第3章

控制结构和数据文件

——学习任何知识的最佳途径是由自己去发现，因为这种发现理解最深、也最容易掌握其中的规律、性质和联系。

学时分配

课堂教学：16学时。

自学：18学时。

自我上机练习：10学时。

本章教学目标

(1) 理解自顶向下和结构化程序设计思想；

(2) 理解并能构造关系表达式和逻辑表达式；

(3) 应用选择结构解决问题；

(4) 应用循环结构，根据指定条件重复执行一系列步骤解决问题；

(5) 掌握几种常用算法和Code；

(6) 应用结构化程序设计方法解决实际问题；

(7) 理解并会简单使用从信息文件中读取数据和向信息文件中写入数据的方法。

本章项目任务

“学生信息管理系统”软件界面设计——进一步优化。

3.1 算法开发

3.1.1 自顶向下设计的算法思想

“自顶向下设计(Top-down Design)”相对于“自底向上设计(Bottom-up Design)”，它们均是一种软件设计策略。自顶向下设计是一种逐步求精的程序设计过程和方法。对要完成的较大规模的总任务进行分解，先对最高层次中的问题进行要领定义、参数设计、编程和测试，将其中暂不能明确设计解决的问题作为一个子任务放到下一层次中去解决。这样逐层、逐个地进行定义、设计、编程和测试，直到所有层次上的问题均由实用程序来实现，从而设计出一个具有完整、详细的层次结构的程序。一般经过概念设计、参数化设计和详细设计3个阶段。这个过程存在着多次反复和修改。这就是自顶向下、逐步求精的程序设计方法。

自顶向下、逐步求精的程序设计方法一般有以下几个步骤。

(1) 对实际问题进行全局性分析，确定问题的总体解决结构，分解为若干个相对独立的子问题。这称为问题的模块化设计。

(2) 对每个子模块按步骤(1)的方法，进行分析和细化，把问题解决方案细分为越来越小的部分。这个过程称为分治法策略。

(3) 用算法描述把每个小“子问题”细化为更多具体步骤。

(4) 用计算机语言描述，并最终解决问题。

【例 3-1】 用自顶向下、逐步求精的方法求解一元二次方程 $ax^2+bx+c=0$ 的根。

(1) 问题陈述和需求分析：求根问题“抽象”为一个黑盒子，输入该方程的系数参数 a，b，c 即输入一元二次方程，输出方程的根，如图 3-1 所示。这是对问题求解的整体描述。

(2) 数学模型和问题处理过程。

① 进一步细化“黑盒子”部分，求出问题解决方案。

步骤 1：对于输入的参数，判断能否构成一元二次方程，若 a 为 0，则退出程序。

步骤 2：利用数学中的求根公式求解一元二次方程的根。

细化求精后，出现了分支选择结构。用流程图描述，如图 3-2 所示。

② 细化子步骤 2 中的“黑盒子”，具体步骤如下。

步骤：确定数学中的求根公式(建立数学模型)。

$x_1=\dfrac{-b+\sqrt{\delta}}{2\times a}$，$x_2=\dfrac{-b-\sqrt{\delta}}{2\times a}$，用流程图描述如图 3-3 所示。

