电子计算机软件

编译原理例解析疑

赵雄芳　白克明　易忠兴
张克强编

湖南科学技术出版社

电子计算机软件

编译原理例解析疑

赵雄芳　白克明　易兴忠　张克强编

湖南科学技术出版社

电子计算机软件
编译原理例解析疑
赵雄芳　白克明　易兴忠　张克强编
责任编辑：周翰宗
*
湖南科学技术出版社出版
（长沙市展览馆路14号）
湖南省新华书店发行　湖南省新华印刷二厂印刷
*
1986年12月第1版第1次印刷
开本：787×1092毫米 1/16　印张：18　字数：445,000
印数：1——3,700
统一书号：15204·181　　定价：4.40元
征订期号：湖南新书目86——16(26)

前　言

《编译原理》是目前高等院校计算机专业均已讲授的一门重要的基础理论课程，兼有很强的理论性与实践性。但许多读者在学习时感到难度大，不知怎样理解和运用所学概念和知识，来求解练习与实践中所面对的课题。为此，本着如何正确理解概念和原理、掌握编译技巧以突破难点、强化能力培养的原则，笔者就学习该课程时，对那些具有普适性的典型问题，及研究生考试中的一些疑难问题，概括为八章编写成册，详尽地予以阐述、讨论，然后以多种方式求解，使能学以致用，学用结合，以帮助读者掌握重点，抓住要点，提高析疑与求解能力。

全书在每章之前，都简要地介绍了相应的编译理论与方法，起到指导复习和巩固概念的作用。然后由易到难，由简及繁，讨论与解析了一批有典型和普遍意义的习题作为示范，供广大读者借鉴和参考。附录部分还给出了一个简单的课程实习方案和程序实例，供组织课程实习时参考。故本书适合计算机专业广大师生读用，特别是对于日益增多的，有志于计算机科学发展的研究生报考者来说，无疑是一册全面而又系统的难得的复习资料，能收全面加深理解而又融汇贯通之效。这也是笔者编写此书的主要目的之一。

全书在编写过程中，得到了国防科大陈火旺教授、复旦大学钱家骅教授的帮助和支持，钱教授为本书提供了一些很好的解题方法。此外，国防科大、广西大学和长沙铁道学院的许多老师和同学对本书的编写提供了许多宝贵意见。对此笔者深表感谢！

由于编者水平有限，恳望广大读者对本书存在的谬误与不足之处，予以指正和批评，则不胜感激之至！

作　者

1986.5.于长沙

目　录

第一章　高级语言与表格处理

内容提要：

要构造一个好的编译程序，首先必须要掌握好三方面的知识，即源语言、目标语言和编译方法。

源语言即所谓高级语言。目前国内外高级语言种类繁多，比较流行的有：便于数值计算的FORTRAN,ALGOL 60；便于事务处理的COBOL；便于符号处理的SNOBOL及适于表处理的语言LISP等。源语言是一个记号系统，它是由语法和语义两方面定义的。

语法是指语言的构成规则。其中一部分称为词法规则，亦即单词构成规则。例如构成一个常数、名字、运算符等的规则；另一部分称为语法规则，根据它可以构成源语言中的各种语法单位(语法范畴)。例如表达式、语句、过程、程序等。语法单位的基本成分是单词。语法的描述比较容易，已经有了公认的形式系统，例如产生式、巴科斯范式等。

语义则指的是程序的含义。描述一个语言的语义目前还没有公认的形式系统，多数仍使用自然语言描述语义。

目标语言就是源语言被编译的对象语言。一般来说，微型机和小型机的目标语言就是机器语言，而大型机和巨型机的目标语言是某种汇编语言。

构造一个编译程序已经积累了很多相当成熟的技术，我们将陆续讨论这些技术。首先，讨论编译过程中常见的数据结构形式。

编译程序在其工作过程中使用最多的数据结构形式就是表。各种各样的表中记录着源程序的各种信息，以便需要时可以随时查询或者修改。这些表中尤以符号表最为重要，它的生存期最长，使用也最频繁。

符号表中登录着源程序中出现的各种名字的属性和特征。每当扫描器识别出一个名字后，编译程序就查阅符号表，如果是新的名字就填入符号表中；如果是符号表中已有的名字，就可得到有关它的各种信息(属性和特征)。在语法分析和语义分析过程中，一方面陆续登记有关信息，另一方面按表中已记录的信息进行语法和语义检查，并产生中间代码。在目标代码生成阶段，符号表是数据存贮分配的依据。由此可见，编译程序花在查填符号表上的时间，往往在整个编译时间中占有很大的比重。因此改善符号表的组织，加快查填符号表的速度对于提高编译速度是至关重要的。

