第 3 章　程序的控制结构

本章导引

利用计算机程序来解决实际问题大致需要经历分析问题、选择解决方案、编写程序、调试程序及测试程序等几个重要的阶段。其中，解决方案是指为解决问题所采用的基本方法和操作步骤，人们常把它称为计算机算法，在程序设计的整个过程中，算法设计的正确与否直接决定着程序的正确性，算法质量的优劣直接影响着程序的最终质量，因此，要想完成一个优秀的程序设计，设计一个优秀的算法是不容置疑的基本前提。本章将介绍算法设计的概念及其表示方法。

在选择好解决方案(即设计好算法)后，就进入编写程序阶段，此时需要利用 C 语言提供的各种语句来实现算法中描述的每项操作步骤。其中，使用到控制语句的结构化，即将顺序结构、选择结构和循环结构作为程序流程的基本控制结构，且每种结构均只有一个入口、一个出口。本章也将介绍 C 语言对上述三种控制结构的支持手段，并通过列举一些实例加深读者对它们的理解。

3.1　程序灵魂：算法

可以把“算法”理解为完成一件事情或解决一个问题而采取的方法和步骤。由此可知，算法这一概念早已融入我们的学习与生活中：如何求解一个方程、如何安排去某地旅游的路线或行程等都包含着某种算法。我们现在只讨论计算机算法，即计算机可以实现的算法。

按数据的处理方式，计算机算法可分为以下两种。

(1)数值运算：目的是求数值解，如求方程的根、求函数的定积分等。

(2)非数值运算：目前使用范围广泛，如办公自动化处理、图书情报检索等。

一般来说，不同的问题有不同的解决方法和步骤，而对同一问题也可能有不同的解决方法和步骤，也就是有不同的算法。算法有优劣，一般而言，应当选择简单的、运算步骤少的，运算快、内存开销小的算法。

3.1.1　算法的特性

1．有穷性

算法包含的操作步骤是有限的，每一步都应在合理的时间内完成。

2．确定性

算法中的每一步骤都应是唯一的和确定无误的，不允许有歧义性。例如，“输出成绩优秀的同学名单”就是有歧义的，“成绩优秀”的含义不明确。

3．有效性

算法中每一步骤都应是能有效地执行，且能得到确定的结果。例如，求一个负数的对数就是一个无效的步骤。

4．没有输入或有多个输入

有些算法不需要从外界输入数据，如计算 6!；而有的算法需从外界输入数据，如计算 n!，则需从键盘上输入 n 的具体值后才能进行计算。

5．有一个或多个输出

算法必须得到结果，没有结果的算法是毫无意义的。一个算法有一个或多个输出，以反映对输入数据加工后的结果。

3.1.2 算法的表示

为了描述一个算法，可以采用多种不同的方法，常用的有自然语言、传统流程图、N-S 结构化流程图、伪代码和计算机语言等。

1．用自然语言表示算法

所谓自然语言，即人们日常使用的语言，可以是汉语、英语、其他语言及其混合体。用自然语言表示通俗易懂，但文字冗长，容易出现歧义。用自然语言表示算法的含义不太严格，往往需要根据上下文才能判断其含义。因此，除那些很简单的问题外，一般不用自然语言描述算法。

【案例 3.1】 计算 1+2+3+…+100 的和。

方法 1：可以采用最原始方法：1+2，+3，+4，一直加到 100，加 99 次。将这一思路用自然语言描述为如下算法。

算法：

s1: 计算 1 + 2;
s2: 计算 s1 + 3;
s3: 计算 s2 + 4;
…
计算 100 以内自然数的和，需要 99 个步骤。
即 s99: s98 + 100。

这样的算法虽然正确，但太烦琐。特别是当累加的数很多时，就会非常麻烦。因此，必须找到一种更为简单的通用算法。

观察上述的算法发现，从第二个步骤起，每一次求和时所用被加数都是上一次求和的结

果。回忆前面学习过的变量中的值可以不断发生变化的特点，可以考虑使用一个变量(sum)既用来存放每一次的求和结果，又用来表示每一次求和时的被加数。这样，方法 1 中的 99 个步骤可以表示为：

s1: sum=sum+2；　(在这之前先把 sum 初始化为 1)

s2: sum=sum+3;

s3: sum=sum+4;

…

s99: sum=sum+100;

再次观察上述的算法发现，上述的 99 个式子除加数不一样外，其他都一样。并且，加数变化很有规律，从 2 开始，每次增加 1，直到 100。可以考虑再使用一个变量 i 来表示加数，那么上述的 99 个式子可以用同一种形式：sum=sum+i 来表示。也就是说，我们可以让 i 从 2 开始，不断地做 sum=sum+i 这一个相同的循环操作，直到 i 超过 100 为止。于是，可以得到改进的算法如下。

方法 2：用变量 sum 和 i 分别表示两个加数，和也用变量 sum 表示。用自然语言描述如下。

s1:　sum = 1;

s2:　i = 2;

s3:　若 i<=100，重复步骤 s4～s5，否则转去执行 s6;

s4:　sum = sum + i;

s5:　i = i + 1;

s6:　输出 sum 的值。

上述算法是一个循环算法：s3 到 s5 组成一个循环，在实现算法时要反复多次执行 s3，s4，s5 等步骤，直到某一时刻，执行 s5 步骤时经过判断，加数 i 已超过规定的数值而不返回 s3 步骤为止。此时，算法结束，变量 sum 的值就是所求结果。显然，此算法比前一种算法简练。实际上，此算法还具有通用性、灵活性，因为它只需要稍做修改即可适用于若干个有规律的数的求和。例如求 1+3+5+7+…+99 等。

