C#语法基础

学习内容

C#语言的产生和发展

C#语言的基础知识

C#流程控制语句

学习目标

掌握C#程序的开发步骤

掌握C#基本语法

掌握流程控制

C#简介sharp:锋利

产生与发展

发展历史

- Anders Hejlsberg--C#之父：丹麦人 sharp
- 1981年将Pascal编译器卖给了Borland，并加入Borland公司
- 1995年2月14推出Delphi1.0
- 1996年，Anders 离开Borland加盟Microsoft，主持开发了J++和WFC，
- 2000年6月，创造了C#语言

特征

- C#语言是一种简单、现代、优雅、面向对象、类型安全、平台独立的新型组建编程语言
- Microsoft公司针对Microsoft .NET框架开发的一种语言
- Microsoft对C#的描述为
- 简单、现代化、面向对象
- 语法简单易学
- 功能强大

Visual Studio 2012简介

- 作为Microsoft推出的新一代集成开发工具
- 提供统一的集成开发环境及工具，大大提高了开发的效率
- 集成多种语言的支持
- 提供了用于创建不同类型应用程序的多种模板
- 提供很多应用程序的快速开发工具

创建C#控制台应用程序

项目下的Program.cs是程序的入口，即若要运行项目，必须有该文件（类）

Main方法

- Main首字母大写
- 可以不使用public修饰
- 返回值可以为 void 或者 int
- 命令行参数是可选的

语法：

static void Main (string[ ] args) { }

static int Main (string[ ] args) { }

static void Main ( ) { }

static int Main ( ) { }

C#基础知识

常量与变量

分类	C#类型	.NET Framework类型	范围
整数类型	byte	System.Byte	$0\sim2^8-1$
	int	System.Int32	$-2^{31}\sim2^{31}-1$
	short	System.Int16	$-2^{15}\sim2^{15}-1$
	long	System.Int64	$-2^{63}\sim2^{63}-1$
浮点类型	float	System.Single	$\pm1.5\times10^{-45}$到$\pm3.4\times10^{38}$
	double	System.Double	$\pm5.0\times10^{-324}$到$\pm1.7\times 10^{308}$
字符类型	char	System.Char	一个字符
字符串类型	string	System.String	任意长度的任意字符
布尔类型	bool	System.Boolean	true和false

变量：在程序的运行过程中值可以被改变的量，一般用于保存程序中的临时数据

语法：

***访问修饰 数据类型 变量名; ***

示例：

```
int studentAge;
studentAge = 18;
string studentName = "Jack";
double markChinese = 89;
float markMaths = 92.5f;
```

变量只有声明且初始化后方可使用

变量的命名规则

- 变量名必须以字母开头
- 变量名只能由字母、数字和下划线组成
- 变量名不能与C#中的关键字名称相同
- 变量名不能与C#中的库函数名称相同
- 变量名称第一个单词全小写，从第二个单词开始，每个单词首字母要大写（小驼峰）
- 变量名区分大小写

常量：在程序的运行过程中值不能改变的量

- 常量在编译时确定它的值，在整个程序中不允许修改
- 常量声明的同时必须赋值
- 常量名一般大写
- 常量值的类型要和常量数据类型一致

语法：

const 数据类型 常量名 = 常量值;

运算符与表达式

类型	运算符
算术运算符	+、-、*、/ 、%、++、--
关系运算符	<、>、<=、>=、!=、==
逻辑运算符	&&、||、!
赋值运算符	=、*=、/=、+=、-=、%=
条件运算符	?:(三元运算符)