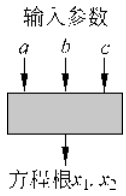

图 3-1 输入输出

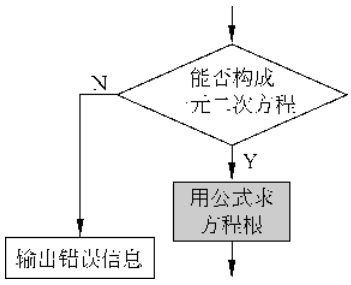

图 3-2 选择结构流程图

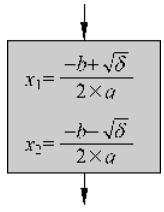

图 3-3 求根公式

③ 对求根公式再继续分析并细化。中间数据 δ 设为变量 delt，需进一步计算并判断。

步骤 1：delt=b * b-4 * a * c

步骤 2：delt≥0，且 delt=0，有两个相等实根，否则，有两个不等实根；否则，有两个虚根。

(3) 用流程图描述算法步骤，如图 3-4 所示。

```
/*算法开始*/
float a,b,c,x1,x2,delt;
用 scanf 函数输入 a,b,c 的值;
if(a == 0)
    输出提示信息并退出程序;
if(b*b-4*a*c>=0)
```

```
{   if(b * b - 4 * a * c == 0)
        输出方程两个相等根;
    else
        输出方程的两个实根;
else
    输出两个虚根;
}
/ * 算法结束 * /
```

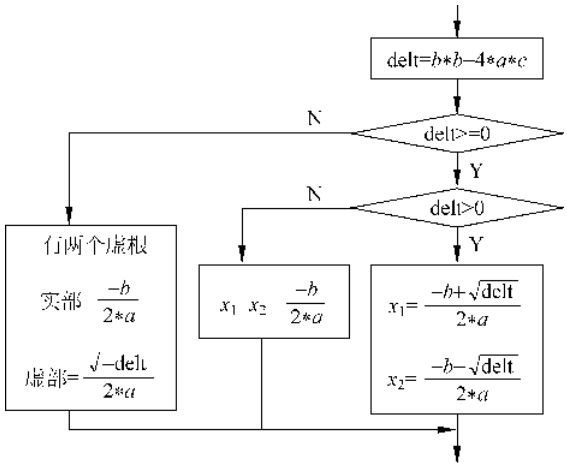

图 3-4　例 3-1 的算法流程图

使用流程图或伪代码表示算法只是个人爱好不同。算法只是问题求解的处理模型,并不是唯一的。算法不同,开发的 C 语言程序也会有所不同。

(4) 编码如下。

```
/ * program ch3 - 1.c * /
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main(void)
{   float a,b,c,x1,x2,shi,xu;
    printf("请输入方程的系数:");
    scanf("%f%f%f",&a,&b,&c);
    if(fabs(a)<1e-5)
    {    printf("不是一元二次方程! 程序结束\n");
         exit(0);
    }
    if((b * b - 4 * a * c)>=0)
        if((b * b - 4 * a * c) == 0)
        {   printf("方程有两个相等的实根\n");
            x1 = x2 =- b/(2.0 * a);
            printf("x1 = x2 = %7.2f\n",x1);
        }
        else
```

```
        {   printf("方程有两个不相等的实根\n");
            x1 = ( - b + sqrt(b * b - 4 * a * c))/(2.0 * a);
            x2 = ( - b - sqrt(b * b - 4 * a * c))/(2.0 * a);
            printf("x1 = %7.2f\nx2 = %7.2f\n",x1,x2);
        }
    else
    {   printf("方程有两个虚根\n");
        shi = - b/(2 * a);                              /* 实部 */
        xu = sqrt( - (b * b - 4 * a * c))/(2.0 * a);    /* 虚部 */
        printf("虚根的实部 = %7.2f\n",shi);
        printf("虚根的虚部 = %7.2f\n",xu);
    }
}
```

3.1.2 结构化程序设计思想

程序设计是给出解决特定问题程序的过程，该过程应当包括分析、设计、编码、测试和排错等不同阶段。专业的程序设计人员常被称为程序员。

程序设计有许多技巧和规则需要记忆。在编写程序时，经常会遇到许多相似的模式，应该学会识别这些模式并将它们看做一个个概念单元去使用。通常所说的概念单元可以是一条或一组语句，它们的大体结构是固定的，只在特定情况下做一些修改。要编写出高效的程序，必须学会在问题求解时灵活使用这些概念单元——功能模块。

结构化程序设计(Structure Programming)是一种程序设计技术，通常采用自顶向下、逐步求精的设计方法和单入口、单出口的控制结构。使用顺序、选择和循环 3 种基本控制结构或它们的嵌套结构构成具有复杂层次的“结构化程序”。结构化方法编出的程序在结构上具有以下效果：

