第 3 章　程序设计初步

学习要点

- ❖ 掌握 C++ 语言 I/O 流类库中常用控制符的使用方法。
- ❖ 理解算法的含义。
- ❖ 掌握 if 语句三种形式的特点及一般使用方法。
- ❖ 掌握 while、do-while 和 for 三种循环语句的特点和用法。
- ❖ 掌握 switch、continue 和 break 语句的用法。
- ❖ 初步掌握程序设计的方法。

3.1　在输出流中使用控制符

有时人们在数据输出时有一些特殊的要求，如：在输出时规定字段宽度、设置填充字符、需要数据向左或向右对齐等。这就要使用输出格式控制了，C++ 提供了在输出流中使用的格式控制符（有的地方称为操纵符）见表 3-1。

表 3-1　输出流格式控制符

控 制 符	作　　用
endl	控制换行
dec	设置数值的基数为 10
hex	设置数值的基数为 16
oct	设置数值的基数为 8
setfill (c)	设置填充字符 c，c 可以是字符常量或字符变量
setw(n)	设置字段宽度为 n 位

下面介绍几个输出流控制符的含义。

(1) setw 可以看成是 setwidth 的缩写，是设置输出的宽度之意。

(2) setfill 是设置填充符号之意。

需要注意的是：如果使用了控制符（endl、dec、hex、oct 除外），在程序的开头除了要增加输入输出流预处理命令头文件 iostream.h 外，还要增加控制符预处理命令头文件 iomanip.h。

例 3-1　设置输出值的输出宽度示例。

除了使用空格来强行控制输出间隔外，还可以用 setw(n)控制符。setw 控制符主要用来对输出预留空间格数(或者说是输出列数)，若空间有余，则向右对齐；若空间不够，按数据长度输出。注意：使用 setw 控制符时程序必须包括头文件 iomanip.h。

源程序如下：

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
    int a=80;
    int b=8000;
    cout<<setw(5)<<a<<endl;
    cout<<setw(2)<<b<<endl;
}
```

程序运行结果在屏幕上显示是：

```
   80
8000
```

setw(5)预留的是 5 位空间格数，由于 a 的值本身仅有两位字符，右对齐后前面留有 3 个空格；b 的本身就有 7 位字符，而 setw(2)预留的是两位空间格数，小于 b 本身的占位数，所以按数据的实际长度输出，前面没有空格位。

setw 控制符只对紧接其后的待输出变量有效。如以上源程序改成：

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
    int a=80,b=8000;
    cout<<setw(5)<<a<<b<<endl;
}
```

由于 setw(5)只对 a 有效，所以程序运行结果在屏幕上显示是：

```
808000
```

例 3-2　输出八进制和十六进制数示例。

控制符 hex、oct 和 dec 分别对应十六进制、八进制和十进制数的显示，这三个控制符已在 iostream.h 头文件中定义。

源程序如下：

```
#include <iostream.h>
void main()
{
    int number=1001;
    cout<<"十进制:"<<dec<<number<<endl
```

```
        <<"十六进制:" <<hex<<number<<endl
        <<"八进制:" <<oct<<number<<endl;
}
```

运行结果为：

```
十进制:1001
十六进制:3e9
八进制:1751
```

1001 是一个十进制数，在输出时，I/O 流根据控制符进行过滤，使其按一定的进制来显示。

例 3-3 设置填充字符示例。

setw 可以用来确定显示的宽度。默认时，I/O 流使用空格符来保证字符间的正确间隔。用 setfill 控制符可以确定一个非空格的别的字符。setfill 在头文件 iomanip. h 中定义。

源程序如下：

```
#include <iostream.h>
#include <iomanip.h>
void main()
{
   cout<<setfill('*')                    //设置填充的字符
       <<setw(2)<<21<<endl
       <<setw(3)<<21<<endl
       <<setw(4)<<21<<endl;
}
```

运行结果为：

```
21
*21
**21
```

3.2 算法概述

任务 3-1 从键盘读入一系列整数，计算并输出前十个正整数的和。

初学者常常会有这样一种感觉：读别人编的程序比较容易，等到自己遇到问题编写程序时就难了。虽然学了程序设计语言，可还是不知从何下手，这是为什么呢？其中一个重要的原因就是没有掌握基本的算法。事实上，在生活中每做一件事情，都要遵循一定的步骤。例如：你来到一座城市，虽然有公共汽车，但是你不知道按照怎样的路线走才能找到你要去的地方。当别人告诉你一条路线，如先乘什么车，在什么站下车，再换乘什么车等。这就好比告诉了你一个解决乘车问题的算法，于是你就可以沿着这条路线到达目标，下次再来时，你就不会感到为难了。所以，请读者一定要重视算法的设计，多了解、掌握和积累一些计算机常用算法，不要急于编写程序，应养成编写程序前先设计好算法的习惯。

