# 第3章 chapter 3

# 选择结构

顺序结构、选择结构和循环结构是程序设计的三种基本结构，掌握了这三种基本结构，便可以设计解决任何复杂问题的程序。顺序结构是按书写顺序执行语句；选择结构是有条件地选择要执行的程序段；循环结构是有条件地重复执行循环体的程序段。前两章涉及的程序均为顺序结构的实例，本章介绍选择结构。

**内容导读：**

- 认识C语言的语句分类。
- 掌握关系运算符、逻辑运算符、条件运算符和表达式。
- 掌握选择结构的if语句，包括if语句的单分支、双分支、多分支和嵌套结构。
- 掌握选择结构的switch语句和条件表达式。

## 3.1 C语言语句分类

C程序的执行部分是由语句组成的，程序的功能也是由执行语句实现的，C语言语句分类如图3-1所示。

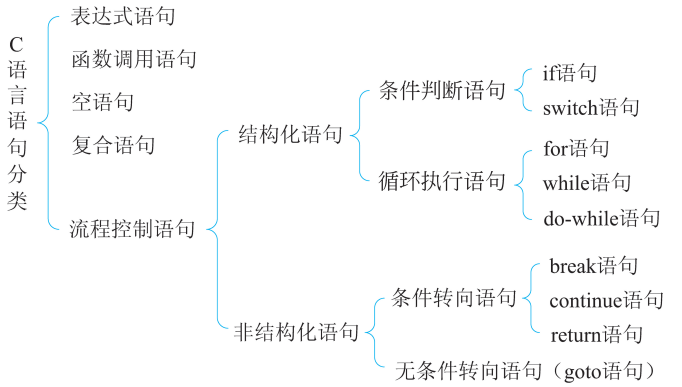


图3-1 C语言语句分类

### 1. 表达式语句

表达式语句是指在表达式后面加一个分号所构成的语句，其格式如下：

```
表达式;
```

C语言中有实际使用价值的表达式语句主要有三种：

（1）赋值语句。例如：

```
sum = a + b;
```

（2）自增、自减运算符构成的表达式语句。例如：

```
i++;
```

（3）逗号表达式语句。例如：

```
x = 1, y = 2;
```

### 2. 函数调用语句

函数调用语句是指在函数调用后面加一个分号所构成的语句，其格式如下：

```
函数名([参数表]);
```

例如：

```
printf("a=%d, b=%d\n", a, b);          //printf()是标准屏幕输出函数
x = sqrt(9.0);                         //sqrt()是求平方根的数学库函数
```

### 3. 空语句

空语句就是一个单独的分号，在语法上是一条语句。

空语句什么也不做，但是也要像其他普通语句一样被执行。在实际应用中，空语句放在循环体里可实现延时的功能。例如：

```
for(i = 1; i <= 100; i++)
{
    ;                                  //循环体是空语句
}
```

以上代码段使用了循环结构（详见第4章），执行100次空循环，实现一定时间的延时。

**注意：**虽然空语句什么都不做，但是在程序中不能随意添加，如果在不适宜的地方添加了空语句，会引起语法错误或者逻辑错误，具体注意事项参看本书选择结构和循环结构中的有关说明。

### 4. 复合语句

复合语句是指用花括号将一条或若干语句包含起来的语句，其格式如下：

```
{
    [数据说明部分]
    执行语句;
}
```

例如：

```
{
    double l, s;                   //此变量为复合语句内的局部变量，作用域仅在复合语句内
    l = 2 * 3.14159 * r;
    s = 3.14159 * r * r;
}
```

**注意：**

(1) 复合语句在语法上是一条语句，不能看作是多条语句。

(2) 复合语句常用于流程控制语句(如 if 语句、switch 语句、for 语句、while 语句、do-while 语句)中，由于这些流程控制语句都只能自动结合一条语句，因此它们的语句体常写为复合语句的形式(流程控制语句是指选择结构和循环结构的语句)。

(3) 复合语句内的各条语句都必须以分号结尾。

**5. 流程控制语句**

流程控制语句用于实现程序的各种结构方式，控制程序的流向。

流程控制语句有九种，分为以下三类：

(1) 条件判断语句：if 语句、switch 语句。

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

(3) 转向语句：break 语句、continue 语句、return 语句、goto 语句(该语句不提倡使用)。

C 语言提供顺序、选择和循环三种结构化语句来控制程序的执行流程。

## 3.2 条件判断表达式的设计

C 语言提供了关系运算符和逻辑运算符，用于书写具有条件选择的表达式，从而实现程序的选择结构和循环结构，构成流程控制语句。

关系运算和逻辑运算的结果只能是逻辑值真或假，由于 C 语言中没有逻辑型数据，于是使用 **1 表示逻辑真，0 表示逻辑假**，因此关系表达式和逻辑表达式的值只有 1 和 0 两种结果。

### 3.2.1 关系运算符及表达式

**1. 关系运算符**

关系运算是对两个运算对象进行大小比较的运算，其实际是比较运算。C 语言提供了 6 个**关系运算符(relation operator)**，如表 3-1 所示。

表 3-1 关系运算符

| 运 算 符 | 名 称 | 优 先 级 | | 运算对象的数量 | 结 合 性 |
|---|---|---|---|---|---|
| > | 大于 | 较高 | 优先级相同 | 双目 | 自右向左 |
| >= | 大于或等于 | | | | |
| < | 小于 | | | | |
| <= | 小于或等于 | | | | |
| == | 等于 | 较低 | 优先级相同 | | |
| != | 不等于 | | | | |

