第3章 程序结构与调试

程序设计是计算机科学的核心领域，指通过系统化方法将问题转化为计算机可执行的指令序列，涵盖需求分析、算法设计、代码实现及调试优化等环节。程序结构指程序选择什么样的路径进行下去，是通过合理的逻辑组织和代码设计，确保程序高效、可靠、可维护，是程序设计者的主体思路，是算法的主要体现。

3.1 程序结构

常见的程序结构有顺序结构、分支结构、循环结构等。

3.1.1 顺序结构

如前所述程序均为顺序结构，顺序结构程序代码严格按照程序代码出现的先后顺序从前到后逐条语句执行。

如：

```
static void Main(string[] args)
{
    Console.WriteLine("Please input your name!");
    string name = Console.ReadLine();
    Console.WriteLine("Hello " + name);
    Console.WriteLine("Press any key to exit...");

    Console.ReadKey();
}
```

3.1.2 分支结构


视频讲解

在程序运行过程中，许多情况下需要根据程序运行产生的新环境有选择地执行相应代码语句，此时需要用到程序结构中的分支结构，分支结构又分为单分支结构和多分支结构。

1. if 语句

由 if 语句引导的分支结构通常称为单分支结构，if 语句完整结构如下：

```
if(条件表达式)
{
    //要执行的程序段;
```

```
}
else
{
    //要执行的程序段;
}
```

如要解决如下数学问题：

```
y=1(x>=0);y=-1(x<0)
```

用 if 语句实现如下：

```
double x = Convert.ToDouble(Console.ReadLine());
int y;
if (x>=0)
{
    y = 1;
}
else
{
    y = -1;
}
```

在很多时候还会用到 if 语句的嵌套。如：

```
y=1(x>0);y=0(x=0);y=-1(x<0)
```

用 if 语句嵌套解决如下：

```
double x = Convert.ToDouble(Console.ReadLine());
int y;
if (x>0)
{
    y = 1;
}
else
{
    if (x == 0)
    {
        y = 0;
    }
    else
    {
        y = -1;
    }
}
```

if 语句还有另外一种结构，如上问题用 if 语句的另外一种结构解决如下：

```
double x = Convert.ToDouble(Console.ReadLine());
int y;
if (x>0)
{
```

```
    y = 1;
}
else if (x == 0)
{
    y = 0;
}
else
{
    y = -1;
}
```

提示：以上 if 语句结构在编程实践中不建议使用，尤其是在更复杂的问题中需要使用更多的 else if 语句时，会使得程序结构变得异常复杂，很难理清程序结构中的对应关系。

2. 三元运算符

在 C# 中，还定义了一个三元运算符来实现和 if 语句相似的功能。结构如下：

```
表达式 1?表达式 2:表达式 3
```

如果表达式 1 为真，则执行表达式 2，否则执行表达式 3。如：

```
static void Main(string[] args)
{
    Console.WriteLine("Please input first value:");
    int i1 = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Please input second value:");
    int i2 = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("The bigger value is:");
    Console.WriteLine(i1 > i2 ? i1 : i2);
    Console.ReadKey();
}
```

输出结果如图 3-1 所示。

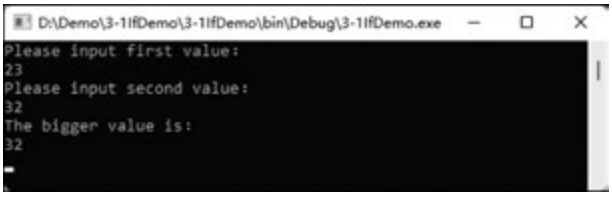

图 3-1　三元运算符的执行结果

3. switch 语句

switch 语句被称为多分支语句，用于当选择较多的情况。结构如下：

```
switch(表达式)
{
    case 值 1:
        程序段;
        break;
```

```
        case 值 2:
             程序段;
             break;
        case 值 3:
             程序段;
             break;
             //…
        default:
             程序段;
             break;
    }
```

