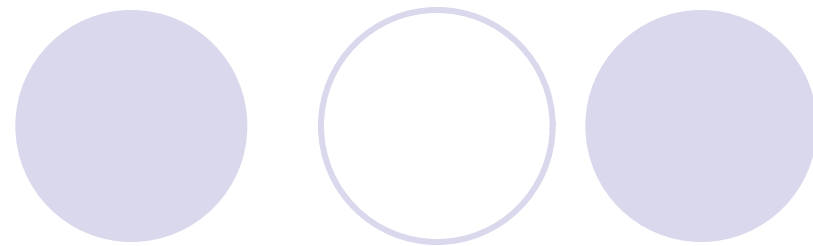


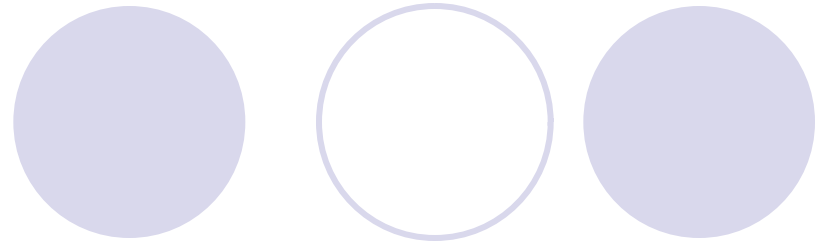
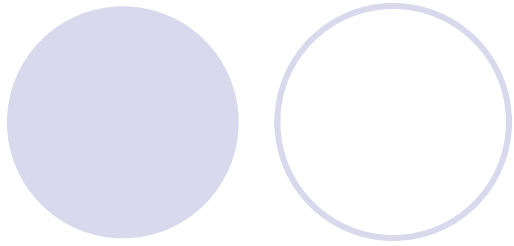
第4章 C#控制语句及数组的使用

- 程序包含若干语句，而语句的执行流程分为3种基本结构，即顺序结构、选择结构和循环结构，顺序结构是顺序执行的一组语句，容易理解。本章主要介绍选择结构和循环结构。另外还介绍了跳转语句及数组的使用

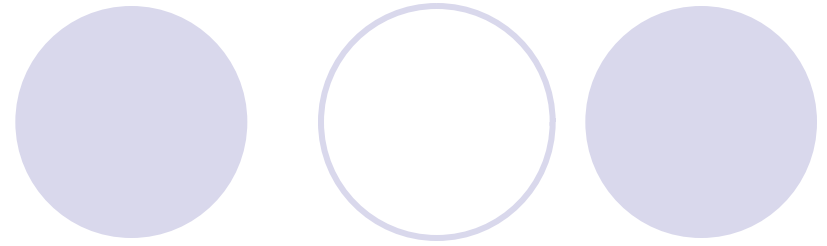
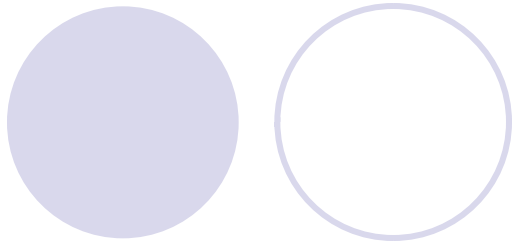
4.1 选择结构语句



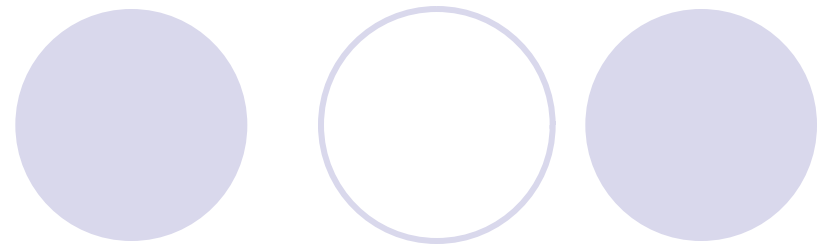
- 选择结构(语句)就是条件判断语句，它能让程序在执行时根据特定条件是否成立而选择执行不同的语句块。**C#**提供两种选择语句结构，**if**语句和**switch**语句。
- 4.1.1 **if**语句
- **if**语句在使用时可以有几种典型的形式，它们分别是：**if**框架、**if_else**框架、**if_else if**框架，以及嵌套的**if**语句。



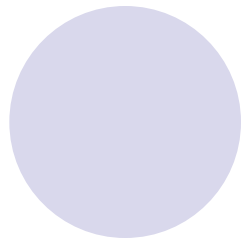
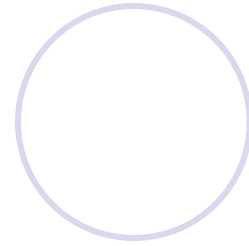
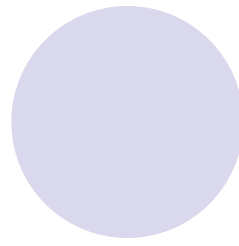
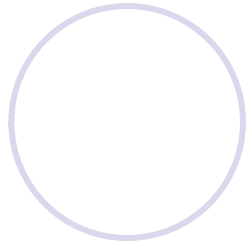
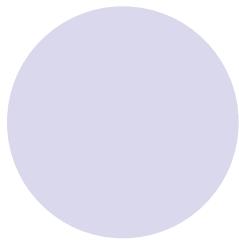
- 1. if框架
- 形式: if (条件表达式) 语句;
- 若条件表达式的值为真, 则执行if后的语句(块), 否则不做任何操作, 控制将转到if语句的结束点。
- 【例4-1】 求一个数的绝对值。
- if ($x < 0$) $x = -x$; //取x的绝对值
- 【例4-2】 计算三角形面积。