(1) 以控制结构为单位，只有一个入口，一个出口，所以能独立地理解每一部分。

(2) 能够以控制结构为单位，从上到下顺序地阅读程序文本。

(3) 程序的静态描述与执行的控制流程容易对应，所以能方便正确地理解程序的动作。

“自顶而下、逐步求精”使设计者能把握主题，高屋建瓴，避免一开始就陷入复杂的细节中。而独立功能、单入口单出口的模块结构减少了模块的相互联系使模块可作为插件使用，降低程序的复杂性，提高可靠性。

1. 顺序结构

第 2 章中的程序都是顺序结构(Sequence Structure)程序。顺序结构是按照语句书写顺序执行的程序结构。

2. 选择结构

选择结构(Select Structure)也称为分支结构。选择结构测试一个条件，该条件结果或为真，或为假，依据条件的结果选择执行路径，而不是严格按照语句出现的顺序执行。

选择结构程序的设计方法关键在于构造合适的选择条件和适当的分支语句。

【例 3-2】 考试成绩处理之一——判断学生成绩是否及格。

该问题要求先输入一个学生的成绩，然后判断是否小于 60，如果小于 60 则输出成绩不及格，否则输出成绩及格。用流程图描述的算法如图 3-5 所示。

判断学生成绩是否及格构成了双分支选择结构。实际处理问题时，还会遇到根据多种不同条件选择，即构成多分支的选择结构。

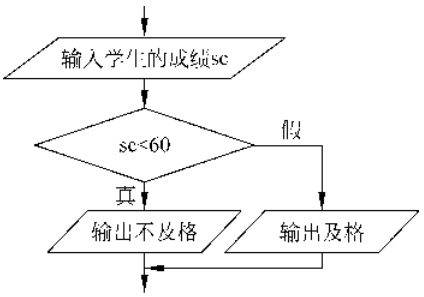

图 3-5　例 3-2 的算法流程图

3. 循环结构

循环结构(Loop Structure)允许在测试条件为真时重复执行某一组语句，可以减少源程序重复书写的工作量。循环结构用来描述被重复执行的程序段，这种结构充分发挥了计算机的特长。

【例 3-3】 考试成绩处理之二——存储全班学生的成绩。

假设全班有 30 名学生，通过键盘输入 30 个学生的成绩，然后将学生的成绩存储在数据文本文件中以便以后查询。用流程图描述该例的算法如图 3-6 所示。

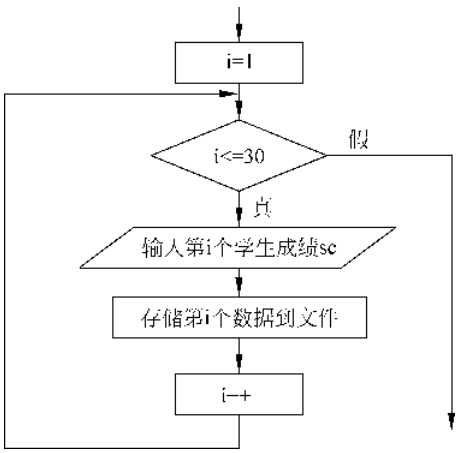

图 3-6　例 3-3 的算法流程图

3.2　构造程序中的条件

选择和循环结构都用到了测试条件(Condition)。条件是能够计算出真或假的表达式，它由关系运算符、逻辑运算符、其他运算符和运算对象构成。

例如，利用海伦公式计算三角形的面积。

解决该问题首先应该输入三角形的三条边。但输入的三个数不一定能构成三角形，所

以需要判断能否构成三角形。若 a,b,c 代表三角形三条边，构成三角形三条边的逻辑表达式应为：$a>0\&\&b>0\&\&c>0\&\&a+b>c\&\&b+c>a\&\&a+c>b$。

3.2.1 关系运算符与关系表达式

程序中经常需要比较两个量的大小关系，以决定程序下一步的工作。比较两个量的运算符称为关系运算符(The Operator of Relations)，由关系运算符将操作数连起来的表达式称为关系表达式(Expressions of Relations)。

