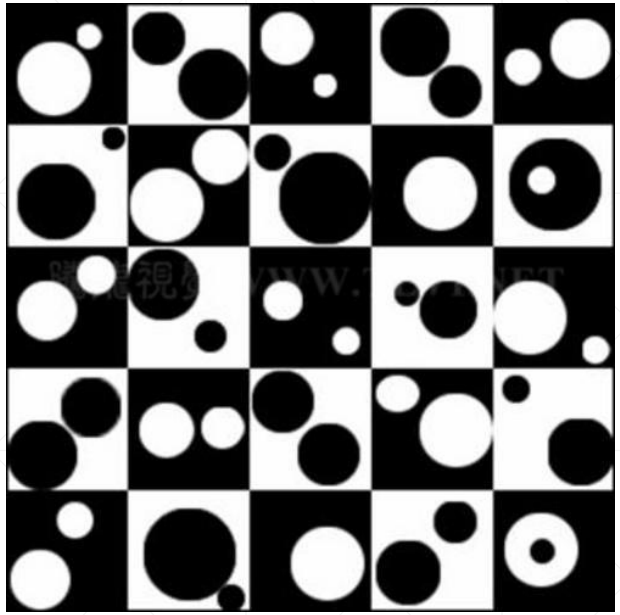
C#中循环语句的使用方法



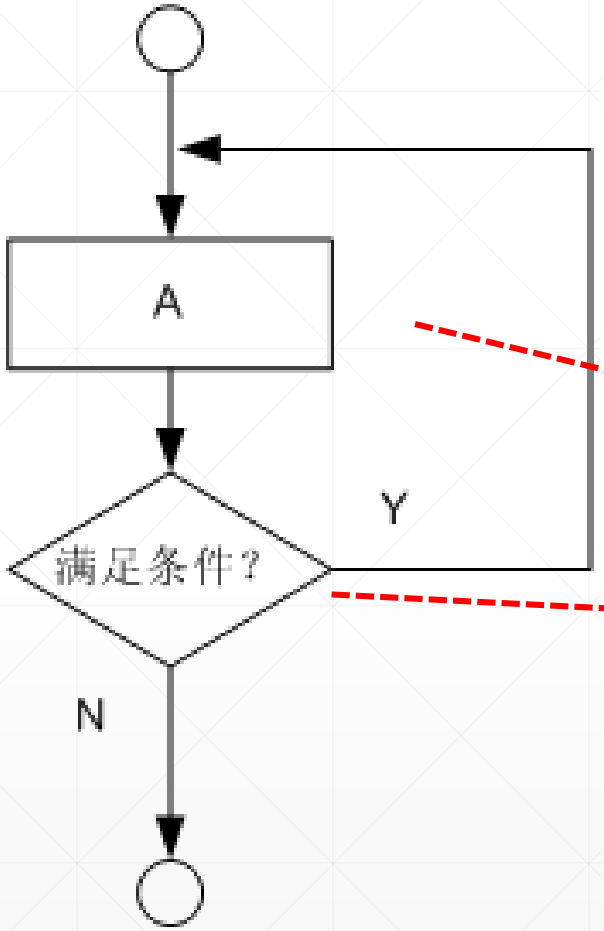
# 重复的艺术

北京理工大学计算机学院
金旭亮

# 程序设计中的循环结构

所谓“**循环结构**”，其实就是在特定的场景下，某些语句将被反复地执行多次。

# do/while循环结构的流程图




```
int i = 0;
do
{
    Console.Write("{0}", i) ;
    i++;
}
while ( i < 10 );
```

循环体

循环判断条件

do/while循环的特点：

先做事，再判断循环条件是否满足，满足就继续执行循环体。

# while/do循环结构

## 语法

```
while(逻辑表达式)
{
    循环体语句;
}
```

while/do循环的特点：
**先判断循环条件，满足了再做事。**

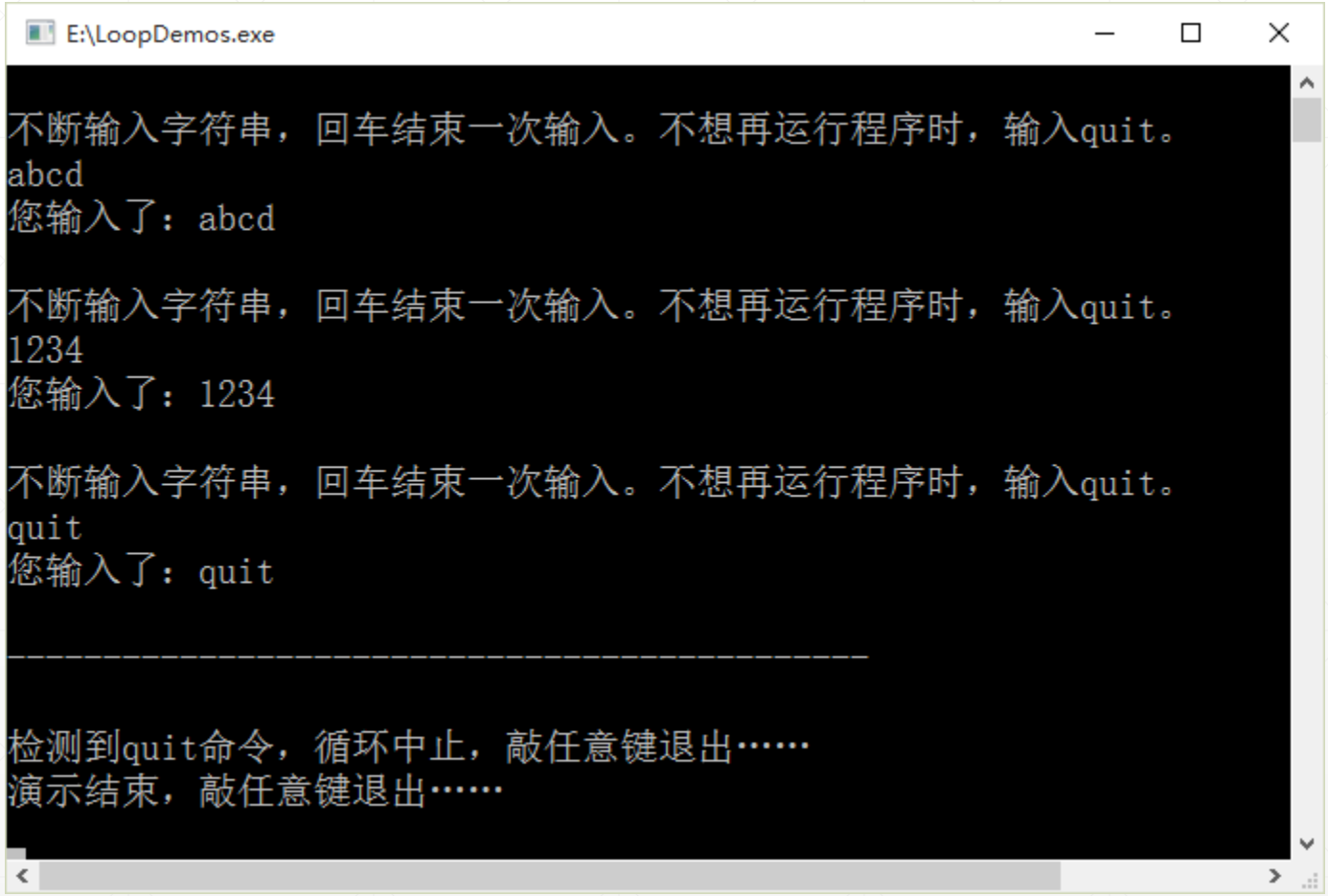
**实例：使用While循环计算1到100的整数之和。**

```
static void SumFrom1To100()
{
    //此变量将用于保存求和结果
    int sum = 0;
    //此变量将作为循环变量
    int counter = 1;
    //只要counter变量的值小于100，就执行循环体语句
    //counter从1到100，循环体语句一共执行了100次
    while (counter <= 100)
    {
        //使用sum保存每次累加的结果
        sum = sum + counter;
        //上述代码可以简写为：
        //sum += counter;
        //循环变量自增1
        counter++;
    }
    //输出结构
    Console.WriteLine(sum);
}
```

参看测试方法SumFrom1To100()

# 循环结构的一个应用场景

我想编写一个控制台程序，让用户不断地输入，直到他输入某个特殊值.....