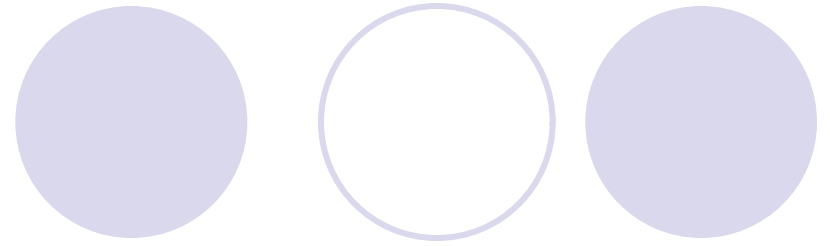
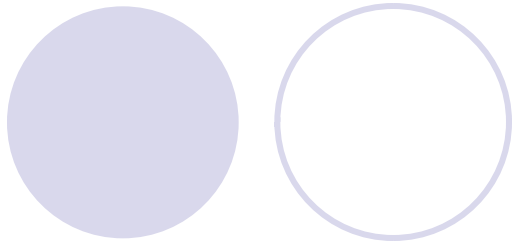
- `int a,b,c;`
- `double p, s;`
- `a=Convert.ToInt32(Console.ReadLine());` //ReadLine()从键盘接收一个字符串
- `b=Convert.ToInt32(Console.ReadLine());` //Convert.ToInt32()转换为整型
- `c=Convert.ToInt32(Console.ReadLine());`
- `if(a+b>c && b+c>a && a+c>b)` //判断读入的三个数是否构成三角形
- {
- `p = (a+b+c) / 2 ;`
- `s = Math.Sqrt (p * (p-a) * (p-b) * (p-c)) ;` //Sqrt求平方根
- `Console.WriteLine("{0}",s);` //输出三角形的面积
- `Console.ReadLine();`
- }



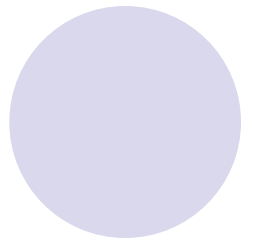
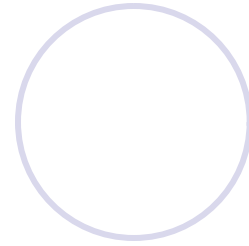
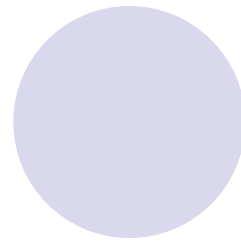
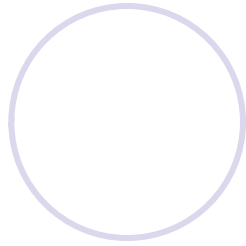
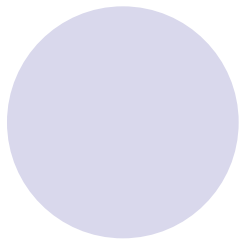
- 2. **if_else**框架
- 形式:
- **if** (条件表达式)
- 语句1;
- **else**
- 语句2;
- 该语句的作用是当条件表达式的值为真时，执行**if**后的语句(块)1，否则执行**else**后的语句(块)2。



- 【例4-3】 使用if_else框架计算三角形面积。
- //判断数据合法性，关系运算符优先级高于逻辑运算符
- if (a+b>c && b+c>a && a+c>b)
 //如果条件成立
- {
- p = (a+b+c) / 2 ;
- s = Math.Sqrt (p * (p-a) * (p-b) * (p-c)) ; //求三角形
面积
- Console.WriteLine("{0}",s);
- }
- else //如果条件不成立(若数据不构成三角形)，则输
出提示信息
- Console.WriteLine (" 三角形的三条边数据有错！ ");



- 3. **if_else** if框架
- 形式:
- **if** (条件表达式1)
- 语句1 ;
- **else if** (条件表达式2)
- 语句2 ;
- **else if** (条件表达式3)
- 语句3 ;
-
- **[else**
- 语句n ;]



- **if_else if** 框架执行过程：从上向下顺序计算相应的条件表达式，如果相应的条件表达式为真则执行相应语句，然后跳过框架的剩余部分，**if_else if** 框架就此结束，直接执行框架后面的语句；如果结果为假，则继续向下计算相应的条件表达式，直到所有的条件表达式都不成立，则执行这个语句的最后部分**else**所对应的语句，如果没有**else** 语句就什么也不做，结束**if_else if** 框架。
- **【例4-4】** 用户从控制台输入一个分数(0~100)，系统根据分数判断分数等级(A~E)并输出。如果分数范围不对，则输出错误提示信息。


```
static void Main(string[] args)
```

```
{
```

```
    String grade;
```

```
    int score;
```

```
    Console.WriteLine("Input your score:");
```

```
    score = Convert.ToInt32(Console.ReadLine());
```

```
    if (score > 100 || score < 0)
```

```
    {
```

```
        Console.WriteLine("False score range!");
```

```
        return;
```

```
    }
```

```
    if (score > 90)
```

```
        grade = "A";
```

```
    else if (score > 80)
```

```
        grade = "B";
```

```
    else if (score > 70)
```

```
        grade = "C";
```

```
    else if (score > 60)
```

```
        grade = "D";
```

```
    else
```

```
        grade = "E";
```

