变量 X 减去 N。N 的缺省值为 1。

从功能来看,如果 X 是整数,则 Inc(X,N)等价于 X:= X + N;Dec(X,N)等价于 X:= X - N。如果 X 是有序类型,则 Inc(X,N)等价于 X:= Succ(X);Dec(X,N)等价于 X:= Pred(X)。

之所以要有 Inc 和 Dec,是由于它们所生成的代码比较高效。这是因为系统在汇编表达式时,对于形如 A:= B+C 的表达式所生成的典型的代码是:

载入 B 的值到寄存器中;

将 C 的值加到寄存器中;

将寄存器的值保存到 A 中。

而对于 A:= A+B 类型的表达式,代码可以简化为:

载入 B 的值到寄存器中

将寄存器的值加到 A 中

Inc 和 Dec 的参数 N 都可以是负数。

这里提醒读者一点:Inc 和 Dec 不会对操作进行溢出检查。

【例程】