第四章 控制结构–选择结构

选择选择结构是程序设计结构中的另一种重要结构，也是现实生活中经常遇到的情况之一。C 语言提供了两类处理方法：一是使用 if 语句，二是使用 switch 语句。本章将详细介绍这几类选择语句的用法。

4.2 选择结构概述

选择结构（*selection*）是高级语言三大控制结构之一，用于需要选择判断的场合。

选择结构程序中，程序员根据实际需求，设置一个选择判断条件，并对选择的每一种可能情况（称为分支）编码以处理不同的流程。这些流程在编码时是平行的，没有哪一种处于主导地位。而程序在执行的时候，会根据选择条件的不同而执行不同的分支，从而导致结果的不同。图 4-1 是选择结构的流程图。

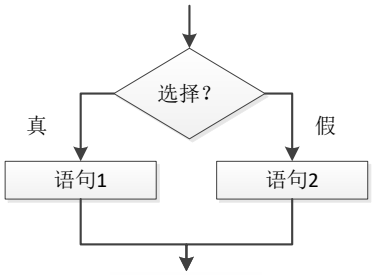

图4-1 选择结构

多数情况下，选择结构的不同分支不会全部都执行，而是根据条件是否成立，只执行众多流程当中的一个，以完成不同的功能。例外主要发生在使用多路选择 switch 语句的情况下(4.4.2)。

4.3 if 语句

C 语言提供的 if 语句有两个变种：单/双路选择 if…else 和多路选择 if…else if，以适应不同的场合。

4.3.1 单/双路选择 if 语句

单路选择 if 语句的语法为：

if (表达式) 语句 1

[else 语句 2]

其功能是：如果表达式的值为真（非 0 值），则执行语句 1；否则，如果存在 else 子句则执行其后的语句 2。如果没有 else 子句，则结束该 if 语句。该 if 语句执行完后，就接着执行紧跟在该 if 语句后面的一条语句。if 语句的流程图如图 4-1 所示。

例如：

```
x = 3;
if (x == 3)
    ++x;
else
    --x;
printf("%d", x);
```

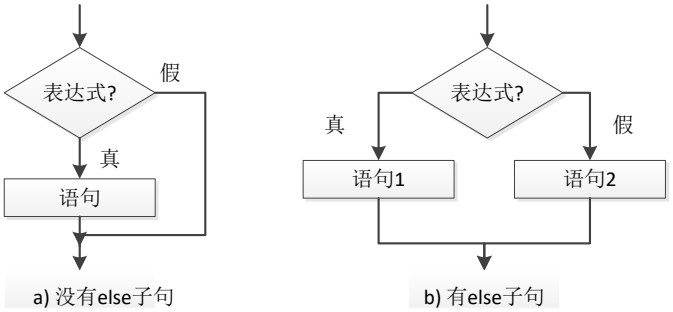

图4-2 if语句的流程图

使用 if 语句要注意以下几点：

(1) if 语句的选择条件必须用一对圆括号括起来；

(2) 从原则上讲，圆括号中的表达式应该是一个关系或者逻辑表达式，其值是一个逻辑类型（真或假）。但由于 C 语言没有提供表示逻辑真假的类型（C99 提供了_Bool 类型），因此这个表达式可以是任何类型表达式，如关系表达式、逻辑表达式以及算术表达式（其类型甚至可以是浮点型!）等。在这种情况下，任何非 0 值都被认为是逻辑真，0 值则被认为是逻辑假；

(3) 在判断相等的时候，请一定要注意使用关系运算符“==”，而不是赋值运算符“=”。例如，有 int

a = 1, b = 2，那么判断 a、b 是否相等的有逻辑错误的语句 if (a = b)是能够被编译器接受的，但结果却与初衷相悖：很明显的，a 和 b 不相等，因此该 if 之后的选择是不会被执行的；但在==误写成=之后，赋值表达式 a=b 将会得到结果 2，而所有非 0 值都被认为是逻辑真。这是初学者常犯的错误之一，而且很不容易察觉；