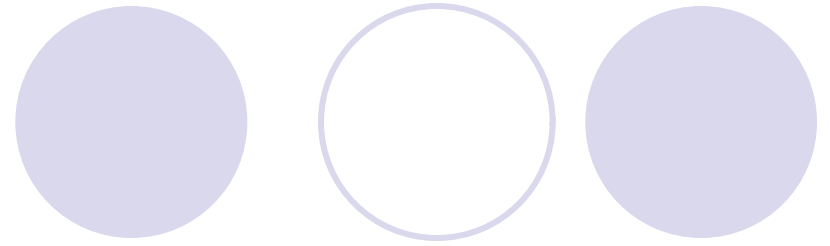
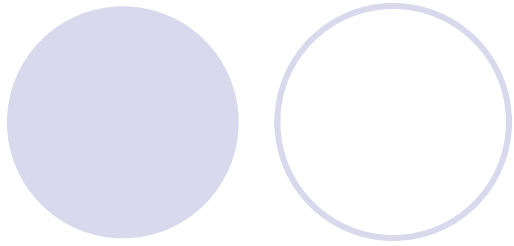
```
    Console.WriteLine("Your grade is {0}!", grade);
```

```
    Console.ReadLine();
```

```
}
```

```
}
```

- 下面根据mark分数，输出不同等级。但该程序段运行结果只有“及格”和“不及格”两种情况。请大家思考为什么会出现这样的错误。
- if (mark >= 60)
- Console.WriteLine ("及格");
- else if (mark >= 70)
- Console.WriteLine ("中");
- else if (mark >= 80)
- Console.WriteLine ("良");
- else if (mark >= 90)
- Console.WriteLine ("优");
- else
- Console.WriteLine ("不及格");



- 4. 嵌套的if语句
- 在if语句框架中，无论条件表达式为真或假，将要执行的语句都有可能又是一个if语句，这种在if语句块中又包含if语句的结构称为嵌套的if语句。为避免产生歧义，**C#**规定**else**语句与和它处于同一模块最近的if相匹配。

【例4-5】 假设有一函数

$$y = \begin{cases} a & (x > 0) \\ 0 & (x = 0) \\ -a & (x < 0) \end{cases}$$

- `y=0;`
- `if(x>=0)`
- `if(x>0)`
- `y=a;`
- `else y=-a;`
- 应修正为:
- `y=0;`
- `if(x>=0)`
- `{ if(x>0)`
- `y=a;`
- `}`
- `else y=-a;`

4.1.2 switch语句

- switch语句是一个多重选择语句，它可以根据表达式的值从多个分支中选择一个来执行，其形式为：

- switch(表达式)

- {

- case 常量1:

- 语句块1; //由零个或多个语句组成

- break(或goto,return);

- case 常量2:

- 语句块2;

- break(或goto,return);

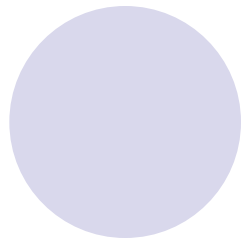
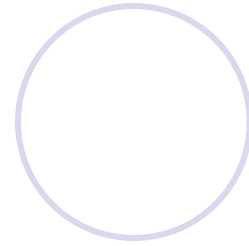
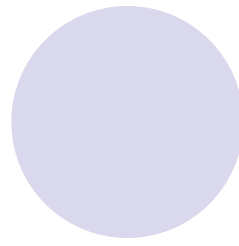
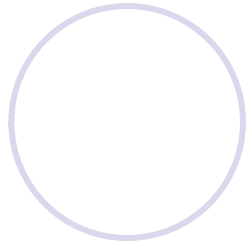
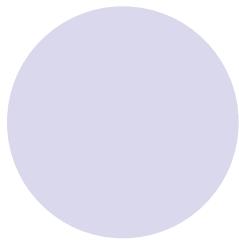
-

- [default: //default是任选项，可以不出现

- 语句块n;

- break(或goto,return);]

