第 3 章　判断与决策——选择程序设计


电子教案

学习目标：

- 掌握用流程图描述算法的方法。
- 理解关系运算、逻辑运算和条件运算。
- 掌握几种形式的逻辑判断条件。
- 能用选择结构解决逻辑判断与决策问题。
- 理解复合语句的概念。
- 熟悉现有各种运算的优先级。

通过第 2 章的入门学习，大家已经能够用计算机解决一些比较简单的问题了。请回顾第 2 章解决的问题的特点：先给定一些数据，然后按照某个公式计算出一些结果，最后把结果输出到屏幕上，告知用户。这个过程可以说是直线型的，很固定，每个步骤的先后顺序是固定不变的、依次进行的，在这个过程中不需要做任何判断，没有任何智能在里面，对应的程序结构是顺序结构。实际上计算机不仅能计算，按照公式计算，而且还能够有选择地、有判断地采取不同的计算方案，也就是计算机具有判断决策能力，能像人一样思考。本章要展现的就是计算机是如何表示判断条件，如何做判断决策的。

本章要解决的问题是：

- 让成绩合格的学生通过；
- 按成绩把学生分成两组；
- 按成绩把学生分成多组；
- 判断某年是否为闰年。


视频

3.1　让成绩合格的学生通过

问题描述：

假设有一个计算机打字训练教室，刚入学的大一学生都要到这个训练教室练习打字。计算机自动考核，成绩在 60 分以上视为合格。训练教室的门口有一个计算机控制的栏杆，它是一个“智能栏杆”，知道每一个参加训练的学生的当前训练成绩，因此当有人走进它时，它会获取学号，并要求输入成绩，然后计算机会检查输入的成绩是否属实，如果属实并大于或等于 60，栏杆将自动打开，允许通过。可想而知，如果成绩小于 60，智能栏杆会是什么样子。注意，键盘输入成绩的时候必须要诚实，别忘了它是智能栏杆，不然就会有不良的记录。写一个程序模拟这个“智能栏杆”。

输入样例 1：

```
70
```

输出样例 1：

```
good you passed!
```

输入样例 2：

```
50
…
```

输出样例 2：

无

问题分析：

问题似乎很复杂，控制栏杆的起落其实就是一个简单的判断，“成绩是大于或等于 60 吗?”，如果大于或等于 60，它就升起，几秒钟后落下，等待其他学生的到来。它始终处于监控状态。“允许通过”可以用输出“good! you passed!”信息表示。获取成绩，可以用 scanf 语句来模拟，获得成绩后，进行判断，如果成绩大于或等于 60，才打印通过信息，不然什么都不做，继续等待别的学生输入。

算法设计(伪码)：流程图如图 3.1 所示。

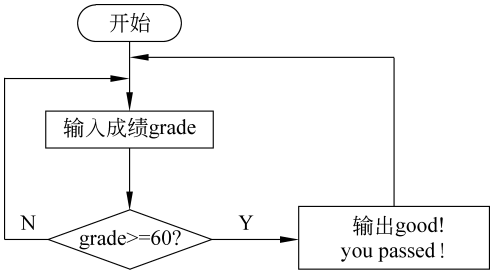

图 3.1　让成绩合格的学生通过的流程图

① 计算机等待输入成绩；

② 如果成绩大于或等于 60，输出“good! you passed!”，回到①；

③ 否则，回到①。

程序清单 3.1


源码 3.1

```
#001 /*
#002 * stupassed.c:判断学生成绩是否通过
#003 */
#004 #include<stdio.h>
#005 int main(void)
#006 {
#007     int grade;
#008     while( 1 ){
#009           scanf("%d",&grade);
#010           if(grade >=60)
#011                printf("good!you passed!\n");
#012     } //此循环只能采用 Ctrl-C 强行中断
#013     return 0;
#014 }
```

3.1.1　关系运算与逻辑判断

程序清单 3.1 中#010 行有一个式子 “grade >= 60”，它是**关系表达式**，其中出现的运

算符 >= 称为**关系运算符**。这个表达式是把 grade 变量中的值与 60 进行大于或等于比较,比较的结果有两种可能,或者为真,或者为假,当大于或等于成立时为真,否则为假。由此可见,关系表达式的真假是表示逻辑判断的条件,使用关系表达式就可以让计算机具有一定的"智能"。

C/C++ 支持 6 种关系运算,其中大于(>)、小于(<)、大于或等于(>=)、小于或等于(<=)与数学上两个数的比较运算>、<、≥、≤相对应,但要注意写法上有所不同。此外还有等于(==)、不等于(!=)两种运算,这与数学上的写法大不相同。这里有两个容易犯的错误:一是把由两个符号构成的关系运算如<=分开书写成< =,中间多了一个空格;二是非常容易把判断相等的关系运算==写成一个=号,而且这个错误比较隐蔽,因为编译器不会知道你要进行关系运算,而是认为你要进行赋值运算,不会报错。

显然,关系运算像算术四则运算一样,都是**双目运算**,即它们都有两个操作数。由关系运算符连接起来的式子称为**关系表达式**,如 grade >= 60 就是一个关系表达式。

到此为止,已经有三类主要的运算了,它们是算术运算、赋值运算、关系运算等。在一个表达式中既可以出现算术运算,也可以出现关系运算,甚至它们的混合运算,因此不同类型的运算符之间必须规定严格的优先级。即使是同一类运算,不同的运算之间也要有严格的结合性。这三类运算的优先级是:

关系运算的优先级低于算术运算,但高于赋值运算,而关系运算中比较大小的四个运算>、<、>=、<=的优先级又高于判断相等的两个运算==和!=。

关系运算是左结合的,但一般很少用到,因为常常造成误解,真正使用的时候多加一层括号更清楚。表 3.1 扩展了表 2.3,给出了当前各种运算符的优先级和结合性。

表 3.1 运算的优先级和结合性(优先级从高到低)