**2. 关系表达式**

以下为关系表达式的一般格式：

**表达式 1　关系运算符　表达式 2**

以下是一些关系表达式的实例，假设有定义：

```
int a = 3, b = 5, c = 15;
```

a ＞ b 的值为 0(假)。

a !＝ b 的值为 1(真)。

a * b ＞＝ c 的值为 1(真)。

c % 5 ＝＝ 0 的值为 1(真)，该表达式用于判断变量 c 的值能否被 5 整除。

**敲重点：**

(1) 关系表达式的值只有 1(真)和 0(假)两种结果。

(2) 注意区分赋值运算符＝和关系运算符＝＝，表 3-2 对二者进行了比较。

表 3-2　比较赋值运算符＝与关系运算符＝＝

| 比较内容 | ＝ | ＝＝ |
|---|---|---|
| 名称 | 赋值号，属于赋值运算符 | 等于号，属于关系运算符 |
| 功能 | 将右侧表达式的值赋给左侧的变量 | 比较左、右两侧表达式的值是否相等 |
| 运算结果 | 赋值表达式的值是赋值号右侧表达式的值 | 关系表达式的值只有 1(真)和 0(假)两种结果 |
| 举例 | 假设有定义<br>int a ＝ 0，b ＝ 1，c ＝ 2；<br>① c ＝ a ＋ b<br>//将 a+b 的值赋给变量 c<br>② a ＋ b ＝ c<br>//表达式错误，赋值号左侧只能是变量 | 假设有定义<br>int a ＝ 0，b ＝ 1，c ＝ 2；<br>① c ＝＝ a ＋ b<br>//比较 c 和 a ＋ b 的值是否相等<br>② a ＋ b ＝＝ c<br>//比较 c 和 a ＋ b 的值是否相等 |

## 3.2.2　逻辑运算符及表达式

### 1. 逻辑运算符

C 语言提供了三个**逻辑运算符(logic operator)**，其运算规则如表 3-3 所示。

表 3-3　逻辑运算符的运算规则

| 运算符 | 名称 | 运 算 规 则 | 优先级 | 操作数 | 结合性 |
|---|---|---|---|---|---|
| ! | 逻辑非 | ! 真 ＝ 0，! 假 ＝ 1 | 最高 | 单目 | 自右向左 |
| && | 逻辑与 | 真 && 真＝ 1，真 && 假＝0，假 && 真＝0，假 && 假＝0 | 其次 | 双目 | 自左向右 |
| \|\| | 逻辑或 | 真 \|\| 真＝ 1，真 \|\| 假＝1，假 \|\| 真＝1，假 \|\| 假＝0 | 最低 | | |

表 3-3 中的真表示非 0，非 0 的值参与逻辑运算都视为真；假表示 0。而逻辑运算的结果用 1 表示真，0 表示假。

### 2. 逻辑表达式

以下为逻辑表达式的一般形式：

```
!表达式
表达式 1 && 表达式 2
表达式 1 || 表达式 2
```

以下是一些逻辑表达式的实例，假设有定义：

```
int a = 0, b = 50, c =100;
```

！a 的值为 1，该表达式判断变量 a 的值是否为 0。

(a <= b) && (b <= c)的值为 1，该表达式判断变量 b 的值是否介于 a、c 之间。

(b % 4 == 0) || (b % 5 == 0)的值为 1，该表达式判断变量 b 的值是否能被 4 或 5 整除。

**敲重点：**

(1) 逻辑表达式的值只有 1(真)和 0(假)两种结果。

(2) 逻辑与(&&)、逻辑或(||)的结合性都是自左向右，一般情况，运算时先计算左侧表达式，再计算右侧表达式。但是运算符 && 和 || 具有特殊的短路特性，即右侧表达式不一定被执行，其是否被执行取决于左侧表达式的值。

- 逻辑与(&&)的短路特性：当左侧表达式值为假时，可确定逻辑表达式值为假，于是右侧表达式不被执行，称右侧表达式被“短路”了。
- 逻辑或(||)的短路特性：当左侧表达式值为真时，可确定逻辑表达式值为真，于是右侧表达式不被执行，称右侧表达式被“短路”了。

假设有定义：

```
int a = 1, b = 2;
```

a == 0 && ++b 的值为假，右侧表达式++b 被“短路”未执行，因此变量 b 的值仍然为 2。

a == 1 || ++b 的值为真，右侧表达式++b 被“短路”未执行，因此变量 b 的值仍然为 2。

### 3.2.3 关系表达式和逻辑表达式常见错误

关系表达式和逻辑表达式多用于选择结构和循环结构的条件判断，对初学者来说，能够正确书写各种复杂表达式，是程序正确执行的关键步骤。表 3-4 举例了一些错误的关系表达式或逻辑表达式，这些错误不易察觉，时常引起程序的逻辑错误，而编译系统不会报错，但最终程序运行的结果却是错误的。

表 3-4 常见关系表达式和逻辑表达式错误举例

| 描　　述 | 表达式及解析 |
| --- | --- |
| 判断变量 x 中的字符是不是大写字母？假设有定义<br>char x = 'a'; | 错误的表达式：'A' <= x <= 'Z'或 65 <= x <= 90<br>解析：第一步执行 'A' <= x，值为 1(真)；第二步执行 1 <= 'Z'，值为 1(真)。第二步本应执行 x <= 'Z'，却因少写了 &&，导致运行结果错误 |
| | 正确的表达式：x >= 'A' && x <= 'Z'或 x >= 65 && x <= 90 |