当表达式值等于某 case 语句值时,则执行该 case 语句下的程序段,程序段必须以 break 结束;当表达式值不等于任一 case 语句值时,则执行 default 语句中语句。default 语句不是必需的。如:

```
static void Main(string[] args)
{
    Console.WriteLine("Please input confirm、retry or cancel,inputting other is invalid:");
    string s = Console.ReadLine();
    switch (s)
    {
        case "confirm":
            Console.WriteLine("You choose confirm!");
            break;
        case "retry":
            Console.WriteLine("You choose retry!");
            break;
        case "cancel":
            Console.WriteLine("You choose cancel!");
            break;
        default:
            Console.WriteLine("Invalid value!");
            break;
    }
    Console.ReadKey();
}
```

输出结果如图 3-2 所示。

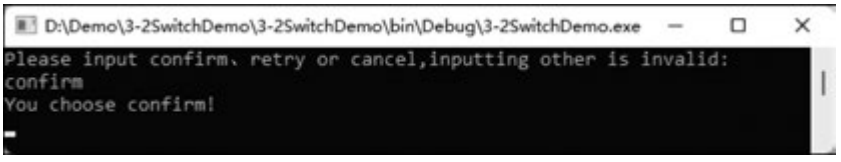

图 3-2 switch 语句/执行效果

提示:switch 语句只能进行等值判断进入分支语句,不能进行大小比较进入分支语句。如:

```
string s = Console.ReadLine();
switch (s)
```

```
{
     case != "confirm":
          Console.WriteLine("You choose confirm!");
          break;
     case >"abc":
              Console.WriteLine("You choose retry!");
              break;
       //…
}
```

以上判断将导致程序报错。

case 语句不能出现重复值。同时，不管 switch 语句有多少个分支，只有一个分支会被执行，由 break 语句结束 switch 语句执行。

3.1.3　循环结构


视频讲解

循环结构用于不断执行重复操作。循环结构可以使用 for 语句、while 语句和 do…while 语句来实现。

1. for 语句

for 语句通常用来执行固定次数的循环。结构如下：

```
for (整型计数器; 循环结束条件判断; 步长)
{
    //循环执行的代码段
}
```

整型计数器用来对 for 语句的每一次循环进行计数，循环结束条件用于判断循环何时结束，步长用来控制计数器每一次循环计数器的增幅。

如：

```
string[] s = new string[10];
for (int i = 0; i < s.Length; i++)
{
    s[i] = Console.ReadLine();
}
```

如上代码定义了一个字符串型一维数组，并通过循环执行方式为数组每一个元素赋值。

提示：一维数组通常和 for 语句一块使用，用于读取一维数组每一个元素的值，或者为一维数组的每一个元素赋值。

在程序设计中，嵌套的 for 语句也经常会被使用到。如对二维数组嵌套循环赋值或者输出：

```
static void Main(string[] args)
{
    int[,] s = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 } };
    for (int i = 0; i < 3; i++)
    {
```

```
            for (int j = 0; j < 4; j++)
            {
                Console.WriteLine(s[i, j]);
            }
        }
        Console.ReadKey();
    }
```

输出结果如图 3-3 所示。

图 3-3 for 循环语句嵌套输出

如上代码定义了一个二维数组并为二维数组每一个元素进行了赋值，通过嵌套 for 语句对二维数组每一个元素进行了输出。

2. while 语句

while 语句通常用于非固定次数的循环。结构如下：

```
while(条件表达式)
{
    //循环执行的代码段
}
```

条件表达式用作循环结束的判断，当条件表达式值为 true 时，循环继续，否则，循环结束。

如：

```
string s = Console.ReadLine();
ArrayList list = new ArrayList();
while (s!= "z")
{
    list.Add(s);
    s = Console.ReadLine();
}
```

如上代码定义了一个动态数组，如果通过键盘获取的字符串不等于"z"，则将输入的字符串添加到动态数组，循环继续；当输入的字符串等于"z"时，则循环结束。

3. do…while 语句

do…while 语句与 while 语句一样，一般也用于非固定次数的循环。结构如下：

```
do
{
    //循环执行的程序段
}while(条件表达式);
```

