第3章

选择结构程序设计

在前面两章中，所有的程序都是顺序结构，程序从 main()函数开始进入，然后逐条指令开始执行，直到所有指令都执行完。但是，就像日常生活一样，不是所有的事情都是按照计划执行的，许多事情会随着条件的变化，面临着各种选择。程序亦是如此，也会根据不同条件执行相应的行为。例如在“俄罗斯方块”游戏中，按下不同的按键，方块的运动方向会随之而改变。本章主要讨论选择结构程序设计的实现方法。

3.1 if 语句

最常见的选择语句是 if 语句，if 语句是根据给定的条件进行判断，以决定执行对应分支的程序段。C 语言提供了三种形式的 if 语句。

3.1.1 单分支结构

单分支结构基本形式如下：

```
if(条件表达式){
    语句;
}
```

if 语句用于选择是否执行一个行为，执行过程是先计算条件表达式的值，如果条件表达式为真，则执行其后的复合语句；否则，就直接跳过其后的复合语句。

单分支结构非常简单，if 后面括号里的条件表达式为真，也就是如果条件满足了，就执行{}里的语句，否则，直接越过{}里的语句。{}里的语句相当于一个整体，表示是复合语句。{}里的语句可以是多条，也可以只有一条，当只有一条语句时，可以省略{}。


视频讲解

【例 3.1】 编写程序，实现通过按键控制方块运动，当按下按键 W 时，控制方块向上运动，如图 3.1 所示。

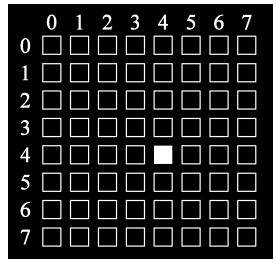

图 3.1 按键控制方块运动

为了实现按键控制方块运动，需要获得按键输入信息，获得按键输入信息之后，根据判断信息执行相应的行为。C 语言标准库中有 getch()函数，它的作用是从输入设备获得输入的字符。例如，在键盘上按下按键 W，getch()函数返回的值就是字符'w'，使用方法为：

```
char ch = getch();
```

通过 getch()函数可以获得按键的输入信息，然后根据键值执行相应的操作。代码如下：

```
#include"screen.h"
# define SIZE 8

int main(){
    initGame(SIZE);                    //初始化游戏,设置 8×8 矩阵的屏幕
    int row = 4;
    int col = 4;
    turnOn(row, col);                  //点亮第 4 行第 4 列位置的灯


    char ch = getch();                 //获得按下了按键的值

    if( ch == 'w'){                    //判断是否按下了按键 W
        clearScreen();                 //清屏,关闭屏幕上所有的灯

        row = row - 1;                 //向上运动 1 行
        turnOn(row, col);              //点亮第 3 行第 4 列位置的灯
    }
    return 0;
}
```

编译并运行代码，在键盘上按下按键 W 时，方块向上运动。有时不小心将键盘上大写键打开了，测试程序时，按下按键 W，方块并没有向上运动，这个错误不易被发现，所以将代码修改成大写键打开了也能控制方块向上运动，代码如下：

```
#include"screen.h"
#define SIZE 8

int main(){
    initGame(SIZE);                    //初始化游戏,设置 8×8 矩阵的屏幕
    int row = 4;
    int col = 4;
    turnOn(row, col);                  //点亮第 4 行第 4 列位置的灯

    char ch = getch();

    if( ch == 'w' || ch == 'W'){       //判断是否按下了按键 W
```

```
            row = row - 1;
            clearScreen();
            turnOn(row, col);
        }
        return 0;
    }
```

编译并运行代码，当在键盘上按下按键 W 时，小方块向上运动。

3.1.2　双分支结构

除了上述单分支选择结构，C 语言提供了 if-else 双分支语句，在两条语句之中进行选择，格式如下：

```
if(条件){
    语句 1;
}
else{
  语句 2;
}
```

if-else 语句的执行过程：如果满足条件，则执行语句 1，否则执行语句 2，语句 1 和语句 2 均可以由多条语句组成。需注意，else 语句不能单独使用，必须与 if 语句配对使用。

