第 3 章　代数思维与计算机解题

教学目标

- C/C++程序的基本结构
- 变量的定义和使用
- 代数思维

内容要点

- 程序的基本结构
- 程序说明部分
- 预编译命令
- 主函数
  - ▶ 声明部分
  - ▶ 执行部分
- 变量与数据类型
  - ▶ 变量的基本概念
  - ▶ 数据类型与变量的地址空间
  - ▶ 定义变量和赋初值
  - ▶ 变量赋值的 5 个要点
  - ▶ 指针变量
  - ▶ 指针定义与初始化
  - ▶ 指针赋值

在讲述这一章之前，先举一个例子。

【任务 3.1】 王小二同学是一个聪明的孩子，他到超市去买东西，看到电子计价器算账方便快捷，就想编程模拟操作一下。下面先请读者看程序，然后我们再做解释。

3.1　程序的基本结构

先来读任务 3.1 的程序，为了易于说明，程序清单的左边加注了表示行号的数字。

```
1 //*****************************************
2 //* 程 序 名：3_1.cpp                      *
3 //* 作    者：王小二                       *
4 //* 编制时间：2002 年 7 月 7 日            *
5 //* 主要功能：计算应付款                   *
6 //*****************************************
7 #include <iostream>     //预编译命令
```

```
8 using namespace std;
9
10 int main()                                    //主函数
11 {                                             //主函数开始
12     float applePrice = 3.5;                   //苹果单价，3.5 元/千克
13     float bananaPrice = 4.2;                  //香蕉单价，4.2 元/千克
14     float appleWeight = 0.0;                  //苹果重量，初始化为 0
15     float bananaWeight = 0.0;                 //香蕉重量，初始化为 0
16     float total = 0.0;                        //总钱数，初始化为 0
17     cout << "请输入苹果重量" << endl;          //提示信息
18     cin << appleWeight;                       //输入苹果重量
19     cout << "请输入香蕉重量" << endl;          //提示信息
20     cin << bananaWeight;                      //输入香蕉重量
21     total = applePrice * appleWeight + bananaPrice * bananaWeight;
22                                               //计算应付款
23     cout << "应付款" << total << endl;        //输出应付款
24     return 0;
25 }   //主函数结束
```

1．程序说明部分

写好的程序通常还会再修改，或与他人交流。由于程序语言与人类自然语言在表达上的差异，程序代码示意图不太容易从代码直接反映出来，有些重要信息（如作者、时间、目的等）也不属于源码性质，所以，一般要在源程序开头几行对程序作一个简要的说明。这些说明是以 C++的注释形式出现的。所谓注释，是指计算机不对其进行计算的部分，在 C++中以“//”符号引导的同一行内的文字都是注释。

在读程序时，首先要看程序说明，这件事十分重要。此程序的说明有 4 项：程序名、作者、编制时间、主要功能。在源程序清单中占 6 行，这 6 行纯粹是为了说明用的，不属于机器要计算的内容，因此，在每一行的前面冠以注释符号“//”。为了节省篇幅，本书后续章节中的源程序代码都省略了程序说明。

2．预编译命令

在第 7 行以“#”开头的是预编译命令。“#include <iostream>”意思是将程序库中的输入输出流文件作为头文件加入到现在要编写的程序中。

3．输入流对象 cin

cin 表示输入流对象，它也是输入输出流库的一部分，与 cin 相关联的输入设备是键盘。当从键盘输入字符串时，形成了输入流（数据流），用提取操作符“>>”将数据流存储到一个事先定义好的变量中，例如下面两条语句：

```
float x;
cin >> x;
```

第一条语句定义了一个浮点数类型的对象，即变量 x；第二条语句是用键盘输入一个带小数点的数，例如 3.14159。图 3.1 描述了提取输入流的示意图。

4．主函数

从第 10 行到第 25 行是主函数。主函数是以“main()”为标识的，这是每一个程序都

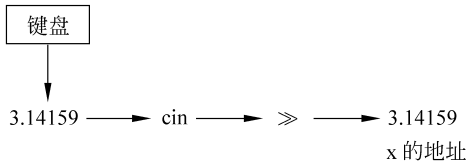

图 3.1　cin 输入流

必须有的标识。

主函数 main()的函数体由一对大括号{}括起，函数体包含两个部分：前面是声明部分，后面是执行部分。按规定声明在前，执行在后。不声明者，不能执行。

在任务 3.1 中声明了以下 5 项内容，依次在程序的第 12～16 行：

（1）变量名 applePrice 是苹果单价，在其前的 float 是该变量所取的数据类型，3.2 节将详细讲解。在其后的“=3.5”是赋初值给该变量，这里苹果的单价是每千克 3.5 元。