如：

```
string s = Console.ReadLine();
ArrayList list = new ArrayList();
do
{
    list.Add(s);
    s = Console.ReadLine();
} while (s!= "z");
```

提示：for 语句一般用于固定次数的循环，而 while 语句、do…while 语句一般用于非固定次数的循环。while 语句先判断条件再决定是否执行其中的代码段，循环执行的代码段可能一次也不被执行，而 do…while 语句先执行一次再进行条件判断，循环执行的代码段至少被执行一次。如以上两例，while 语句中无论如何也不可能把字符串“z”添加到动态数组，而 do…while 语句则会。

4. foreach 语句

foreach 语句通常与集合型数据一起使用，用于对集合中元素的遍历。foreach 语句结构如下：

```
foreach (集合元素数据类型 item in 集合)//item 为集合中的当前元素
{
  //循环执行的代码段
}
```

如：

```
static void Main(string[] args)
{
    List<string> list = new List<string>();
    list.Add("abc");
    list.Add("efg");
    foreach (string item in list)
    {
        Console.WriteLine(item);
    }
    Console.ReadKey();
}
```

输出结果如图 3-4 所示。

图 3-4　foreach 语句执行结果

5. continue、break、return 命令

在程序设计中,经常看到 continue、break 和 return 命令,如 switch 分支语句、for 循环语句等,三者分别定义了语句执行的结束方式。

在循环语句中,continue 用于控制循环结构结束本次循环,继续下一次循环;break 用于控制结束整个循环,继续执行循环结构后的代码;return 用于结束当前程序体执行,返回到上一级,当前程序体中 return 后的代码将不被执行。如:

```
for (int i = 0; i < 100; i++)
{
    if (i % 2 == 0)
    {
        Console.WriteLine(i);
    }
    else
    {
        continue|break|return;
    }
}
```

在程序中若使用 continue 则输出 1～100 的所有偶数;若使用 break,则只输出一个 0;若使用 return,则控制台窗口一闪而过。

提示:在后续内容中,我们还会经常使用 return 从一个方法中返回想要的数据。


视频讲解

3.2 程序调试

在程序设计时,无论设计得多么完美,在程序运行过程中可能都会出现这样或者那样的问题,因此,程序调试可能会占用程序设计者很大一部分时间用来修改程序中出现的 bug,也使得程序调试成为每个程序员必备的技能。

3.2.1 调试

在程序运行时,为便于观察程序运行过程和细节,往往使用添加断点的方式对程序运行加以控制。可以单击代码行左侧边栏灰色区域为当前程序行添加断点,添加断点后,当前代码行前端出现一个红色圆点标记,如图 3-5 所示。

图 3-5 添加断点

再次单击红色圆点，添加的断点会被取消。添加断点后，当程序运行到当前代码行时，程序运行被中断（当前行代码并未被执行），通过按 F11 键可以使程序进入逐行执行模式。

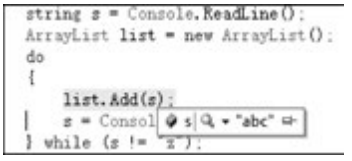

图 3-6　查看变量值

在程序逐行执行模式下，将鼠标指针移动到任意变量或者对象，将会显示该变量或者对象当前的值，如图 3-6 所示。

提示：值得注意的是，在程序逐行运行模式下，当连续按 F11 键时，如果不小心越过了需要处理的内容，需启动调试模式重新进行调试。换句话说，就是程序运行只能前进，不能后退。

3.2.2　异常

在程序设计时，难免会出现预料之外的情况，造成程序运行时出现异常，也就是通常所说的 bug，我们应尽可能发现和避免程序中出现 bug，这也是程序员面临的一大难题。

1. 系统异常

在程序运行时，一个健壮的应用程序即使出现问题也应该能够给出相应提示而不是直接崩溃或者让用户摸不着头脑，除了书写大量注释外，还需要对程序运行过程中可能出现的情况进行预处理。对一些不能确定或者无法预期的情况，通常使用 try…catch…finally 语句为相应代码设置异常捕获，try…catch…finally 语句结构如下：

```
try
{
        //可能出现异常的程序代码
}
catch (Exception e)
{
        //异常信息的显示
}
finally
{
        //无论何种情况出现都要执行的代码，finally 部分可以不出现，不是必需的
}
```