续表

| 描　述 | 表达式及解析 |
| --- | --- |
| 判断变量 score 的值是不是一个百分制分数？假设有定义<br>int score = 101; | 错误的表达式：0 <= score <= 100<br>解析：第一步执行 0 <= score，值为 1(真)，第二步执行 1 <= 100，值为 1(真)，结果错误 |
| | 正确的表达式：(0 <= score) && (score <= 100) |
| 判断变量 a、b、c 能否构成三角形的三边？假设有定义<br>int a = 1, b = 2, c = 3; | 错误的表达式：(a + b > c) \|\| (a + c > b) \|\| (b + c > a)<br>解析：构成三角形的条件是任意的两边之和均要大于第三边，因此以下三个表达式 a+b>c、a+c>b、b+c>a 的关系是逻辑与 |
| | 正确的表达式：(a + b > c) && (a + c > b) && (b + c > a) |
| 判断变量 a、b、c 能否构成等边三角形？假设有定义<br>int a = 2, b = 2, c = 1; | 错误的表达式：a == b == c 或 a = b = c<br>解析：若写成 a == b == c，则第 1 步执行 a == b，值为 1(真)；第 2 步执行 1 == c，值为 1(真)，结果错误。若写成 a = b = c，则是将== 误用为=，同时缺少 && |
| | 正确的表达式：(a == b) && (a == c)　　或　　(a == b) && (b == c) |
| 判断变量 a、b、c 能否构成等腰三角形？假设有定义<br>int a = 1, b = 2, c = 3; | 错误的表达式：a == b<br>解析：构成等腰三角形的条件是任意两条边相等，需以逻辑或的形式将以下三个表达式 a==b、a==c、b==c 连接起来 |
| | 正确的表达式：(a == b) \|\| (a == c) \|\| (b == c) |

## 3.3 if 语 句

**选择结构又称分支结构，选择结构中最常用的语句是 if 语句，if 语句包含以下四种基本形式：单分支 if 语句、双分支 if 语句、多分支 if 语句、if 语句的嵌套结构。**

单分支 if 语句——仅对判断条件为真的情况进行处理。

双分支 if 语句——选择执行判断条件为真或假两种情况当中的一个。

多分支 if 语句——有多个条件需要判断，自上而下依次判断各条件，当某个条件为真时，就执行该条件对应的分支，其余分支不再执行。

if 语句的嵌套结构——有多个条件需要判断，条件之间有层次关系，判断有先后。

### 3.3.1 单分支 if 语句

#### 1. 单分支 if 语句的一般形式

单分支 if 语句是最简单的条件判断语句，以下为单分支 if 语句的一般形式：

```
if (表达式)
    语句 A;
```

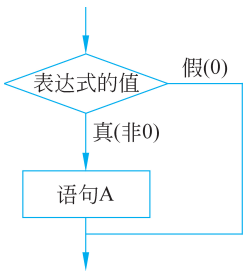


图 3-2　单分支 if 语句流程图

单分支 if 语句的执行流程如图 3-2 所示，先执行表达式，判断其值如果为真(非 0)就执行语句 A；如果为假(0)就跳过语

句 A。

**敲重点：**

(1) if(表达式) 中的表达式可以是任何符合 C 语言语法的表达式，其值为非 0 表示真；其值为 0 表示假。

(2) if(表达式) 只能自动结合一条语句，如果有多条语句，必须用花括号括起来构成复合语句(复合语句在语法上相当于一条语句)，书写形式如下：

```
if(表达式)
{
    语句 A;
}
```

为使程序层次清晰，花括号内的语句都向后缩进。注意：C 语言在语法上是用花括号界定语句间的从属关系，缩进只是在书写上更好地体现语句间的层次感，有助于提高程序的可读性。

(3) 如果语句 A 仅有一条语句，可以省略花括号。特别地，建议无论语句 A 包含多少条语句，都添加花括号，这是一个良好的编程习惯，对初学者而言，添加花括号能避免很多语法错误及逻辑错误。

### 2. 单分支 if 语句举例

**【例 3.1】** 输入两个整数，求其中的较大数。

思路：定义整型变量 a、b 存放两个数，选出较大数放到变量 a 中。于是当 a >= b 时，不做任何处理；当 a < b 时，将 b 的值赋给 a。仅对条件 a < b 为真的情况进行处理，属于典型的单分支选择结构，程序流程如图 3-3 所示。


单分支 if 语句

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int  a, b;
5.      printf("请输入两个整数:");
6.      scanf("%d, %d", &a, &b);
7.      if(a < b)
8.      {
9.          a = b;
10.     }
11.     printf("较大数:%d\n",  a);
12.     return 0;
13. }
```

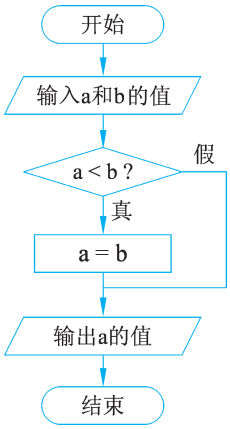

图 3-3 例 3.1 的流程图

程序运行结果：

```
请输入两个整数：2, 7
较大数：7
```

```
请输入两个整数：7, 2
较大数：7
```

### 3. 单分支 if 语句的常见错误

**(1) if(表达式)后面误加分号。**

刚接触 if 语句，不少初学者会在 if(表达式)后面加分号，由于一个单独的分号属于一条空语句，将被 if 结合，导致本来属于 if 的语句体不能被 if 结合，引起逻辑错误。

假设例3.1的第7～10行代码写成如下形式：

```
if(a < b) ;                                    //此分号引起逻辑错误
{
    a = b;
}
```

分析：if(a ＜ b)自动结合了后面多余的分号，而没有结合语句“a ＝ b;”，于是语句“a ＝ b;”的执行不再受条件a ＜ b的控制。此时引起了程序的逻辑错误，但是编译系统不会报错。遇到逻辑错误，可使用单步调试法定位错误。

**（2）if语句体需要加花括号时却没有加花括号。**

C语言使用花括号界定语句的从属关系，当if语句体中有多条语句时，必须用花括号将多条语句界定为一条复合语句，从而能够被if(表达式)自动结合。但是很多初学者总是忘记添加花括号，导致程序出现逻辑错误。

修改例3.1的代码，思路：判断当a ＜ b时，就交换a、b的值，保证较大值总是放在变量a中。修改的代码如下，注意第7～10行的改变。

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int  a, b, t;
5.      printf("请输入两个整数:");
6.      scanf("%d,%d", &a, &b);
7.      if(a < b)
8.      {
9.          t = a;      a = b;      b = t;   //交换a、b的值
10.     }
11.     printf("较大数:%d\n",  a);
12.     return 0;
13. }
```