运 算 符	含 义	结 合 性
()	括号	最近的括号配对
+,-	单目运算,取正、负	从右向左
*,/,%	双目运算,乘、除、求余	从左向右
+,-	双目运算,加、减	从左向右
>,<,>=,<=	双目运算,比较大小	从左向右
==,!=	双目运算,判断是否相等	从左向右
=	双目运算,赋值	从右向左

【例 3.1】 设有"int a, b, c, status; a=1; b=2; c=3;",分析下面两个语句中各种运算的顺序:

```
① printf("%d\n",a+b>c);      //算术运算与关系运算混合
② status=a>b;                //赋值运算与关系运算混合
```

分析如下:因为算术运算优先于关系运算,所以① 的运算顺序为先 a+b,其结果再与 c 比较;因为关系运算优先于赋值运算,所以② 的运算顺序为先 a>b,结果再赋值给 status。

【例 3.2】 设有 "int a=30,b=20, c=2,status;",下面的语句正确吗?

```
status=a >b >c;
```

如果正确，status 的值会是多少？

首先肯定连续使用关系运算是没有语法错误的，但是很容易让人造成误解，因为从数学上来看，b 是介于 a 和 c 之间的，所以比较的结果应该为 1。但在 C/C++ 中是不能按照数学上的关系理解级联的多次比较，编译器会按照左结合的方式先求 a ＞ b 的值为 1，求得的结果 1 再与 c 比较，1 不大于 c，所以 1 与 c 比较的结果为 0，然后 0 赋给 status，最后 status 的值是 0。因此该语句不能表达数学上介于两个数之间的不等式。

3.1.2 逻辑常量与逻辑变量

1. 逻辑常量

任何表达式都是有值的，算术表达式的值是算术运算的结果，赋值表达式的值是赋值的结果。C/C++ 中关系表达式的值应该是比较的结果。关系表达式比较的结果不是真就是假，因此它的值或者为 1，或者为 0。一个关系表达式的值在逻辑上只有两种可能，因此逻辑常量只有 0 和 1。程序清单 3.2 对 a＝10 和 b＝20 分别输出了关系表达式 a＞b、a＜b、a＝＝b、a＞＝b、a＜＝b 及 a!＝b 的值。注意，逻辑值 0 和 1 是特殊的整型数，因此输出的时候仍然使用％d。

程序清单 3.2


源码 3.2

```
#001 /*
#002 * compare2numbers.c:比较两个数
#003 */
#004 #include<stdio.h>
#005 int main(void){
#006     int a=10,b=20;
#007     printf("%d ",a>b);
#008     printf("%d ",a>=b);
#009     printf("%d ",a<b);
#010     printf("%d ",a<=b);
#011     printf("%d ",a==b);
#012     printf("%d\n",a!=b);
#013     return 0;
#014 }
```

运行结果：

```
0 0 1 1 0 1
```

2. 逻辑变量

标准 C 语言中没有提供逻辑数据类型，因此如果要用一个变量存储逻辑常量 0 或 1 的话，只能用整型变量来模拟。在程序清单 3.2 中，如果声明一个整型变量 status 存放逻辑比较的结果，那么就有程序清单 3.3 的程序。

程序清单 3.3

```
#001 /*
```


源码 3.3

```
#002 * logicAssign.c:两个数的比较结果暂存到一个逻辑变量中或者整型变量中
#003 */
#004 #include<stdio.h>
#005 #include<stdbool.h>
#006 int main(void){
#007     int a=10,b=20;
#008     int status;
#009     //_Bool status;
#010     // bool status;
#011     status=a>b;
#012     printf("%d ",status);
#013     status=a>=b;
#014     printf("%d ",status);
#015     status=a<b;
#016     printf("%d ",status);
#017     status=a<=b;
#018     printf("%d ",status);
#019     status=a==b;
#020     printf("%d ",status);
#021     status=a!=b;
#022     printf("%d\n",status);
#023     return 0;
#024 }
```

注意：程序中的第#011行赋值运算的优先级低于关系运算，因此先进行关系运算，再把比较结果0或1赋值给status。

C99专门为逻辑数据提供了一种数据类型_Bool，称为**布尔型**，它是用英国数学家Geordge Boole命名的。程序清单3.3的#008行、#009行及#010行都是等价的，即逻辑值可以用bool型或_Bool型，也可以使用int型代替，会得到同样的结果。值得注意的是，如果在程序中使用bool型，必须包含头文件stdbool.h，这个头文件虽然是C++的，但在C程序中也可以使用。打开stdbool.h就会发现，bool就是_Boole的另一个名字而已，在这个头文件还定义了符号常量true和false，用它们来表示逻辑值1和0。

3.1.3 单分支选择结构

关系表达式的真或假构成了逻辑判断的条件。C/C++的选择结构就是通过判断条件的真和假有选择地执行某些语句(分支)。按照分支的多少分为三种选择结构：单分支、双分支和多分支。本节介绍单分支的选择结构。

单分支选择结构用if语句表达，具体格式如下：

```
if(判断条件表达式)
    条件为真时执行的语句或语句块;
其他语句
```

其中，判断条件表达式可以是3.1.2节介绍的关系表达式，也可以是其他形式(详见

3.1.4 节)；条件为真时执行的语句可以是任何可执行语句，甚至可以是多个语句构成的**语句块**(也称为**复合语句**，详见3.1.4 节)。单分支选择结构的流程图如图 3.2 所示。

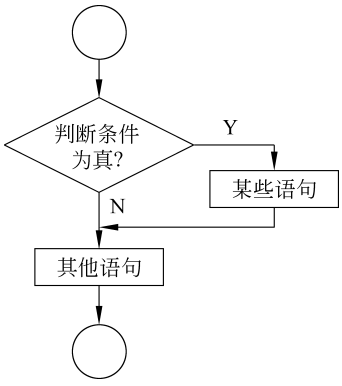

图 3.2 单分支选择结构流程图

注意 if 语句的结构和写法。从结构上来看，可以认为 if 语句由两部分组成：一部分是 if 判断行，if 后面的判断条件一定要用小括号括起来；另一部分是条件为真时要执行的语句，这一部分可以是一个单句，也可以是一组语句。因为 **C/C++ 语言对行不敏感，写成多行还是一行都是一样的**，所以 if 语句写成两行或多行完全是为了阅读上的清晰，完全可以把它们都写在一行里。不管把它写成几行，if 语句的两部分合起来才是一个 if 语句。**注意 if 判断行的末尾是没有分号的**。程序清单 3.1 的 #010 行和 #011 行合起来才是一个单分支选择结构，或者叫 if 语句。注意它的写法

```
if(grade>=60)
    printf("good,you passed!\n");
```