因为计算机是高速运算的自动机器，实现循环是计算机特别擅长的工作之一，因此上述算法不仅正确，而且是计算机能实现的较好算法。

多学一点

方法 2 中的步骤 s1 和 s2 也可以修改为 s1:sum=0; s2: i=1;结果相同。本算法中，sum 用来存放每次累加后的和，经常称为累加器，并且一般初始化为 0。

2．用传统流程图表示算法

传统流程图采用一组规定的图形符号、流程线和文字说明来表示各种操作算法。美国国家标准协会(American National Standard Institute，ANSI)规定了一些常用的流程图符号，已为世界各国程序工作者普遍采用，如表 3-1 所示。

表 3-1　传统流程图常用的图形符号

符号	名称	用途
	起止框	用于描述控制流程的开始和结束：开始框内标注“开始”字样，结束框内标注“结束”字样
	输入/输出框	用于表示数据的输入和输出：框内标明输入/输出的变量
	处理框	用于描述数据加工和处理：常采用文字加符号来表示计算公式和赋值操作
	判断框	用于描述条件判断和转移关系：框内描述条件关系，两个流出边分别标注 Yes/No、Y/N、True/False 或“真/假”，表示条件成立或不成立时的转移关系
	调用框	用于描述过程调用或模块调用：框内标注函数或模块名
	流程线	用于连接两个图形框：箭头描述处理过程的转移方向
	连接框	用于描述多个流程图的连接：应附加文字标识连接关系

【案例 3.2】 用传统流程图描述【案例 3.1】的算法，如图 3-1 所示。

用传统流程图表示算法直观形象，易于理解，能够比较清晰地表达各种处理之间的逻辑关系，是表示算法的较好的工具。但由于对流程线的使用没有严格限制，易造成流程的随意转移，不能保证是结构化的，从而难以阅读和修改。

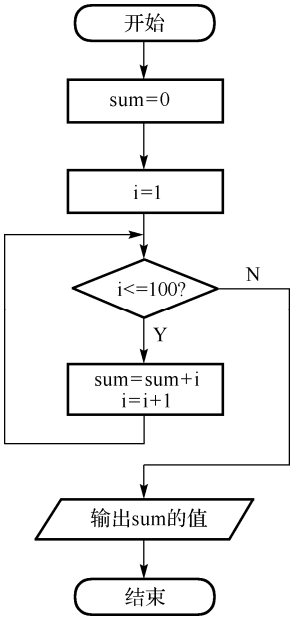


图 3-1　传统流程图

3．用 N-S 流程图表示算法

1973 年，针对传统流程图存在的问题，美国学者 I.Nassi 和 B.Shneiderman 提出了一种新的结构化流程图形式。在这种流程图中，完全去掉了带箭头的流程线。全部算法写在一个矩形框内，在该框内还可以包含其他的从属于它的框，或者说，由一些基本的框组成一个大的框。该矩形框以三种基本结构(顺序、选择、循环)描述符号为基础复合而成，这种流程图又称 N-S 流程图。

【案例 3.3】 用 N-S 流程图描述【案例 3.1】的算法，如图 3-2 所示。

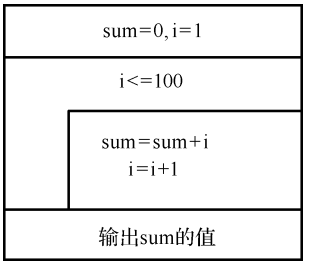


图 3-2　N-S 流程图

用 N-S 流程图表示算法既比自然语言描述直观、形象、易于理解，又比传统流程图紧凑易画。尤其是它废除了流程线，整个算法结构是由各个基本结构按顺序组成的，N-S 流程图中的上下顺序就是执行时的顺序。用 N-S 流程图表示的算法都是结构化的算法，因为它不可能出现流程无规律的跳转，而只能自上而下地顺序执行。

4．用伪代码表示算法

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，特别适用于设计过程中需要反复修改时的流程描述。

【案例 3.4】 用伪代码描述【案例 3.1】的算法。

```
begin
    sum←0;
    i←1;
    while  i≤100
        sum←sum+i;
        i←i+1
    print  sum
end
```


5．用计算机语言表示算法

计算机是无法识别流程图和伪代码的。只有用计算机语言编写的程序，经编译成目标程序后，才能被计算机执行。因此，在用流程图或伪代码描述出一个算法后，还要将它转换成计算机语言程序。用计算机语言表示算法必须严格遵循所用语言的语法规则，这是和伪代码不同的。

【案例 3.5】 用 C 语言描述【案例 3.1】的算法。

```
#include <stdio.h>
int main()
{
    int i, sum;
    sum=0;
    i=1;
    while (i<=100)
    {
        sum=sum+i;
        i++;
    }
    printf("sum=%d", sum);
    return 0;
}
```

