第 5 章 选择结构

本章导读

本章介绍一种应用非常广泛的控制结构，即选择结构。选择结构可以根据某一个表达式的值，选择执行程序中的某组语句。也就是说，选择结构可以控制程序中的语句，令其在满足某种条件时执行，不满足条件时不执行。

本章主要内容

- 什么是选择结构。
- 如何使用 if-else 语句实现选择结构。
- 如何使用 switch 语句实现选择结构。

5.1 选择的过程

判断和选择是现实生活中的常见动作。例如，如果今天下雨，那么李明出门时要带伞；买车票时，如果购票者是身高不足 1.2 米的小朋友，将给予车票半价优惠；如果购票者是 70 岁以上的老人，也将给予一定的票价减免。我们分析这些日常生活中常见的判断和选择过程，发现它们总是呈现图 5.1 所示的判断流程。

有些时候，如果满足条件，就完成相应的任务，反之则什么都不做，这是图 5.1(a)；还有些情境下，无论条件是否满足，都会完成不同的任务，这就是图 5.1(b)。

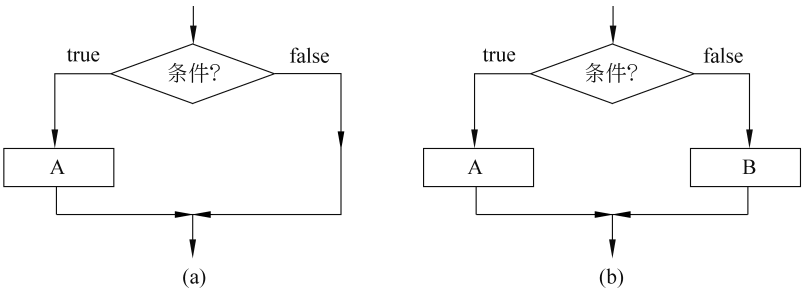

图 5.1 两种常见的判断流程

5.2　if-else 控制结构

在 C 语言中，可以用 if 语句来实现选择结构。C 语言的 if 语句有两种基本形式。

5.2.1　if 语句的两种形式

1. if

if 是最简单的一种选择结构，其流程图如图 5.2 所示。如果表达式的值为 true，则执行语句 A，否则选择 false 分支，即不执行任何操作。

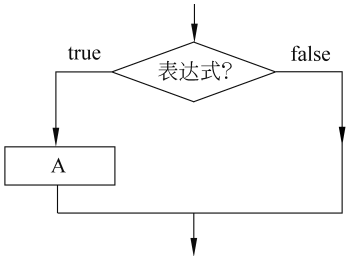

图 5.2　if 控制结构的流程图

if 语句的语法如下：

```
if(expression)
    statement;
```

该语句表示：如果(条件为真)，则执行语句 statement；否则什么都不执行。

其中：

(1) 必须将条件表达式 expression 写在小括号中，且小括号的末尾没有分号。

(2) expression 通常是逻辑表达式或关系表达式，但也可以是其他类型的表达式。例如，以下写法都是正确的，执行时，只要 if 括号中的 expression 的值为非 0，即表明条件为真。

```
if(a=5)                                    //等价于 if(5)
if(b)                                      //等价于 if(b 为真)
if(m>n)
if('a')
if(5)
if(0)
```

(3) 原则上，statement 为一条语句，但如果分支上有多条语句需要执行，则需将所有要执行的语句放在一对配对的大括号中，大括号中的语句被称为代码段。

代码段的形式如下：

```
if(expression)                                    //代码段开始
{
    statement1;
    statement2;
}                                                 //代码段结束
```

上述代码表示如果条件为真,则执行代码段中的全部语句,否则代码段中的语句全部不执行。

下面结合一个例子来分析 if 语句的用法。

【例 5.1】 根据分数输出成绩的等级和评语。

本例问题的要求是:输入一个学生的分数,如果大于或等于 90 分,则输出学生成绩的等级为 A,同时输出评价“Great!”。

本例中有一个简单的判断和决策过程,即

```
如果(score>=90)
    输出 A
    输出"Great!"
```

用 if 语句实现该操作的代码为:

```
if(score>=90)
{
    printf("A\n");
    printf("Great!");
}
```