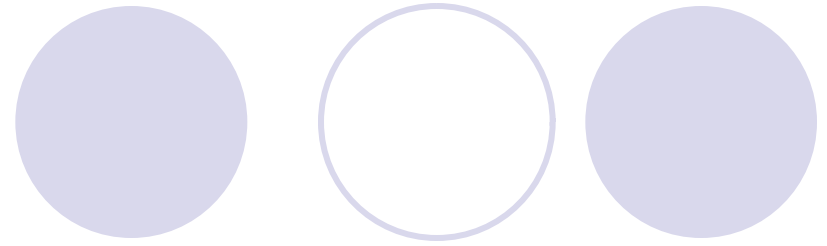
- }



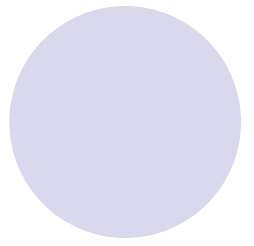
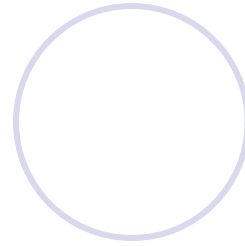
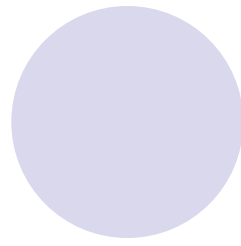
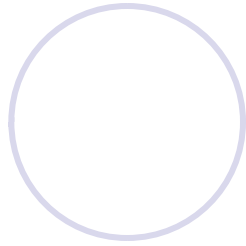
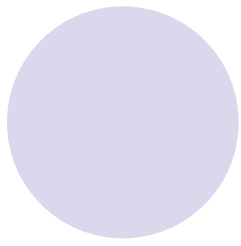
- **switch**语句的执行流程是，首先计算**switch**后表达式的值，然后把该值与各个**case**子句中的常量值比较，如果相等，则执行相应的**case**子句中的语句块，直到遇到跳转语句(**break**)，**switch**语句执行结束；如果与**case**子句都不等，则执行**default**子句中的子句块；如果没有**default**，则执行**switch**模块后面的代码。
- **switch**语句的表达式必须是整数类型，如**char**、**sbyte**、**byte**、**ushort**、**short**、**uint**、**int**、**ulong**、**long**或**string**、枚举类型。**case**常量必须与表达式类型相兼容，**case**常量的值必须互异，不能重复。
- 可以有二个或多个**case**前缀指向相同的语句序列。

- **【例4-6】** 从键盘输入一个整数，如果该整数值是1、3，则输出该值；如果是4，则输出“x=1”；如果是其他值，则显示“x为其他值”。
- `int x;`
- `x = Convert.ToInt32(Console.ReadLine());`
`//Convert.ToInt32()转换为整型`
- `switch (x)`
- `{`
- `case 1: Console.WriteLine("x=1"); break;`
- `case 2: goto default;`
- `case 3: Console.WriteLine("x=3"); break;`
- `case 4: goto case 1;`
- `default: Console.WriteLine("x为其他值"); break;`
- `}`
- `Console.ReadLine();`
- `//为了看清显示结果，等待键盘输入，也表示switch模块后面的代码`

4.2 循环控制语句



- 循环语句用于反复执行一些操作，可以指定循环次数或循环条件。**C#**提供了四种循环语句：**while**，**do while**、**for**和**foreach**。**foreach**语句主要用于遍历集合中的元素，如对数组对象，可以用**foreach**遍历数组的每个元素。
- 4.2.1 **while**语句
- **while**语句用于循环执行指定的语句块，直到给定表达式的值为假(**False**)。**while**语句在执行指定语句块之前首先要判断表达式的值。若条件表达式的值为真(**True**)，则执行循环体语句，否则结束循环。
- 形式：
- **while** (条件表达式)
- { 循环体语句; }



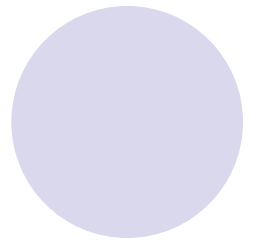
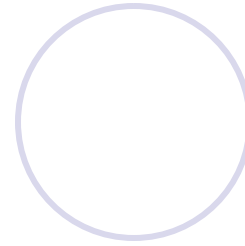
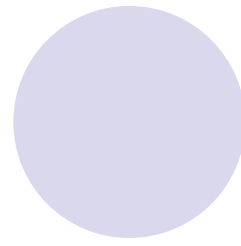
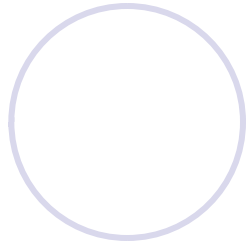
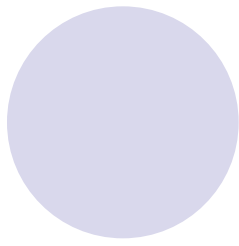
- 【例4-7】 使用while语句输出0~9的整数。
- `int x=0;`
- `while (x < 10)`
- `{`
- `Console.WriteLine(x); //执行10次循环`
- `x++;`
- `}`
- `Console.ReadLine();`

4.2.2 do while语句

- do while语句首先执行循环体语句，再判断条件表达式。如果条件表达式为真(True)，则继续执行循环体语句；如果表达式的值为假(False)，则退出循环。
- 形式：
- do
- { 循环体语句 ; }
- while (条件表达式);
- 【例4-8】 使用do while语句输出0~9的整数。
- int x=0;
- do
- {
- Console.WriteLine(x);
- x++;
- } while (x < 10);
- Console.ReadLine();

4.2.3 for语句

- **for**语句是构成循环的最灵活简便的方法，其一般格式为：
- **for** (表达式1; 表达式2; 表达式3)
- {
- 语句块;
- }
- 表达式1完成循环变量的初始化。表达式2判断循环是否继续执行。表达式3修改循环变量的值，控制循环条件
- 3个表达式之间用分号隔开，它的执行过程是：
- (1) 先计算表达式1的值。
- (2) 求解表达式2的值，若表达式2条件成立，则执行**for**语句的循环体语句块，然后执行第(3)步；若条件不成立，则转到第(5)步。
- (3) 求解表达式3。
- (4) 转回到第(2)步执行。
- (5) 循环结束，执行**for**语句后面的语句。




- **【例4-9】** 编程计算10以内自然数的累加和。
- `private void button1_Click(object sender, EventArgs e) //`
在按钮的单击事件中编写代码
- `{`
- `int s = 0;`
- `for (int i = 1; i <= 10; i++) //`如果想了解循环的详细过程，请按单步运行F11
- `{`
- `s += i;`
- `}`
- `label1.Text = "1~10的累加和= " + s; //`在label1中显示循环后的计算结果
- `}`