表1.1

名字栏 (NAME)	信息栏 (INFORMATION)

概括地说，一张符号表的每一项(或称入口)包含两大栏(或称区段，字域)，即名字栏和信息栏，其表格形式如表1.1所示。名字栏记录名字，信息栏记录有关名字的各种不同属性。编译程序往往按名字的不同属性及类型将符号表分为：常数表，变量名表，过程名表，标号表等。这种区分方法在处理上比较方便。

对于符号表，一般有三种构造和处理法。即线性查找，二叉树和杂凑技术。

线性表的构造最为简单，其办法是按名字出现的顺序填写各个项，当碰到一个新名字时

就按顺序将它填入表中，若要了解某个名字的有关信息，就从该表的第一项开始顺序查找，这就是所谓线性查找。

线性查找因为每查找一项都是从表头开始，因此效率很低，但是线性表具有结构简单及节省存贮空间的优点，所以编译程序在仍采用线性表的同时，设法提高线性表的查表速率。

为了提高查表的速度，可以在造表的同时，把表格中各项按某种顺序排列，这种经过顺序化整理了的表格可用对折法查找，即从表的中项开始查找。当所查找的项目不是中项时，再在表的上半部分或下半部分的中项查找，每次查找时将查找范围缩小一半，故称对折查找。对折查找的速度比线性查找要快得多，但是相对地却使填表速度慢了下来，因为边填表还要边作顺序化。如果把符号表组织成一棵二叉树，这样就能达到既提高填表速度又提高查找速度的目的。

表格处理的杂凑技术是一种在查表、填表两方面都能高速进行的统一技术。这种办法是：假定有一个足够大的区域，这个区域可以填写一张含N项的符号表。则希望构造一个地址函数H，对任何名字SYM，H(SYM)取值于0～(N－1)之间。也即对任意SYM，无论是查表或填表，都能立即从H(SYM)获得它在表中的位置。地址函数H也称为杂凑函数。这种技术的不足之处就在于需要相当大的空间。

三种查填表的技术各有长处和不足。在构造编译程序的各种表格时，究竟应使用何种查填技术，要具体分析，灵活运用，扬长避短。提高了查填表的效率相对地也就提高了整个编译程序的效率。

题例与解析：

1.1 设计一个能按字典顺序输出二叉树各结点名字的算法。

解：设有如图1-1所示的二叉树(以字典排序法定义大小关系)。我们把符号表的每一项均

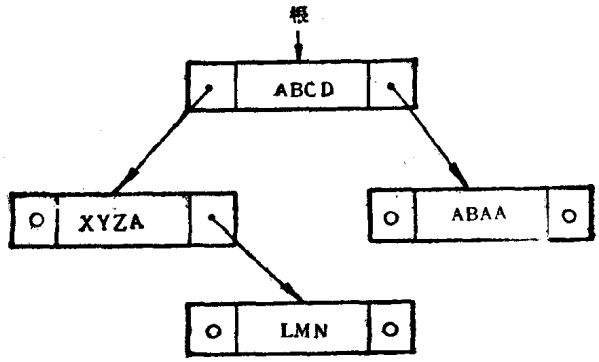

图1—1 按字典排序的二叉树

视为二叉树的一个结点，而把符号表主栏(名字栏)内部码值看成是代表该结点的值。每个结点(P)均附设LEFT(左技)和RIGHT(右技)两个指示器栏，并规定：其右技所有结点的值均小于，而左技所有结点的值均大于该结点(P)的值。因此按字典顺序输出二叉树上各结点名字的具体算法如下：

```
procedure PRINTBIN (root);
  begin
```

```
        p: = root;
    Repeat
      while p≠null  do
            begin
            将P压入STACK 栈;
            P: = RIGHT (P)
            end;
      Repeat
        把STACK栈顶项上托出去并放在P中;
        print(p.NAME);
        P: = LEFT(P)
      until  p≠null或者STACK栈为空;
    until  p = null
  end;
```