参看测试方法：
InputQuitToStop()

# 学了马上用……

编写一个考试成绩平均分计算程序。

用户不断地输入考试成绩，直到他输入“-1”，结束输入。
程序给出以下处理结果：

您一共输入了XX个考试成绩，平均分为XXX.XX

# 针对不同的场景，选择合适的处理方式

- 当需要执行次数不定的循环时，使用do/while或while/do循环是最自然的选择。
- 当需要执行次数固定的循环时，使用for循环就变成了最自然的选择。

# for循环语句

```
for（循环初始表达式 ； 循环结束条件 ； 执行完循环体后才执行的表达式）
{
    循环体语句块;
}
```

示例： 使用for循环从1加到100

```
71          static void SumFrom1To100UseFor()
72          {
73              //此变量将用于保存求和结果
74              int sum = 0;
75              for(int i = 1; i <= 100; i++)
76              {
77                  sum += i;
78              }
79              //输出结果
80              Console.WriteLine(sum);
81          }
```

**可以使用断点，单步跟踪执行代码，**
**了解for循环各组成部分的执行顺序**
**下图为Visual Studio中的调试工具栏**


单步执行（调试时会跳入被调用方法的内部代码）

单步执行（不进入方法内部）

# 循环的退出与提前中止.....

**break;**　　**continue;**

**break**：提前结束循环，后面还没有执行的循环也不再执行。

**continue**：提前结束当前循环（本轮循环中还没有执行的代码不再执行），后面的循环继续执行。

参看测试方法：
BreakAndContinue

# “永远不结束”的“死循环”

在某些场景，无法预知到底要执行多少轮循环，也不知道要运行多长时间，举几个例子……

Windows需要时刻监控鼠标与键盘，以便及时地响应用户操作。

某股票软件需要及时地提取股票信息，将涨跌信息及时地通知用户

防病毒软件需要在后台监控各种活动，发现危险时向用户报警

在这些场景中，使用“死循环”是合理的

# 使用C#实现“死循环”

```
while( true )
{
    循环体;
}
```

```
for( ; ; )
{
    循环体;
}
```

```
do
{
    循环体;
} while( true );
```

使用这种编程方式，在循环体中，一定要有一条检测语句，检测特定的条件是否满足，当条件满足时，使用“break”（或“return”）退出循环……

# 示例与试验

使用“死循环”的方法编程计算1到100000的和。

```
static void SumUseInfinteLoop()
{
    long sum = 0;
    int counter = 1;
    while (true)
    {
        sum += counter;
        counter++;
        if (counter > 100000)
            break;
    }
    Console.WriteLine(sum);
}
```

**动手试验指导：**

1 输入左边代码，查看运行结果
2 把变量sum的类型由long改为int，再看运行结果
3 把break;改为continue，运行，出现什么情况？
4 如何用for的“死循环”方式实现同样的功能？

请认真做完上述试验，并尝试回答相关问题。如果回答不出，请通过互联网或询问老师同学解决之。

# 使用循环结构访问数据集合

在面向对象的软件中，我们会经常遇到“数据集合”这一概念。“数据集合”，顾名思义，就是“数据的集合”，在实际开发中，有两种类型的数据集合我们经常会遇到……

保存int、float等值类型数据的集合，如“List<int>”

保存string和自定义类等引用类型数据的集合，如List<MyClass>，又称为“对象集合”

要遍历这两种类型的数据集合，我们可以使用**foreach**循环……

# 数据集合的“遍历”

所谓“**遍历**”，换一个说法，就是“**逐个地访问**”。

遍历数据集合最自然的方式，就是使用**foreach**循环。

```
//遍历整数集合
var IntValues = new List<int>() { 1, 2, 3, 4 };
foreach (var value in IntValues)
{
    Console.WriteLine(value);
}
```