如：

```
static void Main(string[] args)
{
    Console.WriteLine("Please input an int value:");
    int x = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Please input an int value again:");
    int y = Convert.ToInt32(Console.ReadLine());
    try
    {
        Console.WriteLine(x / y);
    }
    catch (Exception e)
```

```
        {
            Console.WriteLine(e.Message);
        }
        Console.ReadKey();
    }
```

当 y=0 时,输出结果如图 3-7 所示。

图 3-7 异常信息显示

提示:C#定义了大量系统异常,可以满足程序运行过程中大部分异常信息的捕获。

2. 自定义异常

虽然 C#定义了大量系统异常以帮助在程序设计时及时了解问题产生的原因,但是,这些异常也并不能解决所有的问题,有时候还需要程序设计者自定义一些异常来解决程序设计中出现的个性化问题。自定义异常一般结构如下:

```
//自定义异常
public class 异常类名 : Exception
{
        //字段
        //方法
}
//自定义方法,在方法中抛出异常
public static void 方法名()
{
    //抛出异常
}
```

如:

```
class Program
{
    enum Days : int { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };
    static void Main(string[] args)
    {
        Console.WriteLine("Please input Sunday - Saturday:");
        string day = Console.ReadLine();
        try
        {
            method(day);
        }
        catch (MyException e)
        {
            Console.Write(e.message);
        }
```

```
            Console.ReadKey();
        }
        public static void method(string day)
        {
            bool bl = false;
            foreach (Days item in Enum.GetValues(typeof(Days)))
            {
                if (day != item.ToString())
                {
                    continue;
                }
                else
                {
                    bl = true;
                }
            }
            if (!bl)
            {
                throw new MyException("Invalid enum value!"); //抛出异常
            }
        }
    }

    public class MyException : Exception
    {
        public string message;
        public MyException(string m)
        {
            message = m;
        }
    }
```

以上自定义异常定义了对枚举类型数据的有效性检测，当输入星期日期不是 Sunday～Saturday 时，则输出异常信息“不是有效枚举值!”。

以上程序代码执行效果如图 3-8 所示。

图 3-8　抛出自定义异常

提示：异常捕获有时也被用在程序运行过程中获取变量的值或者相应的实时数据。异常捕获只能捕获程序运行中逻辑或者计算性等错误，对程序书写或者语法性错误仍然会以语法报错的方式给出提示。

3.3　程序结构应用


视频讲解

Demo1：**星期提醒**

使用枚举类型数据，根据星期日期不同，给出友好提示。

实现代码如下：

```
class Program
{
    enum Days : int { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };
    static void Main(string[] args)
    {
        Days day = (Days)DateTime.Now.DayOfWeek;
        switch (day)
        {
            case Days.Sunday:
                Console.WriteLine("Today is sunday,you have a happy time!");
                break;
            case Days.Monday:
                Console.WriteLine("Today is monday,start to work!");
                break;
            case Days.Tuesday:
                Console.WriteLine("Today is tuesday,mind to have a meeting at afternoon!");
                break;
            case Days.Wednesday:
                Console.WriteLine("Today is wednesday,a half of week has left!");
                break;
            case Days.Thursday:
                Console.WriteLine("Today is thursday,mind class at afternoon!");
                break;
            case Days.Friday:
                Console.WriteLine("Today is friday,had you a plan of the weekend?");
                break;
            case Days.Saturday:
                Console.WriteLine("Today is saturday,you have a rest!");
                break;
        }
        Console.ReadKey();
    }
}
```