1.2 假定数组ARR按递增排序，数组的下标界限为lower和upper，现在值key可能是数组的分量，试编写一个二分查找的递归过程和递推过程。

解：二分查找法就是所谓的对折查找法或平分查找法。

(1) 递归过程：

```
procedure  ARRE (key);
 begin
   if key<arr[lower] OR arr[upper]<key
     then write (key, 'is not arr')
     else  begin
           middle: = (lower + upper) div 2;
           while arr[middle]≠key do
               begin
                 if key>arr[middle]
                 then lower: = middle + 1
                 else upper: = middle - 1;
                 ARRE(key)
               end;
             writeln(key, 'mthe arr[', middle,']')
           end
 end;
```

(2) 递推过程：

```
procedure ARRE (key);
 begin
   if key<arr[lower] OR arr[upper]<key
       then writeln(key,'is not arr')
       else  begin
               middle: = (lower + upper) div  2 ;
               while key≠arr[middle] do
                 begin
```

```
        if key>arr[middle]
              then lower:=middle+1
              else  upper:=middle-1;
        middle:=(lower+upper) div 2
        end;
        writeln (key,'in the arr[',middle,']')
      end
  end;
```

1.3 给出自适应线性表的查填算法（注意修改自适应键）。

解：自适应线性表的形式如表1.2：

表1.2

序号	NAME	INFORMATION	
			LINK
1			
2			
⋮			
n			
AVAIL→			

查填方法为：自适应线性表中的每一项都设一链域（link），用以将源程序中出现的符号名连结成一条链，此链是按“最新最近”的原则拉成的，即对于源程序中出现的每个符号名从链头往前查，若是一新名则将其填入AVAIL所指处，作为新的链头项；若是已出现过的符号名（即从表中查到了）。则修改自适链，使刚刚查到的项为链头项。这就是自适应线性表的构造与查填方法。

下面给出的查填算法中用到一链头指针变量Head，它的链域Head·Link指向自适应线性表的链头项。若自适应线性表为空，则Head·Link为null。sym是存放当前面临符号的变量。

查填算法：

```
procedure SEARCH (Head, Sym)
  begin
    P:=Head·Link;
    while P≠null do
      if P·NAME≠sym then      /*查不到，则继续找*/
        begin
          Q:=P; P:=P·Link
        end
      else     /*查到了，准备修改自适应链*/
        begin  /*若sym不是链头项，则修改自适应链*/
          if P≠Head·Link      then
            begin
              Q:=P·Link;  /*Q临时单元*/
              P·Link:=Head·Link;
              Head·Link:=Q
            end;
          RETURN (Q)          /*回送新链头项*/
        end;
      AVAIL·NAME:=sym;        /*将新名字填入表中*/
```

```
            AVAIL·Link: = Head·Link;
            Head·Link: = AVAIL;        /·Head指向新项·/
          end;
            AVAIL: = AVAIL + 1;
            RETURN(Head·Link)
      end
```

查填算法二 引进三个变量：链头（headlink）变量和临时变量P、Q。

```
Procedure search (headlink, sym);
 begin
   if AVAIL = 1 then
     begin
       AVAIL·name: = sym;
       AVAIL·link: = null;
       headlink: = AVAIL;
       AVAIL: = AVAIL + 1;
       RETURN (headlink)
     end
   else
 begin  Q: = head link;
   while headlink·name≠sym and headlink≠null do
     begin
       P: = headlink;
       headlink: = headlink·link
     end;
   if headlink = null then
     begin
       AVAIL·name: = sym;
       AVAIL·link: = headlink;
       headlink: = AVAIL;
       AVAIL: = AVAIL + 1
     end
   else begin
       P: = headlink·link;
       headlink·link: = Q
       end;
   end
   RETURN (P)
 end;
```

1.4 假定我们有一张10个单元的杂凑（链）表和一个足够大的存区用于登记符号名和连结指示器（此处用自然数作“名字”），杂凑函数为

$$H(i) = i(\text{mod } 10)$$

当最初的10个素数2，3，5，…，29进入符号表后，试给出杂凑链表和符号表的内容。当更多的素数进入符号表后，你能期望它们随机地分布在十个子表中吗？为什么？

解：表1.3是由杂凑函数H(i)＝i(mod 10) 得到的对于最初的10个素数2，3，5，…，29的杂凑链表和符号表。

表1.3

HASH表

0	null
1	5
2	1
3	9
4	null
5	3
6	null
7	7
8	null
9	10

符号表

序号	NAME	INFORMATION	
		·····	LINK
1	2		0
2	3		0
3	5		0
4	7		0
5	11		0
6	13		2
7	17		4
8	19		0
9	23		6
10	29		8
AVAIL			

对于更多的素数，它们也不能随机地进入这十个子表之中。因为任何素数都不以0，4，6或8结尾，因此对于任何素数a，其杂凑值H(a)都不可能等于0，4，6或8。另外，除素数2和5外，其他任何素数的杂凑值都不可能等于2或5。所以在0，4，6，8这四个子表区中没有任何素数进入，而2和5两个子表区中只有一个素数。因此，素数不能随机地分布在这十个子表区中。