4.3 跳转语句

- 跳转语句用于改变程序的执行流程，转移到指定语句之处。跳转语句包括**continue**语句、**break**语句、**return**语句、**goto**语句。
- 1. **break**语句
- 如果循环语句执行到**break**语句，则从循环体跳出，结束循环。不管循环有多少层，**break**语句只能使程序跳出**break**本身所在的循环，而不能跳出所有的循环。
- 如果在**switch**语句中执行到**break**语句，则立刻从**switch**语句中跳出，转到**switch**语句的下一条语句。

- 2. **continue**语句
- **continue**语句用于循环语句之中，作用是结束本轮循环(剩余语句没有执行的必要，而跳过剩余语句)，然后执行下一轮循环(没有从循环结构中退出)。
- 如果**continue**语句在多重循环结构之中，它只对包含它最内层循环有效。
- 3. **return**语句
- 语法形式：
- **return**;
- 或
- **return** 表达式;
- **return**语句出现在一个方法内，在方法中执行到**return**语句时，程序流程转到调用这个方法处。如果方法没有返回值(返回类型修饰为**void**)，则使用**return**返回；如果方法有返回值，那么使用**return** 表达式的形式，其后面跟的表达式就是方法的返回值。

- 
- 4. **goto**语句
 - 语法形式:
 - **goto** 标号;
 - 其中标号就是定位在某一语句之前的一个标识符, 这称为标号语句, 它的格式是:
 - 标号: 语句;
 - 它给出**goto**语句转向的目标。值得注意的是, **goto**语句不能使控制转移到另一个语句块内部, 更不能转到另一个方法内部。
 - 另外**goto**语句如果用在**switch**语句中, 它的格式是:
 - **goto case** 常量;
 - **goto default** ;
 - 它只能在本**switch**语句中从一种情况转向另一种情况。

- 【例4-10】 从键盘输入不同的数，用单步运行(F11)看看运行结果。
- `int x;`
- `x = Convert.ToInt32(Console.ReadLine());`
- `while(x<10)`
- `{ x++;`
- `Console.WriteLine(x);`
- `if (x == 3) continue;` //不执行后面的语句，直接跳到下一轮循环
- `if (x == 5) break;` //跳出循环，执行while后面的语句
`Console.ReadLine();`
- `if (x == 8) return;` //跳出循环，并退出本循环体所在的子程序
- `}`
- `Console.ReadLine();`

4.4 数组

- 数组是一种数据结构，它包含相同类型的一组数据。
- 4.4.1 数组的定义
- 数组必须先声明。因为数组类型为引用类型，数组变量的声明是为数组实例的引用留出空间。
- 1. 一维数组
- (1) 一维数组声明语法形式
- 类型名 [] 数组名;
- 例如： `int[]A;` 声明一个数组名为A的一维整型数组。

(2) 创建数组对象

数组在声明后必须实例化才能使用。数组实例在运行时使用**new**运算符动态创建。声明数组和创建数组分别进行，**new**运算符自动将数组的元素初始化为相应的默认值。实例化一维数组的一般形式：

- 类型名称 [] 数组名; //数组声明

● **数组名 = new 类型名 [数组长度]; //创建数组实例**

- 例如: `int[] A; A=new int[6];`

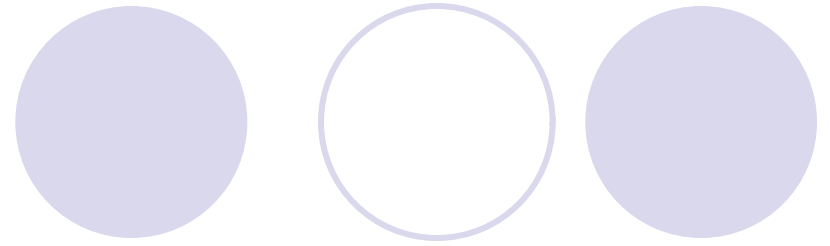
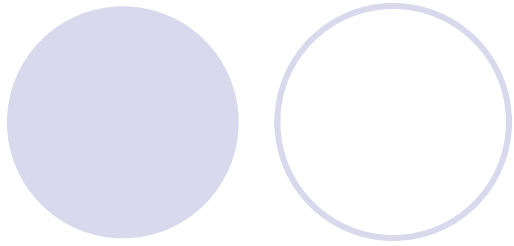
- 数组长度表明数组元素的个数，它是一个大于或等于零的整数。

● (3) 声明数组和创建数组实例也可以合在一起写

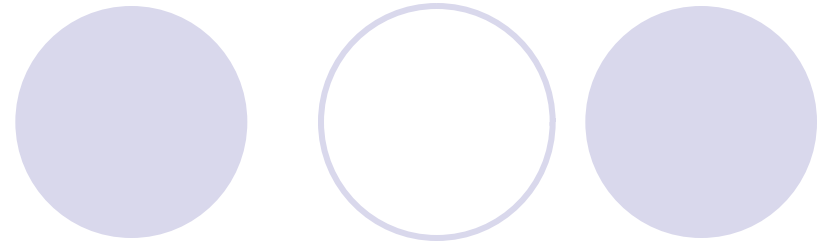
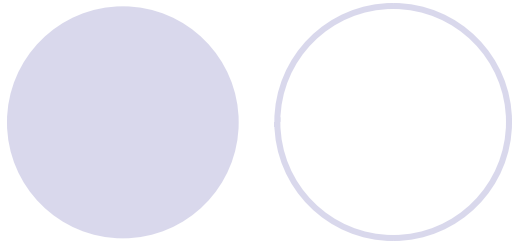
- **类型名 [] 数组名 = new 类型名 [数组长度];**