（2）变量名 bananaPrice 是香蕉单价。

（3）变量名 appleWeight 是苹果重量。

（4）变量名 bananaWeight 是香蕉重量。

（5）变量名 total 是总钱数。

声明部分之后是对 5 个变量的操作，即执行部分。

第 17 行和第 19 行是让屏幕显示提示信息，告诉程序的使用者下面准备用键盘输入苹果和香蕉的重量。这两条语句用 cout 输出流。

第 18 行和第 20 行是用 cin 输入流将键盘输入的实数分别赋给变量 appleWeight（苹果重量）和 bananaWeight（香蕉重量）。

第 21 行用来计算应付款 total，这是一条赋值语句，计算购买香蕉和苹果的钱，相加后将总钱数赋给 total 变量。

第 23 行是输出语句，将应付款显示到屏幕上。

第 24 行要有“return 0;”。

第 11 行与第 25 行所包含的一对大括号是主函数 main()所必需的，被这一对大括号所括起的语句，就是主函数的内容。任务 3.1 是一个完整的例子，在对变量有了一些初步的感性认识之后，下面再深入讲述有关变量的重要概念和一些特点。

3.2　变量与数据类型

3.2.1　变量的基本概念

变量是相对于常量而言的，在程序中经过操作其值允许改变的量称为变量。

变量在使用前必须加以定义。

每一个变量要有一个与其他变量不相同的名字，称为变量名。

变量在计算机中需要占据存储空间，这些空间都有各自不同的内部编号。这些编号称为变量在计算机内存中存储单元的地址，简称变量的地址。

变量名的第一个字符必须是字母或下画线，其后的字符只能是字母、数字和下画线，且所用的名字不得与 C/C++语言系统所保留的关键字相同。变量中的字母是区分大小写的，

即大小写有差异的两个变量是不同的变量。

建议

在给变量命名时应考虑实际含义，以便提高程序的易读性。例如任务 3.1 中的苹果单价用 applePrice。

3.2.2 数据类型与变量的地址空间

程序中的变量取什么数据类型是由工程任务的需要决定的。C/C++中的数据类型可分为以下两大类：第一类是基本数据类型，包括整型、浮点型和字符型；第二类是构造数据类型，包括数组、结构、联合、枚举等。所谓构造数据类型，是指这种类型的数据是由若干个基本数据类型的变量按特定的规律组合构造而成的。

各种数据所能表示的数据精度不同，因而它所占用的内存空间的大小不同。下面仅就基本数据类型来分析所能表示的数的精度和所占用的内存空间。

基本数据类型有：

（1）整型。即整数类型，它又可分为 4 种。

① int：整型，占用 4 字节，数的表示范围是－2 147 483 648～2 147 483 647。

② unsigned int：无符号整型，占用 4 字节，数的表示范围是 0～4 294 967 295。

③ long int：长整型，占用 4 字节，数的表示范围是－2 147 483 648～2 147 483 647。

④ unsigned long int：无符号长整型，占用 4 字节，数的表示范围是 0～4 294 967 295。

（2）实型。即实数类型，它又可分为 3 种。

① float：浮点型，占用 4 字节，数的表示范围是-3.4×10^{38}～-2.2×10^{-38}，1.2×10^{-38}～3.4×10^{38}，有效位为 7 位。

② double：双精度型，占用 8 字节，数的表示范围是-1.7×10^{308}～-2.2×10^{-308}，2.2×10^{-308}～1.7×10^{308}，有效位为 15 位。

③ long double：长双精度型，占用 16 字节，数的表示范围是-1.2×10^{4932}～-3.3×10^{-4932}，3.3×10^{-4932}～1.2×10^{4932}，有效位为 19 位。

（3）bool：逻辑型。占用 1 字节。

（4）char：字符型。占用 1 字节。

3.3 定义变量和赋初值

在主函数 main() 中要对一些变量进行定义，提出合适的精度要求，指出这些变量的数据类型，目的是为变量分配内存单元并赋予初始值。例如定义变量名为 a 的整型变量，程序写成：

```
int a = 30;
```

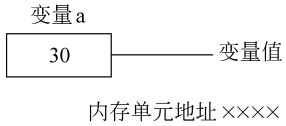

图 3.2 变量的定义和内存地址的关系

系统会根据这个精度要求，安排 4 个字节的内存单元存放 a 变量的整数值，见图 3.2。在图中变量名 a 是这个内存单元的符号地址。在本节的学习中建立起变量与变量地址的概念会对以后的学习