(4) 如果各子句中的语句多于一条时，必须使用复合语句，即用{}将这些语句括起来。

【例 4-1】输入两个整数 a 和 b，若 a<b，交换两个数，并输出交换后 a、b 的值。

【解题思路】题目中出现了这样一种要求，就是当满足一定条件后就要执行相应的操作，如果不满足则不需要有任何动作。根据题目的特点，应该选择使用没有 else 子句的 if 语句。

```
//4-1.c
#include <stdio.h>

int main()
{
    int a, b, x;           //x 作为交换的中间变量
    scanf("%d,%d", &a, &b);

    if (a < b)
    {
        x = a;
        a = b;
        b = x;
        printf("交换后 a=%d,b=%d\n", a, b);
    }

    return 0;
}
```

程序运行结果如下：

```
12,16↙
交换后 a=16,b=12
```

【例 4-2】任意输入一个整数，判断其奇偶性。

【解题思路】偶数的定义是：能被 2 整除的数。而判断一个数 a 是否能被 b 整除所用的方法就是看看二者相除的余数是否为 0。C 语言恰好提供了一个求两数余数的运算符符：模运算符%。所以，如果 a % 2 == 0，那么 a 一定是偶数；否则一定是奇数。根据这个特点，应该选择使用有 else 子句的 if 语句。

```
//4-2.c
#include <stdio.h>

int main()
{
    int data;

    scanf("%d", &data);
    if (data % 2 == 0)              //判断 data 能否被 2 整除
        printf("%d 是偶数\n", data);
    else
        printf("%d 是奇数\n", data);
```

```
    return 0;
}
```

程序运行结果如下：

```
24↙
24 是偶数
```

如果 if 语句分支中的语句比较简单，那么 if…else 语句可以用条件表达式实现。如例 4-2 中的 if 语句可以改为：

```
printf("%d 是%s\n", data, data%2 == 0 ? "是偶数" : "是奇数");  //条件表达式
```

程序的结果不变，但在表达上更加简洁和高效。

> 初学者在使用 if…else 语句时，常犯一些错误。
>
> 1. 存在多余的分号，例如：
>
> ```
> if (data % 2 == 0); //logical error
>     printf("%d 是偶数\n", data);
> else; //error
>     printf("%d 是奇数\n", data);
> ```
>
> if 后多余的分号并没有使其存在语法错误。编译器认为那是一条空语句，从而将其后的 printf()排除在该 if 语句之外，即使程序员采用了缩进的形式也无济于事。此外，另一个多余的分号使 else 成为"无主"的子句。同样地，else 后面的分号也会将其后的 printf()排除在整条 if 之外。
>
> 2. 在 else 子句后加上多余的条件判断，例如：
>
> ```
> if (data % 2 == 0) …
> else (data % 2 != 0) … //error
> ```
>
> 实际情况是：else 子句隐含的条件就是 if 的否条件，因此其后的条件完全是多余的，并且是语法错误的。

4.3.2 多路选择 if...else if 语句

当我们面临被选方案多于两个的场合时，可以使用多路选择 if…else if 语句，其语法如下：

```
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 n)   语句 n
[else                语句 n+1]
```

该语句的功能是：首先计算表达式 1，如果其结果为真，则执行语句 1；否则依次往下计算各表达式的值，直到某个表达式 i 的值为真，并且执行相应的语句 i。如果所有表达式的值都为假，则执行最后的 else 子句后的语句 n+1（如果存在这种无条件的 else 子句）。整个 if 语句的流程图如下所示（n=4）：