也可以把它写成一行

```
if(grade>=60)  printf("good,you passed!\n");
```

但是这种写法对于条件为真时要执行的语句是多个的时候就不方便了，因此不提倡这种写法。

程序结构通常用**流程图**表示可以直观地认识它的执行过程，if 语句对应的流程图如图 3.2 所示。在流程图中，一般把判断条件表达式置于菱形框之内，用菱形框表示判断条件，所以通常称菱形框为**判断框**。判断框有且只有一个入口，有且只有一个可以选择的出口，“真”或者“假”。框图中的两个小圆圈表示在其之前或之后是程序的其他部分，它是连接其他部分的关节，称为**连接框**。从框图可以看出 if 语句的执行过程是：条件为真时执行某个可执行语句或复合语句；条件为假时跳过那个可执行语句或复合语句去执行 if 结构下面的其他语句。

【例 3.3】 对键盘输入的两个整数 a 和 b，输出它们所具有的大小关系。

先简单进行分析。两个数的大小关系可能有多种，大于、小于、大于或等于、小于或等于、不等于、等于，不能只给出其中的一种判断。如果大于关系成立，大于或等于也成立，不等于也成立。输入的数据不同，大小关系也不同。因此需要列出所有可能的大小关系。具体实现如程序清单 3.4。

程序清单 3.4


源码 3.4

```
#001 /*
#002 * compare2numbers2.c:比较两个数
#003 */
#004 #include<stdio.h>
#005 int main(void){
#006     int a,b;
```

```
#007    scanf("%d%d",&a,&b);
#008    if(a>b)
#009        printf("%d>%d\n",a,b);
#010    if(a>=b)
#011        printf("%d>=%d\n",a,b);
#012    if(a<b)
#013        printf("%d<%d\n",a,b);
#014    if(a<=b)
#015        printf("%d<=%d\n",a,b);
#016    if(a==b)
#017        printf("%d==%d\n",a,b);
#018    if(a!=b)
#019        printf("%d!=%d\n",a,b);
#020    return 0;
#021 }
```

运行结果：

输入样例 1：

```
2 3
```

输出样例 1：

```
2<3
2<=3
2!=3
```

输入样例 2：

```
3 2
```

输出样例 2：

```
3>2
3>=2
3!=2
```

输入样例 3：

```
3 3
```

输出样例 3：

```
3>=3
3<=3
3==3
```

3.1.4 特殊形式的判断条件

除了利用像 grade>=60 这样的关系表达式的值作为逻辑判断条件之外，还有一些特殊形式，它们不是关系表达式，但当它们用作判断条件时，系统就会把它们转换为逻辑值。如算术表达式的值、一个整型变量或常量的值、一个字符常量值，甚至是浮点型变量或常量。C/C++ 规定一个表达式的值或某个变量的值或常量，当它们出现在 if 语句或其他含有判断条件的语句中时，只要它们的值非零，就转换为逻辑真，只有它们的值为零时才为逻辑假。简单来说，**非 0 即为逻辑真，0 为逻辑假**。下面分别举例说明。

【例 3.4】 判断一个整数不是偶数(或者是奇数)。

一个数 x 如果它能被 2 整除它就是偶数，也就是说如果 x%2 等于 0，那么 x 就是偶数，那么一个数不是偶数的判定条件为真怎么写呢？答案是 x%2 != 0，即 x 除以 2 其余数不为零，不为零的数当然不等于 0，因此结果为真。下面的 if 语句

```
if( x%2!=0 )
    printf ("%d 不是偶数\n",x);
```

按照**非零即为逻辑真**的原则，可以把它简写为

```
if( x%2 )
    printf ("%d 不是偶数\n",x);
```

这里用算术表达式 x%2 作为逻辑判断条件，当它的值非零时就转化为逻辑真。

【例 3.5】 判断一个整数不是零。

很容易写出“一个整数 x 不是零”为真的条件 x!= 0，因此有下面的 if 语句

```
if( x !=0 )
    printf ("%d 不是零\n",x);
```

与例 3.4 类似，同样有下面的简写形式：

```
if( x )
    printf ("%d 不是零\n",x);
```

这里直接使用了变量的值作为判断条件，当 x 不是零时它的值就转换为逻辑真。

现在回头看看 while(1)，这个 1 就是一个常量作为逻辑判断条件的例子，1 不为零，所以当然就是逻辑真了。其实，如果把 1 换成任何一个其他不为零的整数，负数也可以，其效果都是一样的，如 while(2)、while(－100)。而且这个常数作为逻辑条件是不变的，所以它永远为真。

还有更多形式的判断条件，后面章节会陆续介绍。

3.1.5 比较两个实数的大小

在数学上，比较两个实数与比较两个整数没什么区别，但在计算机中，比较两个实数要特别小心了。由于实数在计算机中存储是有精度的，float 类型的数据只有 6 位或 8 位有效数字，double 类型的数据也只有 16 位或 17 位有效数字。从下面的例子可以看到，单精度变量 a 虽然赋值是 3.1，但实际内存中存放的确是 3.099999905，单精度变量 b 是 3.099999991397669，但实际在内存中存放的也是 3.099999905，不管是内部还是外部，a 与 b 小数点后前 7 位是相等的，所以结果是 a 等于 b，而表面看上去应该 a 大于 b。双精度的变量 x，程序中给的也是 3.1，但内存中存储的是 3.10000000000000100，而双精度的 y＝3.099999991397669，但内存里存储的是 3.09999991397668800，由于双精度的变量精度比较高，所以比较的结果与实际数据的比较结果一致，即 x＞y。从这两个例子可以看出，两个实数是否相等是与精度密切相关的，在单精度时两个数据相等，在双精度时两个数据可能不等。