大有用处。一讲到变量就要想到有一个地址与之联系。

为了更清楚地讲清变量内存单元的大小与存储单元地址的关系，我们来看如下的示例程序：

```
#include <iostream>
using namespace std;

int main()
{
    int i;
    long m;
    float f;
    double d;
    cout << "sizeof(int)=" << sizeof(int) << ", sizeof(i)="
         << sizeof(i) << ", &i=" << &i << endl;
    cout << "sizeof(long)=" << sizeof(long) << ", sizeof(m)="
         << sizeof(m) << ", &m=" << &m << endl;
    cout << "sizeof(float)=" << sizeof(float) << ", sizeof(f)="
         << sizeof(f) << ", &f=" << &f << endl;
    cout << "sizeof(double)=" << sizeof(double) << ", sizeof(d)="
         << sizeof(d) << ", &d=" << &d << endl;
    return 0;
}
```

在上面的程序中，“&”表示取变量地址的操作符，运算结果是返回变量在内存中存储单元的位置，即地址值。

程序运行结果如下：

```
sizeof(int)=4, sizeof(i)=4, &i=0x7fff5fbffa9c
sizeof(long)=8, sizeof(m)=8, &m=0x7fff5fbffa90
sizeof(float)=4, sizeof(f)=4, &f=0x7fff5fbffa98
sizeof(double)=8, sizeof(d)=8, &d=0x7fff5fbffa88
```

在上面的运行结果中，地址是通过 cout 输出到屏幕的，通常是以十六进制的格式表示的。需要说明的是，上述地址信息在读者自己机器上可能会有所不同。

3.4　变 量 赋 值

给变量赋值是一个非常重要的概念。

3.4.1　赋值符号与赋值表达式

在 C/C++中赋值符号为“=”，赋值表达式的一般格式为：

```
<变量> = <表达式>
```

例如：

```
PI = 3.14159;   //读作将表达式的值 3.14159 赋给变量 PI
C = sin(PI/4);  //读作将表达式 π/4 的正弦函数值赋给变量 C
```

3.4.2 变量赋值的 5 要素

给变量赋值时要注意以下 5 点：

（1）变量必须先定义再使用。

（2）在变量定义时就赋给初值，是良好的编程习惯。

（3）对变量的赋值过程是“覆盖”过程。所谓“覆盖”是在变量地址单元中用新值去替换旧值。

（4）读出变量的值后，该变量保持不变，相当于从中复制一份出来。

（5）参与表达式运算的所有变量都保持原来的值不变。

下面举例说明上述特点。

```
#include <iostream>
using namespace std;

int main()
{
    int a = 0, b = 0, c = 0;    //定义 a、b、c 为整型变量，均初始化为 0
    a = 7;        //a 赋值为 7，覆盖了原来的 0
    b = a;        //b 赋值为 a，a 中的值覆盖了 b 中的值，但 a 中的值不变
    c = a + b;    //将 a+b 的值赋给 c，a+b 的值为 14 去覆盖 c 中的 0，a 与 b 保持 7 不变
    a = a + 1;    //将 a+1 的值赋给 a，a+1 的值为 8 覆盖了原来的 7
    cout << a << ' ' << b << ' ' << c << endl;
    return 0;
}
```

上例中，“a = a + 1”可简化写作“a++”，图 3.3 说明了这 5 条语句的执行过程。

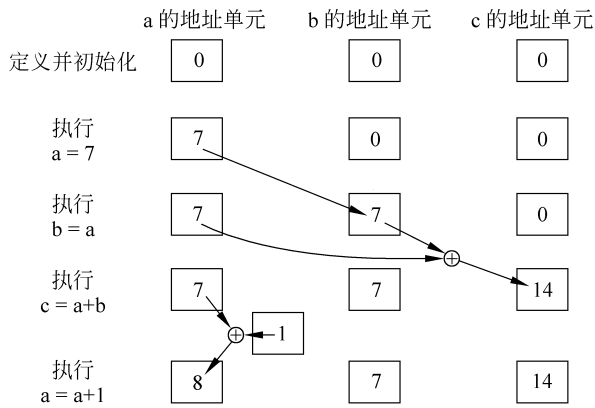

图 3.3　变量赋值过程

3.5　指 针 变 量

指针是 C/C++语言中的一个重要概念。掌握指针的用法，可使程序简洁、高效、灵活。指针看似复杂，但并不难学。

为了了解什么是指针，先看一个小故事。

地下工作者阿金接到上级指令，要去寻找打开密电码的密钥，这是一个整数。几经周折，才探知如下线索：密钥藏在一栋 3 年前就被贴上封条的小楼中。一个风雨交加的夜晚，阿金潜入了小楼，房间很多，不知该进哪一间，正在一筹莫展之际，忽然走廊上的电话铃声响起。艺高人胆大，阿金毫不迟疑，抓起听筒。只听一个陌生人说：“去打开 211 房间，那里有线索。”阿金疾步上楼，打开 211 房间，用电筒一照，只见桌上赫然写着：地址 1000。阿金眼睛一亮，迅速找到 1000 房间，取出重要数据 66，完成了任务。