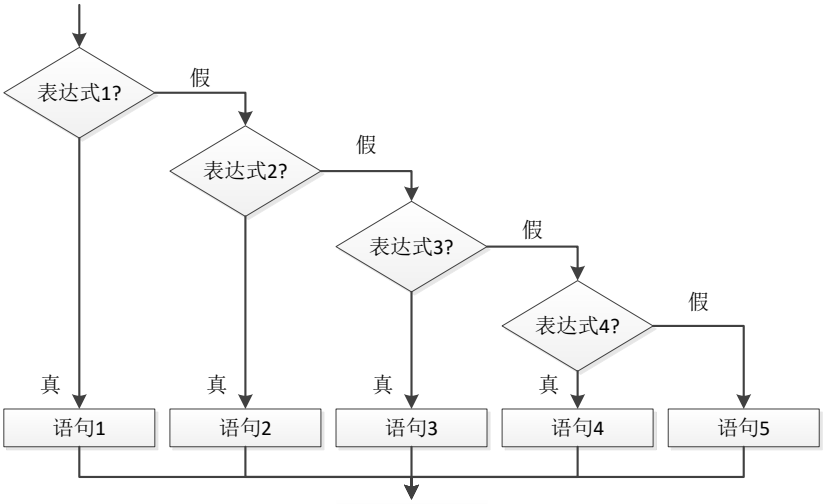

图4-3 if...else if语句的流程图

if…else if 语句的特点是：

(1) 整条 if 语句中只有一个分支能被执行。当执行完某个分支后，整条 if 语句也就结束了；

(2) else if 子句不能作为语句单独出现，必须与 if 配对使用；

(3) 最后的 else 子句是可省略的。

> 初学者在使用多路选择 if 语句的时候也常犯前面曾经描述的错误。此外，另一个错误是将两个分开的关键字 else 和 if 写在一起，即写成 elseif。

【例 4-3】编程实现一个人体型判断。医学上根据身高和体重，可以计算出“体重指数”，从而实现对人肥胖程度的划分。计算公式为：体型指数 t=体重 w/(身高 h)^2

其中，w 单位是千克，h 单位是米。体型判断依据如下：

- 当 t<18 时，为偏瘦；
- 当 18≤t<25 时，为标准；
- 当 25≤t<27 时，为偏胖；
- 当 t≥27 时，为肥胖。

从键盘输入某人的身高 h 和体重 w，计算出体指数 t，然后判断他们属于何种体型。

【解题思路】这是一个典型的需要根据条件进行判断选择的题目，因此要用到 if 语句。但通过分析可以发现：需要判断的条件超过了两个，因而要使用 if…else if 语句。

```
//4-3.c
#include <stdio.h>

int main()
{
  float h, w, t;

  scanf("%f,%f", &h, &w);
  t = w / (h * h);
  if (t < 18)
     printf("偏瘦\n");
  else if (t < 25)
    printf("标准\n");
  else if (t < 27)
     printf("偏胖\n");
```

```
    else
       printf("肥胖\n");

    return 0;
}
```

运行结果如下：

```
1.2,40
偏胖
```

> 因为多路选择 if 语句的执行是从第一个选择判断依次往下执行的，所以当不满足 t<18 这个条件后，后续的判断一定是建立在这个条件不成立（也就是 t>=18 成立）的基础上的，因此不需要使用 t>=18 && t<25 这样的复合条件了。当然，如果写成这样也没有任何错误。
>
> 如果一定要使用复合条件，那么请读者注意：在 C 语言中，判断一个值是否介于一个区域之间时不能使用数学上的类似于 18<=t<25 这样的方式，必须使用逻辑运算符&&或者||来连接两个关系表达式。

4.3.3 if 语句的嵌套

在上述各类 if 语句的分支中，可以包含任意合法的 C 语句。如果该语句也是一条 if 语句，那么称这是 if 语句的**嵌套**，其形式为：

```
if (表达式 1) //outer if
    if (表达式 2) 语句 1 //nested if 1
    else  语句 2
else
    if (表达式 3) 语句 3 //nested if 2
    else  语句 4
```

在上面的语法中，标记为 nested if 1 的 if 语句（包括其 else 子句）属于 outer if 的真分支，而 nested if 2（包括其 else 子句）属于 outer if 的假分支。

例如，判断一个数是正数、负数、零，我们可以用这样的方式来完成：

```
if (x != 0)
   if (x > 0)
     printf("x 是正数");
   else
     printf("x 是负数");
else
   printf("x 是零");
```

if 语句的嵌套需要注意以下几点：

(1) if 的数目必须大于等于 else 的数目。也就是说，有多少个 if，不一定有相同数目的 else；但反过来，有多少个 else，必须有同样数目的 if 与之匹配。

(2) if 与 else 的配对采用最近匹配原则：else 总是与它上面最近的未配对的 if 配对，而与书写的缩进格式无关。假设将上题设计如下，会出现什么问题？

```
if (x != 0) //outer if
   if (x > 0) //nested if
     printf("x 是正数");
else
   printf("x 是零");
```