1. 关系运算符

关系运算符也称为比较运算符，用于比较两个数据之间的关系。C 语言提供了 6 个关系运算符，均为双目运算符，其含义见表 3-1。

表 3-1 关系运算符的含义

关系运算符	名 称	案 例	含 义
>	大于	x>y	如果 x 大于 y，结果为真(1)；否则结果为假(0)
>=	大于等于	x>=y	如果 x 大于等于 y，结果为真(1)；否则结果为假(0)
<	小于	x<y	如果 x 小于 y，结果为真(1)；否则结果为假(0)
<=	小于等于	x<=y	如果 x 小于等于 y，结果为真(1)；否则结果为假(0)
==	等于	x==y	如果 x 等于 y，结果为真(1)；否则结果为假(0)
!=	不等于	x!=y	如果 x 不等于 y，结果为真(1)；否则结果为假(0)

关系运算符的操作数可以是数值型、字符型、指针类型或枚举类型等。

关系运算符的优先(Priority)级低于算术运算符，高于赋值运算符。其中<，<=，>，>=是同级运算符，==和!=是同级运算符，且前 4 个运算符的优先级高于后 2 个运算符，关系运算符的结合性是自左向右。

注意：==和=是两个不同的符号，后者指赋值运算。

2. 关系表达式

关系表达式是由关系运算符连接起来的表达式，它的一般形式为

表达式 关系运算符 表达式

功能：比较关系运算符两边表达式的值，结果为逻辑值(也称布尔值)。

说明：

(1) 表达式可以是常量、变量、算术表达式或关系表达式等合法 C 语言表达式。

(2) 逻辑值有两个：指定关系式成立时，结果为真，以 1 表示；指定关系不成立时，结果为假，以 0 表示。实际运行过程中，非零数据值表示真，零表示假。

例如：int a=10,b=9,c=8,d=7;，则构成的关系表达式及其值见表 3-2。

表 3-2　关系表达式及其运算结果示例

关系表达式	表达式的值	说　明
a==b+1	真(1)	先计算 b+1,然后 a 和 b+1 比较
a==b	假(0)	a 和 b 进行比较
a>b	真(1)	a 和 b 进行比较
a>=b	真(1)	a 和 b 进行比较
b>a	假(0)	a 和 b 进行比较
a>=b+1	真(1)	先计算 b+1,然后 a 和 b+1 比较
a<=b+1	真(1)	先计算 b+1,然后 a 和 b+1 比较
a!=b	真(1)	a 和 b 进行比较
d=a<b<c	真(1)	相当于 d=((a<b)<c),即 d=(0<c),运算结果：d=1

上例中关系运算符两边操作数的类型都是整型。又如，

```
int a = 2;
float b = 3.4;
```

分析表达式 a<b 和 a+b<b 的结果是多少？

若关系运算符两侧的数据类型不统一，首先将运算符两边操作数的类型转换成相同的数据类型，然后进行比较。所以 a<b 的结果值为真；对于 a+b<b，首先进行类型转换，然后计算 a+b 的值 2.0+3.4=5.4，由于它大于 b，所以表达式的结果值为假。

使用关系表达式要注意以下几点：

(1) 字符变量是以它对应的 ASCII 码值参与运算的。

(2) 不等于用!=表示，而不是常见的<>。

(3) 运算符>=，==，!=，<=用空格分开或将运算符写反都会产生语法错误，例如：= >，= <，= ! 等形式会产生语法错误。但在运算符的两侧增加空格会提高可读性。

(4) 注意实数相等的比较。实际中会有许多实数进行相等的比较，由于实数在内存中的存储误差，因此避免直接使用运算符==和!=对两个实数进行比较。判断两个实数是否相等，一般通过判断它们差的绝对值是否小于一个较小的数来确定。