**IntValues**是被遍历的集合，在每轮循环中，**value**变量依次引用集合中的每个对象。

```
var MyClasses = new List<MyClass>();
//向集合中追加5个对象
for (int i = 0; i < 5; i++)
{
    MyClasses.Add(new MyClass()
    {
        Id = i,
        Description = "MyClass对象" + i
    });
}
//遍历对象集合
foreach (var obj in MyClasses)
{
    Console.WriteLine("{0}:{1}",obj.Id,obj.Description);
}
```

参看测试方法：ForEachDataCollection()

# “遍历数据集合”时的注意事项

使用foreach遍历数据集合时，不要向集合中增删数据。

```
//遍历整数集合
var IntValues = new List<int>() { 1, 2, 3, 4 };
foreach (var value in IntValues)
{
    //取消以下注释，尝试在遍历过程中从集合中移除数据，
    //将会遇到“System.InvalidOperationException”异常
    //if (value == 2)
    //    IntValues.Remove(value);
    Console.WriteLine(value);
}
```

foreach访问数据集合时，应该让数据集合保证稳定不变。

# 使用foreach循环遍历一个数组

“数组（Array）”也可以看成是一个数据集合。

```
//定义一个整数数组
int[] numbers = { 1, 3, 5, 7, 9 };

//输出所有数组元素
foreach (var number in numbers)
{
    Console.WriteLine(number);
}
```