计算机是能够直接进行算术运算和逻辑运算的机器。算术运算是指加、减、乘、除等

运算，运算结果是一个数值。如计算机能够进行 5＋4×3 的算术运算，求得的结果为 17。逻辑运算是指比较两个数值或一串字符的大小、判断一个条件（或称命题）是否成立等运算，运算结果是逻辑值“真”或“假”，又称为“是”或“否”、“成立”或“不成立”等。如计算机能够判断命题 10＞5 为真，能够判断命题 x＜10 是否成立。当 x 的值确实小于 10，则判断出该命题成立；当 x 的值大于或者等于 10，则判断出该命题不成立。

计算机除了能够对数据进行算术运算和逻辑运算外，还能够进行数据存储、传送等操作。

数据能够被临时性地保存在内存中，能够被永久性地保存在外存中。数据传送是指数据从一种设备传送到另一种设备，或从同一个设备的一个存储位置传送到另一个存储位置中去。人们经常需要把数据从输入设备传送到内存，从内存传送到输出设备，在内存和外存之间相互传送，在内存内部不同位置之间的进行传送等。

为了能很好地完成给定的任务，程序设计过程大致需要以下三步。

（1）确定算法与数据结构。

（2）用流程图表示程序的思想。

（3）用程序设计语言编制计算机程序。

那么，什么是算法呢？简言之，算法就是解决问题的步骤和方法。

利用计算机解决问题，首先要设计出适合计算机执行的算法，此算法包含的步骤必须是有限的，每一步都必须是明确的，且计算机最终能够执行。因此算法中的每一步都只能是一些基本操作或它们的不同组合。这些操作包括数据存储、数据传送、算术运算、逻辑运算等。

著名的计算机科学家沃思曾提出过一个经典公式：

程序＝数据结构＋算法

这个公式说明一个程序应由两部分组成：

（1）数据的描述和组织形式，即数据结构。

（2）对操作或行为的描述，即操作步骤，也称算法。

正如前面所说，知道乘车路线是找到目的地的关键，编写一个程序的关键就是合理地组织数据和设计算法。去一个地方可能会有多条路线，同样地，解决一个问题也会有多种算法。比如，排序算法就有很多，有冒泡法、交换法、选择法等。程序设计语言好比是汽车，它仅仅是实现算法（到达目的地）的工具。到达目的地，可以利用各种交通工具。同理，对于同一个算法，可以利用各种程序设计语言来实现，比如，用 C 语言、FORTRAN 语言、Pascal 语言、汇编语言等。用不同的算法以及不同的程序设计语言解决同一个问题，只是速度和时间、效率上不同而已。每个程序都要依靠算法和数据结构，在某些特殊领域，如计算机图形学、数据结构、语法分析、数值分析、人工智能和模拟仿真，解决问题的能力几乎完全依赖于最新的算法和数据结构。因此，针对某个应用领域，要想开发出高质量、高效率的程序，除了要熟练掌握程序设计语言这种工具和必要的程序设计方法以外，更重要的是要多了解、多积累并逐渐学会自己设计一些好的算法。

在面向过程的程序设计中，程序设计者必须拟定计算机的具体步骤，不仅要考虑程序应“做什么”，还要解决“怎么做”的问题，根据程序要“做什么”的要求，写出一个

个语句，安排好它们的执行顺序。怎样设计这些步骤，怎样保证它的正确性和具有较高的效率，这就是算法需要解决的问题。所谓算法，就是一个有穷规则的集合，其中的规则确定了解决某个特定类型问题的运算序列。简单地说，就是为解决一个具体问题而采取的确定有限的操作步骤。当然，这里所说的算法仅仅是指计算机算法，即计算机能够执行的算法。