若 a，b 为实数，检测 $|a-b|<\varepsilon$(ε 为很小的正数，表示 a 和 b 之间的误差)，若该式成立(即结果为真)，则认为 a 与 b 之间误差不超过 ε，近似相等；否则 a 和 b 不相等。ε 可根据实际要求进行调节，ε 越小，a 和 b 之间的误差就越小。例如：

```
fabs(x - y)<1e - 5
```

表示当两个实数 x，y 的差绝对值小于 10^{-5}时，可判断这两个实数相等。其中 fabs 是取绝对值函数，包含在头文件 math.h 中。

(5) 注意区别==和=。如果将 $x==y$ 写成 $x=y$，C 语言会将该表达式作为赋值表达式处理，将 y 的值赋给 x。

3.2.2　逻辑运算符与逻辑表达式

数学中可以用 $1<x<100$ 来表示介于 1 和 100 之间的数。C 语言中 $1<x<100$ 是关系

表达式，不论 1<x 的结果是真或假，0(或 1)<100 是恒为真的。

1. 逻辑运算符

C 语言中有 3 个逻辑运算符：&&(逻辑与，并且)，||(逻辑或，或者)，!（逻辑非，取反)。其中，! 为单目运算符，&& 和|| 为双目运算符。

它们的优先级顺序为：! >&&>||。! 的优先级高于算术运算符，结合性是自右向左。&& 和|| 的优先级低于关系运算符，其结合性为左结合。逻辑运算符及其含义见表 3-3。

表 3-3 逻辑运算符及其含义

逻辑运算符	名 称	目 数	案 例	含 义
!	逻辑非	单目	!a	操作数为真时，结果为假；操作数为假时，结果为真；
&&	逻辑与	双目	a&&b	两个操作数都为真时，结果真；否则结果为假；
||	逻辑或	双目	a||b	两个操作数中有一个为真，结果为真；两个都为假时，结果为假

2. 逻辑表达式

逻辑表达式是由逻辑运算符将逻辑型数据连接起来的式子。C 语言没有逻辑型数据，C 语言将所有非 0 的数据看成逻辑量的逻辑真值(1)，将 0 看成逻辑量的逻辑假值(0)，因此整型、实型、字符型都可以作为逻辑量。逻辑表达式的运算结果只有逻辑真和逻辑假两个。

逻辑运算符常用在选择和循环结构中，用于构造条件，程序根据判断的结果决定程序流程应该如何进行。如：

```
(1) x>1&&x<100              /*判断x值是否介于1和100之间，或位于数轴中间的开区间*/
(2) x>=1&&x<=100            /*判断x值是否介于1和100之间，位于数轴中间的闭区间*/
(3) x<1||x>=100             /*判断x值是否小于1或大于等于100，位于数轴的两端*/
(4) x>=0||y>=0              /*判断x和y值是否都大于等于0，或位于第一象限，含数轴*/
(5) x+y>z&&x+z>y&&y+z>x     /*可用于判断三个数是否满足构成三角形的条件*/
(6) ch>='A'&&ch<='Z'        /*用于判断变量ch是否为大写字母*/
```

例如，判断字符变量是否为英文字母。

(1) 英文字母有大写字母和小写字母两类，在 ASCII 码表中，所有的小写字母是连续的，所有的大写字母也是连续的，但小写字母和大写字母是不连续的。

(2) 对于一个字符变量 c，数学表达式'a'≤c≤'z'或者'A'≤c≤'Z'，说明 c 是一个英文字母。

(3) 将数学中表示英文字母的数学关系式转换为 C 语言的逻辑表达式，即为 C 语言中判断英文字母的表达式：c>='a'&&c<='z'|| c>='A'&&c<='Z'。

注意：*在将数学关系式'a'≤c≤'z'转换为逻辑表达式时，不能将表达式写为'a'<=c<='z'，因为该表达式的值恒为真。请读者特别注意数学关系式与 C 语言逻辑表达式的转换。*

3.2.3　控制条件的描述与表示

例如，构造一个判断三角形属何种类型的控制条件的描述。

数学中，常见的特殊类型的三角形有等腰三角形、等边三角形、直角三角形等。要分析一个三角形属于哪种三角形，应该在三条边可以构成三角形的基础上，继续分析三条边可以构成哪种特殊类型的三角形。