# 循环语句的等价性原则

各种类型的循环语句都是等价的，可以把一种类型的循环语句用另一种类型的循环语句替换而不会影响到程序的功能。

# 检测一下你的学习成果

1. 请编程计算出以下表达式的值：

   **2-4+6-8+10-…+100=?**

2. 使用控制台程序，生成以下图形

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
```

```
* * * * * *
   * * * * * *
* * * * * *
   * * * * * *
* * * * * *
   * * * * * *
```

# foreach的背后不简单

从表面上看，foreach与前面介绍过的for, do/while等循环并没有什么区别，但实际上，两者是有很大区别的。

C#编译器在编译foreach语句时，会将其转换为使用IEnumerable与IEnumerator的相应代码，并且C# 8还引入了序列和异步循环迭代等新特性，这些内容，当前暂不介绍，而安排在后继的其他课程中，等你掌握了相应的知识之后再学习。

扩充学习

# 控制台应用程序编程小技巧

# 开发场景

当我们编写控制台应用程序时，经常需要知道用户是否按了某些特殊的键（比如F1键），于是问题出来了.....

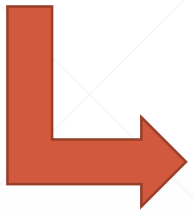
（1）我怎么知道用户按的是哪个键?

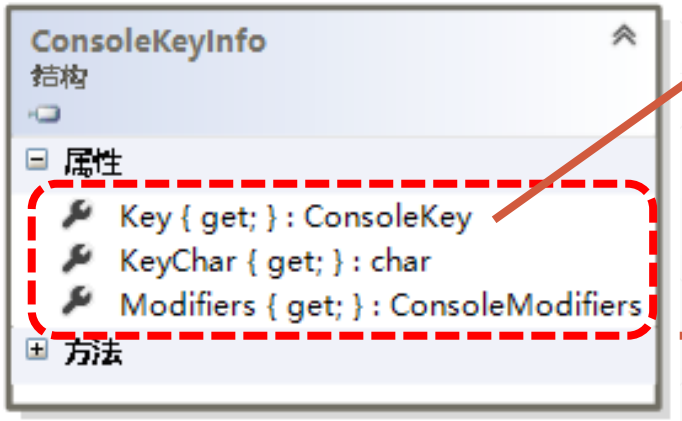
（2）怎样编程来检测用户的按键?