程序清单 3.5

源码 3.5

```
#001 /*
#002 * realCompare.c: 实数比较大小
#003 */
#004 #include<stdio.h>
#005 int main(void){
```

```
#006    float a,b;
#007    a=3.1;
#008    b=3.099999991397669;
#009    printf("%11.9f  %11.9f\n",a,b);
#010    printf("%f  %f\n",a,b);
#011    if(a>b)
#012        printf("a>b\n");
#013    if(a<b)
#014        printf("a<b\n");
#015    if(a==b)
#016        printf("a==b\n");
#017    double x,y;
#018    x=3.1;
#019    y=3.099999991397669;
#020    printf("%20.18f  %20.18f\n",x,y);
#021    printf("%f  %f\n",x,y);
#022    if(x>y)
#023        printf("x>y\n");
#024    if(x<y)
#025        printf("x<y\n");
#026    if(x==y)
#027        printf("x==y\n");
#028    return 0;
#029 }
```

运行结果：

```
3.099999905  3.099999905
3.100000  3.100000
a==b
3.100000000000000100  3.099999991397668800
3.100000  3.100000
x>y
```

从程序清单3.5可以看出，**两个实数是否相等是由指定的精度决定的**。在给定的精度下，两个实数的差如果等于零意味着两个实数相等，否则就是不等。精度常用一个小数表示，如0.01就是精确到2位小数，0.0001就是精确到4位小数，当小数位数更多的时候可以用指数形式表示，如.1e-7就是精确到第8位小数，这种精度数据常常用变量eps表示，即

```
double eps=.1e-7;
```

在程序设计的时候**一般不是直接比较两个实数是否相等，而是通过它们之间的误差的精度来判断**，误差如果在允许范围之内就认为是相等的了。标准数学库math.h提供了一个求绝对值的函数fabs(double x)，在程序中用来求两个数的误差的绝对值，如果这个绝对值不超过给定的精度eps，就可以认为这两个数是相等的。程序清单3.6中给定了一个精度eps=0.001和单精度的pi=3.1415926，用户任意输入一个pi值存入yourPi中，程序通过把

yourPi 与程序中的 pi 值比较，如果它们的误差的绝对值不超过给定的精度 eps，这时就认为 yourPi 符合精度，也可以认为在给定的精度下，yourPi 与程序中的 pi 相同。如果它们的误差的绝对值大于给定的精度，这时认为 yourPi 不符合精度要求。

程序清单 3.6


源码 3.6

```
#001 /*
#002 * realCompare2.c: 判断给定的 PI 值是否符合精度
#003 */
#004 #include<stdio.h>
#005 #include<math.h>
#006 int main(void){
#007     double eps=0.001,yourPi;
#008     double pi=3.1415926;
#009     printf("I have a precision now,pls input your PI value:\n");
#010     scanf("%lf",&yourPi);                    //输入一个 pi 值
#011     double err=fabs(pi-yourPi);              //计算误差
#012     printf("fabs(pi-yourPi)=%10.8f\n",err);//打印误差
#013     if(err<=eps)                             //符合精度
#014     {
#015      printf("%10.8f=%10.8f according to the precision %10.8f\n",pi,yourPi,
         eps);
#016          printf("yourPi %10.8f is met the precision %10.8f\n",yourPi,eps);
#017     }
#018      if(fabs(pi-yourPi)>eps)                 //没有达到精度
#019     printf("yourPi %10.8f is not met the precision eps %10.8f\n",yourPi,eps);
#020      return 0;
#021 }
```

运行结果 1：

```
I have a precision now,pls input your PI value:
3.14(用户输入的)
fabs(pi-yourPi)=0.00159260
yourPi 3.14000000 is not met the precision eps 0.00100000
```

运行结果 2：

```
I have a precision now,pls input your PI value:
3.1415(用户输入的)
fabs(pi-yourPi)=0.00009260
3.14159260=3.14150000 according to the precision 0.00100000
yourPi 3.14150000 is met the precision 0.00100000
```

3.1.6 复合语句

复合语句是多个语句的复合体，它是由**左右两个大括号括起来的语句块**。复合语句本身自成一体，它与程序的其他部分既相互独立又有一定的联系。请看下面的例子。

```
if(grade <60 )
{
     printf("you are not passed\n");
     printf("hope you make great efforts\n");
     nopassed=nopassed +1;
}
```

或者

```
if(grade<60 ){
    printf("you are not passed\n");
    printf("hope you make great efforts\n");
    nopassed=nopassed +1;
}
```

注意大括号的位置,它们可以顶格左对齐,也可以左右错开,是不同程序风格的体现。前者称为 **Allman 风格**(Allman 来源于 sendmail 和其他 UNIX 工具的作者 Eric Allman),后者称为 **K&R 风格**(K&R 是 Kernighan 和 Ritchie 的简称,他是 The C programming language 的作者)。你更喜欢哪一种?还要注意复合语句包含的内部语句采用缩进格式,这样更便于阅读。

上面的复合判断语句,当判断条件 grade<60 为真时执行的是一个复合语句。复合语句可以出现在任何程序结构中,任何单个语句的位置都可以用一个复合语句代替,后面要介绍的循环体或者某个顺序结构的一部分都可以是某个复合语句。


视频

3.2 按成绩把学生分成两组

问题描述:

教师要把参加某次测验的学生按成绩及格与否分成两组,并统计出各组的人数。

输入样例:

```
88 99 77 66 55 44        //或者分多行输入
Ctrl-Z
```

输出样例:

```
you belong in group A
you belong in group A
you belong in group A
you belong in group B
you belong in group B
aNum=4
bNum=2
```

问题分析:

3.1 节的问题只考虑了成绩合格者如何处理,成绩不合格者置之不理,即当判断条件为真时去处理事情,而当判断条件为假时就跳过了,没有做任何事情。很多场合我们不仅要描述判断条件为真时做什么,还要对判断条件为假时的情况做出处理。本节的问题仍然是一个判断决策问题。按学生成绩把学生进行分组,就是成绩大于或等于 60 时的学生去 A 组,成绩小于 60 的学生去 B 组,并统计出每组的学生数。用简单的单分支选择结构可以解决这个问题吗?回答是可以的。大家分析下面的程序是否可行,看看存在什么不足。