以上代码的if语句体中有三条语句，必须添加花括号将三条语句括起来。

请思考，如果没有添加括号，写成如下形式：

```
if(a < b)
    t = a;     a = b;     b = t;
```

程序运行结果正确吗？编译系统会报错吗？会出现语法错误还是逻辑错误？

## 3.3.2 双分支if语句

### 1. 双分支if语句的一般形式

双分支if语句是由某个条件的两种取值(真或假)构建两个分支，即if-else语句，任何时候仅执行其中一个分支，形成二选一的结构。以下为双分支if语句的一般形式：

```
if (表达式)
    语句A;
else
    语句B;
```

双分支 if 语句的执行流程如图 3-4 所示，先执行表达式，判断其值如果值为真（非 0）就执行语句 A；如果为假（0）就执行语句 B。由于表达式的值非真即假，因此语句 A 和语句 B 不会同时执行，也不会都不执行，而是二选一执行。

**敲重点：**

（1）关键字 if 后面必须有表达式，而关键字 else 后面没有表达式。

（2）if 分支和 else 分支都是只能自动结合一条语句，当有多条语句时，必须加花括号构成复合语句。书写形式如下：

```
if(表达式)
{
    语句 A;
}
else
{
    语句 B;
}
```

（3）else 不能单独存在，必须有对应的 if 与之配套使用，即 if 和 else 应成对出现。

### 2. 双分支 if 语句举例


双分支 if 语句

**【例 3.2】** 要求同例 3.1，求两个整数中的较大数，使用双分支 if 语句实现。

思路：判断 a、b 的大小关系，如果变量 a 的值大就输出 a，否则就输出 b，即 a 和 b 二选一输出，该方法是双分支选择结构，流程图如图 3-5 所示。

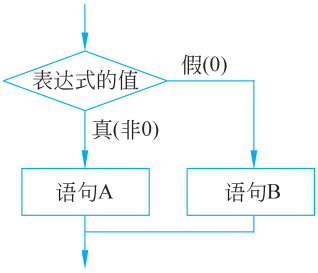

图 3-4 双分支 if 语句流程图

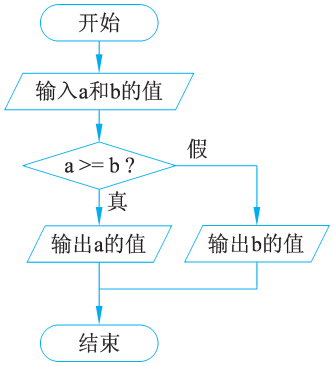

图 3-5 例 3.2 的流程图

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int  a, b;
5.      printf("请输入两个整数:");
6.      scanf("%d, %d", &a, &b);
7.      if(a >= b)
8.      {
9.          printf("较大数:%d\n",  a);
10.     }
11.     else
12.     {
```

```
13.         printf("较大数:%d\n",  b);
14.     }
15.     return 0;
16. }
```

程序运行结果：

```
请输入两个整数：2, 7
较大数：7
```

```
请输入两个整数：7, 2
较大数：7
```

【例 3.3】 输入一个年份，判断是否为闰年。


判断闰年

思路：判断闰年的条件是，能被 4 整除同时不能被 100 整除的年份，或者能被 400 整除的年份。一个年份要么是闰年，要么是平年，属于典型的双分支选择结构。

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int year;
5.      printf("请输入一个年份:");
6.      scanf("%d", &year);
7.      if((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)        //判断闰年
8.      {
9.          printf("%d年是闰年\n", year);
10.     }
11.     else
12.     {
13.         printf("%d年是平年\n", year);
14.     }
15.     return 0;
16. }
```

程序运行结果：

```
请输入一个年份：2020
2020年是闰年
```

```
请输入一个年份：2022
2022年是平年
```

### 3. 双分支 if 语句的常见错误

**（1）if 或 else 分支需要加花括号时却没有加花括号。**


if 语句漏加花括号的常见错误

【例 3.4】 输入两个整数，判断并输出两个数中的较大数和较小数。

流程图如图 3-6 所示。

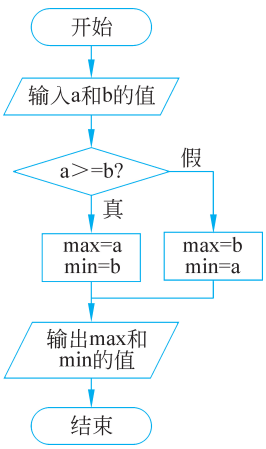

图 3-6 例 3.4 的流程图

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int  a, b, max, min;
5.      printf("请输入两个整数:");
6.      scanf("%d, %d", &a, &b);
7.      if(a >= b)
8.      {
9.          max = a;
10.         min = b;
```

```
    }
    else
    {
        max = b;
        min = a;
    }
    printf("较大数:%d,较小数:%d\n", max, min);
    return 0;
}
```

程序运行结果：

```
请输入两个整数：2, 7
较大数：7，较小数：2
```

```
请输入两个整数：7, 2
较大数：7，较小数：2
```

例 3.4 代码的第 7～16 行是 if-else 语句，if 分支和 else 分支都有两条语句，两个分支都必须添加花括号。思考：如果将以上代码第 7～16 行写成如下不加花括号的形式，将引起语法错误还是逻辑错误？

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

分析：以上漏写花括号的代码将引起语法错误，因为 if 和 else 之间被放置了两条语句，破坏了 if 和 else 的整体性，使得 else 没有 if 与之配套使用。

思考：以上代码缺少花括号，但保留了语句的缩进，缩进能体现语句间的从属关系吗？

**（2）if(表达式)后面误加分号。**

双分支 if 语句的 if(表达式)的后面不能加分号，如果误加，将导致语法错误。例如：