运算符的优先级：

当一个表达式包含多个运算符时，表达式的运算顺序决定于运算符的优先级

注释

- 代码注释可以提高编程的效率，使程序看起来更加清晰完整
- 在程序代码中添加注释是一个软件工程师应该具备的良好习惯

单行注释： //

多行注释 /**/

文本注释： ///

Console类 ascii

常用方法

方法名称	描述
read	从标准输入流读取下一个字符
readLine	从标准输入流读取下一行字符
write	将指定值的文本表示形式写入标准输出流。不换行
writeLine	将指定的数据（后跟当前行终止符）写入标准输出流
clear	清除控制台缓冲区和相应的控制台窗口的显示信息

```
int.Parse():将字符串转还为int类型数值

float.Parse():将字符串转为单精度的小数

double.Parse():将字符串转为双精度的double类型的小数

将数字转换为字符串:
int num=100;

String  num2=num.ToString();

Float num=10.1;
```

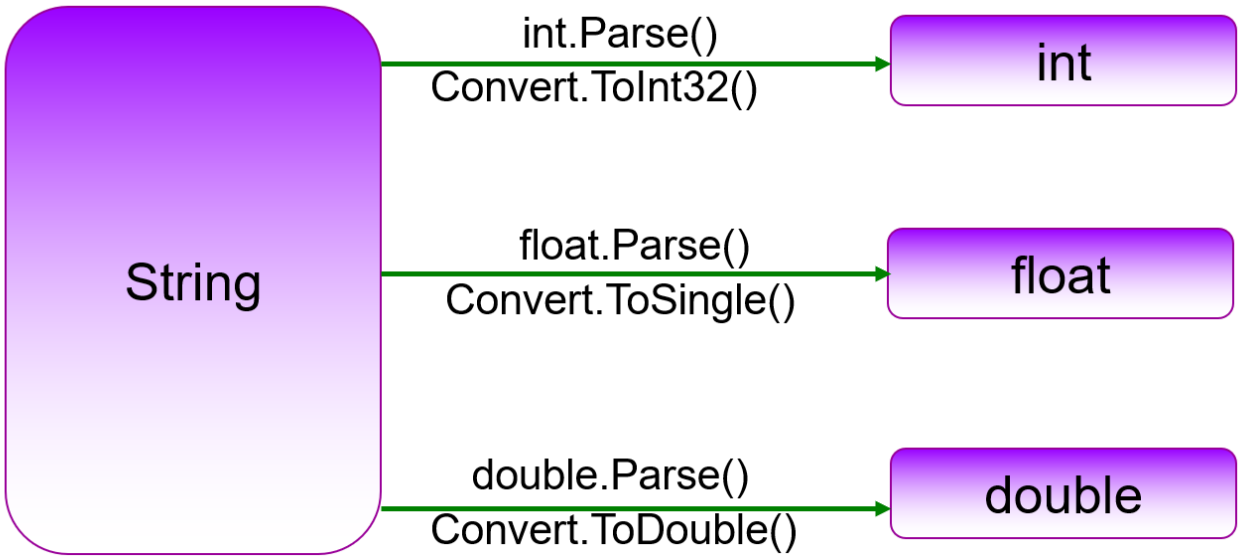
控制台输出

```
Console.WriteLine("Hello");
Console.ReadLine();
我们在使用Console.WriteLine("")输出操作时有两种方法:
 (1) Console.WriteLine("要输出到额字符串");//直接输出
 (2) Console.WriteLine("格式化字符串","值列表");//利用占位符{n}进行输出，n从0开始
示例:
String  name="小明";
Console.WriteLine("您的名字为：{0}",name);
Console.WriteLine("计算相加结果：{0}+{1}={2}", 100,200,100+200);
```

控制台输入

可使用Console.Read()方法或Console.ReadLine()方法接收控制台的输入

在使用Console.ReadLine()方法时，值得注意的是该方法的返回值为String类型，当需要对接受的数据进行计算时必须进行类型转换



程序的结构

顺序结构：默认的程序，由上向下执行的

分支结构：有条件判断，当产生某些条件的时候，就执行对应的程序代码

循环结构：反复不断执行某些代码

分支结构

if

```
格式一:
if(表达式或者bool类型变量){
    代码块儿;
}
格式二:
if(表达式或者bool类型变量){
    代码块一;
 }else{
    代码块二;
 }

格式三:
if(表达式1){
    代码块一;
 }else  if (表达式2) {
    代码块二;
 }else  if (表达式2) {
    代码块三;
 }else  if (表达式2) {
    代码块四;
 }else{
```

```
    代码块五;
}
```

switch

```
switch (int/char/string表达式)  {
    case 常量表达式1:
        语句 2;
        break;//必须有
    case 常量表达式2:
        语句2;
        break://必须有
    default:
        语句n;
        break; //必须有
}
```