算法应具备如下特点。

(1) 有输入：可以有零个或多个输入。

(2) 有输出：必须具有一个或多个输出。

(3) 有穷性：在执行有穷步骤后结束。

(4) 确定性：对处理问题的结果不能出现二义性。

(5) 高效性：执行的时间要短，并且不占用过多的内存。

下面分类介绍算法的描述方法。

1. 自然语言描述

自然语言就是人们日常生活中使用的语言。用自然语言描述算法时，可以使用汉语、英语、数学符号等，比较符合人们日常的思维习惯，通俗易懂，初学者容易掌握。

任务 3-1 的处理过程用自然语言描述如下。

(1) 设置求和变量 sum 并使其初值为零。

(2) 读入键盘输入的数据。

(3) 判断是否是正数，如果是正数则加入 sum 中。

(4) 重复(2)、(3)步，直到加入十个正整数为止。

(5) 输出 sum 的值。

用自然语言表示虽然通俗易懂，但文字冗长，容易出现“歧义性”。自然语言表示的含义往往不大严格，要根据上下文才能判断其正确含义。假如有这样一句话：“张先生对李先生说他的孩子考上了大学”。请问是张先生的孩子考上大学呢，还是李先生的孩子考上大学呢？光从这句话本身难以判断。此外，用自然语言描述结构复杂的算法，不方便。因此，除了很简单的问题以外，一般不用自然语言描述算法。

2. 流程图描述

流程图是一个描述程序的控制流程和指令执行情况的有向图，它是程序的一种比较直观的表示形式。美国国家标准化协会(ANSI)规定了一些符号作为常用的流程图符号，已为世界各国程序工作者普遍采用。用传统流程图描述算法的优点是形象直观，各种操作一目了然，不会产生“歧义性”，便于理解，算法出错时容易发现，并可直接转化为程序；但缺点是所占篇幅较大，由于允许使用的流程线过于灵活，不受约束，使用者可使流程任意转向，从而造成程序阅读和修改上的困难，不利于结构化程序的设计。

3. N-S 结构化流程图(盒图)描述

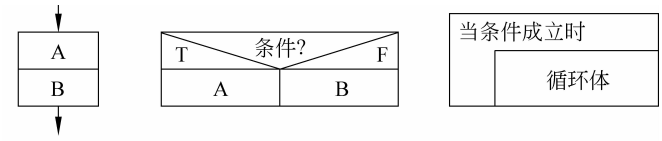
N-S 结构化流程图是由美国学者 I. Nassi 和 B. Schneiderman 于 1973 年提出的，N-S 图就是用这两位学者名字的首字母命名的。它的最重要的特点就是完全取消了流程线，这样算法被迫只能从上到下顺序执行，从而避免了算法流程的任意转向，保证了程序的

质量。

与传统的流程图相比，N-S 图的另一个优点就是既形象直观，又比较节省篇幅，特别适合结构化程序的设计。

结构化程序设计采用单入口单出口的控制方式，程序由顺序、分支选择、循环三种基本控制结构组成。这三种基本结构的特点都是：每一种结构只有一个入口和一个出口，任何一个问题都可以用这三种基本结构实现，任何复杂的程序都可以分解为由这三种基本结构组成。

跟三种基本结构对应的 N-S 图如图 3-1 所示。



(a) 顺序结构　(b) 分支选择结构　(c) 循环结构

图 3-1　三种结构化程序结构的 N-S 图

跟任务 3-1 对应的 N-S 盒图如图 3-2 所示。

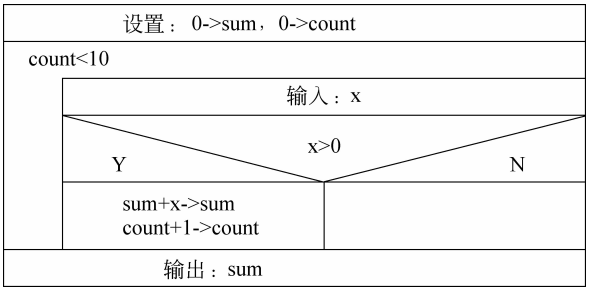


图 3-2　任务 3-1 的 N-S 图

如果将任务 3-1 作一些改动得到下面的任务 3-2。

任务 3-2　从键盘读入一系列整数，计算并输出前十个整数中正整数的和。

仅增加了“整数中”三个字，题意就有所不同。举一个具体例子：

x 为：1、－1、2、3、4、5、－5、6、7、8、9、10；

任务 3-1 是求前十个正整数的和，应该是：

$$sum=1+2+3+4+5+6+7+8+9+10=55;$$

任务 3-2 是求前十个整数中正整数的和，应该是：

$$sum=1+2+3+4+5+6+7+8=36。$$

由此可见，它们的算法是有区别的，从我们设定的数据来看，任务 3-1 是计算前 12 个整数中的 10 个正整数的和，累加的是正整数的个数；改动后是计算前 10 个整数中的正整数的和，累加的是整数的个数。