```
if(a >= b) ;                          //此分号引起语法错误
    printf("较大值:%d\n",  a);
else
    printf("较大值:%d\n",  b);
```

此时 if(a >= b) 后面相当于有两条语句：一条是空语句；另一条是 printf() 语句，使得 else 没有 if 与之配套使用，编译系统将报错。

**（3）else 后面误加分号。**

关键字 else 后面不能加分号，误加会导致逻辑错误。例如：

```
if(a >= b)
    printf("较大值:%d\n",  a);
else ;                                //此分号引起逻辑错误
    printf("较大值:%d\n",  b);
```

此时 else 自动结合空语句，使得本应从属于 else 的语句“printf("较大值：%d\n", b);”被分隔开，编译系统不会报错，属于逻辑错误。

### 3.3.3 多分支 if 语句

#### 1. 多分支 if 语句的一般形式

多分支 if 语句是对某个条件的多种取值情况进行判断，每个情况对应一个分支，构建为 if-else if-else 语句，任何时候仅执行其中一个分支，形成多选一的结构。以下为多分支 if 语句的一般形式：

```
if (表达式 1)
    语句 1;
else if (表达式 2)
    语句 2;
…
else if (表达式 n-1)
    语句 n-1;
else
    语句 n;
```

多分支 if 语句的执行流程如图 3-7 所示，首先判断表达式 1，若值为真就执行语句 1 并结束多分支结构，反之继续判断表达式 2；若表达式 2 的值为真就执行语句 2 并结束多分支结构，反之继续判断表达式 3；依次类推。

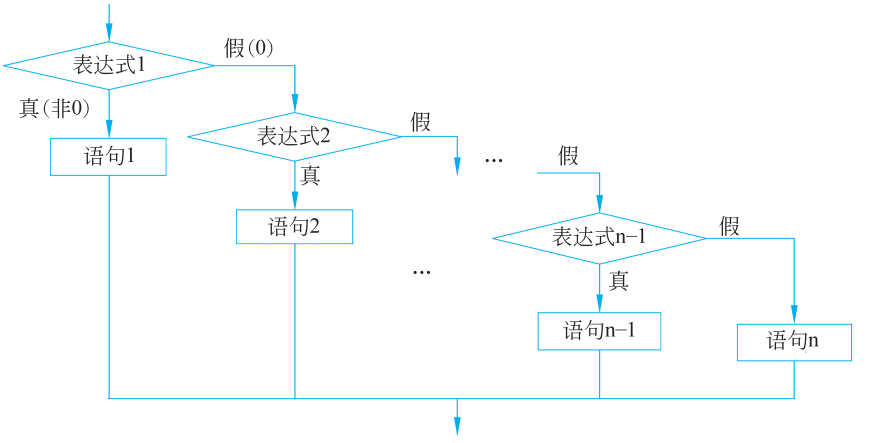


图 3-7 多分支 if 语句流程图

#### 2. 多分支 if 语句举例

【例 3.5】 输入一个成绩，如果介于 60～100 分，输出“成绩合格”；如果介于 0～59 分，就输出“成绩不及格”。同时，判断输入的成绩是否为有效的百分制分数。

思路：根据题意，将输入的数值范围分为以下三个区间 0～59、60～100、小于 0 或大于 100，对应三个分支进行判断，因此使用多分支 if 语句，程序流程如图 3-8 所示。


多分支 if 语句

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      float  score;
5.      printf("请输入一个成绩:");
```

```
6.      scanf("%f", &score);
7.      if(score < 0 || score > 100)
8.      {
9.          printf("成绩超范围\n");
10.     }
11.     else if(score >= 60)
12.     {
13.         printf("成绩合格\n");
14.     }
15.     else
16.     {
17.         printf("成绩不及格\n");
18.     }
19.     return 0;
20. }
```

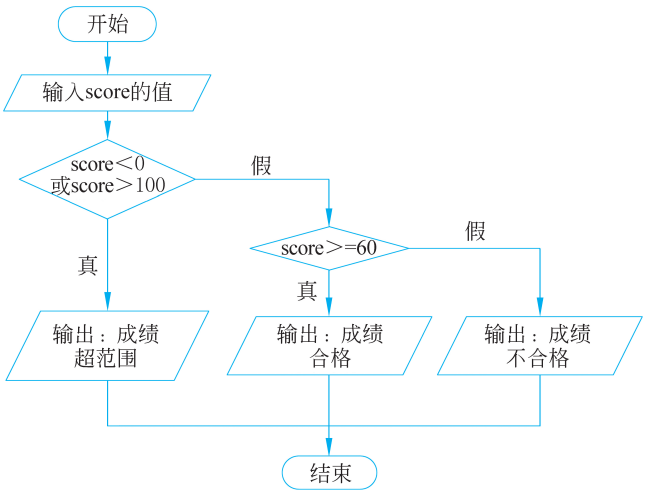


图 3-8 例 3.5 的流程图

程序运行结果：

```
请输入一个成绩：103
输入成绩超出范围
```

```
请输入一个成绩：86
成绩合格
```

```
请输入一个成绩：45
成绩不及格
```