程序清单 3.7


源码 3.7

```
#001 /*
#002 * twoif.c: 使用 if(单分支选择)语句实现学生按成绩及格与否分组
#003 */
#004 #include<stdio.h>
#005 int main(void){
#006     int aNum =0, bNum =0; //声明两个计数变量,并初始化为 0
#007     int grade;              //学生成绩变量
#008     while(1) { //无限循环,只能在输入结束时用 Ctrl-C 结束
#009         scanf("%d",&grade) ;
#010         if(grade >=60 ) {     //及格分组统计
#011            printf("you belong in group A\n");
#012            aNum =aNum +1;
#013         }
#014         if(grade <60 ) {      //不及格分组统计
#015            printf("you belong in group B\n");
#016            bNum =bNum +1;
#017         }
#018         printf("aNum =%d\n", aNum);  //输出当前统计结果
#019         printf("bNum =%d\n", bNum);
#020     }
#021     return 0;
#022 }
```

这个程序实现的正确性是没有问题的,但仔细看会发现,不管你的成绩是大于或等于60,还是小于 60,#010 行、#014 行的两个判断都要进行。例如,现在一个学生的成绩是90,首先经历#010 行的判断,grade>=60 为真,执行#011 行和#012 行。紧接着就要执行#014 行的判断,grade < 60 为假,因此不执行#015 行和#016 行。同样,如果一个成绩是 50,也要经历同样的两次判断。每个成绩都要判断两次,显然是一种浪费。实际上,成绩大于或等于 60 和成绩小于 60 这两个判断条件之间是紧密相连的,是恰好相反的。如果第一个条件为假,自然就有另一个条件为真,没有必要再去重复判断。对于具有这样性质的判断问题,C/C++ 提供了一种双分支选择结构 if-else。

这个实现还有一个问题就是 while(1)无限循环结构,一个班级的实际学生数是有限的,当没有数据输入的时候,while(1)要停下来,如果没有其他的控制手段只能在运行时按 Ctrl-C 结束。在前面这个实现里,输出部分也在 while(1)循环结构里,每个同学分组之后都打印当前的统计结果,如果有这样的需要当然没有问题,但如果只要求所有的同学分组之后给出最终的统计结果就不好办了。这时势必要把输出部分拿到 while 之外,但由于不能控制结束 while 所以输出就做不到了。因此我们必须给出控制 while 的方法,scanf 函数有一个特点,就是当没有数据要输入的时候,如果输入了 Ctrl-Z,这时 scanf 函数就读不到需要的数据了,它会返回一个特殊的信息 EOF(它是一个用#define 定义的符号常量,其值为-1),它表示输入已经结束。所以,我们现在可以用它来控制循环的结束,关于循环方面的细节在第 4 章进行详细介绍。

算法设计：流程图如图 3.3 所示。

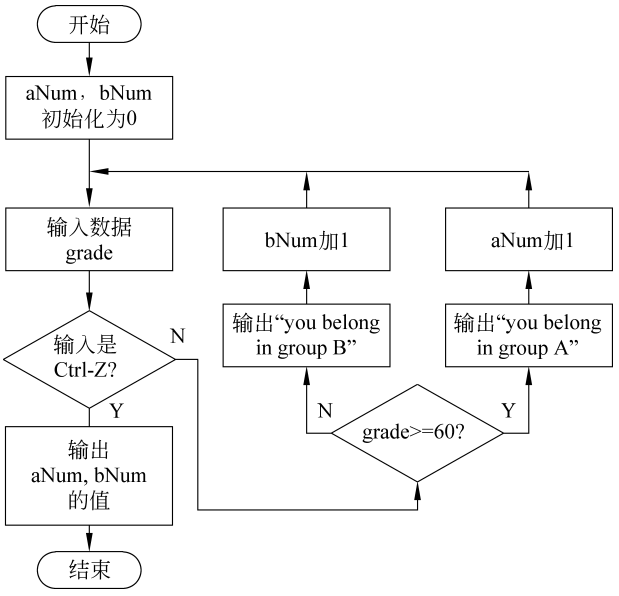

图 3.3　按成绩把学生分成两组的流程图

① 把统计求和变量 aNum,bNum 初始化为 0；
② 输入学生成绩,如果输入了 Ctrl-Z,执行⑤,否则转③；
③ 如果成绩大于或等于 60,输出分到 A 组信息,a 加 1,返回②；
④ 否则,输出分到 B 组信息,b 加 1,返回②；
⑤ 输出最终统计结果程序结束。

程序清单 3.8

源码 3.8

```
#001 /*
#002 * ifelse.c: 使用 if-else(双单分支选择)语句实现学生按成绩及格与否分组
#003 *           通过 Ctrl-Z 结束键盘输入
#004 */
#005 #include<stdio.h>
#006 int main(void) {
#007     int aNum = 0, bNum = 0;                          //声明两个计数变量,并初始化为 0
#008     int grade;                                       //学生成绩变量
#009     while(scanf("%d",&grade) !=EOF) {                //读学生成绩直到输入 Ctrl-Z 为止
#010         if(grade >= 60 ) {                           //及格分组统计
#011            printf("you belong in group A\n");
#012            aNum = aNum + 1;
#013         } else{                                      //不及格分组统计
#014            printf("you belong in group B\n");
#015            bNum = bNum + 1;
#016       } }
#017     printf("aNum = %d\n", aNum);                     //输出统计结果
#018     printf("bNum = %d\n", bNum);
```

```
#019     return 0;
#020 }
```

3.2.1 双分支选择结构

程序清单 3.8 的＃010 行到＃016 行是一个双分支的选择结构，条件“grade＞＝60”为真时执行一个分支，否则执行另一个分支。双分支选择结构用 if-else 语句表示，其一般形式为

```
if(判断条件表达式)
   条件为真时要执行的语句
else
    条件为假时要执行的语句
其他语句
```