可用图 3.4 来描述这几个数据之间的关系。

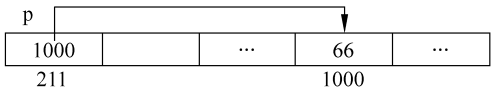

图 3.4　数据存放

说明

（1）数据藏在一个内存地址单元中，地址是 1000。

（2）地址 1000 又由 p 单元所指认，p 单元的地址为 211。

（3）66 的直接地址是 1000；66 的间接地址是 211；211 中存的是直接地址 1000。

（4）称 p 为指针变量，1000 是指针变量的值，实际上是有用数据在存储器中的地址。指针变量就是用来存放另一变量地址的变量（变量的指针就是变量的地址）。

3.5.1　指针定义与初始化

指针是一种特殊的变量，特殊性表现在类型和值上。从变量角度看，指针也具有变量的 3 个要素：

（1）变量名，这与一般变量命名相同，由英文字符开始。

（2）指针变量的类型，是指针所指向的变量的类型，而不是自身的类型。

（3）指针的值是某个变量在内存中的地址，简称变量的内存地址。

例如下面的语句是定义一个名为 p 的指针，该指针指向一个整数类型的变量，且被初始化为 NULL。

```
int *p = NULL;
```

一旦指针 p 被定义，系统会为 p 分配一个内存单元，该单元的地址可以用“&p”表示（符号&p 表示 p 的地址），如图 3.5 所示。

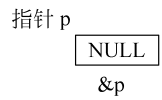

图 3.5　指针 p 被分配一个内存单元

在 p 中赋予一个符号化的常量 NULL，称之为将

指针 p 初始化为 0。这个符号化的常量 NULL 是在头文件<iostream>中定义的。将指针初始化为 NULL 等于将指针初始化为 0，在这里整数 0 是 C/C++系统唯一一个允许赋给指针类型变量的整数值。除 0 以外的整数值是不允许赋给指针变量的，因为指针变量的数据类型是内存的地址，而不是任何整数。要记住：值为 NULL 的指针不指向任何变量。在定义时让指针初始化为 NULL 可以防止其指向任何未知的内存区域，以避免产生难以预料的错误。定义指针并将其初始化为 NULL 是一个值得提倡的好习惯。

3.5.2 指针赋值

前已述及指针变量是一个特殊的变量，其值是内存的地址，给指针赋值，就是将一个内存地址装入指针变量，这件事一做完就意味着指针指向了该内存地址。请看下例：

```
int a = 66;                    //定义一个整型变量 a，并将其初始化为 66
int *p = NULL, *q = NULL;      //定义 p，q 为指向整型变量的指针变量并初始化为 0
p = &a;                        //将变量 a 地址赋给 p
q = p;                         //将 p 的值赋给 q
```

在图 3.6 中，当 a 变量的地址赋给指针 p，意味着让指针 p 指向 a。

在图 3.7 中，当执行 q = p 后，p 中所存的 a 变量的地址值也就被放到 q 变量中，意味着让指针 q 也指向 a。这里用到了一个取地址运算符“&”，“&a”表示取变量 a 所在的内存的地址。

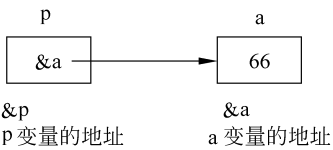

图 3.6 指针赋值

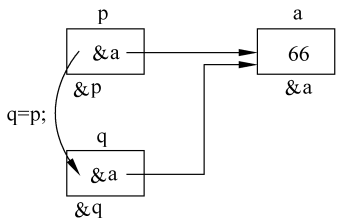

图 3.7 赋值

3.5.3 在赋值语句中使用间接访问运算符

下面给出一个程序，在程序中定义两个指针变量 p 和 q，在第 14 行和第 15 行，使用取地址运算符将整型变量 akey 的地址赋给 p，b 的地址赋给 q，即让 p 和 q 分别指向 akey 和 b。出现在第 16 行语句中的*p 和*q 代表指针所指向的变量单元中的内容。换句话说*p 和 akey 等价，*q 与 b 等价。赋值表达式

```
*q = *p;
```