### 3. 注意区分多分支 if 语句和多个单分支 if 语句

多分支 if 语句的各分支之间是互斥的、多选一的关系，其中一个分支被执行后，其他分支将不会被执行。

多个单分支 if 语句的各 if 分支之间是无关联的、并列的关系，其中一个 if 分支执行与否，不会影响其他 if 分支。

**【例 3.6】** 输入三个整数到变量 a、b、c 中，判断它们的大小关系，按从大到小的顺序将三个数依次放到变量 a、b、c 中。

思考：阅读以下程序一、程序二，分析哪个程序能完成题目要求？为什么？

程序一：多个单分支 if 语句

```
1.  #include <stdio.h>
2.  int main()
```

区分多个单分支 if 语句和多分支 if 语句

```
3.  {
4.      int a, b, c, t;
5.      printf("请输入三个整数:");
6.      scanf("%d%d%d", &a, &b, &c);
7.      if(a < b)
8.      {
9.          t = a;  a = b;  b = t;
10.     }
11.     if(a < c)
12.     {
13.         t = a;  a = c;  c = t;
14.     }
15.     if(b < c)
16.     {
17.         t = b;  b = c;  c = t;
18.     }
19.     printf("按由大到小的顺序输出三个数:%d  %d  %d\n", a, b, c);
20.     return 0;
21. }
```

程序二：多分支 if 语句

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int a, b, c, t;
5.      printf("请输入三个整数:");
6.      scanf("%d%d%d", &a, &b, &c);
7.      if(a < b)
8.      {
9.          t = a;  a = b;  b = t;
10.     }
11.     else if(a < c)
12.     {
13.         t = a;  a = c;  c = t;
14.     }
15.     else if(b < c)
16.     {
17.         t = b;  b = c;  c = t;
18.     }
19.     printf("按由大到小的顺序输出三个数:%d  %d  %d\n", a, b, c);
20.     return 0;
21. }
```

分析：使用多个单分支 if 语句的程序一能够完成题目要求，三个数的大小关系需要经过两两比较才能确定，即（a，b）、（a，c）、（b，c）三个组合都需要被比较，使用三个 if 语句依次完成，因此该题不能使用多分支 if 语句设计程序。

### 3.3.4 if 语句的嵌套结构

#### 1. if 语句的嵌套结构一般形式

如果判断条件之间有层次性，当某一条件满足时，才能进一步判断其他条件，这便是

嵌套结构。单分支、双分支、多分支 if 语句之间可以相互嵌套，内层 if 语句既可以嵌套在 if 子句中，也可以嵌套在 else 子句中。以下列出了三种 if 语句的嵌套结构。

```
if(表达式 1)
{
    if(表达式 2)
      …
    else
      …
}
```

```
if(表达式 1)
{
     …
}
else
{
    if(表达式 2)
      …
    else
      …
}
```

```
if(表达式 1)
{
    if(表达式 2)
      …
    else if(表达式 3)
      …
    else if(表达式 4)
      …
}
else
{
    …
}
```

### 2. if 语句的嵌套结构举例


if 语句的嵌套结构

【例 3.7】 输入一个成绩，输出该成绩对应的级别。0～59 分为“成绩不及格”，60～79 分为“成绩中等”，80～89 分为“成绩良好”，90～100 分为“成绩优秀”，同时判断输入成绩是否为有效的百分制分数。

思路：首先判断输入成绩是否为有效的百分制分数。如果是，再对成绩进行四个等级的判断；反之就输出提示信息。本题使用嵌套结构编程，外层为双分支 if 语句，内层为多分支 if 语句。程序流程如图 3-9 所示。

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int score;
5.      printf("请输入一个成绩: ");
6.      scanf("%d", &score);
7.      if(score >= 0 && score <= 100)      //外层:判断是否为百分制分数
8.      {
9.          if(score<60)                    //内层:将成绩分为不同等级
10.         {
11.             printf("成绩不及格\n");
12.         }
13.         else if(score<80)
14.         {
15.             printf("成绩中等\n");
16.         }
17.         else if(score<90)
18.         {
19.             printf("成绩良好\n");
20.         }
21.         else
22.         {
23.             printf("成绩优秀\n");
24.         }
25.     }
26.     else
```

```
27.     {
28.         printf("无效的成绩\n");
29.     }
30.     return 0;
31. }
```

程序运行结果：

```
请输入一个成绩：98
成绩优秀
```

```
请输入一个成绩：82
成绩良好
```

```
请输入一个成绩：75
成绩中等
```

```
请输入一个成绩：50
成绩不及格
```

```
请输入一个成绩：-78
无效的成绩
```

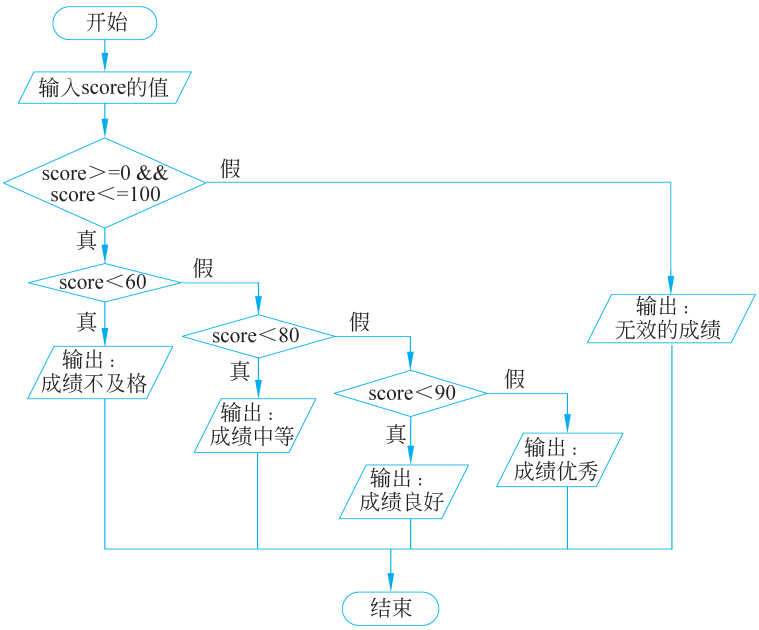


图 3-9　例 3.7 的流程图

### 3. if 语句的嵌套结构的就近配对原则

if 语句的嵌套结构不是刻意去追求的，是在解决问题的过程中随实际需要而采用的。嵌套结构的语句层次相对复杂，一般应为各子句添加花括号以增加程序的可读性，同时注意语句的缩进，通过书写格式更好地体现语句间的从属关系。