当 x 的值为 0 时，本应该显示"x 是零"，但结果却是没有任何输出，显然与预期的结果不一样。

究其原因可以发现：虽然 else 在格式上对齐 outer if，但因最近匹配原则，实际它匹配的是 nested if。当 outer if 条件不满足时，理应执行它所对应的 else，但程序中并未出现与之对应的 else，所以没有产生屏幕输出。

为避免出错，使用时应该适当加上｛｝，明确匹配关系。上述程序应改为：

```
if (x != 0)
{
  if (x > 0)
     printf("x 是正数");
}
else
  printf("x 是零");
```

这样一来，显式地明确了 if 与 else 的配对关系，使程序员在阅读程序时不会产生歧义。

(3) if 嵌套关系比较复杂，但在程序中却有经常出现，因此在使用时要清楚配对关系。一个常用的做法就是为 if 和与之配对的 else 加上注释记号，例如：

```
if (x != 0)    //outer
{
  if (x > 0)   //nested
     printf("x 是正数");
}
else   //of outer
  printf("x 是零");
```

【例 4-4】if 语句的嵌套：从键盘输入 3 个整数，输出其中最小的那个。

【解题思路】在第二章里读者做过相似的题目，在实现时用了两个条件表达式分两次来求解的。这里用 if 语句来求解。根据题目，可以设计这样的算法：

(1) 输入 a、b、c 三个整数；
(2) 如果 a < b，那么 （此时 a 不一定是最小的）
(3)　　如果 a < c，那么 a 一定是最小的，输出 a，算法结束；
(4)　　否则，c 一定是最小的，输出 c，算法结束；
(5) 否则（a>b），那么 （此时 b 不一定是最小的）
(6)　　如果 c > b，那么 b 一定是最小的，输出 b，算法结束；
(7)　　否则，c 一定是最小的，输出 c，算法结束。

现在来验证这个算法的正确性。

(1) 假设 a < b，那么最小数一定是 a 和 c 中的一个。如果 a < c，那么有 a < c 并且 a < b 成立（此时 b 和 c 的关系不明，但这无关既要），则 a 是最小的，算法的第 3 步覆盖了这一情况。如果 a >= c，那么而有 b > a >= c 成立，则 c 一定是最小的，算法的第 4 步覆盖了这一情况。
(2) 假设 a > b，那么最小数一定是 b 和 c 中的一个。如果 c > b，那么有 b < a 并且 b < c 成立（此时 a 和 c 的关系不明，但这也无关紧要），则 b 一定是最小的，算法的第 6 步覆盖了这一情况。如果 b >= c，那么有 a > b >= c 成立，则 c 一定是最小的，算法的第 7 步覆盖了这一情况。

除上述两种情况之外，再无别的情况出现，因此可以保证算法的正确性。

> 验证的目的是为了保证 if 语句的分支能覆盖所有的情况。这在选择程序设计中有非常重要的意义。我们常犯的错误之一就是分支未能覆盖所有可选择的情况。

现在来进行编码。很明显，这需要用到 if 语句的嵌套。

```
//4-4.c
#include <stdio.h>
```

```
int main()
{
    int a, b, c;

    printf("Please input 3 ints:");
    scanf("%d%d%d", &a, &b, &c);

    printf("The minimum is ");
    if (a < b) //level 1
           if (a < c) printf("a=%d", a);
           else       printf("c=%d", c);
    else   // of level 1
           if (c > b) printf("b=%d", b);
           else       printf("c=%d", c);

    return 0;
}
```

运行结果如下：

```
Please input 3 ints:23 45 17↙
The minmum is c=17
```

4.4 多路选择 switch 语句

读者已经知道，if…else if 语句和 if 语句的嵌套可以用来实现多路选择结构。但是当选择越多，if…else if 语句就越冗长，或者 if 语句嵌套的层数也就越多，这些都大大降低了的程序的可读性和可维护性。因此，C 语言又提供了另一种多选择 switch 语句，可以更方便地处理多路选择。