1.5 假定我们有如下的ALGOL程序

```
begin {B1}
    integer a,b;
    ……
    begin  {B2}
      real  a,c;
        ⋮
    end;
```

```
      ……
L1: begin  {B3}
      integer b,d;
    ……
L2:   begin  {B4}
        real  c,e;
        ……
      end
    ……
    end
    ……
L3: end
```

采用杂凑链法的符号表结构，试指出在到达标号L1、L2和L3各点时信息区，符号表，分程序表和字符串数组的内容。

解：（一）到达标号L1时：这时已从B2分程序退出，刚刚进入B3分程序，但尚未扫描到B3分程序的说明部分。于是得到表1.4

分程序表

	P3	局部名个数	P4
b+1	C+1	2_{B_1}	S+1
BP→			
BP′→			
b+n		2_{B_2}	S+m

符号表

	P1	层号	P2
S+1	C+1	1	i+1
S+2	C+3	1	i+2
AVAIL→			
END→			
S+m-1		2	i+3
S+m		2	i+4

信息区

	TYPE	CAT	…
i+1	int	v	
i+2	int	v	
i+3	real	v	
i+4	real	v	
i+5			

字符串数组

↓csp											
	1	a	1	b	1	a	1	c			
	C+1		C+3		C+5						

表1.4

（二）到达L2时：这时刚刚进入B4分程序，但尚未扫描到B4分程序的说明部分。于是得到表1.5：

分 程 序 表

	P3	局部名个数	P4
b+1	C+1	2_{B1}	S+1
b+2	C+5	2_{B3}	S+3
BP→			
BP′→			
b+n		2_{B2}	S+m

符 号 表

	P1	层 号	P2
S+1	C+1	1	i+1
S+2	C+3	1	i+2
S+3	C+5	2	i+5
S+4	C+7	2	i+6
AVAIL→			
END→			
S+m−1		2	i+3
S+m		2	i+4

信 息 区

	TYPE	CAT	···
i+1	int	v	
i+2	int	v	
i+3	real	v	
i+4	real	v	
i+5	int	v	
i+6	int	v	
i+7			

字 符 串 数 组

								↓csp			
1	a	1	b	1	b	1	d				
C+1		C+3		C+5		C+7		C+9			

表1.5

（三）到达L3时：这时扫描到B1分程序的end之前，B2、B3和B4分程序的均已退出，但B1分程序尚未退出。于是得到表1.6：

分 程 序 表

	P3	局部名个数	P4
b+1	C+1	2_{B1}	S+1
BP→			
BP′→			
b+n-2		2_{B3}	S+m−4
b+n-1		2_{B4}	S+m−2
b+n		2_{B2}	S+m

符 号 表

	P1	层 数	P2
S+1	C+1	1	i+1
S+2	C+3	1	i+2
AYAIL→			
END→			
S+m−5		2	i+5
S+m−4		2	i+6
S+m−3		3	i+7
S+m−2		3	i+8
S+m−1		2	i+3
S+m		2	i+4

信 息 区

	TYPE	CAT	···
i+1	int	v	
i+2	int	v	
i+3	real	v	
i+4	real	v	
i+5	int	v	
i+6	int	v	
i+7	real	v	
i+8	real	v	
i+9			

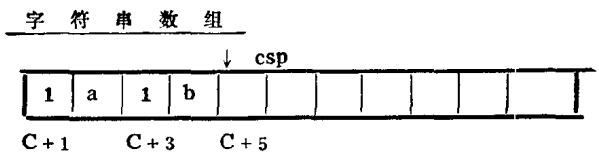

表1.6

1.6 何谓“标识符”，何谓“名字”，两者的区别是什么？

解：在程序设计语言中，标识符是一个最基本的概念，定义为：凡是以字母开头的字母数字序列（有限个字符）都是标识符。当给予某标识符以确切的涵义时，这个标识符就叫做一个名字。程序语言中各种名字都是用标识符表示的。只是后者是一个没有意义的字符序列，而名字却有着确切的意义和属性（即类型和作用域）。如：A，B1，…作为标识符时没有什么意思，但作为名字时，却可以代表变量名、数组名、函数名或过程名等。