其中，表达式和语句的含义同单分支选择结构一样。它的执行过程如图 3.4 所示，当判断条件为真时执行 if 和 else 之间的语句，否则(隐含着判断条件为假)，执行 else 后面的语句。在这个结构中存在两个分支，对于给定数据，只能有一个分支符合判断条件。不管是哪个分支，执行完毕之后都应该执行 if-else 结构下面的“其他语句”。这种双分支选择结构是对称的。

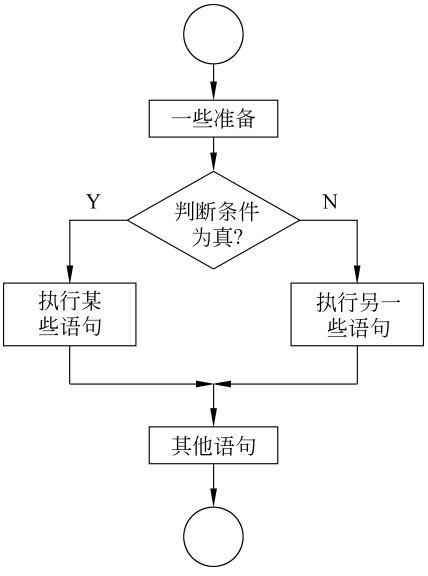

图 3.4 双分支选择结构流程图

从程序清单 3.7 和程序清单 3.8 的运行结果可以看到，两个平行的单分支选择结构和一个双分支选择结构都能实现本节的问题，其结果是完全一致，但是它们的运行过程有很大的不同。两个单分支要判断两次，而一个双分支只判断一次。下面再看几个小例子。

【例 3.6】 判断一个数是奇数还是偶数。

程序实现的片段如下：

```
scanf("%d",&num);
if(num%2 )
    printf("num is odd\n");
else
    printf("num is even\n");
```

当用户输入一个整数之后，如果输入的是奇数，则 num％2 不为 0，判断条件为真，输出信息 num is odd，如果输入的是偶数，num％2 为 0，判断条件为假，输出 num is even。输出哪条信息(进入哪个分支)由判断条件 num％2 的真假决定。

【例 3.7】 判读一个数是大于或等于零还是小于零。

程序实现的片段如下：

```
scanf("%d",&num);
if(num >=0 )
    printf("num is equal to 0 or positive number;\n");
else
    printf("num is a negative number;\n");
```

【例 3.8】 判断一个人的体重是否过大,判断标准是身体指数 t 是否大于 25,其中 $t=w/h^2$(w 为体重,h 为身高)。

程序实现的片段如下:

```
scanf("%f %f", &w, &h);
t=w/(h * h);
if(t>25 )
    printf("your weight is higher \n");
else
    printf("your weight is not higher,
           but I could not know if your weight is lower \n");
```

3.2.2 条件运算

由于双分支选择结构应用比较频繁,C/C++ 提供了一种特别的运算,称为**条件运算**,用来表达对称的双分支选择结构,具体格式如下:

```
表达式 1? 表达式 2 : 表达式 3
```

这里? 与 : 是**条件运算符**,它需要三个操作数:第一个操作数是表达式 1,第二个操作数是表达式 2,第三个是表达式 3。这个运算是唯一的三目运算,由条件运算符连起来的式子称为**条件表达式**。条件表达式的值由表达式 1 的真假来决定是取表达式 2 的值还是表达式 3 的值。当表达式 1 为真时,条件表达式的值等于表达式 2 的值,否则等于表达式 3 的值。表达式 1 是判断条件,表达式 2 和表达式 3 是对称的两个选择。条件表达式的使用非常灵活简洁,它可以独立作为一个语句,也可以作为其他运算的操作数。它出现在普通表达式可以出现的任何地方。下面看 6 个例子。

【例 3.9】 用条件表达式判断一个数是奇数还是偶数。

```
num%2? printf("num is odd\n"): printf("num is even\n");
```

根据 num%2 的真假执行不同的打印输出,当 num%2 不等于 0 时输出 num is odd,否则输出 num is even。

【例 3.10】 用条件表达式判断一个数是正还是负。

```
num >=0
    ? printf("num is equal to 0 or positive number;  \n")
    : printf("num is a negative number;  \n");
```

这里把一个条件表达式语句写在了多行,但要注意它是一个语句,只能在末尾有一个“;”号。

【例 3.11】 打印两个数中的较大者,对于给定的 i 和 j,

```
printf("%d\n", i >j? i : j);
```

根据 i>j 的真假，输出 i 或 j，当 i>j 时输出 i，否则输出 j。

【例 3.12】 返回两个数中的最大者，对于给定的 i 和 j，

```
return( i >j ? i : j) ;
```

根据 i>j 的真假，返回 i 或 j，当 i>j 时返回 i，否则返回 j。

【例 3.13】 求两个数的最大值，对于给定的 a 和 b，

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

max 的值由 a>b 的真假决定，当 a>b 时 max 的值为 a，否则为 b。

注意：条件表达式一般多用于三个表达式比较简单的情形。

【例 3.14】 条件运算是右结合的，对于给定的 a、b、c、d 条件表达式。

```
a >b ? a : c >d ? c : d
```

按照右结合的原则，c >d 先作为第二个条件表达式的判断条件，如果它为真，取 c，否则取 d，然后再看 a>b 是否为真，如果为真，则取 a，否则取 c>d 时条件运算的值。它相当于

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

3.3 按成绩把学生分成多组（百分制）


视频

问题描述：

写一个程序帮助教师把学生按分数段（90 分以上，80～89 分，70～79 分，60～69 分，小于 60 分）分成多组，统计各组的人数。

输入样例：

```
please input grades:
44 55 77 88 99 98 78 67
^Z
```

输出样例：

```
Failed! group F
Failed! group F
Middle! group C
Better! group B
Good! group A
Good! group A
Middle! group C
Pass! group D
aNum =2
bNum =1
cNum =2
dNum =1
FNum =2
```