Demo2：九九乘法口诀表

使用 for 语句嵌套实现九九乘法口诀表输出。

实现代码如下：

```
static void Main(string[] args)
{
    for (int i = 1; i < 10; i++)
    {
        for (int j = 1; j <= i; j++)
        {
            Console.Write(i.ToString() + "*" + j.ToString() + "=" + (i * j)
.ToString() + " ");
        }
        Console.WriteLine();
    }
    Console.ReadKey();
}
```

Demo3：冒泡排序

实现代码如下：

```
static void Main(string[] args)
{
    int[] a = { 23, 45, 67, 89, 76, 54, 32 };
    for (int i = 0; i < a.Length; i++)
    {
        for (int j = i + 1; j < a.Length; j++)
        {
            if (a[i] < a[j])
            {
                int t;
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
    }
    for (int i = 0; i < a.Length; i++)
    {
        Console.WriteLine(a[i]);
    }
    Console.ReadKey();
}
```

Demo4：退出管理

使用分支结构、循环结构对 1.3 程序实例输入和输出应用进行修改，只有当输入“6”时程序才会退出。

实现代码如下：

```
Console.WriteLine("1.Create DOS partition or Logical DOS Drive");
    Console.WriteLine("2.Set active partition");
    Console.WriteLine("3.Delete partition or Logical DOS Drive");
    Console.WriteLine("4.Display partition information");
    Console.WriteLine("5.Change current fixed disk drive");
    Console.WriteLine("6.Exit");
    Console.WriteLine();
    Console.WriteLine("Please input a number and input enter to execute the operation:");

    string s = Console.ReadLine();
    while (s != "6")
    {
        switch (s)
        {
            case "1":
                Console.WriteLine("Enter choice:" + s);
                break;
            case "2":
                Console.WriteLine("Your choice:" + s);
                break;
            case "3":
```

```
                Console.WriteLine("Your choice:" + s);
                break;
            case "4":
                Console.WriteLine("Your choice:" + s);
                break;
            case "5":
                Console.WriteLine("Your choice:" + s);
                break;
            case "6":
                Console.WriteLine("Your choice:" + s);
                break;
            default:
                Console.WriteLine("Please choose 1 - 6:");
                break;
        }
        if (s == "6")
        {
            break;
        }
        else
        {
            s = Console.ReadLine();
        }
    }
    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();
```

Demo5：字符统计

输入一串字符,统计 a～z 26 个英文字符出现的频率。

实现代码如下：

```
static void Main(string[] args)
{
    Console.WriteLine("Please input a lower string...");
    string s = Console.ReadLine();
    char[] c = s.ToCharArray();
    int[] a = new int[26];
    for (int i = 0; i < c.Length; i++)
    {
        switch (c[i])
        {
            case 'a':
                a[0]++;
                break;
            case 'b':
                a[1]++;
                break;
            case 'c':
                a[2]++;
                break;
            case 'd':
                a[3]++;
```

```
            break;
        case 'e':
            a[4]++;
            break;
        case 'f':
            a[5]++;
            break;
        case 'g':
            a[6]++;
            break;
        case 'h':
            a[7]++;
            break;
        case 'i':
            a[8]++;
            break;
        case 'j':
            a[9]++;
            break;
        case 'k':
            a[10]++;
            break;
        case 'l':
            a[11]++;
            break;
        case 'm':
            a[12]++;
            break;
        case 'n':
            a[13]++;
            break;
        case 'o':
            a[14]++;
            break;
        case 'p':
            a[15]++;
            break;
        case 'q':
            a[16]++;
            break;
        case 'r':
            a[17]++;
            break;
        case 's':
            a[18]++;
            break;
        case 't':
            a[19]++;
            break;
        case 'u':
            a[20]++;
            break;
        case 'v':
```

```
                    a[21]++;
                    break;
                case 'w':
                    a[22]++;
                    break;
                case 'x':
                    a[23]++;
                    break;
                case 'y':
                    a[24]++;
                    break;
                case 'z':
                    a[25]++;
                    break;
            }
        }
        Console.WriteLine("a - z 出现的频率依次为:");
        foreach (int item in a)
        {
            Console.WriteLine(item);
        }
        Console.ReadKey();
    }
```