编程时如果在嵌套结构中缺省花括号，系统会根据就近配对原则进行各分支的匹配。就近配对原则：else 子句总是与前面距离它最近的但是又不带其他 else 子句的 if 子句配对使用，与书写格式无关，如图 3-10 所示。

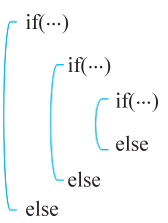


图 3-10　就近配对原则

如果将例 3.7 程序中第 7～29 行的花括号省去，写成图 3-11(a)的格式，系统将根据就近配对原则划分语句的从属关系，仍然是外层双分支、内层四个分支。然而，缺少花括号的代码，语句层次变得不明了，不恰当的缩进还会影响对程序的正确理解。

如果将图 3-11(a)代码中的“else　printf("成绩优秀\n");”修改为

图 3-11(b)的“else if(score>=90) printf("成绩优秀\n");”,根据就近配对原则,语句的嵌套结构被改变了,变成外层单分支、内层五个分支。可见,缺省花括号既不利于程序的理解,又可能在不经意间改变程序的结构。

```
if(score >= 0 && score <= 100)
if(score<60)
      printf("成绩不及格\n");
else if(score<80)
      printf("成绩中等\n");
else if(score<90)
      printf("成绩良好\n");
else
      printf("成绩优秀\n");
else
      printf("无效的成绩\n");
```

(a)

```
if(score >= 0 && score <= 100)
      if(score<60)
            printf("成绩不及格\n");
      else if(score<80)
            printf("成绩中等\n");
      else if(score<90)
            printf("成绩良好\n");
else if(score>=90)
      printf("成绩优秀\n");
else
      printf("无效的成绩\n");
```

(b)

图 3-11 if 语句嵌套结构缺省花括号的不良书写习惯示例

因此,在这里再次强调,为子句添加花括号,以及将内层语句向后缩进是良好的编程习惯,初学者应注意培养自己养成这样的习惯,良好的编程习惯能够让语句间的逻辑关系一目了然,便于他人理解自己编写的程序,同时也有利于自己维护程序。

## 3.4 switch 语句

### 1. switch 语句的一般形式

switch 语句是一种结构整齐的多分支选择结构,以下为 switch 语句的一般形式:

```
switch(表达式)
{
    case 常量表达式 1:  <语句体 1;>  <break;>
    case 常量表达式 2:  <语句体 2;>  <break;>
        …
    case 常量表达式 n:  <语句体 n;>  <break;>
    <default: 语句体 n+1;>
}
```

switch 语句的执行流程如图 3-12 所示。其执行过程:首先计算 switch 表达式的值,将该值与各个 case 后的常量表达式的值进行比较,如果 switch 表达式的值与某个 case 常量表达式的值相等,就执行其后的语句体。如果语句体后面有 break 语句,则结束 switch 语句;如果没有 break 语句,则继续执行后续的 case 分支。如果 switch 表达式的值与所有 case 常量表达式的值都不相等,就执行 default 分支。以上过程可比喻为一个多路开关,当某个结点的开关闭合时,电流就从这个结点流过,因此 switch 语句也称开关语句。

说明:switch 语句格式中加角括号(< >)的地方表示可缺省。

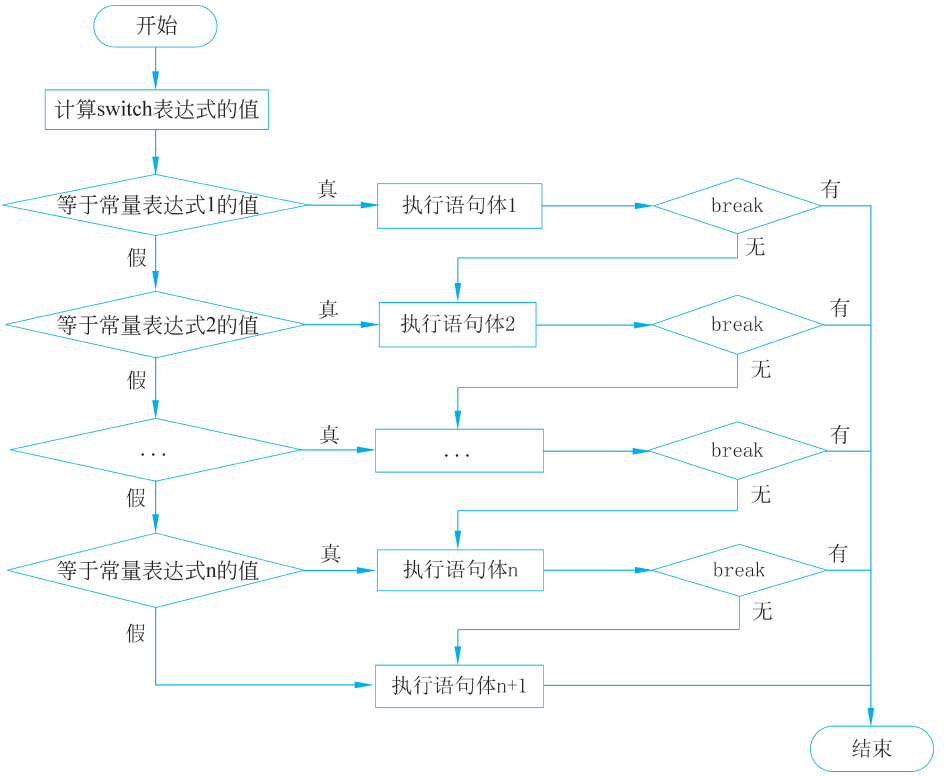


图 3-12　switch 语句流程图

**敲重点：**

（1）switch 语句中有四个关键字：switch、case、break、default。

（2）switch 后面的表达式必须是整型或字符表达式。

（3）case 与其后面的常量表达式之间必须有空格，常量表达式后面是冒号。常量表达式中不能含有变量或者函数。case 后面的常量表达式也必须是整型或字符表达式。

（4）所有 case 分支后面的常量表达式必须不同。

（5）每个 case 分支的语句体均可缺省。如果缺省语句体，则表示该 case 分支与后续的 case 分支共用后面的语句体。

（6）每个 case 分支末尾的“break;”语句均可缺省。如果有 break，则执行完该 case 分支就跳出 switch 语句；如果缺省 break，则执行完该 case 分支，继续执行后续的 case 分支。