问题分析：

在 3.2 节已经使用双分支选择结构解决了把学生按成绩分为两组的问题。本节的问题要求进行更细致的划分，即根据成绩的分数段（90 分以上，80～89 分，70～79 分，60～69 分，小于 60 分）把学生划分为多个组，不妨设为 A、B、C、D、F 组，并分别统计各组的人数。问题中有多个判断条件，而且不是简单的关系表达式所能表达的。90 分以上和 60 分以下比较容易表达，用一个简单的关系表达式即可。其他几种情况都是一种复合条件，即两个条件要

同时满足,如成绩为80～89分的复合条件是“成绩大于或等于80分”并且“成绩小于90分”,前面曾经指出这样的条件不能写成

```
80<=grade<90
```

如果这样写,它并不表示成绩grade为80～90分,它将从左至右分两段进行计算。如果一定要用一个表达式表达这种比较复杂的条件,要用到3.5.1节介绍的逻辑运算,这里暂时不考虑这种方法,下面从两个方面进行讨论。

第一种方法:仍然采用简单的关系表达式作为条件。由于介于两者之间分数段判断是一个复合条件,这种复合条件判断可以通过两个简单判断的**嵌套**来实现,见3.3.1节。若干个复合条件可以彼此独立,也可以按照相邻分数段之间彼此存在的联系用双分支中的else体现出来,具体见算法设计1,详细讨论见3.3.2节。同一个问题,不一定必须先判断什么,后判断什么,但判断顺序的不同,将导致不同的嵌套顺序,算法设计2与算法设计1的判断顺序不同,因此嵌套结构也不同。

第二种方法:是对分数间接地进行判断,把输入的成绩进行适当的转换(除以10再取整),再去判断它属于哪一组。这时所有的成绩转换为有限的几种情况,即10、9、8、7、6、5、4、3、2、1、0。解决这类问题引进一种新的选择结构,switch-case多分支选择结构,具体见算法设计3,详细讨论见3.3.3节。

算法设计1:流程图如图3.5所示。

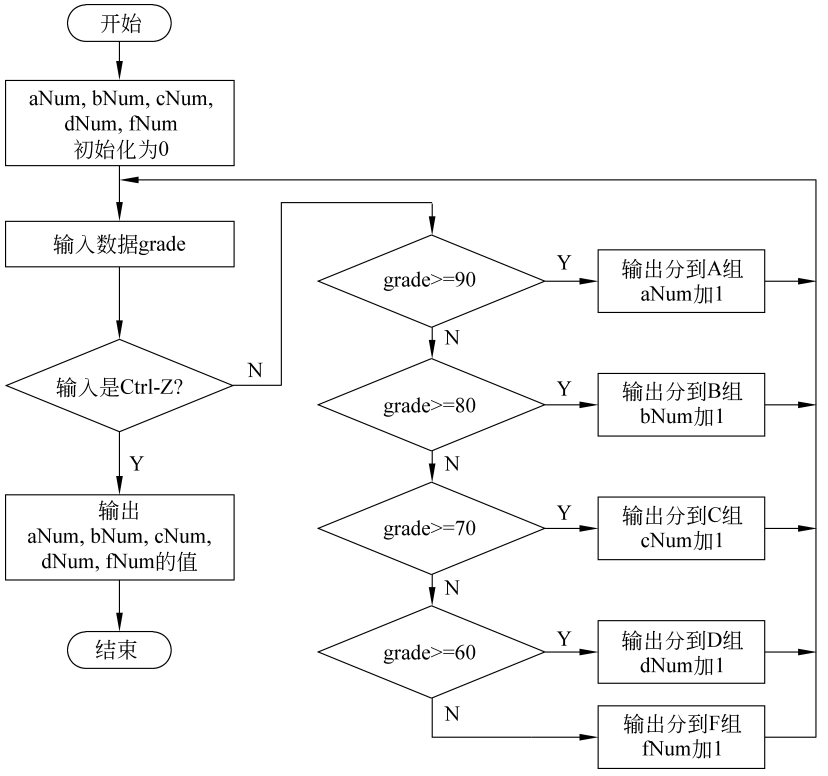

图3.5　按成绩把学生分成多组算法1的流程图

如果按照成绩从大到小或从小到大进行判断，则有比较整齐的嵌套描述。本算法描述按成绩从大到小的顺序来判断，即先判断成绩是否大于或等于90，如果不是，再看是否大于或等于80，以此类推。具体算法描述如下。

① 把统计求和变量aNum，bNum，cNum，dNum，fNum初始化为0；
② 输入学生成绩；
③ 如果输入没有结束则执行④，否则执行⑨；
④ 如果成绩大于或等于90，输出分到A组信息，aNum加1，返回到②；
⑤ 否则如果成绩还大于或等于80，输出分到B组信息，bNum加1，返回到②；
⑥ 否则如果成绩还大于或等于70，输出分到C组信息，cNum加1，返回到②；
⑦ 否则如果成绩还大于或等于60，输出分到D组信息，dNum加1，返回到②；
⑧ 否则输出分到F组信息，fNum加1，返回到②；
⑨ 输出统计结果。

程序清单 3.9


源码 3.9

```
#001 /*
#002 * ifelse_nest.c: 使用双分支嵌套，把学生按成绩的分数段分组
#003 */
#004 #include<stdio.h>
#005 int main(void){
#006     int aNum =0, bNum =0, cNum =0, dNum =0, fNum =0;
#007     int grade;
#008     printf("please input grades:\n");
#009     while( scanf("%d",&grade) !=EOF){
#010         if( grade >=90 ){
#011             printf("Good! group A\n");
#012             aNum=aNum +1;
#013         }else if( grade >=80 ){
#014             printf("Better! group B\n");
#015             bNum=bNum +1;
#016         }else if( grade >=70 ) {
#017             printf("Middle! group C\n");
#018             cNum=cNum +1;
#019         }else if( grade >=60 ) {
#020             printf("Pass! group D\n");
#021             dNum=dNum +1;
#022         }else {
#023             printf("Failed! group F\n");
#024             fNum=fNum +1;
#025         }}
#026     printf("aNum=%d\n", aNum);
#027     printf("bNum=%d\n", bNum);
#028     printf("cNum=%d\n", cNum);
#029     printf("dNum=%d\n", dNum);
#030     printf("FNum=%d\n", fNum);
```

```
#031     return 0;
#032 }
```