等价于

```
b = akey;
```

以下是完整的源程序。

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int akey = 0, b = 0;          //定义整型变量
    int *p = NULL, *q = NULL;     //定义指针变量
    akey = 66;                    //赋值给变量 akey
    p = &akey;                    //赋值给指针变量 p，让 p 指向变量 akey
    q = &b;                       //赋值给指针变量 q，让 q 指向变量 b
    *q = *p;                      //将 p 所指向的 akey 的值赋给 q 所指向的变量 b
    cout << "b=" << b << endl;    //输出 b 的值
    cout << "*q=" << *q << endl;//输出 b 的值
    return 0;
}//函数体结束
```

程序操作如图 3.8 所示。

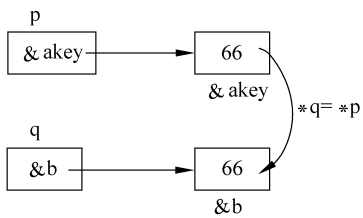

图 3.8　程序 3_3 说明

3.6 小　　结

（1）掌握变量的概念和变量赋值的 5 个要素，对于程序设计是十分重要的。

（2）比较第 2 章与第 3 章的内容可见，仅仅能够将计算机当作功能强大的计算器是远远不够的，只是使用了输出流 cout 和一些数学函数，基本上是算术运算。当引入变量后，就有了质的飞跃，上升到代数运算水平，编写程序用计算机自动解题才有了一个初步的基础。

（3）指针是一个特殊的变量，其值是内存的地址。给指针赋值就是将一个内存地址装入指针变量。如果这个内存地址是某变量的地址，则该指针就指向了该变量。

（4）指针变量的类型是指针所指向的变量的类型。

（5）在定义一个指针变量时，将其初始化为 NULL 是一个好习惯。

（6）对指针赋值是将它所指向的变量的地址赋给指针变量。这时要用到取地址运算符 &。&a 表示取变量 a 的地址。如果有

```
int a = 66;
int *p = NULL;
```

则

```
p = &a;
```

就是将变量 a 的地址赋给指针变量 p。

（7）在赋值语句中可使用间接访问运算符*，如

```
int a = 66;
int b = 0;
int *p = &a;
b = *p;
```

则会将 p 指针所指向的变量 a 的值 66 赋给变量 b。

习　　题

新年就要到了，假定你要负责筹备一个晚会，晚会上要有抽奖活动，奖品由你设计，例如一等奖是一个双肩背的包，二等奖是一件普通的背心，三等奖是一个笔记本。奖项的人数、钱数都由你来筹划，原则是注重实效和节约。请编写一个程序计算要筹集多少钱。

当然，这是你第一次编写这类程序，建议你借鉴王小二编写的程序。

第 4 章　逻辑思维与计算机解题

教学目标

- 将实际问题抽象为逻辑关系
- 枚举法解题思路
- 关系与关系表达式
- 程序的循环结构与分支结构

内容要点

- 关系运算符与关系表达式
- 从人的思维到用计算机语言的表示
- 枚举的概念与思路
- 循环结构
- 分支结构

本章试图以“任务驱动”的方式来引出如下内容：

（1）介绍关系运算符与关系表达式；

（2）介绍枚举法的解题思路；

（3）为了枚举引出程序的循环结构；

（4）为了判断和裁决引出程序的分支结构。

按以上这种讲法目的明确，有的放矢，思路清晰，易于上手。

计算机强大的逻辑分析功能是由人通过程序赋给它的，一些逻辑问题必须转换成计算机能够看得懂的数学表达式和一定的程序指令。这一章通过例子来介绍如何将人对问题的思考转换为让计算机能解的数学表达式，同时给出一些通常要用到的程序结构和 C/C++ 语句。

【任务 4.1】 清华附中有 4 位同学中的一位做了好事不留名。表扬信来了之后，校长问这 4 位同学是谁做的好事。

A 说：不是我。

B 说：是 C。

C 说：是 D。

D 说：他胡说。

已知 3 个人说的是真话，一个人说的是假话。现在要根据这些信息，找出做了好事的人。

为了解这道题，需要学习如何通过逻辑思维与判断找到解这类问题的思路。

4.1 关系运算和关系表达式

4.1.1 关系运算符

关系运算符有如下 6 个：>=（大于等于），>（大于），==（等于），<=（小于等于），<（小于）和 !=（不等于）。

为了讲解关系运算符和关系表达式，先在机器上建立和运行程序 4_1.cpp。

```
#include <iostream>
using namespace std;

int main()
{
    cout << "3>2 的逻辑值是" << (3 > 2) << "，1 为真。" << endl;
    cout << "3>=2 的逻辑值是" << (3 >= 2) << "，1 为真。" << endl;
    cout << "3==2 的逻辑值是" << (3 == 2) << "，0 为假。" << endl;
    cout << "3<2 的逻辑值是" << (3 < 2) << "，0 为假。" << endl;
    cout << "3<=2 的逻辑值是" << (3 <= 2) << "，0 为假。" << endl;
    cout << "3!=2 的逻辑值是" << (3 != 2) <<  "，1 为真。" << endl;
    return 0;
}
```

程序运行结果