【例 3.2】 编写程序，实现按键 W 控制方块向上运动，其他按键控制方块向下运动。


视频讲解

按下按键 W 向上运动，按下其他键则向下运动，这是非常明显的二选一的情况，要么向上，要么向下，所以可以使用 if-else 语句完成任务，代码如下：

```
#include"screen.h"
#define SIZE 8
int main(){
    initGame(SIZE);                    //初始化游戏,设置 8 行 8 列的屏幕
    int row = 4;
    int col = 4 ;

    turnOn(row,col);

    char ch = getch();                 //获得按键的键值

    if( ch == 'w'){                    // 判断按下的是不是按键 W
       row = row - 1;                  //方块向上运动
    }
    else{
       row = row + 1;                  //方块向下运动
    }
    clearScreen();
    turnOn(row,col);

    return 0;
}
```

if 语句用于选择是否执行一个行为，而 if-else 语句用于在两个语句之间进行选择。对于 if-else 语句，C 语言还提供了条件表达式这种便捷表达方式。条件表达式由条件运算符?：组成，通用形式为：

```
表达式 1 ? 表达式 2: 表达式 3
```

执行过程：先求解表达式 1 的值，如果为真，则求解表达式 2 的值，并且将表达式 2 的值作为整个条件表达式的值。否则，求解表达式 3 的值，并将表达式 3 的值作为整个条件表达式的值。

例如代码：

```
if( ch == 'w'){                          //判断按下的是不是按键 W
   row = row - 1;                        //方块向上运动
}
else{
   row = row + 1;                        //方块向下运动
}
```

使用条件运算表达式为：

```
ch == 'w'? (row = row+1 ) : (row = row - 1);
```

一般情况下，条件运算符能完成的，if-else 也能完成。但是使用条件运算符的代码更加简洁。对于初学者，建议使用 if-else 语句，其更容易掌握。

3.1.3 多分支结构

在程序中，经常会遇到面临很多选择，例如按下不同的按键控制不同方向的运动，C 语言提供了 if-else if-else 语句，格式如下：

```
if(条件 1){
    语句 1;
}
else if(条件 2){
    语句 2;
}
…
else{
语句 n;
}
```

执行过程：如果条件 1 满足，则执行语句 1，否则计算条件 2，如果条件 2 满足，则执行语句 2，否则计算条件 3，如果条件 3 满足，则执行语句 3……如果所有条件均不满足，则执行 else 所对应的语句 n。


视频讲解

【例 3.3】 编写程序，实现按键 W 控制方块向上运动，按键 S 控制方块向下运动，其他按键斜向右下运动。

根据题意可知，这是多种选择的情况，可以使用 if-else if-else 语句，代码如下：

```
#include"screen.h"
#define SIZE 8
```

```
int main(){
    initGame(SIZE);                    //初始化游戏,设置 8 行 8 列的屏幕
    int row = 4;
    int col = 4 ;

    turnOn(row,col);

    char ch = getch();                 //获得按键的键值

    if( ch == 'w'){                    //判断按下的是不是按键 W
       row = row - 1;                  //方块向上运动
    }
    else if (ch == 's'){               //判断按下的是不是按键 S
       row = row + 1;
    }
    else{
       /* 方块斜向右下运动 */
       row = row + 1;
       col = col + 1;
    }

    clearScreen();
    turnOn(row,col);

    return 0;
}
```

编译并运行代码,按下按键 W,方块向上运动,按下按键 S,方块向下运动,按下其他按键,方块斜向右下运动。

对于多分支选择语句,可以将其修改成多个单分支语句,代码如下:

```
#include"screen.h"
#define SIZE 8

int main(){
    initGame(SIZE);                    //初始化游戏,设置 8 行 8 列的屏幕
    int row = 4;
    int col = 4 ;

    turnOn(row,col);

    char ch = getch();

    if( ch == 'w'){                    //判断按下的是不是按键 W
       row = row - 1;
    }

    if (ch == 's'){                    //判断按下的是不是按键 S
       row = row + 1;
```

```
    }

    if( ch != 'w' && ch != 's'){          //判断按下的键既不是按键 W,也不是按键 S

       row = row + 1;
       col = col + 1;
    }

    clearScreen();
    turnOn(row,col);

    return 0;
}
```

通过这个例子可以感受到无论是双分支语句，还是多分支语句，最后都可以改编成单分支语句实现，只不过判断条件可能较为复杂。

3.1.4 if 语句的嵌套

有时候，在一个特定选择中又引出新的选择，这种情况可以使用 if 嵌套语句。在 if 语句中又嵌套一个或者多个 if 语句称为 if 语句嵌套。其一般形式为：

```
if(条件){
  if(条件){
    语句块;
  }
  else{
    语句块;
  }
}
```

在 if 语句的嵌套结构中，要非常注意 if 和 else 的匹配关系。C 语言规定：每一个 else 总是与它前面最近的同一复合语句内的不带 else 的 if 结合。为了避免引起歧义，在书写代码时，不要省略{}，这样就能有效避免 else 的匹配问题。