- 例如: `int [] B=new int[6];`

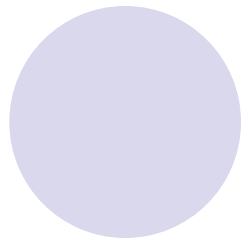
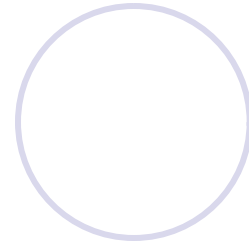
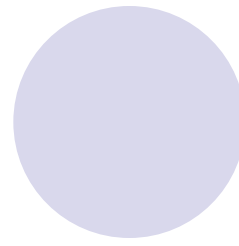
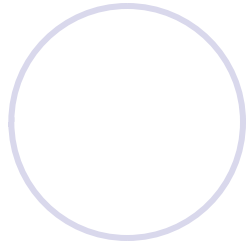
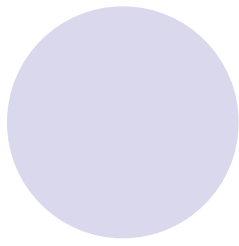
- **//B是一个有6个int类型元素的数组**



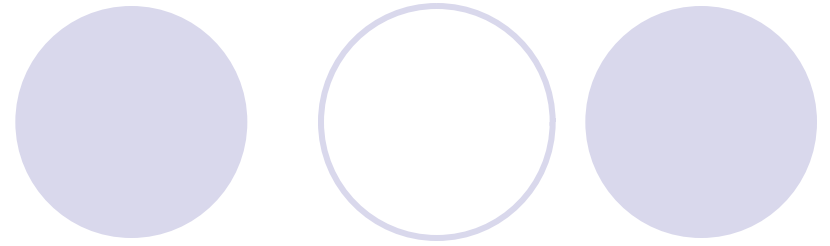
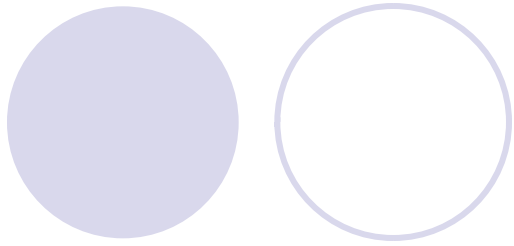
- 2. 多维数组
- (1) 多维数组声明语法形式
- 类型名 [, , , ...] 数组名;
- 多维数组是指能用多个下标访问的数组。在声明时方括号内加逗号，就表明是 multidimensional array，有 n 个逗号，就是 $n+1$ 维数组。
- 例如：
- `int [,] A;`
 二维数组
 //A是一个int类
- `float [, ,] B;`
 三维数组
 //B是一个float类型的



- (2) 创建数组对象
- 声明数组和创建数组分别进行:
- 类型名 [, , ...] 数组名 ; // 声明数组
- 数组名 = new 类型名 [长度1, 长度2, 长度3, ..., 长度n+1];
// 创建数组实例
- 长度1、长度2、长度3、...、长度n+1分别表明多维数组每一维的元素个数。
- 例如:
- int [,] A ;
- A = new int [3, 4] ; // A是一个3行4列的二维数组



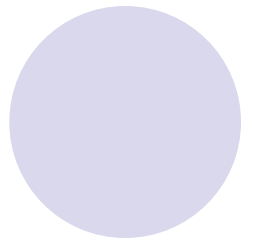
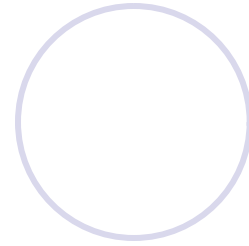
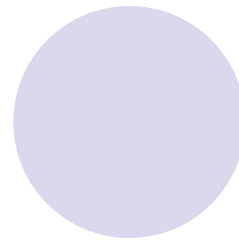
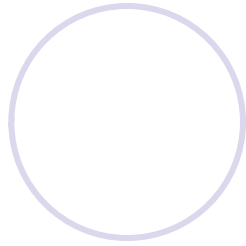
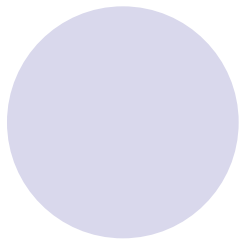
- (3) 声明数组和创建数组实例也可以合在一起写
- 类型名 [, , ...] 数组名 = new type [size1, size2, size3, ..., size n+1];
- 例如:
- `float [, ,] B=new float [2, 3, 4]` //B是一个三维数组, 每维长度分别是2、3、4
- (4) 不规则数组
- 一维数组和多维数组都属于矩形数组, 而C#所特有的不规则数组是数组的数组, 它的内部每个数组的长度可以不同, 就像一个锯齿形状。
- 不规则数组的声明:
- 类型名[][][] 数组名;
- 方括号[]的个数与数组的维数相同。