循环

for

```
for（表达式一：进行变量的声明及赋值；进行判断条件；表达式三：进行变量值得更改）{

  循环体;

}
```

while

do-while

break/continue

案例：得到1-100累加和超过30的当下那个数字是几?

foreach语句是C#引入的新的循环结构，主要用于遍历数组或集合

语法:

```
foreach(数据类型 循环变量 in 数组或集合){

      语句块;
}
```

案例:

输入字符串，判断奇数、偶数的个数

char.IsLetter(s):判断遍历s的值是否是字母

char.IsDigit(s):判断遍历s的值是否是数字

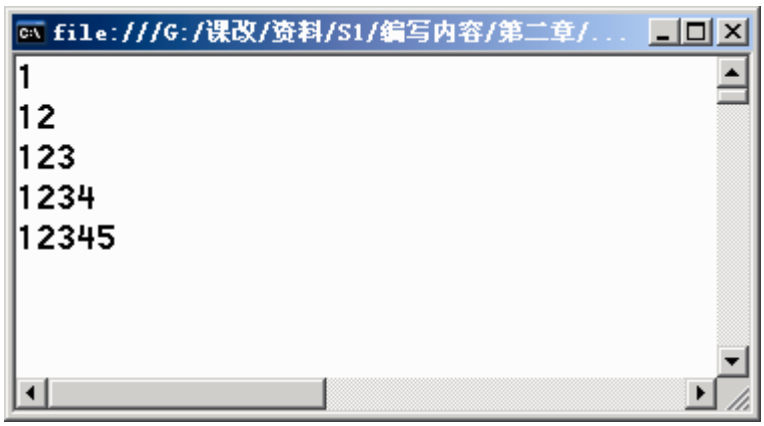
char.IsPunctuation(s):判断遍历s的值是否是标点

```
String str1 = "123aasdasd...11?";
int num1 = 0,num2=0,num3=0;
foreach(char s in str1){
   if (char.IsLetter(s)) {
          //是字母
          num1++;
    }
    if (char.IsDigit(s)) {
        //是数字
         num2++;
    }
    if (char.IsPunctuation(s)) {
         //是标点
         num3++;
     }

 }
 Console.WriteLine("字母个数：{0}，数字个数：{1}，标点个数：{2}",num1,num2,num3);
```

循环嵌套

实现如下效果：



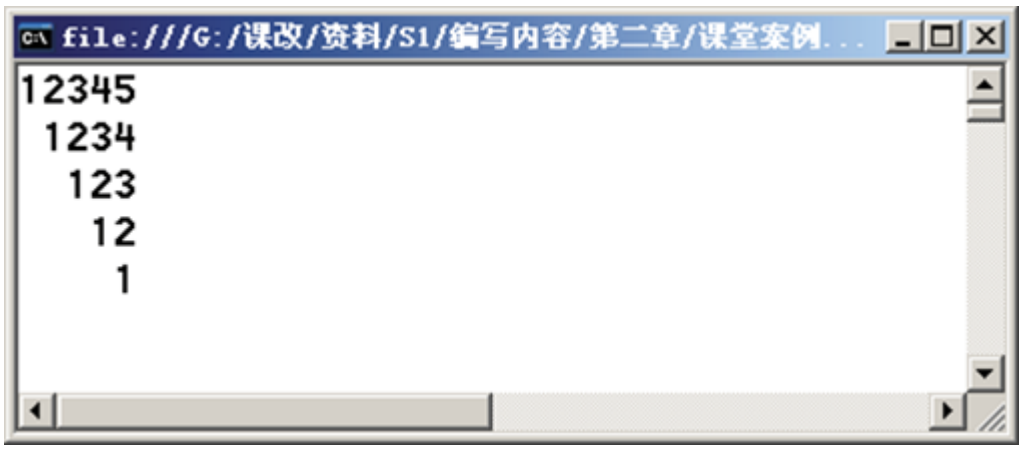
共同的规律，就是：第n行从1输出到n。

```
for (int i=1;i<=n:i++){
    Console.Write(i);
    Console.WriteLine();
}
```