1.7 关于变量、数组说明，FORTRAN和ALGOL有什么不同？下面的FORTRAN说明有什么错误？

```
REAL   MAXELE
DIMENSION   MAXELE (10, 10)
COMMON   MAXELE (10, 10)
```

下面的ALGOL说明正确吗？

```
integer  I;
array I[0:9, 0:9];
```

解：ALGOL语言规定：对于程序中引用的所有变量和数组元素，都必须在说明部分给出相应变量和数组的说明，并且一个变量或者数组至多只能被说明一次。变量或数组的作用域（有效范围）局限于说明它的那个最小分程序。例如：

```
begin  integer  X, Y;
   ⋮
Y:=X+Y
   ⋮
  begin  real  X, Z;
B1   ⋮
 B2   Z:=X+Y
  end;
  X:=Y
end
```

在分程序B1中，说明了X，Y两个整型变量，在分程序B2中，说明了X，Z两个实型变量。因此，整型变量X的作用域是除了分程序B2之外的所有区域，整型变量Y的作用域是整个分程序B1（包括B2），实型变量X，Z的作用域是分程序B2。

FORTRAN语言对于变量和数组的说明较之ALGOL语言要灵活一些。变量和数组的类型可以显式说明，也可以隐式说明。对于数组的说明，不仅可以用DIMENSION（维数）语句，还可以用COMMON（公用）语句或者类型说明语句说明之。当然，一个变量或数组的说明不得多于一次。变量和数组的作用域是该程序块（程序段）。

另外，ALGOL和FORTRAN语言对于变量和数组的说明在形式上、及在每一维的下限约定方面也不一样。如下面的FORTRAN说明

```
REAL MAXELE                          REAL MAXELE
DIMENSION MAXELE (10, 10)  正确 ⟹ 写法  DIMENSION MAXELE (10, 10)
COMMON MAXELE (10, 10)               COMMON MAXELE
```

前者对10×10的数组MAXELE重复说明了两次，这是不允许的。正确的写法是后者。也可写成

```
REAL MAXELE                    或    REAL MAXELE (10, 10)
COMMON MAXELE (10, 10)               COMMON MAXELE
```

以上语句说明MAXELE是一个有10×10个元素的实型二维数组，并且将全部元素分配在无名公用区。

下面的ALGOL说明

```
integer I;                   正确 ⟹ 写法  integer I1;
array I[0:9, 0:9];                        array I[0:9, 0:9];
```

前者是不正确的。ALGOL规定不能用同一标识符表示不同的量，除非它们有不相交的作用域。而该说明使标识符I既为整型变量又为实型数组(有10×10个元素)，且作用域相同，因此是错误的。正确的写法还可以是：

```
integer I1;                  或    integer I1;
array I[0:9, 0:9];                 integer array I[0:9, 0:9];
```

1.8 对于下面的ALGOL说明语句所定义的数组A

```
array A[-4:5, -3:3]
```

假定数组是按行为序存放的，存贮器是按字节编址的，每六个字节为一机器字，令A的首地址为1000，问A[I, J]，A[I+1, J+1]的地址是什么？试用符号语言写出执行赋值句A[I, J]:=0的指令序列。

解：不妨设A是下面说明语句所定义的一个ALGOL n维数组

```
array A[l1:u1, l2:u2, ···, ln:un]
```

先求出在一般情况下的数组元素地址计算公式，令$d_i = u_i - l_i + 1$，$i = 1, 2, \cdots, n$；a为数组的首地址，即A[l_1, l_2, ···, l_n]的地址。那么，在按行排列的前提下，数组元素A[i_1, i_2, ···, i_n]的地址D为

$$D = a + (i_1 - l_1)d_2d_3\cdots d_n + (i_2 - l_2)d_3d_4\cdots d_n + \cdots + (i_{n-1} - l_{n-1})d_n + (i_n - l_n)$$

对于题1.8所定义的数组A array A[-4:5, -3:3]，其元素A[I, J]的地址是

$$\begin{aligned} D1 &= a + ((I-(-4))*(3-(-3)+1) + (J-(-3)))*6 \\ &= a + ((I+4)*(3+3+1) + (J+3))*6 \\ &= 1000 + ((I+4)*42 + (J+3))*6 \\ &= 1186 + 42I + 6J \end{aligned}$$