由于满足条件时要做两个输出操作,因此必须把两条 printf()调用语句写在代码段中。

本例完整的代码如程序 5.1 所示。

【程序 5.1】 根据分数输出成绩等级和评语。

```
/*****************************************
  Program 5.1:  根据分数输出成绩的等级和评语
  written by Sky.
  12/10/2020. Copyright 2020
*****************************************/
#include <stdio.h>
int main()
{
    int score;
    scanf("%d",&score);
    if(score>=90)                          //如果大于或等于 90 分,则输出等级和评语
    {
        printf("A!\n");                    //代码段中的语句进行缩进
        printf("Great!");
```

```
    }
    return 0;
}
```

从本章开始，示例程序中将出现分支、循环等更加复杂的控制结构，一定要注意保持良好的编程风格。

如果程序中有分支结构或代码段时，分支上的语句、代码段中的语句应缩进。处于同一分支上的语句应对齐，配对的大括号可单独成行、单独成列对齐，以使程序的控制结构更清晰，令程序便于阅读和理解。

另外，必须特别注意合理地使用代码段。在本例中，如果没有写出代码段的大括号，程序的执行将受到什么影响呢？

以下左边为有大括号的情形，右边为缺失大括号的情形：

```
if(score>=90)           //代码段开始
{
    printf("A\n");      //语句①
    printf("Great!");   //语句②
}                       //代码段结束
```

```
if(score>=90)
    printf("A\n");      //语句③
printf("Great");        //语句④
```

从左边的代码看，由于语句①和②被放在代码段中，当 score 大于或等于 90 时，语句①和②将同时执行，否则同时不执行。

而右边的情形则完全不同。由于没有写成代码段，因此，如果 score 低于 90，则语句③不执行，但语句④却仍将无条件执行。这是因为，此时语句④并不在 if 语句的分支上，它与 if 语句是顺序关系，因此，无论 score 为多少分，始终会输出"Great!"。

可见，代码段外面的大括号并非是一种书写格式，而是一种程序执行的控制方法。缺失代码段外面的大括号时，编译器不会报错，也不会报警，大多数时候，程序将成功编译执行，但运行结果可能出错（因为运行逻辑错误）。

由代码段引发的错误非常难以察觉，出错后也较难排查，因此，养成良好的编程习惯非常重要。一般地，无论程序的分支上有多少条语句，建议都写成代码段的形式。

2. if-else

if-else 是 if 语句的另一种形式，其流程图如图 5.3 所示。如果表达式的值为 true，则执行语句 A，否则执行语句 B。

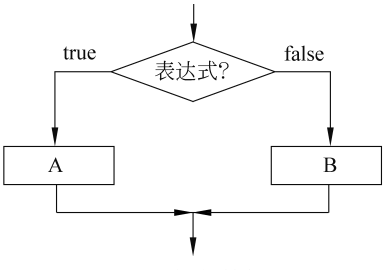

图 5.3　if-else 控制结构的流程图

if-else 语句的语法如下：

```
if(expression)
    statement1;
else
    statement2;
```

该语句表示“如果(表达式为真)，则执行语句 statement1，否则执行语句 statement2”。其中，如果 statement1、statement2 表示多条语句，则需写成代码段的形式，其形式如下：

```
if(expression)
{                                                    //代码段开始
    statement1;
    statement2;
}                                                    //代码段结束
else
{
    statement3;
    statement4;
}
```

下面结合例 5.2 来分析 if-else 语句的用法。

【例 5.2】 分段函数求解问题。

本例问题的要求是：有一个函数：$y=\begin{cases}x & (x<1)\\ 2x-1 & (x\geqslant 1)\end{cases}$

要求从键盘输入 x，计算并输出函数 y 的值。

根据题目要求，无论 x 的值小于 1，还是大于或等于 1，都需要计算 y 的值，因此可使用 if-else 结构来描述，核心代码片段如下：

```
if(x < 1)
    y = x;
else
    y = 2 * x-1;
```