以上效果代码实现：

```
  int n, i;
            for (n = 1; n <= 5; n++)
            {
                for (i = 1; i <= n; i++)
                {
                    Console.Write(i);
                }
                Console.WriteLine();
            }
```

案例：实现如下效果：



第1行 需要打印0个空格，同时数字从1-5；
第2行 需要打印1个空格，同时数字从1-4；
第3行 需要打印2个空格，同时数字从1-3；
第4行 需要打印3个空格，同时数字从1-2；
第5行 需要打印4个空格，同时数字从1-1；
这是就意味着在内层循环中需要两个循环来帮助我们分别控制空格和数字

```
for(int j=1;j<=n-1;j++)
      { 输出空格;  }
  for(int i=1;i<=5-n+1;i++)
     { 输出数字;    }
```

```
static void Main(string[] args)
{
    int n, j , i ;
    for (n = 1; n <= 5; n++)
    {
        for (j = 1; j <= n - 1; j++)
        {
            Console.Write(" ");
        }
        for (i = 1; i <= 5 - n+1 ;i++ )
         {
             Console.Write(i);
        }
        Console.WriteLine("");
```

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

解决嵌套循环问题

- 外循环的循环变量一般决定了内循环的循环次数，
- 内循环是具体执行任务的操作
- 先找出内循环的实现方式
- 接着找到外循环的循环变量和内循环循环变量之间的联系规律，找到它们的公式
- 写出正确的内层循环体

金字塔：

```
            /*
             *     1:1个*
             **    2:2
             ***   3:3
             ****  4:4
             ***** 5:5
             内容与行号有关，则将内层的遍历的内容个数写成行号j
             */
            //外层控制行数
            for (int j = 1; j <=5;j++ )
            {
                //内层：每行的内容
                for (int i = 1; i <= j; i++)
                {
                    Console.Write("*");
                }
                //一行结束要换行
                Console.WriteLine("");
            }
```

靠右直角边三角形

```
            /*
             *4个空格  1个
             * 1:4   1
             ** 2:3   2
            *** 3:2   3
           **** 4:1   4
          ***** 5:0   5
             */

            for (int i = 1; i <= 5;i++ )
            {
                for (int j = 1; j <= 5 - i;j++ )
                {
                    Console.Write(" ");
```

```
                }
                for (int j = 1; j <= i;j++ )
                {
                    Console.Write("*");
                }
                Console.WriteLine("");
            }
```

九九乘法表

```
for (int j = 1; j <=9;j++ ) {
                //内层：每行的内容
                for (int i = 1; i <= j; i++)
                {
                    Console.Write("{0}*{1}={2}\t",i,j,i*j);
                }
                //一行结束要换行
                Console.WriteLine("");
            }
```

```
1*1=1
1*2=2   2*2=4
1*3=3   2*3=6   3*3=9
1*4=4   2*4=8   3*4=12  4*4=16
1*5=5   2*5=10  3*5=15  4*5=20  5*5=25
1*6=6   2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7   2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8   2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9   2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

规则:

每行的第一个数字就是

1 2 3 。 。 。 。直至行数的那个数字终止

第二个数字:

就是行数

数组

一堆相同类型的数据存放在一个变量里，这个变量就是数组

初始化:

数据类型[ ] 数组名;