元素 A[I+1, J+1] 的地址是

$$\begin{aligned} D2 &= a + (((I+1)+4)*7 + ((J+1)+3))*6 \\ &= 1000 + 234 + 42I + 6J \\ &= D1 + 48 \end{aligned}$$

赋值语句 A[I, J]:=0 的符号语言指令序列如下：

```
LD  R1, =F'3'        /*3⇒R1*/
-   R1, =F'-3'       /*(R1)-(-3)⇒R1*/
+   R1, =F'1'        /*(R1)+1⇒R1*/
LD  R2, I            /*(I)⇒R2*/
-   R2, =F'-4'       /*(R2)-(-4)⇒R2/
*   R1, R2           /*(R1)×(R2)⇒R1*/
LD  R2, J            /*(J)⇒R2*/
-   R2, =F'-3'       /*(R2)-(-3)⇒R2*/
+   R1, R2           /*(R1)+(R2)⇒R1*/
*   R1, =F'6'        /*(R1)×6⇒R1*/
+   R1, =F'1000'     /*(R1)+1000⇒R1*/
LD  R2, =F'0'        /*  0⇒R2*/
ST  R2, *(R1)        /*(R2)⇒((R1)): 间接传送*/
```

其中，=F'C'形式表示C为一常数，指令中的R_1表示寄存器(可作累加器和变址器)。

1.9 令+，*和↑代表加，乘和乘幂，按如下的非标准优先级和结合性质的约定，计算 1+1*2↑1*1↑2 的值：

(1)优先顺序(自高至低)为+、*和↑，同级优先采用左结合。

(2)优先顺序为↑、+和*，同级优先采用右结合。

解：$1+1*2\uparrow 1*1\uparrow 2=(((1+1)*2)\uparrow(1*1))\uparrow 2=(4\uparrow 1)\uparrow 2=16$ (1)

$1+1*2\uparrow 1*1\uparrow 2=(1+1)*((2\uparrow 1)*(1\uparrow 2))=2*2=4$ (2)

1.10 列出下面循环语句的循环体的执行情况及执行次数：

```
t:=0;
for i:=0 step t until 10 do
    t:=t+sign(5-i) ;
```

解：题中循环语句在ALGOL语言和PL/I语言中的含义是各不相同的，故执行情况和执行次数也不相同。现分别讨论如下：

在ALGOL意义下，该循环的执行情况相当于执行如下的语句序列。

```
t:=0  ;
i:=0  ;
while (i-10)*sign(t)≤0  do
  begin
    t:=t+sing(5-i);
    i:=i+t
  end;
```

表1.7

i	t	次数
0	0	1
1	1	2
3	2	3
6	3	4
8	2	5
9	1	6
9	0	7
8	-1	

因此，循环执行情况如表1.7所示。

共计执行七次循环。

在PL/I的意义下，该循环的执行情况相当于执行如下语句序列：

```
t:=0;
i:=0;
```

```
        INCR:=t;
        LIMIT:=10;
        goto over;
AGAIN:  i:=i+INCR;
over:   if (i-LIMIT)*sign(INCR)≤0
        then begin
             t:=t+sign(5-i);
             go to AGAIN
             end;
```

显然 INCR = 0，所以 sign(INCR) = 0。因此条件

(i − LIMIT) * sign(INCR) = 0，始终不变，从而产生无限次循环。

1.11 对于程序：

```
procedure p(x, y, z);
  begin
    y:=y+1 ;
    z:=z+x
  end;   {p}
begin
  A:=2 ;
  B:=3 ;
  P(A+B, A, A) ;
  write(A)
end
```

若参数传递的办法分别为(1)传名(Call by name)，(2)传地址(Call by reference)，(3)得结果(Call by result)，以及(4)传值(Call by value)，试问，程序执行时所输出的A分别是什么？

解：(1)传名(Call by name)。即当过程调用时，其作用相当于把被调用段的过程体抄到调用出现处，但须将其中出现的任一形参都代之以相应的实参。因此，该程序的执行等价于执行如下语句：

```
A:=2;
B:=3;
A:=A+1;
A:=A+(A+B);
write(A);
```

输出的A值为9。

(2)传地址(Call by reference)。即当程序控制转入被调用段之后，被调用段首先把实参地址抄进相应形参的形式单元中，过程体对形参的任何引用或赋值都被处理成对形式单元的间接访问。当被调用段工作完毕返回时，形式单元（它们都是指示器）所指的实参单元就持有所期望的值。因此该程序的执行等价于执行如下语句：

```
① A:=2;
  B:=3;
  T:=A+B;
```