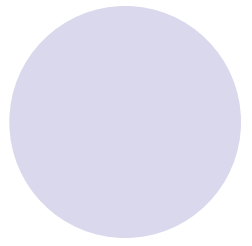
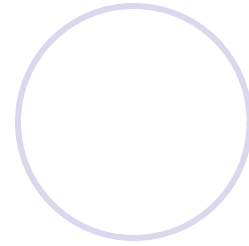
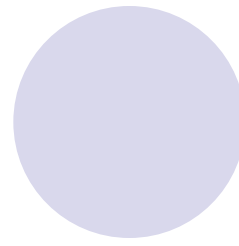
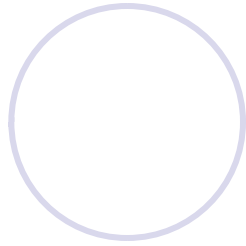
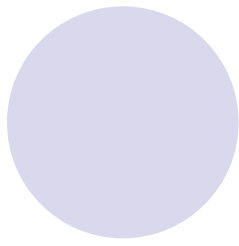
- 例如:
- `int [][] B;` //B是一个int类型的二维不规则数组
- 创建数组对象，以二维不规则数组为例:
- `int [][] B;` //声明二维不规则数组
- `B = new int [3][];` //实例化B，B有3行
- `B[0] = new int [4];`//B的第一行有4个元素
- `B[1] = new int [2];`//B的第二行有2个元素
- `B[2] = new int [3];`//B的第三行有3个元素

4.4.2 数组的初始化

- 1. 一维数组初始化
- 语法形式1：类型名 [] 数组名 = new 类型名 [长度] { 初始值设定 };
- 数组声明与初始化同时进行，长度即数组元素的个数，与大括号内的数据个数一致。
- 语法形式2：类型名 [] 数组名 = new 类型名 [] { 初始值设定 };
- 省略了长度，由编译系统根据初始化表中的数据个数，自动计算数组的大小。
- 语法形式3：类型名 [] 数组名 = { 初始值设定 };
- 数组声明与初始化同时进行，还可以省略new运算符及new后面的类型名。
- 语法形式4：类型名 [] 数组名;
- 数组名 = new 类型名 [长度] { 初始值设定 };
- 把声明与初始化分开在不同的语句中进行时，长度可以默认。

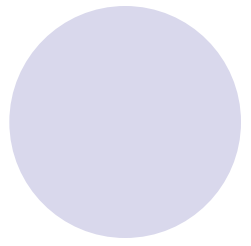
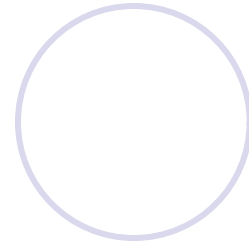
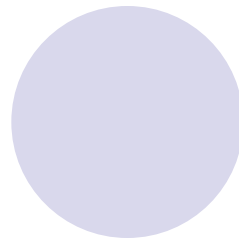
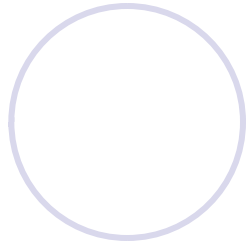
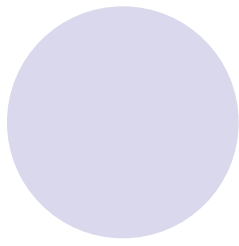


- 例如： 以下数组初始化实例都是等同的。
- `int [] A = new int [10] {0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };`
- `int [] A = new int [] {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};`
- `int [] A = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };`
- `int [] A ;`
- `A = new int [10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };`



- 2. 多维数组初始化
- 多维数组初始化通过将每维数组元素设置的初始值放在各自的一个大花括号内完成。现以最常用的二维数组为例来讨论。
- 语法形式1:
- 类型名 [,] 数组名 = new 类型名 [长度1, 长度2] { { 第一维初始值设定 },
- { 第二维初始值设定 } };
- 数组声明与初始化同时进行, 数组元素的个数是长度1×长度2, 数组的每行分别用一个花括号括起来, 每个花括号内的数据就是这行的每列元素的值, 初始化时的赋值顺序按矩阵的“行”存储原则。

- 说明：
- (1) 可以省略长度1、长度2。编译系统根据初始化表中花括号{}的个数确定行数，再根据{}内的数据确定列数，从而得出数组的大小。
- (2) 还可以省略new运算符。
- 例如，以下数组初始化实例都是等同的。
- `int [,] B = new int [3,4] {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}};`
- `int [,] B = new int [,] {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}};`
- `int [,] B = {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}};`
- 语法形式2：
- 类型名 [,] 数组名;
- 数组名 = new 类型名 [长度1, 长度2] {{ 第一维初始值设定 },
- { 第二维初始值设定 } };
- 把声明与初始化分开在不同的语句中进行时，长度1、长度2同样可以省略。
- `int [,] B ;`
- `B = new int [3, 4] {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}};`



- 3. 不规则数组初始化
- 下面以二维不规则数组为例。不规则数组是一个数组的数组，所以它的初始化通常是分步骤进行的。
- 类型名 `[] [] 数组名 = new 类型名 [长度1] [];`
- 长度1可以是常量或变量，后面中括号`[]`内是空着的，表示数组的元素还是数组，且这个数组的长度是不一样的，需要单独再用`new`运算符生成。
- 例如：
- `char [] [] MY1 = new char [3] [];` // MY1是由三个数组组成的数组
- `MY1[0] = new char [] {'R', 'I', 'C', 'H'};`
- `MY1[1] = new char [] {'L', 'U', 'S', 'H', 'A'};`
- `MY1[2] = new char [] {'K', 'K', 'A', 'B', 'B', 'A', 'L', 'A'};`