假设用 x，y 和 z 来表示三角形的三条边，则三种特殊类型三角形应满足的数学式如下：

等腰三角形：$x=y$ 或 $x=z$ 或 $y=z$

等边三角形：$x=y=z$

直角三角形：$x^2+y^2=z^2$ 或 $x^2+z^2=y^2$ 或 $y^2+z^2=x^2$

将上述的数学式转换为合法 C 语言表达式，即可以表示各种类型的特殊三角形。下面各表达式是各种三角形的完整条件：

```
(x+y>z&&x+z>y&&y+z>x)&&(x==y||x==z||y==z)          /*等腰三角形*/
(x+y>z&&x+z>y&&y+z>x)&&x==y&&y==z                  /*等边三角形*/
(x+y>z&&x+z>y&&y+z>x)&&(x*x+y*y==z*z||x*x+ z*z ==y*y||y*y+z*z==x*x)
                                                   /*直角三角形*/
```

上面各个表达式中，有些括号可以去掉有些不可以，如去掉第一表达式中的第一对圆括号不会改变逻辑关系，但如果将第二对圆括号去掉，则不再是表示等边三角形的条件。

C 语言中表示条件的表达式很灵活，可以是常量、变量、任何类型的表达式。常见的控制条件有如下几种形式：

(1) 关系表达式：如 a==0，a!=0。常见的书写形式为 expression 等价于 expression!=0，!expression 等价于 expression==0。

(2) 逻辑表达式：如 a>b && c>d 等，为真时，结果为 1。

(3) 算术表达式：如 a+b 等，算术结果不等于 0 时，逻辑结果为 1。

对于更复杂的控制条件，可以使用()和逻辑、关系运算符构造。

例如，整型变量 year 中存放年份的值，构造条件，判断 year 是否为闰年。根据数学知识，闰年的年份满足的条件有两个：

① 如果年份能被 4 整除但不能被 100 整除，则是闰年；

② 年份能被 400 整除，也是闰年。

第一种情况可以用表达式 year%4==0 && year%100!=0 来表示，第二种情况可以用表达式 year%400==0 表示，将两个表达式进行逻辑或运算可以用来完整地表示闰年条件，即(year%4==0 && year%100!=0) || year%400==0。该条件也可以使用表达式：year%4==0&&year%100!=0||year%400==0 或 year%4==0&&year%100||year%400==0 或!(year%4)&&year%100 || !(year%400)等价表示。

注意：

(1)“短路表达式”。在由若干个子表达式组成的逻辑表达式中，从左向右计算，当计算出一个子表达式的值就能确定整个逻辑表达式的值时，此后就不再计算右边剩下的子表达式。我们将这样的表达式称为“短路表达式”。如：

```
int x = 3,y = 0,z = 6;
!x&&(y + 1)&&(z += 2)          /* 表达式!x的值为0,整个表达式即为假 */
y||(z + 3)||(x -= 3)           /* 表达式(z + 3)的值为1,整个表达式即为真 */
```

第一个表达式计算了!x之后整个表达式的值已经确定为假,后面的y+1和z+=2都没有运算,z的值还是6而不是8。

第二个表达式在计算了子表达式z+3时,整个表达式的值已经确定为真,后面的子表达式x-=3没有计算,x的值仍然为3。

(2) 程序设计时,只提倡用关系运算符和逻辑运算符构成的表达式表示逻辑值,而不提倡使用赋值表达式以及算术运算符组成的表达式表达逻辑值。

(3) C语言的关系表达式与数学上的比较运算的表达式不完全一样,要注意区分,并将数学上的运算转化为合法的C语言关系表达式。

例如,判断一个变量的值是否在12到30之间,数学表达式为:12<a<30,把这样的表达式放在程序里编译一下,没有什么不正常的,编译通过。但是在运行的时候就会出问题。