```
3>2 的逻辑值是 1，1 为真。
3>=2 的逻辑值是 1，1 为真。
3==2 的逻辑值是 0，0 为假。
3<2 的逻辑值是 0，0 为假。
3<=2 的逻辑值是 0，0 为假。
3!=2 的逻辑值是 1，1 为真。
```

4.1.2 关系表达式的一般格式

关系表达式的格式如下：

```
<变量 1>  关系运算符  <变量 2>
```

例如，变量 1 为 b，变量 2 为 c，关系运算符为>，则关系表达式为

```
b > c
```

4.1.3 将“是”“否”写成关系表达式

1．定义字符型变量

为了完成任务 4.1，将 4 个人所说的 4 句话写成关系表达式。为此，要定义一种字符型的变量。这里用 thisman 表示要寻找的做了好事的人。在程序中写入：

```
char thisman = ' ';          //定义字符变量并将其初始化为空
```

接着让“==”在这里的含义为“是”，让“!=”在这里的含义为“不是”。利用关系表达式将 4 个人所说的话表示成如下表：

说话人	说的话	写成关系表达式
A	“不是我”	`thisman != 'A'`
B	“是 C”	`thisman == 'C'`
C	“是 D”	`thisman == 'D'`
D	“他胡说”	`thisman != 'D'`

2．字符型变量在内存中的数据

在 C/C++中，字符在存储单元中是以 ASCII 码的形式存放的（字符的 ASCII 码见附录 C）。C/C++实际上是把 char 型按一个字节大小的整型来处理。因此，用赋值语句 thisman = 'A' 和 thisman = 65 是一样的，见图 4.1。

图 4.1　字符型变量的存储

这两个赋值语句是等效的，在以 thisman 为标识的存储单元中存的是数字 65。65 即是字符'A'的 ASCII 编码值。建议用程序 4_2.cpp 加以验证。

```
#include <iostream>
using namespace std;

int main()
{
    char thisman;   //定义字符变量 thisman
    thisman = 'A';  //thisman 赋值为'A'
    //输出关系表达式"65=='A'"的值
    cout << "65=='A' -- 关系表达式的值为" << (65 == 'A') << "，1 为真。" << endl;
    return 0;
}
```

4.2　枚举法的思路

结合任务 4.1 分析，A、B、C、D 这 4 个人只有一位是做好事者。令做好事者为 1，未做好事者为 0，可以有如下 4 种状态（情况）：

状态	A	B	C	D
1	1	0	0	0

续表

状态	A	B	C	D
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

这 4 种状态可简化写成：

状态	赋值表达式
1	thisman = 'A'
2	thisman = 'B'
3	thisman = 'C'
4	thisman = 'D'

显然第一种状态是假定 A 是做好事者，第二种状态是假定 B 是做好事者，……。所谓枚举是按照这 4 种假定逐一地去测试 4 个人的话有几句是真话，如果不满足 3 句为真，就否定掉这一假定，换下一个状态再试。

具体做法如下：

（1）假定让 thisman = 'A'代入 4 句话中：

说话人	说的话	关系表达式	值
A	thisman != 'A';	'A' != 'A'	0
B	thisman == 'C';	'A' == 'C'	0
C	thisman == 'D';	'A' == 'D'	0
D	thisman != 'D';	'A' != 'D'	1

4 个关系表达式的值的和为 1，显然不是'A'做的好事。

（2）假定让 thisman = 'B'代入 4 句话中：

说话人	说的话	关系表达式	值
A	thisman != 'A';	'B' != 'A'	1
B	thisman == 'C';	'B' == 'C'	0
C	thisman == 'D';	'B' == 'D'	0
D	thisman != 'D';	'B' != 'D'	1

4 个关系表达式的值的和为 2，显然不是'B'做的好事。

（3）假定让 thisman = 'C'代入 4 句话中：

说话人	说的话	关系表达式	值
A	thisman != 'A';	'C' != 'A'	1
B	thisman == 'C';	'C' == 'C'	1
C	thisman == 'D';	'C' == 'D'	0
D	thisman != 'D';	'C' != 'D'	1

4 个关系表达式的值的和为 3，就是'C'做的好事。

综上所述，一个人一个人去试，就是枚举。从编写程序看，实现枚举最好用循环结构。

4.3　循 环 结 构

循环结构是程序中用得最多的一种，它发挥了计算机长于重复运算的特点。下面，还是先结合任务 4.1，写出枚举法所要完成的工作。

4.3.1　使用循环结构的部分程序

使用循环结构的程序如下：