```
// 方式一：先声明数组，然后声明数组的空间发小
int[] array;
array = new int[5];  //创建了一个长度为5的数组。
//方式二：直接在声明数组的同时，就为其开辟存储空间
int[] array = new int[5]; //创建了长度为5的整形数组，没有初始化赋值
//方式三：在声明数组的同时，为其设置数据
int[] array = new int[5]{1,2,3,4,5}; //创建了长度为5的整形数组并为其初始化赋值
int[] array = new int[]{1,2,3,4,5};  //声明了一个整形数组省略了长度
int[] array = {1,2,3,4,5};  //省略了new
```

案例：

定义一个长度为5的string数组，用于存放用户输入的学生的姓名，然后将学生的姓名输出到控制台上。

```
static void Main(string[] args)
        {
            Console.WriteLine("~~请依次输入5位学生的姓名~~");
            //定义了一个长度为5的string类型的数组用于存放学生的姓名
            string[] names = new string[5];
            //用for循环来依次接受每个学生的姓名
            for (int i = 0; i < names.Length; i++)
            {
                Console.WriteLine("第{0}个学生",i+1);
                names[i] = Console.ReadLine();
            }
            Console.WriteLine("您输入的学生信息如下：");
            //使用foreach循环将输入的信息依次显示出来
            foreach (string name in names)
            {
                Console.WriteLine(name);
            }
            Console.ReadLine();
        }
```

冒泡排序

就是对数组的数据进行大小排序

排序规则：

两两比较待排序数组中相邻的值，发现两个值的次序相反时即进行交换，直到没有反序的值为止。

每次比较相邻两数,小的交换到前面

每轮结束后最大的数交换到最后

```
int [] nums={12,2,5,78,23,1,90};
```

第一轮第一次：
【2,12】,5,78, 23,1,90

第一轮第二次:
2,【5,12】,78,23,1,90
第一轮第三次:
2,5,【12,78】,23,1,90
第一轮第四次
2,5,12,【23, 78】,1,90
第一轮第五次
2,5,12,23, 【1, 78】,90
第一轮第六次
2,5,12,23, 1, 【78,90】
第二轮第一次:
【2,5】,12,23, 1, 78,90
第二轮第二次---第三次:
2,5,【12,23】, 1, 78,90
第二轮第四次:
2,5,12, 【1, 23】, 78,90
第二轮第五次
2,5,12, 1, 【23, 78】,90

第三轮　只需要比较四次
第一次:
【2　5】　12　1　23　78　90
第二次:
2　【5　12】 1　23　78　90
　第三次:
 2 5 【12　1】23　78 90
  第三次结束后变为:
  2　5　1　12　23　78　90
　第四次:
　2　5　1 【 12　23】　78　90
　第四次结束后:
　　2　5　1　12　23　78　90


```
int []  num1={2,1,89,45,23}
```
第一轮:
第一次

【2,1】,89,45,23

结束后:
1,2,89,45,23
第二次:
1,【2,89】,45,23
　结束后:
1,2,89,45,23
 第三次:
  1,2,【89,45】,23
　结束:
　1,2,45,89,23
　第四次:
　　1,2,45,【89,23】
　　结束:

```
        1,2,45,23,89
    结论：每轮比较结束后，最后的数就是剩余没有比较数字里边的最大值
    第二轮：】
    第一次：
      【1,2】,45,23,89
结束：
  1,2,45,23,89
    第二次：
      1,【2,45】,23,89
    结束：
      1,2,45,23,89
    第三次：
      1,2,【45,23】,89
    结束：
      1,2,23,45,89
    第三轮：
    只需要比较两次
    第四轮：
    只需要比较依次即可
```

```
scores[j]=23
scores[j+1]=12

第一步：
temp=scores[j];//将前边较大的数字，暂时存放到临时变量里，保证数据不会丢失
第二步：在将后边较小的数字，交换到前边的位置上
scores[j]=scores[j+1];   //scores[j]:12
第三步：
将原来较大的那个数字从临时变量temp里取出来，存放到后边的j+1的位置上
scores[j+1]=temp;
```

代码实现：

```
            int [] scores={89,78,88,90,92,93,91};