改动后描述算法的 N-S 图如图 3-3 所示。

例 3-4　求 5!（此问题的 N-S 盒图如图 3-4 所示）。

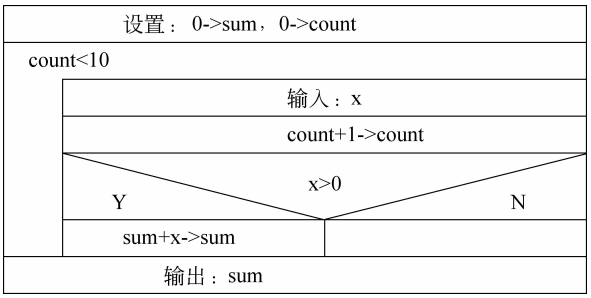


图 3-3 任务 3-2 的 N-S 图

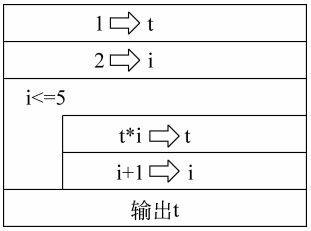


图 3-4 例 3-4 的 N-S 图

4. 伪代码描述

伪代码是指介于自然语言和计算机语言之间的一种代码，是帮助程序员制定算法的智能化语言，它不能在计算机上运行，但是使用起来比较灵活，无固定格式和规范，只要写出来自己或别人能看懂即可，由于它与计算机语言比较接近，因此易于转换为计算机程序。用伪代码表示的计算 5!的算法如下：

```
开始
    置 t 的初值为 1
    置 i 的初值为 2
    当 i<=5,执行下面操作
        使 t=t×i
        使 i=i+1
   (循环体到此结束)
    输出 t 的值
结束
```

也可以写成以下形式：

```
BEGIN(算法开始)
    1=>t
    2=>i
    while(i<=5)
    {
        t×i=>t
        i+1=>i
}
    print t
END(算法结束)
```

在本算法中采用了当型循环(第 3 行到第 7 行是一个当型循环)，while 意思为“当”，它表示当 i＜＝5 时执行循环体(大括弧中两行)的操作。

算法并不涉及具体的编程语言，算法确定之后就可以用任何一门编程语言来编写程序了。

3.3　顺序结构的程序

各执行语句之间存在一定的关系。最简单的一种关系就是：从上到下顺序执行各语句。即先执行第 1 个语句，再执行第 2 个语句，再执行第 3 个语句……直到最后一个语句。这样编写的程序，就是顺序结构的程序。

顺序结构是最简单的 C++ 语言程序结构，也是 C++ 语言程序中最常用的程序结构，其特点是完全按照语句出现的先后次序执行程序。在日常生活中，需要“按部就班、依次进行顺序处理和操作”的问题随处可见。在 C++ 语言中，像赋值操作和输入输出操作等都属于顺序结构，它主要由表达式语句组成。

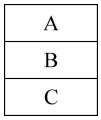


图 3-5　顺序结构的 N-S 图

用 N-S 图表示的顺序结构如图 3-5 所示，表示先执行 A 操作，再执行 B 操作，最后执行 C 操作，三者是顺序执行的关系。

3.4　分支选择结构与 if 语句

任务 3-3　输入一个年份值(year)，编程判断这一年是否为闰年。

闰年的特征是 2 月份有 29 天，如 2004 年、2000 年是闰年，2005 年、2100 年不是闰年。2004 可以被 4 整除但不能被 100 整除，2000 可以被 400 整除，2005 不能被 4 整除，2100 虽可以被 4 整除却又可以被 100 整除。因此，通过对以上特例的分析，不难得出判断闰年的条件应是符合下面两者之一：