视频讲解

【例 3.4】 编写程序，实现按键 W 控制方块向上运动，其余按键控制方块向下运动，并且在运动的过程中要检测是否越界了，保证方块不能运动到屏幕外。

既要判断按键，又要判断是否越界，可以使用嵌套 if 语句，代码如下：

```
#include"screen.h"
#define SIZE 8
int main(){
    initGame(SIZE);              //初始化游戏,设置 8 行 8 列的屏幕
    int row = 0;                 //设置在屏幕最顶行
    int col = 4 ;

    turnOn(row,col);

    char ch = getch();           //获得按键的值
```

```
    if( ch == 'w'){              //判断按下的是不是按键 W
       if (row > 0){             //向上运动需要判断方块是否在屏幕最底部
          row = row - 1;
       }
    }
    else{
       if( row < SIZE - 1){      //向下运动需要判断方块是否在屏幕最底部
          row = row + 1;
       }
    }

    clearScreen();
    turnOn(row,col);

    return 0;
}
```

编译并运行代码，按下按键 W，方块并没有向上运动，因为小方块在屏幕的顶端，再向上运动就越界。代码中 if 的嵌套，也可以使用逻辑运算符替代，代码如下：

```
if(ch == 'w' && row > 0){
    row = row - 1;
}

if(ch != 'w' && row < SIZE - 1){
    row = row + 1;
}
```

C 语言非常灵活，在编写程序时，同样的问题可以有多种解决方法。

3.2 switch 语句

对于在多个选项中选择，不仅可以用 if-else if-else 来完成，而且可以使用 C 语言的提供一种更为方便的 switch 语句，它的结构形式为：

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

执行过程：先计算表达式的值，如果表达式的值与某个常量表达式的值相等，则执行其后控制的语句块，如果所有的常量表达式的值都与表达式的值不相等，则执行 default 后的语句。

需要注意的是：

(1) case 后面必须是常量表达式，不能包含变量，并且每个常量表达式的值都不相同。

（2）default 语句可以省略。如果省略了 default 语句，当表达式的值与所有的常量表达的值都不相等时，则什么也不执行。


视频讲解

【例 3.5】 使用 switch 语句，编写程序，实现按键 W 控制方块向上运动，按键 S 控制方块向下运动，按键 A 控制方块向左运动，按键 D 控制方块向右运动。

使用 switch 语句，代码如下：

```
#include"screen.h"
#define SIZE 8
int main(){
    initGame(SIZE);                          //初始化游戏，设置 8 行 8 列的屏幕
    int row = 4;
    int col = 4 ;

    turnOn(row,col);

    char ch = getch();

    switch(ch){
       case 'w' :
                 row = row - 1;
       case 's':
                 row = row + 1;
       case 'a' :
                 col = col - 1;
       case 'd' :
                 col = col + 1;
    }

    clearScreen();
    turnOn(row,col);

    return 0;
}
```

编译并运行代码之后，按下按键 W，并没有如预期般向上运动一行，其原因是 switch 语句执行的原理是遇到匹配项之后，开始执行其后的控制语句块，如果没有遇到 break 语句，会一直执行到 switch 语句结束的右括号为止。上述代码中，按下按键 W 之后，执行了匹配项的语句，但是也执行了其后所有的语句，变量 row 的值增加了 1 之后，又减少了 1，所以没有变化。解决这个问题的方法是执行完匹配项的语句之后，立即终止 switch 语句。C 语言提供了跳转语句 break 能够达到这个目的。break 语句的调用形式如下：

```
break;
```

在 switch 语句中，break 语句可以终止所在的 switch 语句的执行。

修改后的代码如下：

```
#include"screen.h"
#define SIZE 8
```

```
int main(){
    initGame(SIZE);                //初始化游戏,设置 8 行 8 列的屏幕
    int row = 4;
    int col = 4 ;

    turnOn(row,col);

    char ch = getch();

    switch(ch){
       case 'w' :
                row = row - 1;
                break;             //用 break 跳出 switch 语句
       case 's' :
                row = row + 1;
                break;
       case 'a' :
                col = col - 1;
                break;
       case 'd' :
                col = col + 1;
                break;
    }

    clearScreen();
    turnOn(row,col);

    return 0;
}
```