多学一点

写出了 C 程序，仍然只是描述了算法，并未实现算法，只有运行程序才是实现算法。应该说，用计算机语言表示的算法才是计算机能够执行的算法。

3.2 流水作业：顺序结构

前面已经介绍了算法的概念，那么如何在计算机语言中实现设计好的算法呢？主要是通过各种语言所提供的语句来实现。和其他高级语言一样，C 语言的语句用来向计算机系统发出操作指令。一个语句经编译后产生若干条机器指令。一个为实现特定目的的程序应当包含若干条语句。

C 语言中的语句可以分为以下五类。

1．函数调用语句

该语句由函数名、实际参数加上分号“;”组成。其一般形式为：

```
函数名(实际参数表);
```

执行函数语句就是调用函数体并把实际参数赋予函数定义中的形式参数，然后执行被调函数体中的语句，求取函数值(在 4.1.2 节中详细介绍)。

例如：

```
printf("Hello!"); /*调用库函数，输出字符串*/
```


2．表达式语句

表达式语句由表达式加上分号“;”组成。其一般形式为：

```
表达式;
```

执行表达式语句就是计算表达式的值。

例如：

x=y+z;　　赋值语句。

y+z;　　加法运算语句，但计算结果不能保留，无实际意义。

i++;　　自增 1 语句，i 值增 1。

其中，赋值语句是程序中使用最多的语句之一。表达式能构成语句是 C 语言的一个特色。其实“函数调用语句”也属于表达式语句，因为函数调用也是表达式的一种，只是为了便于理解和使用，我们把“函数调用语句”和“表达式语句”分开来说明。

3．控制语句

控制语句用于控制程序的流程，以实现程序的各种结构方式。C 语言有 9 种控制语句，可以分成以下三类。

(1)条件判断语句：if 语句、switch 语句。

(2)循环执行语句：while 语句、do while 语句、for 语句。

(3)转向语句：break 语句、continue 语句、goto 语句、return 语句。

4．复合语句

把多个逻辑相关的语句用花括号{ }括起来组成一条复合语句。复合语句在逻辑上形成一个整体，在程序中应把它看成单条语句，而不是多条语句。

例如：

```
{
    x=y+z;
    a=b+c;
    printf("%d%d",x,a);
}
```

是一条复合语句。

复合语句内的各条语句都必须以分号“;”结尾，在花括号“}”外不能再加分号。

5．空语句

只有由分号“;”组成的语句称为空语句。空语句是什么也不执行的语句。在程序中，空语句经常被用来作为空循环体。

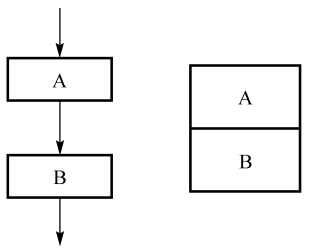
例如：

```
while(getchar()!='\n')
;
```


本语句的功能是，只要从键盘输入的字符不是回车符，则重新输入。

这里的循环体为空语句。有时，空语句也可用来作为流程的转向点或自顶向下程序设计时用在那些未完成的模块中，留待以后对模块逐步求精实现时再进行扩充。

有了上述的这五类语句，就可以组成程序的三种基本控制结构——顺序结构、分支结构、循环结构，从而保证计算机能按照确定的步骤来解决问题，即实现既定的算法。1966 年，C.Bohmt 和 G.Jacopini 首先证明了只用这三种基本结构就可以实现任何“单入口、单出口”的复杂问题，而且编写出来的程序既清晰可读又便于理解，所以提倡使用这三种结构编写程序，并称这样的程序设计为结构化程序设计。下面介绍在结构化程序中最简单、最基本的结构——顺序结构。



(a) 传统流程图　　(b) N-S流程图

图 3-3　顺序结构

顺序结构的特点是，完全按照语句出现的先后次序执行程序。其传统流程图和 N-S 流程图如图 3-3 所示。

任何计算问题的答案都是按指定顺序执行一系列动作的结果，在许多场合顺序与动作同样重要，错误的执行顺序将得不到问题的正确答案。

从程序的整体来看，程序的语句是按顺序执行的，构成了顺序结构，尽管在局部(某些程序段)并不按顺序执行语句，这个过程称为“控制的转移”，它涉及了另外两类程序的控制结构，即选择(分支)结构和循环结构。由此可见，顺序结构也是一种最基本的结构。

【案例 3.6】 输入三角形的三条边长，求三角形面积。为简单起见，设输入的三条线段能构成三角形的三条边。

分析：已知三角形的三条边长为 a、b、c，则该三角形的面积公式如下。

$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

其中，$s = (a+b+c)/2$。

源程序如下：

```
#include <stdio.h>
#include <math.h>                    /*包含使用数学库函数所对应的头文件*/
int main()
{
    float a, b, c, s, area;
    printf("Please input a, b, c:");
    scanf("%f, %f, %f", &a, &b, &c);
    s=1.0/2*(a+b+c);
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("a=%7.2f, b=%7.2f, c=%7.2f, s=%7.2f\n", a, b, c, s);
    printf("area=%7.2f\n", area);
    return 0;
}
```