① 能被 4 整除，但不能被 100 整除；

② 能被 400 整除。

也就是说，判断 year 年是否是闰年的条件应该是 year%4 等于零与 year%100 不等于零要同时满足，或是 year%400 等于零也可以。

这里存在两个问题。

(1) 如何编程来实现判断？这就涉及关系运算和条件运算。

(2) “与”和“或”如何用运算符来描述？这就涉及逻辑运算。

在 C++ 中条件运算除了可以用条件运算符和条件表达式之外，还可以用 if 语句来完成。if 语句是用来判定所给定的条件是否满足，根据判定的结果(真或假)决定执行给出的两种操作之一。

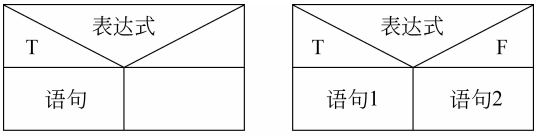
C++ 提供了 2 种基本形式的 if 语句。

(1) if(表达式)语句

例如：

```
if(x>y) cout<<x<<endl;
```

这种 if 语句的执行过程如图 3-6(a)所示。



(a) 单分支if语句　　(b) 双分支if语句

图 3-6　两种 if 语句的执行过程

(2) if(表达式)语句 1

　　else 语句 2

例如：

```
if(x>y)cout<<x;
else cout<<y;
```

这种 if 语句的执行过程如图 3-6(b)所示。

说明：

(1) 从图 3-6(a)和图 3-6(b)可以看出：两种形式的 if 语句都是由一个入口进来，经过对“表达式”的判断，分别执行相应的语句，最后归到一个共同的出口。这种形式的程序结构称为分支选择结构。在 C++ 中 if 语句是实现分支选择结构的主要语句。

(2) 两种形式的 if 语句中在 if 后面都有一个用括号括起来的表达式，它是程序编写者要求程序判断的“条件”，一般是关系表达式或逻辑表达式。例如：

```
if(a==b && x==y) cout<<"a=b,x=y";
```

在执行 if 语句时先对表达式求解，若表达式的值为非 0，按“真”处理，执行 cout 语句；若表达式的值为 0，按“假”处理，不执行 cout 语句。假如有以下 if 语句：

```
if(3)cout<<"ok.";
```

是合法的，执行结果输出“ok”，因为表达式的值为 3，按“非 0”即“真”处理。

(3) 第 2 种形式的 if 语句中，在每个 else 前面有一分号，整个语句结束处有一分号。例如：

```
if(x>0)
    cout<<x;                    //此行末尾有一分号
else
    cout<<-x;                   //此行末尾有一分号
```

这是由于分号是 C++ 语句中不可缺少的部分，这个分号是 if 语句中的内嵌语句所要求的。如果无此分号，则出现语法错误。但应注意，不要误认为上面是两个语句(if 语句和 else 语句)，它们都属于同一个 if 语句，可以在 if 语句中内嵌其他语句。else 是 if 语句中的子句，不能作为独立的语句单独使用，必须与 if 配对使用。

在 C++ 语言中“如果”用 if 表示，“否则”用 else 表示。

如前所述，任务 3-4 提出的判断闰年的条件可以用一个逻辑表达式来表示：

```
((year %4==0 && year %100!=0)|| year %400==0)
```

当给定 year 为某一整数值时，如果上述表达式值为真(1)，则 year 为闰年；否则 year 为非闰年。

请注意以上表达式中不同运算符的运算优先次序由高到低应为!、%、!=、==、&&、||，因为它们的优先级分别为 2、3、6、7、11、12。

完成任务 3-3 的源程序如下：

```
#include<iostream.h>
void main()
{
    int year;
    cout<<"请输入年份:"<<endl;
    cin>>year;
    if(year%4==0 && year%100!=0||year%400==0) cout<<year<<"年是闰年"<<endl;
    else cout<<year<<"年不是闰年"<<endl;
}
```

程序运行结果为：

```
请输入年份:
2008
2008年是闰年
```

再运行一次，程序运行结果为：

```
请输入年份:
2100
2100年不是闰年
```

3.5　if 语句的嵌套

在 if 语句中又包含一个或多个 if 语句称为 if 语句的嵌套。一般形式是在 else 子句中嵌套。

if 语句嵌套的格式为：

```
if(表达式 1) 语句 1
else if(表达式 2) 语句 2
    else if(表达式 3) 语句 3
        ⋮
        else 语句 n
```

任务 3-4　用 if 语句嵌套将用户输入的整数分数转换成等级，分数与等级对应关系如下。

90～100(含 90)	优秀
80～89(含 80)	良好
70～79(含 70)	一般