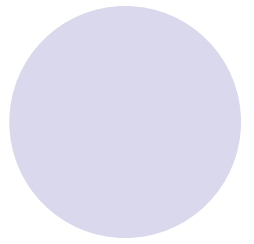
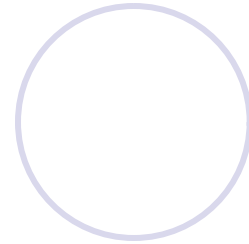
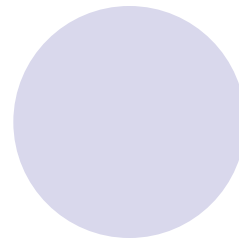
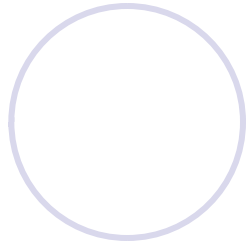
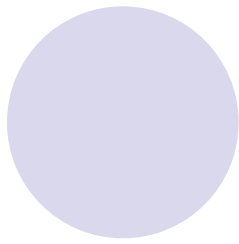
4.4.3 访问数组中的元素

- 引用数组元素与引用普通变量没有本质的区别，只是数组必须通过数组名和下标来引用。下面通过例题了解数组元素的引用。
- 1. 一维数组的引用
- **【例4-11】** 程序运行后，单击按钮，在长度为10的数组a[]中，每个元素存放两位随机正整数，计算它们的累加和，并在标签上显示运算结果

```

private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "随机数: ";
    Random rnd = new Random();           //声明随机数对象
    rnd
    int sum = 0;                          //sum是保存累加和的变量，初值
    为0
    int[] a = new int[10];               //定义数组并实例化
    for (int i = 0; i < 10; i++) //循环语句
    {                                     //a[i]为引用数组中的第i个
    元素
        a[i] = rnd.Next(10, 100);
        //给每个数组元素赋初始值，范围为10~99
        sum += a[i];
        label1.Text += a[i] + " "; //10个随机数在标签上输出成一行
    }
    label1.Text += "\n\n累加和 = " + sum; //\n为换行
}

```



- 2. 多维数组的引用(以二维数组为例)
- 【例4-12】 在标签Label中显示一个由一位随机数组成的 6×6 矩阵，并在矩阵的下方显示每行元素中的最小值。

```
private void button1_Click(object sender, EventArgs e)
```

```
{ label1.Text="";
```

```
Random rnd = new Random();
```

//声明随机数对象rnd

```
int[,] a = new int[6, 6];
```

```
int[] minnum = new int[6];
```

//用来存放每行元素中最小值的一

维数组

```
for (int i = 0; i < 6; i++)
```

```
{
```

```
minnum[i] = 9;
```

```
for (int j = 0; j < 6; j++)
```

```
{
```

```
a[i, j] = rnd.Next(0, 10);
```

//用随机数0~9为二维数组的每个元素赋值

```
label1.Text += a[i, j] + " ";//注意二维数组元素的引用a[i, j]
```

```
if (a[i, j] < minnum[i]) minnum[i] = a[i, j];
```

```
}
```

```
label1.Text += "\n";
```

//\n为换行

```
}
```

```
for (int i = 0; i < 6; i++)
```

```
{
```

```
label1.Text += "第" + i + "行的最小值=" + minnum[i] + "\n";
```

```
}
```

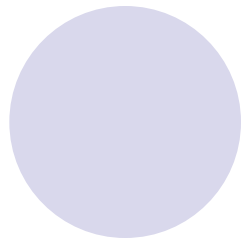
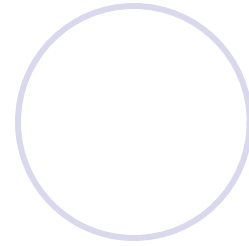
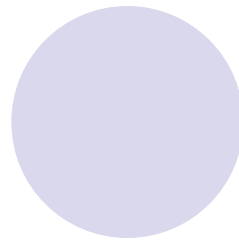
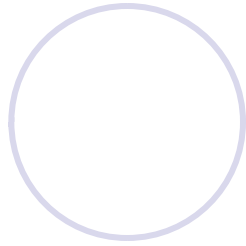
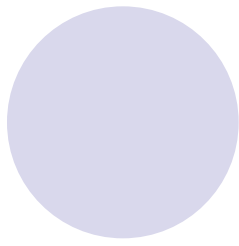
```
}
```

4.4.4 数组与System.Array

- 在C#中，数组实际上是对象，数组类型是从System.Array派生的引用类型。System.Array是所有数组类型的抽象基类型。System.Array类提供了许多实用的方法和属性，可用于数组的复制、排序等操作处理

4.4.5 使用foreach语句遍历数组元素

- C#的foreach语句用于枚举数组或对象集合(后续章节将介绍)中的元素，并对该数组或集合中的每个元素执行一次相关的嵌入语句。foreach语句用于循环访问数组或集合以获取所需信息。当为数组或集合中的所有元素完成迭代后，控制传递给foreach块之后的下一条语句。
- 语法形式：
- 图4.4 例4-13运行界面
- foreach(类型 标识符 in 集合表达式)
- 语句;
- 其中，标识符是foreach循环的迭代变量，它只在foreach语句中有效，并且是一个只读局部变量，也就是说在foreach语句中不能改写这个迭代变量。它的类型应与集合的基本类型一致。



- **【例4-13】** 使用**foreach**语句遍历字符串数组**b**中的每个元素，单击按钮时，在标签上按每行显示一个元素值的格式输出。运行结果如图4.4所示，代码如下：
- `private void button1_Click(object sender, EventArgs e)`
- `{ label1.Text="我的爱好是： \n";`
- `string[] b = { "羽毛球", "游泳", "旅游", "听音乐", "跳舞"`
- `};`
- `foreach (string str in b)`
- `{`
- `label1.Text += str + "\n";` //每行显示
- 一个元素值
- `}`
- `}`