运行结果如图 3-4 所示。

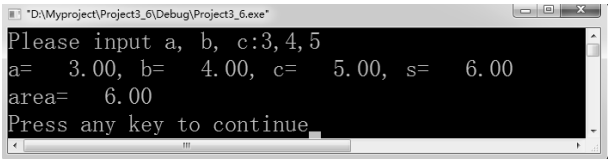


图 3-4 【案例 3.6】运行结果

注意

语句 s=1.0/2*(a+b+c);中的 1.0 可以改成 1 吗？为什么？

多学一点

如何保证所输入的数据能满足本例中的假设呢？一个有效的方法是对所输入的数据进行合法性的检验，而合法性的检验就是判断某些条件是否成立，此时需要用到选择结构中的条件语句。

3.3 择优录取：选择结构

如前所述，在许多实际问题的程序设计中，根据输入数据和中间结果的不同情况需要选择不同的语句组执行。在这种情况下，必须根据某个变量或表达式的值做出判断，以决定执行哪些语句和跳过哪些语句不执行，这就是选择结构(分支结构)。计算机的一个很重要的特征是具有逻辑判断能力，能够灵活处理问题。选择结构中主要用到以下两个语句。

(1)条件语句：根据给定的条件(逻辑值)进行判断，决定执行某个分支和程序段。

(2)开关分支语句：根据给定表达式的值进行判断，然后决定执行多路分支中的一支。

执行条件语句时要进行判断，经常需要使用关系运算符、逻辑运算符及相应的表达式。对于简单的判断条件可用关系表达式来表示，对于较为复杂的条件可用逻辑表达式来表示。

对于一个条件判断的结果只可能有两种：“真”(对应条件成立)或者“假”(对应条件不成立)。而标准 C(C89)没有布尔数据类型，那么用非 0 值表示“真”，用 0 值来表示“假”。

3.3.1 关系运算符和关系表达式

C 语言提供的 6 种关系运算符在所有运算符中的优先级等如表 3-2 所示。

表 3-2 关系运算符

优先级	运算符	名称或含义	使用形式	结合方向	说明
6	>	大于	表达式>表达式	从左到右	双目运算符
	>=	大于等于	表达式>=表达式		双目运算符
	<	小于	表达式<表达式		双目运算符
	<=	小于等于	表达式<=表达式		双目运算符
7	==	等于	表达式==表达式	从左到右	双目运算符
	!=	不等于	表达式!=表达式		双目运算符

注意

(1)关系运算符等于是“==”，而不是“=”(赋值运算符)。

(2)>=、<=、!= 不能写成数学运算符≥、≤、≠。

用关系运算符将两个操作数连接起来组成的表达式，称为关系表达式。关系表达式的一般形式为：

```
表达式 关系运算符  表达式
```

每一个表达式都有一个运算结果，也称为表达式的值。关系表达式的值有两个：“真”(对应关系表达式成立)和“假”(对应关系表达式不成立)，用“1”和“0”表示。

【案例 3.7】 关系运算符和表达式示例。

源程序如下：

```
#include <stdio.h>
int main()
{
    char c='k';
    int i=0,j=1,k=2,m=3;
    float x=0.2,y=0.3;
    printf("(1)%d\n",'a'+5<c);
    printf("(2)%d\n",m=k<j);
    printf("(3)%d\n",k==j==i+5);
    printf("(4)%d\n",x*y==0.06);
    printf("(5)%d\n",-1<=i<=1);
```

```
        return 0;
    }
```

运行结果如图 3-5 所示。

图 3-5 【案例 3.7】运行结果

说明：

与运行情况(1)相对应表达式中的字符变量是以它对应的 ASCII 码参与运算的；与运行情况(2)相对应表达式 m=k<j 中先进行优先级较高的“<”，其结果为 0 赋值给 m 并输出；与运行情况(3)相对应表达式 k==j==i+5 中 j 左右两边的运算符相同(都是“==”)，根据运算符的左结合性，先计算 k==j，该式不成立，其值为 0，再计算 0==i+5，也不成立，故表达式的值为 0。

注意

与运行情况(4)相对应表达式 x*y==0.06，由于浮点数通常不能够在计算机中精确表示；并且在运算过程中，浮点表达式的相对误差是累积的，从而导致以上表达式左右两边不相等。因此应当对两个浮点数 f1、f2(float 或者 double 类型)比较时慎用“==”和“!=”，最好是比较它们差的绝对值是否小于一个非常小的数，如量级在 10^{-7} 以内的数(FLT_EPSILON)，即 fabs(f1-f2)< FLT_EPSILON。

与运行情况(5)相对应关系表达式-1<=i<=1，此时并不是数学表达式来表示 i 落在区间[-1,1]内，如果要表示 i 落在区间[-1,1]内则需要用到下一节介绍的逻辑与运算符 &&，写成：-1<=i && i<=1。

多学一点

关系运算符只能用来比较原子数据值——不能再分解成更小的数据，如整型、浮点型和字符型数据就可以看成原子数据，因为它们不能分解为更小的数据。而字符串就不是原子数据，不能使用关系运算符来进行比较大小，需要使用字符串比较函数 strcmp()来完成。