（7）default 分支通常写在最后，也可以写在前面，它的位置不受限制。default 分支可缺省。

### 2. switch 语句举例

**【例 3.8】** 假设用 0,1,2,…,6 分别表示星期日、星期一、……、星期六。现输入一个数字，使用 switch 语句编程输出对应星期几的英文单词。例如，输入 6，就输出 Saturday。

思路：程序执行流程如图 3-13 所示。将输入的数字作为 switch 判断的表达式，根据表达式的值确定执行哪一个 case 分支，在 case 分支里输出与数字对应的英文单词。

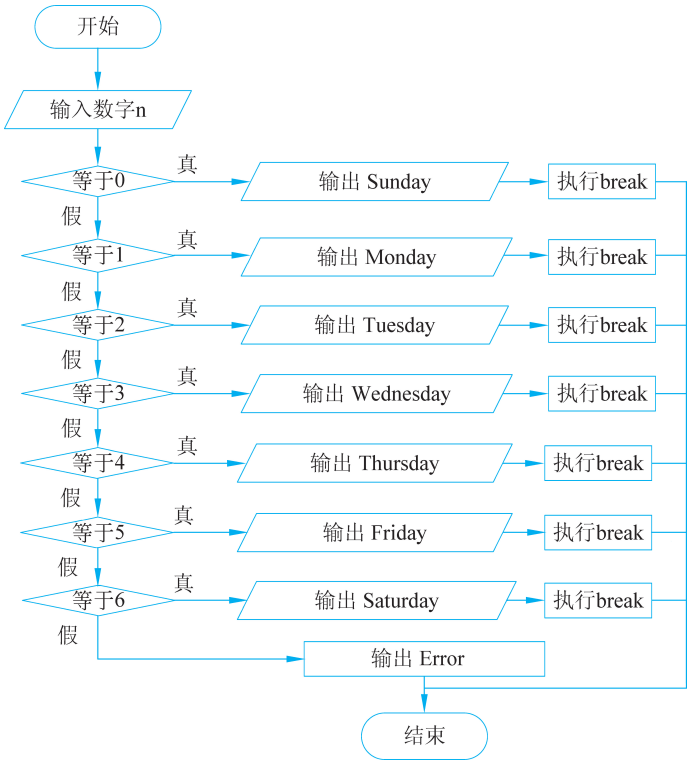


图 3-13 例 3.8 的流程图

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int n;
5.      printf("请输入一个数字: ");
6.      scanf("%d", &n);
7.      switch(n)
8.      {
9.          case 0:   printf("Sunday\n");         break;
10.         case 1:   printf("Monday\n");         break;
11.         case 2:   printf("Tuesday\n");        break;
12.         case 3:   printf("Wednesday\n");      break;
13.         case 4:   printf("Thursday\n");       break;
14.         case 5:   printf("Friday\n");         break;
15.         case 6:   printf("Saturday\n");       break;
16.         default:  printf("Error\n");
17.     }
18.     return 0;
19. }
```


switch 语句

程序运行结果：

```
请输入一个数字: 6
Saturday
```

```
请输入一个数字: 9
Error
```

思考：

修改例3.8的代码，下面程序段一中缺省了case分支的break语句，程序段二中缺省了case分支的语句体，当n = 1时，两个程序段的执行结果分别是什么？

程序段一：

```
switch(n)
{
    case 0: printf("Sunday\n");       break;
    case 1: printf("Monday\n");
    case 2: printf("Tuesday\n");
    case 3: printf("Wednesday\n");    break;
    case 4: printf("Thursday\n");     break;
    case 5: printf("Friday\n");       break;
    case 6: printf("Saturday\n");     break;
    default:  printf("Error\n");
}
```

程序段二；

```
switch(n)
{
    case 0: printf("Sunday\n");       break;
    case 1:
    case 2: printf("Tuesday\n");      break;
    case 3: printf("Wednesday\n");    break;
    case 4: printf("Thursday\n");     break;
    case 5: printf("Friday\n");       break;
    case 6: printf("Saturday\n");     break;
    default:  printf("Error\n");
}
```

分析：当n=1时，程序段一的执行结果为输出Monday、Tuesday、Wednesday，涉及的知识点为case分支缺省break语句，则顺序执行后续的case分支。程序段二的执行结果为输出Tuesday，涉及的知识点为case分支缺省语句体，则共用其后的case分支语句体。

## 3.5 条件运算符及表达式

**条件运算符是C语言中唯一的一个三目运算符，使用问号“?”和冒号“:”将三个运算对象连接起来。**以下为条件运算符构成的表达式，也称条件表达式。

```
表达式1 ? 表达式2 : 表达式3
```

如图3-14所示，条件表达式的执行过程：首先计算表达式1的值，若为真(非0)，就执行表达式2；若为假(0)，就执行表达式3。

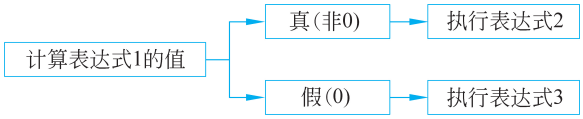


图3-14 条件表达式的执行过程