        //内层循环控制本轮比较的次数，外层循环控制的是比较的轮数
            for (int i = 1; i < scores.Length; i++)
            {
                for (int j = 0; j < scores.Length - i; j++)
                    //  j:0   j<6   0-5
                {
                    if (scores[j] > scores[j + 1])
                    {

                        //若前边的数字>后边的数字，就需要交换前后两个数字的位置

                        // 交换元素
                        int temp = scores[j];
                        scores[j] = scores[j + 1];
                        scores[j + 1] = temp;
```

```
            }
        }
    }
    foreach(int n  in   scores){
        Console.WriteLine(n);
    }
```

作业：

1、冒泡排序代码实现5遍(课前测)

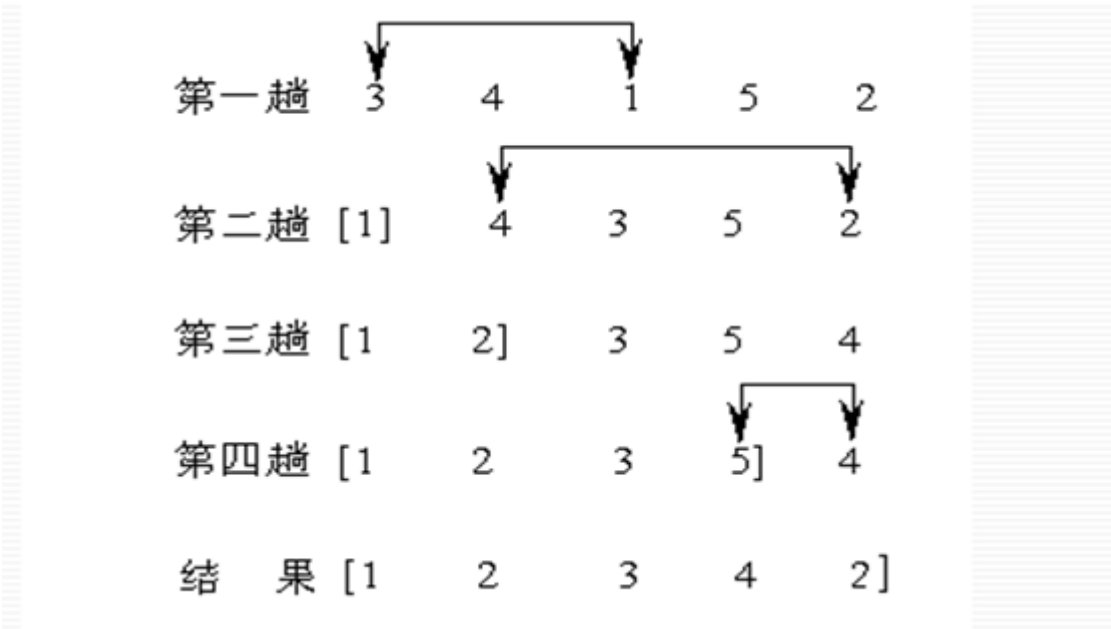
2、手动输入数字的个数，然后声明数组，将输入的数组存储到数组里，并且得到所有输入数字的最大值、最小值、平均值和总和

3、课本50页第2题

4、循环录入5个学生成绩，并且利用冒泡排序进行排序输出

选择排序

在未排序的数据中，选出最小值，放到前边




```
    //选择排序
    int[] nums = { 23,12,2,34,56,76,1,90,79};
    for (int i = 0; i < nums.Length - 1;i++ )
    {
        int minpos = i;//存储最小数据的位置索引
        for (int j = i + 1; j < nums.Length; j++)
        {
            if (nums[minpos] > nums[j]){
                minpos = j;
            }
        }
        //最小数的位置如果不是第一个位置的那个数字，
```

```
            //则将最小数与第一个位置的那个数字交换
            if (minpos != i) {
                int temp = nums[minpos];
                nums[minpos]=nums[i];
                nums[i] = temp;
            }
        }
        foreach(int i  in nums){
            Console.WriteLine(i);
        }
```