3.3.2 逻辑运算符和逻辑表达式

C 语言提供的 3 种逻辑运算符在所有运算符中的优先级等如表 3-3 所示。

表 3-3　逻辑运算符

优先级	运算符	名称或含义	使用形式	结合方向	说明
2	!	逻辑非运算符	!表达式	从右到左	单目运算符
11	&&	逻辑与	表达式&&表达式	从左到右	双目运算符
12	||	逻辑或	表达式||表达式	从左到右	双目运算符

逻辑运算符的运算规则如下：

逻辑非——真变假，假变真。

逻辑与——两者都为真，结果才为真。

逻辑或——只要一个为真，结果就为真。

用逻辑运算符连接操作数组成的表达式称为逻辑表达式。逻辑表达式的值与关系表达式的值一样有两个：“真”和“假”，用“1”和“0”表示。但是，在需要判断一个表达式的值是否为真时并不局限于“1”，只要非 0 值就为真；而“0”仍为假。于是对两个表达式 a 和 b，其逻辑运算的“真值表”如表 3-4 所示。

表 3-4　逻辑运算的“真值表”

a	b	! a	a && b	a || b
非 0(真)	非 0(真)	0	1	1
非 0(真)	0 (假)	0	0	1
0 (假)	非 0(真)	1	0	1
0 (假)	0 (假)	1	0	0

【案例 3.8】 逻辑运算符和表达式示例。

源程序如下：

```
#include <stdio.h>
int main()
{
    char ch='A';
    int i=0,j=1,k=2,year=2016;
    float x=0.2;
    printf("(1)%d,%d\n",i&&j++||!k,j);
    printf("(2)%d,%d\n",(x||i++)&&j==2,i);
    printf("(3)%d,%d\n", 'a'<=ch<='z', ch>='a' && ch<='z');
    printf("(4)%d,%d\n",(year%4==0)&&(year%100!=0)
                        ,(year%4==0)&&(year%100));
    return 0;
}
```

运行结果如图 3-6 所示。

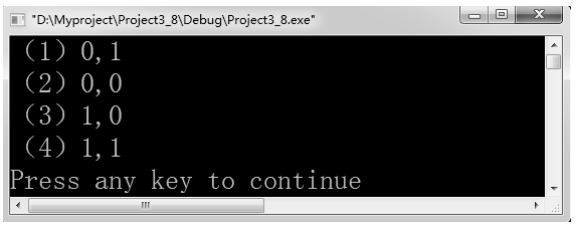


图 3-6 【案例 3.8】运行结果

说明：

与运行情况(1)相对应表达式 i&&j++||!k 中的“||”运算符优先级最低，因此这是一个逻辑表达式，由于 i=0 和 j++进行逻辑与运算的结果为假，再与!k(k=2 为真，!k 为假)进行逻辑或运算的最终结果为假。请注意(1)中后面所输出的变量 j 的值仍为 1，也就意味着前面表达式在运算时 j++并没有执行，这是 C 语言中运算符&&和||都具有的“短路”特性，即如果上述两个运算符的表达式的值可由先计算的左操作数的值单独推导出来，那么将不再计算右操作数的值，如 i&&j++中由于 i=0 即可决定整个运算结果为假，就不需要再做 j++运算。类似情况出现在运行情况(2)的(x||i++)运算中；运行情况(3)中的表达式'a'<=ch<='z'是一个关系表达式，而 ch>='a' && ch<='z'则是一个逻辑表达式，用来判断字符 ch 是否为一个小写字母。

多学一点

在运行情况(4)中对应的两个表达式是等价的，都是用来判断某一个年份是否满足闰年的条件之一，即能否被 4 整除但不会被 100 整除。请注意到当 year=2016 时，算术表达式 year%100 的结果为 16，非 0 为真值；而此时关系表达式 year%100!=0 的结果也是为真值。以上两个表达式之所以等价，主要是因为 C 语言中采用非 0 与 0 作为真假值的判断方法。这种方法也给程序中条件的判断带来灵活性，任何形式的表达式都可以充当判断的条件。

3.3.3 条件语句(if 语句)

if 语句有以下 3 种表现形式。

1. if 语句的简单形式(单分支情况)

其执行过程如图 3-7(a)、图 3-7(b)所示，一般形式如下：

```
if(表达式) 语句;
```

例如：

```
if(x==y) printf("a equal to b");
```

注意

表达式(x==y)能否改为(x=y)？为什么？

说明：

(1)if 后的表达式一定要用圆括号括起来。

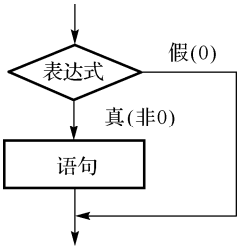
(2)if 后的表达式可以是任意类型的表达式，除常见的关系表达式或逻辑表达式外，也允许是其他类型的数据(包括整型、浮点型、字符型等)。

以上两点说明也适合后面的两种 if 语句形式。

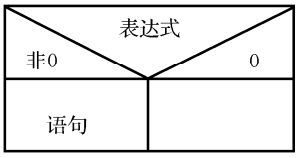
2. if 语句的标准形式(双分支情况)