4.4.1 switch 语句的基本语法

switch 语句特别适用于选择分支多于三种的场合（当然，少于 3 种也适用），其语法如下：

```
switch (控制表达式)
{
    case 常量表达式 1:    语句 1
    case 常量表达式 2:    语句 2
    …
    case 常量表达式 n:    语句 n
    [default:           语句 n+1]
}
```

其中，case 和 default 都是语句标号，它们只能出现在 switch 语句中。图 4-4 是 switch 语句的流程图。

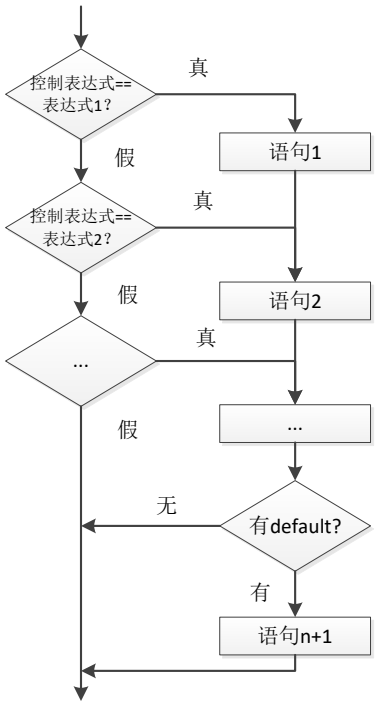

图4-4 switch语句的流程图

switch 语句的功能是：将控制表达式的类型依次推广（*promote*）到每个 case 标号，将该 case 标号后面的常量表达式的值转换成与控制表达式的值相同的类型。如果在推广过程中，两个表达式的值相等，那么就执行该 case 标号后的语句。若所有都不相等，则如果存在 default 标号，那么就执行其后的语句；如果没有，则这条 switch 语句什么都不做。当整条 switch 执行完后，继续执行紧接其后的语句。

例如：要求将考试的英文等级 ABCD 翻译成中文等级 “优秀”、“良”、“及格”、“差”，可以使用如下语句：

```
switch (level)
{
  case ‘A’: printf(“优秀\n”);
  case ‘B’: printf(“良\n”);
  case ‘C’: printf(“及格\n”);
  case ‘D’: printf(“差\n”);
  default:  printf(“error\n”);
}
```

switch 语句在使用时有如下要求：

(1) 控制表达式以及每个 case 标号后的表达式类型都必须是整数类型。C 语言的整数类型有：整型、字符型、枚举型和布尔型。其中布尔型是 C99 标准，有些编译器不支持此类型；

(2) 原则上讲，每个 case 标号的常量表达式的值都应该是唯一的；

(3) 最多只能有一个 default 标号。

初学者使用 switch 语句常犯一些错误。

1. 将控制表达式的类型误用为浮点型，例如：

```
float a;
switch (a) //error
```

2. 将 case 标号后的常量表达式写成一个条件表达式或者区段，或者字符串，例如：

```
switch (a)
{
   case 0<a<9: //error
   case 0-9: //error
   case “abc”: //error
}
```

4.4.2 使用 break 语句终止 switch 语句的执行

实际上，上小节中的那条 switch 语句并不能正确地履行功能。如果 leve 是’A’，那么程序的输出是：

```
优秀
良
及格
差
error
```

显然结果是错误的，虽然代码没有任何的语法错误。为什么会这样呢？

从图 4-4 中可以读者可以看到 switch 语句的流程。当某个 case 执行完成后，switch 语句并没有结束，而是继续下落到其后的 case/default 标号继续执行，并以此类推，直到 switch 语句的最后一个标号语句执行完为止。

为了避免这样的情况发生，可以使用 break 语句来种终止 switch 语句的执行。可以将上例修改为：

```
switch (level)
{
    case ‘A’: printf(“优秀\n”); break;
    case ‘B’: printf(“良\n”); break;
    case ‘C’: printf(“及格\n”); break;
    case ‘D’: printf(“差\n”); break;
    default:  printf(“error\n”);
}
```