编译并运行代码,按键 W、S、A、D 可以控制小方块上、下、左、右运动。switch 和 break 相结合,才能设计出正确的多分支选择结构程序。使用 switch 语句时,要根据情况需要,判断是否要加上 break 语句。例如双人游戏模式,按键 W、I 控制方块向上运动,按键 S、K 控制方块向下运动,按键 A、J 控制方块向左运动,按键 D、L 控制方块向右运动。使用 switch 语句实现,代码如下:

```
#include"screen.h"
#define SIZE 8
int main(){
    initGame(SIZE);                    //初始化游戏,设置 8 行 8 列的屏幕
    int row = 4;
    int col = 4 ;

    turnOn(row,col);

    char ch = getch();

    switch(ch){
       case 'w' :
```

```
        case 'i':
                row = row - 1;
                break;                  //用 break 跳出 switch 语句
        case 's':
        case 'k':
                row = row + 1;
                break;
        case 'a':
        case 'j':
                col = col - 1;
                break;
        case 'd':
        case 'l':
                col = col + 1;
                break;
    }

    clearScreen();
    turnOn(row,col);

    return 0;
}
```

编译并运行代码，按下按键 W 或者 I 都能控制小方块向上运动。当按下按键 W 时，case 'w' 选项后的语句为空，程序会自动往下执行，执行 case 'i' 选项对应的程序段，所以按键 W 和 I 都能控制方块向上运动。

如果给每一个 case 选项的语句段都增加 break 语句，则需要增加不少重复的代码。对于不同条件，执行相同行为的场景下，通过 switch 语句和 break 语句的结合使用，可以写出简洁的代码。

虽然 switch 语句有时比 if-else 语句简洁，逻辑关系一目了然，程序可读性好，但是使用范围较窄。例如选择条件是非常大的范围，如变量 i 是 100～550 的整数，使用 if 语句非常简单，代码如下：

```
if( i > 100 && i < 550 )
```

而使用 switch 语句将会非常麻烦，需要设置几百个 case 选项，因此对于初学者来说，熟练掌握 if 语句即可。

3.3 综合案例：按键控制“俄罗斯方块”运动

编写程序，实现按键控制“俄罗斯方块”运动，按键 W 控制方块向上运动，按键 S 控制方块向下运动，按键 A 控制方块向左运动，按键 D 控制方块向右运动，并且检测“方块”是否越界，如图 3.2 所示。

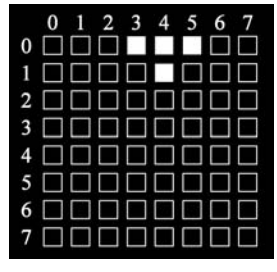

图 3.2　俄罗斯方块

俄罗斯方块由 4 个小方块组成，4 个方块作为一个整体运动，与一个小方块运动没有本质区别。选择一个参照点，使用相对坐标的方法表示 4 个方块，这样运动时，只需要修改参照点的位置即可。判断边界时，要注意哪个方块最先达到边界，代码如下：

```
#include"screen.h"
#define SIZE 8
int main(){
    initGame(SIZE);                          //初始化游戏,设置 8 行 8 列的屏幕
    int row = 0;                             //参照点的行坐标
    int col = 0 ;                            //参照点的列坐标

    /*在屏幕上显示 4 个小方块*/
    turnOn(0 + row,3 + col);
    turnOn(0 + row,4 + col);
    turnOn(0 + row,5 + col);
    turnOn(1 + row,4 + col);


    char ch = getch();

    if( ch == 'w'){
       if (row > 0){                         //判断是否到了屏幕的最顶端
          row = row - 1;
       }
    }

    if( ch == 's'){
       if( row + 1 < SIZE - 1){              //判断最下面的小方块是否到了屏幕的最底端
          row = row + 1;
       }
    }

    if( ch == 'a'){
       if( 3 + col > 0){                     //判断最左边的小方块是否到了屏幕的最左端
          col = col - 1;
       }
    }

    if( ch == 'd'){
       if( 5 + col < SIZE - 1){              //判断最右边的小方块是否到了屏幕的最右端
          col = col + 1;
       }
    }


    clearScreen();
    turnOn(0 + row,3 + col);
    turnOn(0 + row,4 + col);
    turnOn(0 + row,5 + col);
    turnOn(1 + row,4 + col);

    return 0;
}
```

本项目程序虽然实现了以按键的方式控制“俄罗斯方块”运动的功能，但该功能却只能执行一次。如果想要实现多次控制方块运动，需要学习第 4 章循环结构程序设计。

习题

编写程序，按键控制球拍左右运动，如图 3.3 所示。

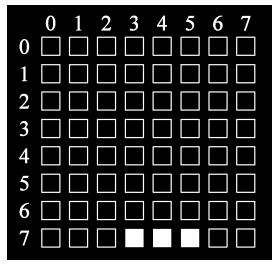

图 3.3　球拍