算法设计2：流程图如图3.6所示。

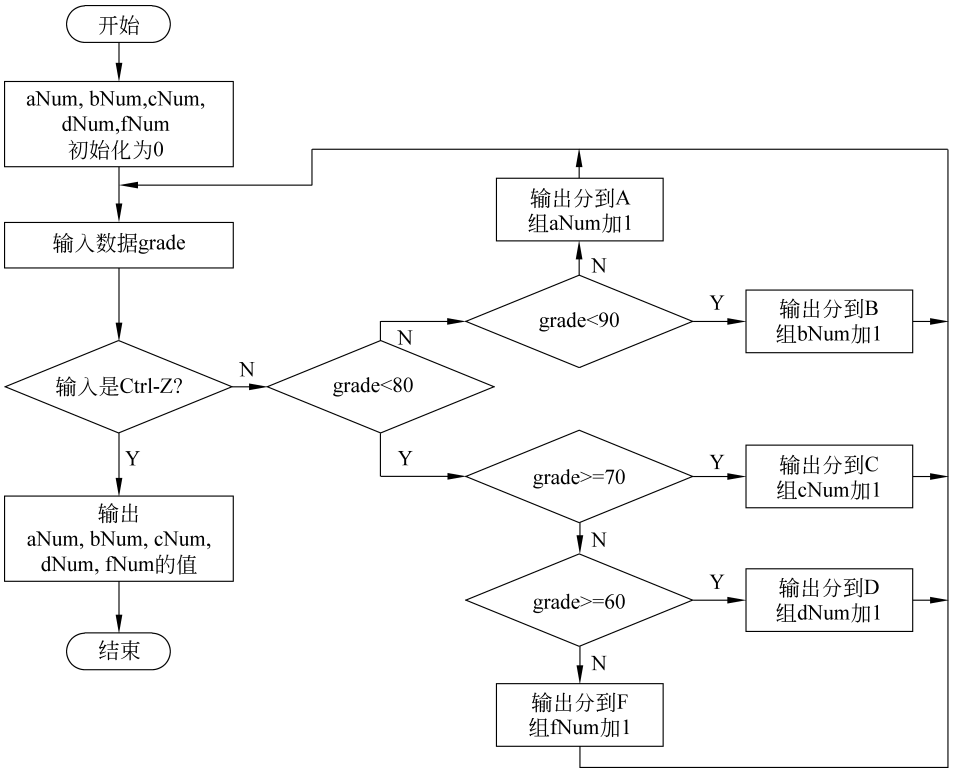

图3.6　按成绩把学生分成多组算法2的流程图

如果按成绩的**客观分布规律**考虑判断的顺序，即可能性比较大的先判断，那么首先应该判断成绩是否介于70分和80分之间，如果不是，则有两种可能，一是大于或等于80分，二是小于70分。如果是大于或等于80分，进一步看是否小于90分，又分两种情况，小于90分和大于或等于90分；如果是小于70分，进一步看是否大于或等于60分，这时又有两种情况，小于和大于或等于60分。这个改进的算法描述如下：

① 把统计求和变量aNum，bNum，cNum，dNum，fNum初始化为0；

② 输入学生成绩；

③ 如果输入没有结束则执行④，否则执行⑨；

④ 如果成绩小于80且大于或等于70，输出分到C组信息，cNum加1，返回到②；

⑤ 否则如果成绩小于90且大于或等于80，输出分到B组信息，bNum加1，返回到②；

⑥ 否则(成绩大于或等于90)输出分到A组信息，aNum加1，返回到②；

⑦ 否则如果成绩小于70且大于或等于60，输出分到D组信息，dNum加1，返回到②；

⑧ 否则(成绩小于60)，输出分到F组信息，fNum加1，返回到②；

⑨ 输出统计结果。

程序清单 3.10


源码 3.10

```
#001 /*
#002 * ifelse_nest_better.c: 使用双分支嵌套实现学生分组,最易发生的放在最外层
#003 */
#004 #include<stdio.h>
#005 int main(void){
#006     int aNum=0, bNum=0, cNum=0, dNum=0, fNum=0;
#007     int grade;
#008     printf("please input grades:\n");
#009     while(scanf("%d",&grade)!=EOF){
#010         if( grade <80 )
#011         if( grade <70 )
#012         if( grade <60){
#013             printf("failed! group F\n");
#014             fNum=fNum +1;
#015         } else{
#016             printf("Pass! group D\n");
#017             dNum=dNum +1;
#018         } else{
#019             printf("Middle! group C\n");
#020             cNum=cNum +1;
#021         }else if( grade <90 ){
#022             printf("Better! group B\n");
#023             bNum=bNum +1;
#024         }else {
#025             printf("Good! group A\n");
#026             aNum=aNum +1;
#027         }
#028     }
#029     printf("aNum=%d\n", aNum);
#030     printf("bNum=%d\n", bNum);
#031     printf("cNum=%d\n", cNum);
#032     printf("dNum=%d\n", dNum);
#033     printf("FNum=%d\n", fNum);
#034     return 0;
#035 }
```

算法设计 3：流程图如图 3.7 所示。

上述两种算法都是通过直接判断学生成绩进行分组。如果把成绩除以 10 取整,可以发现,100 分对应 10,90～99 分对应 9,80～89 分对应 8,以此类推,0～9 分对应 0,也就是说,对于每个学生的成绩,可以把它们除以 10 取整之后对应到 0,1,2,3,…,10 的整型常量上,依据这个整型常量即可断定该同学应该分到哪一组。如果是 0,1,2,3,4,5 就都分到 F 组,如果是 9 和 10 都归为 A 组,其他的 8、7、6 依次对应 B、C、D 组。

算法描述如下:

① 把统计求和变量 aNum,bNum,cNum,dNum,fNum 初始化为 0;

② 输入学生成绩 grade;