完整代码如程序 5.2 所示。

【程序 5.2】 分段函数求解问题。

```
/*****************************************
  progra5.2:  根据 x 的值求解并输出函数值
  written by Sky.
  12/08/2020. Copyright 2020
*****************************************/
```

```
#include <stdio.h>
int main()
{
    int x, y;
    scanf("%d",&x);
    if(x<1)
        y = x;
    else
        y = 2 * x-1;
    printf("y=%d\n",y);
    return 0;
}
```

本例中，由于无论 x 取什么值，都需要计算 y 值，因此选用了 if-else 控制语句进行编码。

在使用 if-else 语句时，应注意 else 分支的条件是隐含的，因此，在 else 的后面不必显式地写出条件。如下写法是错误的，将引发编译器报错。

```
else (x>=1)                                    //这里的写法是错误的
    y = 2 * x-1;
```

虽然本例采用了 if-else 语句来解题，但实际上也可以用多条并列的 if 语句来解决本题。两种解法在执行上存在什么差异，请读者自行尝试分析。

5.2.2　if 语句嵌套

当解题步骤中存在多个分支时，可以采用嵌套的选择结构，其流程如图 5.4 所示。

首先判断表达式 1 的值，如果为真，则执行语句 1；否则判断表达式 2 的值，如果为真，则执行语句 2；否则执行语句 3。

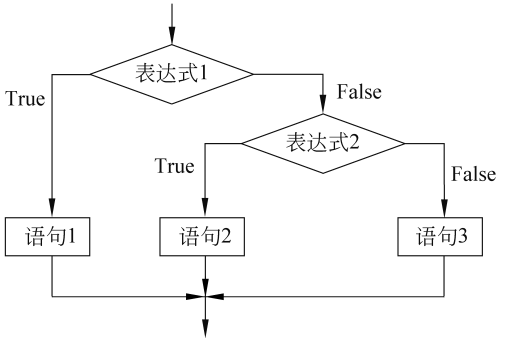

图 5.4　嵌套的选择结构流程图

对应的语法如下：

```
if(expression1)
    statement1;
else if(expression2)
        statement2;
    else
        statement3;
```

以下是一个选择结构嵌套的示例，根据学生的成绩 score，判断并输出分数等级。根据如下左栏的判断逻辑，对应的代码如下面的右栏所示。

```
如果(score 大于或等于 90 分)
    输出"A"
否则
    如果(score 大于或等于 60 分)
        输出"B"
    否则
        输出"C"
```

```
if(score>=90)
    printf("A");
else
    if(score>=60)     //隐含条件为 score<90
        printf("B");
    else
        printf("C");
```

注意，else 分支隐含着 if 的条件为假的先决条件，因此，对分支上的 if 语句的条件表达式，无须重复描述隐含条件，以减少计算量，并且令代码的逻辑更清晰。例如，将上例改为如下代码是不合适的：

```
if(score>=90)
    printf("A");
else
    if(score<90 && score>=60)                    //这里的条件表达式设计不合理
        printf("B");
    else
        printf("C");
```

在嵌套的 if 语句中将会出现多个 if 和 else，这时，要特别注意 if 和 else 的配对问题。C 语言规定，else 总是与它前面最近的、尚未与其他 else 配对的 if 配成一对。例如，以下代码段中，else 被理解为与 if(expression2)配对。

```
if(expressioin1)
    if(expression2)
        statement1;
    else
        statement2;
```

对多层的嵌套结构，其代码的可读性和易维护性尤其重要。编程时，除对不同层次的语句进行缩进外，还可用配对的大括号标明嵌套关系、标明代码段，以使代码结构更加清晰。例如：

```
if(expressioin1)
{
    if(expression2)
    {
        statement1;
        statement2;
    }
    else
    {
        statement3;
        statement4;
    }
}
```