```
for (k = 0; k < 4; k = k + 1)   //计数型循环，循环的控制变量为 k
{   //循环体开始
    thisman = 'A' + k;          //产生被试者，依次为'A','B','C','D'
                                //赋值给 thisman
    sum = (thisman != 'A')      //测试'A'的话是否为真
          + (thisman == 'C')    //测试'B'的话是否为真
          + (thisman == 'D')    //测试'C'的话是否为真
          + (thisman != 'D');   //测试'D'的话是否为真
}   //循环体结束
```

4.3.2　for 语句的格式和执行过程

for 语句的格式为：

```
for (表达式 1; 表达式 2; 表达式 3)
{
    循环体（语句组）
}
```

结合图 4.2 来看 for 循环的执行过程，从而了解表达式 1、表达式 2 与表达式 3 各起什么作用。

for 循环的执行过程如下：

（1）求解表达式 1，赋值表达式，置循环控制变量的初值。

（2）测试表达式 2，关系表达式，测试是否未到循环控制变量的终值。

① 如果表达式 2 的值为真，则执行（3）；

② 如果表达式 2 的值为假，则退出循环转至（5）。

（3）执行循环体语句组之后转至（4）。

（4）求解表达式 3，赋值表达式，让循环控制变量增值或减值，再转至（2）。

（5）执行下一条语句。

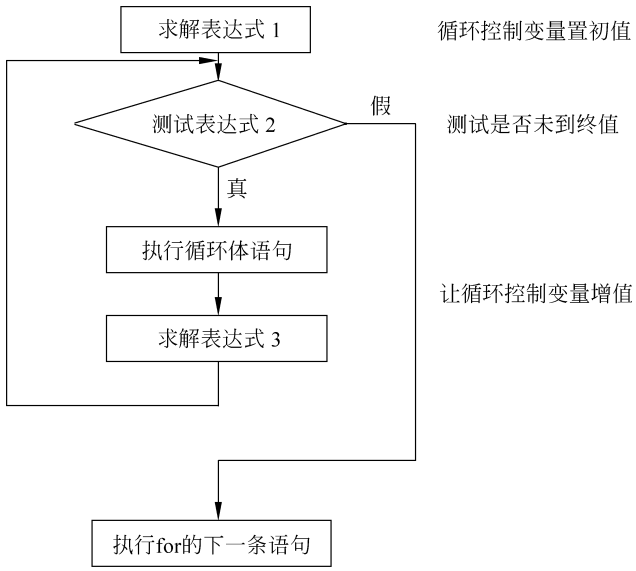

图 4.2　for 循环结构

4.3.3　使用 for 循环解题实例

1．求自然数 1～100 之和

程序 4_3.cpp 如下：

```
#include <iostream>
using namespace std;

int main()
{
    int sum = 0;                            //定义 sum 为整型变量，并初始化为 0
    for (int i = 0; i < 100; i = i + 1) //for 循环
    {   //循环体开始
        sum = sum + (i + 1);                //累加求和
    }   //循环体结束
    cout << "自然数 1～100 之和为" << sum << endl;    //输出累加结果
    return 0;
}
```

【思考】

在程序 4_3.cpp 中做如下修改：

（1）将原来的

```
for (int i = 0; i < 100; i = i + 1)
```

修改为

```
for (int i = 0; i < 100; i = i + 2)
```

问：这是哪些自然数在求和，答案是多少？

（2）将原来的

```
for (int i = 0; i < 100; i = i + 1)
```

修改为

```
for (int i = 0; i < 100000; i = i + 1)
```

执行程序能够得出正确结果吗？如果不能，自己想办法解决。

2．求 10 的阶乘

程序 4_4.cpp 如下：

```
#include <iostream>
using namespace std;

int main()
{
    long mul = 1;                          //定义长整型变量，初始化为 1
    for (int i = 10; i >= 1; i = i - 1)  //用 for 循环作累乘运算
      mul = mul * i;
    cout <<< "10!=" << mul << endl;      //输出 10 的阶乘值
    return 0;
}
```

为了详细了解 10!计算过程的每一步；可以在循环语句中增加一些输出语句，将我们关心的一些中间状态信息以 cout 语句输出到屏幕上。修改后的程序如下：

```
#include <iostream>
using namespace std;

int main()
{
    long mul = 1;                          //定义 mul 为长整型变量，并初始化为 1
    for (int i = 10; i >= 1; i = i - 1) //用 for 循环作累乘运算
    {
       cout << "i=" << i ;                //显示 i
      mul = mul * i;                      //每一步乘积
       cout << "\tmul =" << mul << endl; //显示每一步乘积
       for (int j = 1; j <= 5500; j = j + 1)         //用 for 循环延迟时间
           for (int k = 1; k <= 10000; k = k + 1);   //用 for 循环延迟时间
    }
    cout << "10!=" << mul << endl;                   //显示运算结果
    return 0;
}
```

程序运行结果