其执行过程如图 3-8(a)、图 3-8(b)所示，一般形式如下：

```
if(表达式) 语句1 ;
else     语句2 ;
```



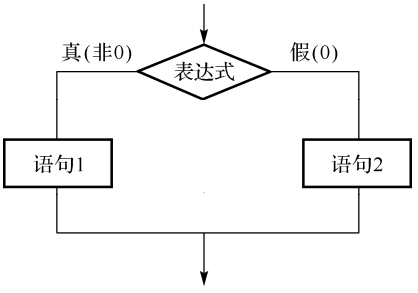
(a) 传统流程图表示



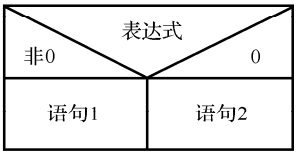
(b) N-S 流程图表示

图 3-7　单分支 if 语句

例如：if(x>y) max=x;
　　else　max=y;



(a) 传统流程图表示



(b) N-S 流程图表示

图 3-8　双分支 if 语句

注意

(1) if 和 else 都属于同一个 if 语句。else 子句不能作为语句单独使用，它必须是 if 语句的一部分，与 if 配对使用。

(2) 在 if 和 else 后面若跟单条语句，则应加分号“;”，若是复合语句，“{}”后面不必另加“;”。

多学一点

在 if 和 else 后面可以只含一个内嵌的操作语句(如上例)，也可以有多个操作语句，此时用花括号“{ }”将几个语句括起来成为一个复合语句。例如：

```
if (a>b) { t=a; a=b; b=t;}
```

【案例 3.9】 输入三角形的三条边长，求三角形面积。

分析：与【案例 3.6】相比，此例去掉假设前提：“设输入的三条线段能构成三角形的三条边”。因此，本题在输入三条线段后要加入条件判断，验证其能否构成三角形的三条边，若可以则计算并输出面积，否则输出提示信息。

源程序如下：

```
#include <stdio.h>
#include <math.h>
int main()
{
    float a, b, c, s, area;
    printf("Please input a, b, c:");
    scanf("%f, %f, %f", &a, &b, &c);
    if (a+b>c && a+c>b && b+c>a)     /*判断三条线段能否构成三角形的三条边*/
    {
        s=1.0/2*(a+b+c);
        area=sqrt(s*(s-a)*(s-b)*(s-c));
        printf("area=%7.2f\n", area);
    }
    else
        printf("it is not a trilateral\n");
    return 0;
}
```

运行结果如图 3-9、图 3-10 所示。

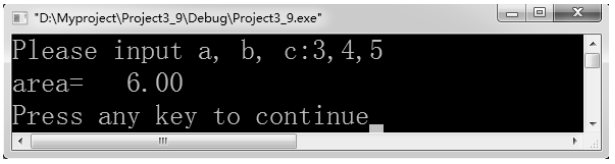


图 3-9 【案例 3.9】运行结果(1)

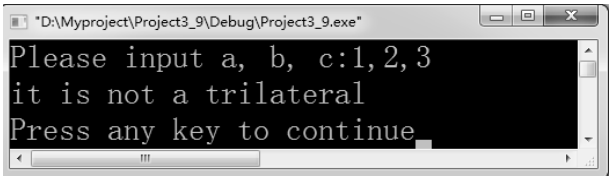


图 3-10 【案例 3.9】运行结果(2)

注意

if 后的“{ }”能否去掉？去掉表示什么意思？

3．条件运算符与条件表达式

条件运算符是 C 语言中唯一的一个三目运算符，它在所有运算符中的优先级等如表 3-5 所示。

表 3-5 条件运算符

优先级	运算符	名称或含义	使用形式	结合方向	说明
13	?:	条件运算符	表达式 1? 表达式 2: 表达式 3	从右到左	三目运算符

由条件运算符组成条件表达式的一般形式为：

```
表达式 1? 表达式 2:表达式 3
```

其求值规则为：如果表达式 1 的值为真，则以表达式 2 的值作为整个条件表达式的值，否则以表达式 3 的值作为整个条件表达式的值。例如：

```
max=(a>b)?a:b;
```

执行该语句的语义是：如 a>b 为真，则把 a 赋予 max，否则把 b 赋予 max。它可以用来替换以下双分支的条件语句，更为简单、直观。

```
if (a>b)  max=a;
else  max=b;
```

多学一点

条件运算符的结合方向是自右至左。例如：

```
a>b?a:c>d?c:d 相当于 a>b?a:(c>d?c:d)
```

这也就是条件表达式嵌套的情形，即其中的表达式 3 又是一个条件表达式。

4．if 语句的嵌套形式(多分支情况)

在 if 语句中又包含一个或多个 if 语句称为 if 语句的嵌套。这种情况主要用来解决多分支情况，if 语句的嵌套有多种表现形式：

1)比较常用的形式

比较常用的形式如下：

```
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 if (表达式 m) 语句 m
else   语句 n
```

其语义是：依次判断表达式的值，当出现某个值为真时，则执行其对应的语句，然后跳到整个 if 语句之外继续执行程序。如果所有的表达式均为假，则执行语句 n，然后继续执行后续程序。if-else-if 语句的执行过程如图 3-11 所示。