5.3　switch

C 语言还提供了另一种用于多分支选择结构的 switch 语句，其一般形式为：

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

其执行过程为：首先计算表达式 expression 的值，将其值逐个与 case 的常量表达式的值相比较。当 expression 的值与某常量表达式的值相等，即执行该 case 子句的冒号后面的语句，一直执行到遇到 break 语句为止，或者执行到 switch 语句结束为止。

使用 switch 语句时应注意以下几点：

(1) switch 的表达式必须为整型或字符型。

(2) 以下称为一个 case 子句：

```
case 常量表达式 1: 语句 1;break;
```

switch 语句中可以包含多条 case 子句，所有 case 子句必须放在一对配对的大括号中。

(3) switch 中可以有一条 default 语句。如果 expression 的值与所有 case 后的常量表达式均不同，则执行 default 的语句。也可以不写 default 语句，这样，如果 expression

找不到相同值的 case 子句，则 switch 语句就什么都不执行。

(4) 一般地，case 子句和 default 子句的末尾可以有一条 break 语句。当遇到 break 语句时，将结束 switch 语句。如果某子句的末尾没有 break 语句，则一旦选中该子句，就会一直执行到遇到 break 语句，或 switch 语句的结束为止。

(5) case 和 default 子句的先后顺序不限，不会影响程序的执行结果。

(6) 每个 case 后只能有一个常量值，多条 case 子句的常量表达式的值不能相同。

(7) 每个 case 后允许有多条语句，可以不用{}括起来。

以下通过例 5.3 来表明 switch 语句的用法。

【例 5.3】 输出当前的月份问题。

万年历程序中有一个功能是输出一年的日历，即根据月份 month 的值(取值为 1～12)，输出 January、February 等月份名称。该流程是一个多分支的判断和选择，用 switch 语句描述的代码如下：

```
switch (month)
{
    case 1: printf("January"); break;
    case 2: printf("February"); break;
    case 3: printf("March"); break;
    case 4: printf("April"); break;
    case 5: printf("May"); break;
    case 6: printf("June"); break;
    case 7: printf("July"); break;
    case 8: printf("August"); break;
    case 9: printf("September"); break;
    case 10: printf("October"); break;
    case 11: printf("November"); break;
    case 12: printf("December"); break;
    default: printf("Error");
}
```

以上代码中，如果 month 的值为 3，则输出 March；如果 month 值为 9，则输出 September；如果 month 的值为 13，则输出 Error，这是因为此时找不到合适的 case 子句，因此执行 default 子句。注意，以上 default 子句的末尾没有 break 语句，但该子句位于 switch 语句的末尾，当 default 后面的语句执行完，switch 语句的执行自然会结束。

如果删掉 case 9 后的 break 语句，再来分析运行结果。修改后的代码形式如下：

```
case 9: printf("September");
case 10: printf("October"); break;
```

此时，如果 month 的值为 9，程序输出：

```
SeptemberOctober
```

由于这里的 case 9 末尾没有 break 语句，因此实际执行的语句为以下三条：

```
printf("September");
printf("October");
break;
```

这表明，在 switch 语句中，“case 常量表达式”只相当于一个分支的入口。当 switch 表达式的值与某个入口值相等，则去执行该入口后的语句，且其后不再进行标号的判断。也就是说，一旦选中子句 case 9，就一直执行其后面的语句，直到遇到 break 语句或 switch 语句结束为止。

本例的完整程序如程序 5.3 所示。

【程序 5.3】 根据月份值输出月份字符串。

```
/*****************************************
  progra5.3: 根据月份值输出对应的字符串
  written by Sky.
  12/08/2020. Copyright 2020
*****************************************/
#include <stdio.h>
int main()
{
    int month;
    scanf("%d",&month);
    switch(month)
    {
        case 1: printf("January"); break;
        case 2: printf("February"); break;
        case 3: printf("March"); break;
        case 4: printf("April"); break;
        case 5: printf("May"); break;
        case 6: printf("June"); break;
        case 7: printf("July"); break;
        case 8: printf("August"); break;
        case 9: printf("September"); break;
        case 10: printf("October"); break;
        case 11: printf("November"); break;
        case 12: printf("December"); break;
        default: printf("Error");
    }
    return 0;
}
```