```
i=10    mul=10
```

```
i=9    mul=90
i=8    mul=720
i=7    mul=5040
i=6    mul=30240
i=5    mul=151200
i=4    mul=604800
i=3    mul=1814400
i=2    mul=3628800
i=1    mul=3628800
10!=3628800
```

【解题思路】

（1）将 10！展开为 10×9×8×7×6×5×4×3×2×1。

（2）让整型变量 i 表示 10，9，…，1。

（3）让长整型变量 mul 表示乘积，初始时让其为 1。

（4)将求 10 的阶乘考虑成累乘问题，让 i = 10 去乘 mul 再将积存至 mul 中，即 mul = mul * i，之后让 i = i – 1，再用上式累乘，不断地反复做这两个运算，从 i = 10，9，…，1，就完成了求 10！的任务。

（5）恰好计算机擅长做这种重复操作，使用 for 循环是最佳选择。

（6）让循环控制变量 i 就是数字 10，9，…，1。i 的初值为 10，终值为 1。for 循环的 3 个表达式中，表达式 1 为 i = 10，表达式 2 为 i >= 1，表达式 3 为 i = i – 1。

4.3.4 for 循环的程序框图

为了以后讲解方便，有必要使用更为简便的程序结构流程图，如图 4.3 所示。

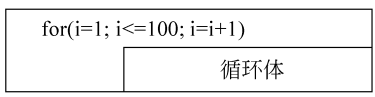

图 4.3　for 循环结构的程序框图

该流程图中突出了 for 循环的 3 个表达式，将循环体表示成放在右下角的一个小一些的矩形方框。循环就是依照条件反复执行这个小矩形方框中的语句组。

4.4 分 支 结 构

为了完成任务 4.1，光有循环结构还不行，还要有用于判断是否已经有 3 个人的话是真话的分支结构。这个程序段如下：

```
if (sum == 3)
{
   cout << "This man is" << 'A' + k << endl;
   g = 1;
}
```

这一段程序可以读作：如果 sum 真的为 3，做下面两件事：
（1）输出做好事的人。
（2）将本题的有解标志置为 1。
其中(sum == 3)为条件判断语句中的条件，根据其真假使程序分支。
图 4.4 是分支程序的流程图，这种图直观清晰，一目了然。

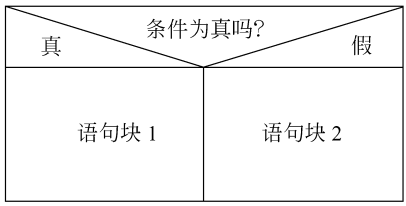

图 4.4　分支结构程序框图

4.4.1　if 语句的格式

1．第 1 种情况

```
if (表达式)
    语句 1;
```

如果表达式为真，则只执行语句 1；否则什么都不做。

2．第 2 种情况

```
if (表达式)
{
    语句块 1;
}
```

如果表达式为真，执行语句块 1（多条语句）的内容；否则什么都不做。

3．第 3 种情况

```
if (表达式)
    语句 1;
else
    语句 2;
```

如果表达式为真，执行语句 1；否则执行语句 2。

4．第 4 种情况

```
if (表达式)
{
    语句块 1;
}
else
{
    语句块 2;
}
```

如果表达式为真，执行语句块 1；否则执行语句块 2。

4.4.2 分支结构的实例

编程实现如图 4.5 所示的函数。

$$y=\begin{cases}1, & x>0\\ 0, & x=0\\ -1, & x<0\end{cases}$$

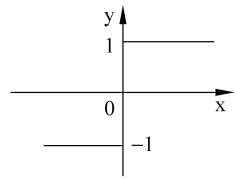

图 4.5　符号函数

参考程序 4_5.cpp 如下：

```
#include <iostream>
using namespace std;

int main()
{
    float x = 0, y = 0;          //定义 x、y 为浮点类型变量，并初始化为 0

    cout << "请输入 x" << endl;   //提示信息
    cin >> x;                    //从键盘输入浮点数送至 x 中

    if (x > 0)                   //如果 x>0，将 1 赋给 y
    {
        y = 1;
    }
    else if (x == 0)             //如果 x==0，将 0 赋给 y
    {
        y = 0;
    }
    else                         //否则(x<0)，将-1 赋给 y
    {
        y = -1;
    }
    //输出 x、y 的值
    cout << "当 x=" << x << "时, y=" << y << endl;
    return 0;
}
```

该程序的框图如图 4.6 所示。