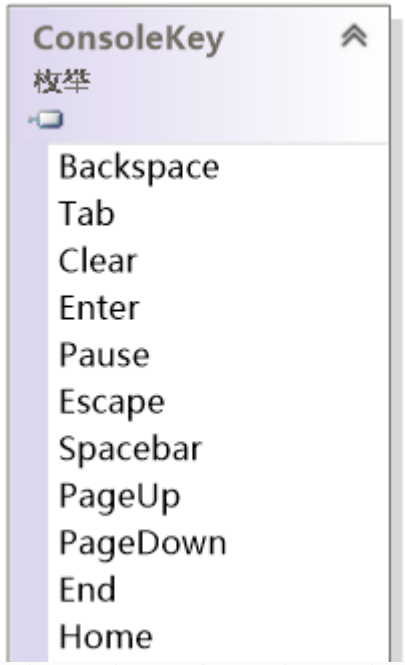
# 问题的解决方案：

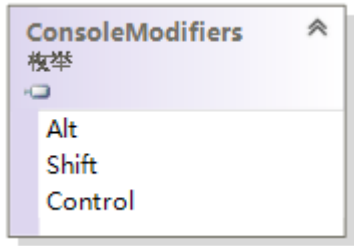
Console.ReadKey()方法，返回一个ConsoleKeyInfo结构……

public static **ConsoleKeyInfo** ReadKey(bool intercept);



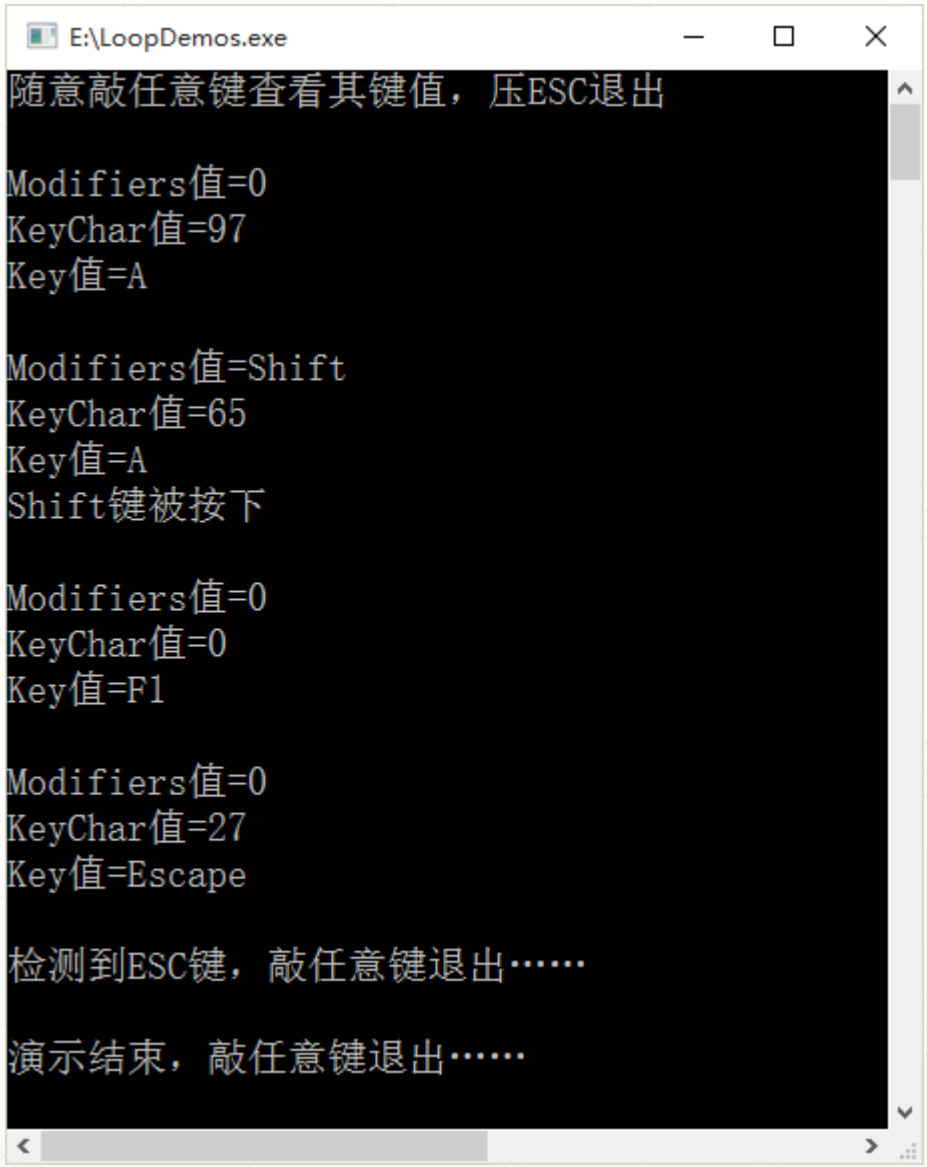



通过这三个属性，可以知道用户按的是什么键。

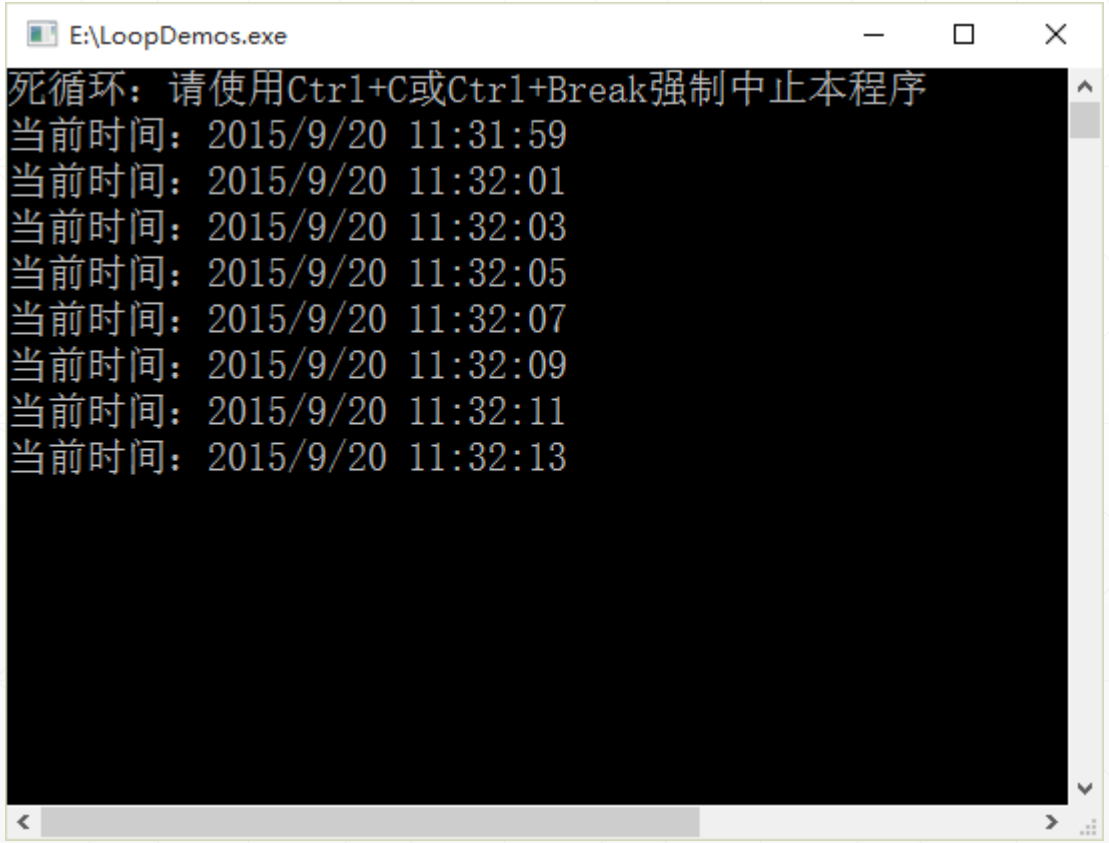
# 检测按键实例




参看测试方法：testKey

# 内置的退出功能键

当一个控制台程序正在运行时，默认情况下，用户可以使用“Ctrl+C”或“Ctrl+Break”强制中止它的运行。




参看测试方法：QuitConsoleApp

要禁用“Ctrl+C”的功能，可以设置以下属性

```
Console.TreatControlCAsInput = true;
```

参看测试方法：DisableControlC

# 屏蔽掉Ctrl+C和Ctrl+Break


注意把握的编程技巧：
（1）如何响应事件？
（2）如何区分是哪个组合键被按下？

参看测试方法：UseCancelKeyPress()

# 学以致用

写一个控制台程序，当它运行时，敲击键盘上的任意一个键，显示当前时间，使用ESC键或Ctrl + Q组合键退出。