上例中，无论哪一个 break 执行都会终止整条 switch 语句，从而在逻辑上保证了程序结果的正确性。

> ⓘ break 只对 switch 和各种循环语句起作用。

没有 break 的 case 标号虽然会导致其下的各 case 标号依次执行而发生逻辑错误，但这种特性并非一无是处。相反，在一些特定的场合，这种特性将会发挥高效的作用。例如，对于得’A’-‘C’的学生都给“通过”的结果，得’D’的给“不合格”的结果，可能会使用如下语句来完成：

```
switch (level)
{
    case ‘A’: printf(“通过\n”); break;
    case ‘B’: printf(“通过\n”); break;
    case ‘C’: printf(“通过\n”); break;
    case ‘D’: printf(“不合格\n”);
}
```

显然，这有代码重复之嫌。为此，利用 case 会顺序执行的特性，可以将代码修改为：

```
switch (level)
{
    case ‘A’:
    case ‘B’:
    case ‘C’: printf(“通过\n”); break;
    case ‘D’: printf(“不合格\n”);
}
```

上例中，如果 level 等于’A’，那么在执行时，case ‘A’标号后紧接着是另一个标号，而标号不是语句，因此将会继续往下，直到遇到第一条可执行语句为止。

【例 4-5】编程模拟一个简易计算器，它能完成整数的加、减、乘、除。（提示：从键盘输入第一个数、运算符、第二个数，然后计算出结果。）

【解题思路】按题目要求，输入的运算表达式应该具有 op1@op2 的形式，其中 op1 和 op2 是两个操作数，@是+-*/四个运算符之一。这样，可以用 scanf()用%d%c%d 的形式来读入操作数和运算符，然后通

过 switch 语句来判断运算符是哪一个，再进行相应的运算。需要处理例外情况，就是输入的运算符不符合要求。

除此之外，还应当处理一些特殊的情况，一个最明显的例子就是在做除法时，除数为 0。为此，我们应当在程序中设置一个判断，如果除数为 0 则显示输入有误。

```
//4-5.c
#include <stdio.h>

int main()
{
  int op1, op2;
  char operator;

  printf("请按 2+5 的格式输入\n");
  scanf("%d%c%d", &op1, &operator, &op2);
  switch (operator)
  {
    case '+':
           printf("%d\n",op1 + op2);
           break;

    case '-':
           printf("%d\n", op1 - op2);
           break;

    case '*':
           printf("%d\n", op1 * op2);
           break;

    case '/':
           if (op2 != 0)
               printf("%d\n", op1 / op2);
           else
               printf("除数不能为 0\n");
           break;

    default:
           printf("输入的符号不是+-*/之一\n");
  }

    return 0;
}
```

运行结果如下：

```
请那 2+5 的格式输入
4*5↙
20
```

以下是另一次运行的结果：

请那 2+5 的格式输入

5/0↙

除数不能为 0

> 在程序代码中对可能预见的错误进行预防处理是保证程序健壮性的有效手段之一，例如上例中对除数为 0 的处理。

4.4.3 switch 语句与 if…else if 的异同

可以看到，switch 语句和 if…else if 语句有异曲同工的效果，但它们也有不同：

(1) if…else if 中的判断条件可以使关系、逻辑或其它类型的表达式，而 case 标号后只能是常量表达式；

(2) if…else if 的条件可以是一个区间，而 switch 只能处理单值相等的情况；

(3) if…else if 满足一个表达式后退出整条语句的执行，而 switch 满足一个 case 的表达式后会继续执行，直到遇到 break 语句或 switch 语句结束；

(4) if…else if 后的语句超过一条要用复合语句，switch 的一个 case 标号后不管多少条语句都可以不要{}。

4.4.4 在 switch 语句中声明变量

一条 switch 语句在其内部形成了一个局部环境，因此允许在其{后面、第一条 case 标号之前声明变量，甚至出现函数调用。例如：

```
switch (level)
{
     int i = 0;
     printf("%d", i);
     case 'A': …
}
```

然而，虽然上面的代码没有任何语法问题，但变量 i 的初始化部分会被编译器忽略，因此 i 的初始值实际上是不确定的；同时函数 printf()永远不会被调用。请读者注意到这个事实，以免在编写程序时发生错误。

> 在 case 和 default 标号后不能有声明变量，除非该声明包含在语句块中。例如：
>
> ```
> case 1: int i; //error
> case 2: { int i; … } //ok
> ```