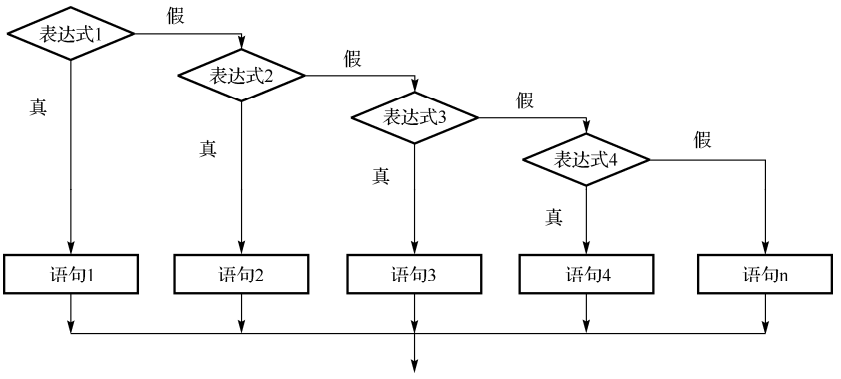


图 3-11　if-else-if 语句的执行过程

【案例 3.10】 已知学生的百分制成绩，编写程序按百分制分数进行分段评定，给出相应的等级：分数大于等于 90，则评定为'A'；分数在 80～89 之间，则评定为'B'；分数在 70～79 之间，则评定为'C'；分数在 60～69 之间，则评定为'D'；分数小于 60，则评定为'E'。

分析：这是一个根据百分制分数进行分段定级的多分支选择问题，可利用上面介绍的 if 语句的嵌套来解决。

源程序如下：

```
#include <stdio.h>
int main()
{
    float score;
    char grade;
    printf("Please enter scores:");
    scanf("%f", &score);
    if(score<0 || score>100)            /*对输入数据的合法性进行检查*/
        printf("Input error!\n");
    else
    {
        if(score>=90)   grade='A';
        else if(score>=80)   grade='B';
        else if(score>=70)   grade='C';
        else if(score>=60)   grade='D';
        else  grade='E';
        printf("%5.1f--%c\n", score, grade);
    }
    return 0;
}
```

运行结果如图 3-12、图 3-13 所示。

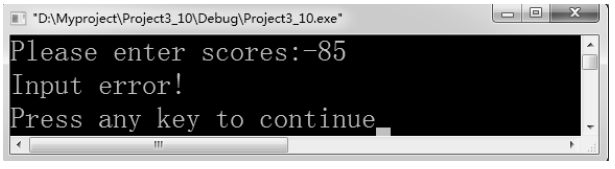


图 3-12 【案例 3.10】运行结果(1)

图 3-13 【案例 3.10】运行结果(2)

2) 其他形式

在嵌套内的 if 语句可能又是 if-else 型的，这将会出现多个 if 和多个 else 重叠的情况，这时要特别注意 if 和 else 的配对问题。

例如：

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

其中的 else 究竟是与哪一个 if 配对呢?

应该理解为：

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

还是应理解为：

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


为了避免这种二义性，C 语言规定，从最内层开始，else 总是与它上面最近的(未曾配对的)if 配对，因此对上述例子应按前一种情况理解。如果 if 与 else 的数目不一样，为实现程序设计者的意图，可以加花括号来确定配对关系。例如：

```
if()
{ if() 语句 1}
else
    语句 2
```

这时，if 限定了内嵌 if 语句的范围，因此 else 与第一个 if 配对。

【案例 3.11】 求三个数的最大数。

分析：假设所求的三个数用变量 x、y、z 表示，三个数的最大数用变量 m 表示。本题有很多种的求法与写法，下面列出其中几种，请读者进行比较理解。

方法 1：采用单分支形式。

```
if(x>y && x>z)   m=x;
if(y>x && y>z)   m=y;
if(z>y && z>x)   m=z;
```

方法 2：采用单分支形式。

```
m=x;
if(y>m) m=y;
if(z>m) m=z;
```

方法 3：采用双分支形式。

```
if(x>y) m=x;
else m=y;
if(m>z) m=m;
else m=z;
```

方法 4：采用条件表达式。

```
m=x>y ? x: y;
m=m>z ? m: z;
```

方法 5：采用 if 语句的嵌套。

```
if(x>y)
    if (x>z) m=x;
    else m=z;
else
    if(y>z) m=y;
    else m=z;
```

方法 6：采用 if 语句的嵌套。

```
m=x;
if (z>y)
    {if(z>x) m=z;}
else
    if(y>x) m=y;
```

注意

方法 6 中的“{ }”若去掉，结果就不正确了，如 x=2,y=3,z=1，请读者自己分析。

3.3.4 开关语句(switch 语句)

C 语言还提供了另一种用于多分支选择的 switch 语句，以代替嵌套的 if 语句，简化程序的设计。switch 语句又称为开关语句，它允许程序根据表达式的计算结果在多个分支中进行选择，常用于各种分类、菜单等程序的设计。它的一般形式如下：

```
switch(表达式)
{
    case 常量表达式 1：语句序列 1
    case 常量表达式 2：语句序列 2
    …
    case 常量表达式 n：语句序列 n
    default        ：语句序列 n+1
}
```

其语义是：计算充当开关角色的表达式的值，并逐个与其后的常量表达式值相比较，当表达式的值与某个常量表达式的值相等时，即按顺序执行此 case 后的所有语句，包括后续 case，

而不再进行判断，直到遇到 break 或右花括号“}”(整个语句执行完毕)为止。如表达式的值与所有 case 后的常量表达式均不相同时，则执行 default 后的语句。