正确的写法应该是:

```
(12<a)&&(a<30)   /* 变量a的值大于12并且小于30 */
```

3.3 选择结构程序设计

C语言利用关系表达式和逻辑表达式来构筑一些复杂的控制条件。这种根据控制条件选择程序中某一部分语句执行的结构叫选择结构(Selection Structure)。选择结构体现了程序的判断能力。选择结构分为单分支、双分支和多分支。C语言中提供了if和switch两种用于实现分支选择结构的控制语句。if语句主要用于单分支、双分支和多分支条件判断结构,而switch语句则用于多分支选择结构。

3.3.1 选择结构语句(Conditional Statements)

先来看这样一个问题,计算分段函数:

$$y=\begin{cases}3-x & x\leqslant 0\\ 2/x & x>0\end{cases}$$

求解问题的流程如下:

(1) 输入 x;

(2) 如果 $x\leqslant 0$ 则 $y=3-x$,否则 $y=2/x$;

(3) 输出 y 的值。

显然程序的流程必须由 x 的值确定,这是分支选择结构,分支的依据是根据表达式x<=0的值做出判断,以决定执行哪个语句。

if语句允许测试控制条件,然后根据控制条件的真假来选择执行语句。选择结构有“if单分支”和“if-else双分支”两种基本形式,语句见表3-4。

表 3-4　基本选择语句

单分支 if 语句	双分支 if-else 语句
if(condition)　statement	if(condition)　statement1　else　statement2

语句中的 condition 是控制条件表达式，圆括号()是必不可少的；statement 在语法上是一条语句，可是任何一条 C 语言语句或一个复合语句。特别地，当 statement 又是分支语句时，称为分支结构嵌套。C 语言标准规定，编译程序必须能支持至少 15 层分支嵌套。

1. 最简单的 if 语句——单分支 if 语句

单分支 if 语句(The if Statement)功能：当控制条件表达式 condition 值为真时，执行 statement，否则跳过该语句，继续执行后续语句。单分支语句执行流程如图 3-7 所示。

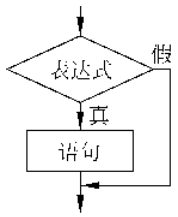

图 3-7　单分支语句流程图

【例 3-4】 某学生在“成绩管理系统”中查询成绩，软件系统要显示学生成绩，同时对成绩高于(包括)90 分的学生表示庆贺。写出程序段。

分析：

这个问题只对符合条件的学生表示庆贺，但所有学生的成绩查询都要应答。可以用单分支语句实现。

```
⋮
if(grade>=90)                          /*检测条件为成绩高于(包括)90分*/
    printf("congratulations! \n");     /*条件满足表示庆贺，不满足就跳过该语句*/
printf("Your grade is %d.\n",grade);   /*与单分支 if 并列，所有学生均能查询成绩*/
⋮
```

也可以这样书写该语句：

```
if(grade>=90) printf("congratulations! \n");      /*单分支语句写为一行*/
printf("Your grade is %d.\n",grade);
```

单分支语句写为一行，程序更易阅读。如果 statement 是复合语句，分行写更好。

注意，if 后表达式必须带括号。例如，

```
if b == a   area = a * a;      /*缺少括号，是错误的*/
```

【例 3-5】 用“假设思想”法求任意两数中的较大数(**必记算法**)。

分析：

“假设思想”算法是指在程序中先假设某事物成立，然后判断事实是否如此，如果事实和假设的条件不符，则修正条件。

设实型变量 x，y 代表两任意数，实型变量 max 表示较大的数。

首先，假设较大的数是 max=x(假设某种情况)；然后比较 max(即 x 的值)和 y 的值(判断假设的正确性)，如果 max<y 成立(反假设：假设的情况不成立)，说明假设错误，则修改假设情况：max=y；否则假设正确。较大的数总是 max。

求 3 个数或更多数中的最大数或最小数都可以使用该算法。本题程序如下：

```
/*program ch3-5.c*/
#include<stdio.h>
```