【案例 3.12】 用 switch 语句改写【案例 3.10】。已知学生的百分制成绩，编写程序按百分制分数进行分段评定，给出相应的等级：分数大于等于 90，则评定为'A'；分数在 80～89 之间，则评定为'B'；分数在 70～79 之间，则评定为'C'；分数在 60～69 之间，则评定为'D'；分数小于 60，则评定为'E'。

分析：使用 switch 语句时要注意到 case 后的常量表达式最终结果只能是某一个值(点)，不能表示区间(范围)。因此在本例中要根据分数进行分段定级，必须设法完成从区间到点的转化。【案例 3.12】传统流程图如图 3-14 所示。

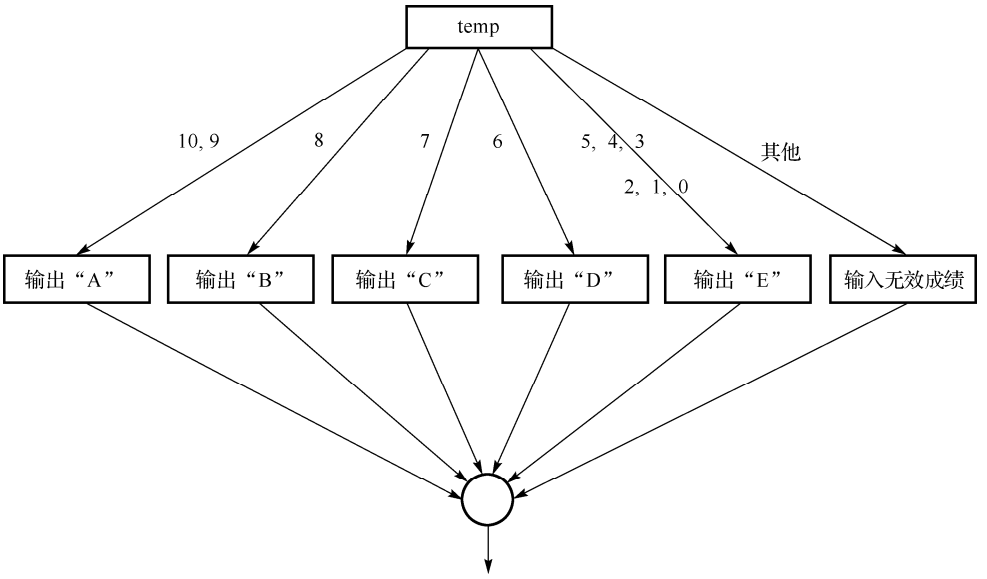


图 3-14 【案例 3.12】传统流程图

源程序如下：

```
#include <stdio.h>
int main()
{
    float score;
    int temp;
    printf("Please input score:");
    scanf("%f", &score);
    if(score<0 || score>100)
        temp=-1;
    else  temp=(int)score/10;     /*采用整除方法，将区间取值转化到点上*/
    switch(temp)
    {
       case 10:
       case  9:  printf("A\n"); break;
       case  8:  printf("B\n"); break;
       case  7:  printf("C\n"); break;
```

```
            case  6:  printf("D\n"); break;
            case  5:
            case  4:
            case  3:
            case  2:
            case  1:
            case  0:  printf("E\n"); break;
            default:  printf("Input invalid score\n");   /*处理非法数据*/
        }
        return 0;
    }
```

运行结果如图 3-15、图 3-16 所示。

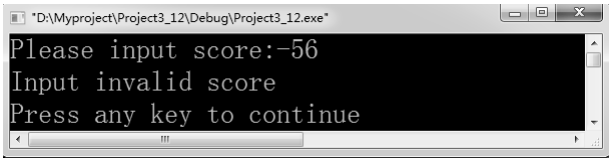


图 3-15 【案例 3.12】运行结果(1)

图 3-16 【案例 3.12】运行结果(2)

说明：在使用 switch 语句时还应注意以下几点。

(1)switch 后面括号内的“表达式”的值一般为整型、字符型或枚举型数据。每个 case 后的“常量表达式”的类型应该与 switch 后面括号内的“表达式”的类型一致。对其他类型，原来的 C 标准是不允许的，而新的 ANSI 标准允许上述表达式和 case 常量表达式为任何类型。

(2)由于每个 case 后的常量只是起到语句标号作用，所以每一个 case 的常量表达式的值必须互不相同，否则就会出现互相矛盾的现象(对表达式的同一个值，有两种或多种执行方案)。

(3)各 case 和 default 子句的先后顺序可以变动，而不会影响程序执行结果。但从程序的执行效率角度考虑，一般将发生频率高的情况放在前面。

(4)在 case 后，允许有多个语句，并且可以不用“{}”括起来。

(5)default 子句可以省略不用。

(6)多个 case 可以共用一组执行语句，例如：

```
    case  'A':
    case  'B':
    case  'C': printf(">60\n");break;
```

grade 的值为'A'、'B'或'C'时都执行同一组语句。

(7)switch 语句不同于 if 语句，switch 语句仅能判断一种逻辑关系，即看括号内的“表达式”的值和指定的常量值是否相等，是一种“单点判断”，而不能进行大于、小于某个值的