C# 编程基础

任课教师：李娜
邮箱：709233393@qq.com

授课安排

- 课时：48学时；

  每周3学时，共计16周

- 教材：零基础学C#　（明日科技编著）

- 考试成绩组成：平时成绩（60%）+期末考试（40%）

平时成绩：考勤+平时作业+提问

- 参考资料：

（1）https://www.icourse163.org/course/XJTU-1002843011?tid=1467021448;
(2)https://www.bilibili.com/video/BV1FJ411W7e5?spm_id_from=333.337.search-card.all.click&vd_source=61e0d65010300db5ea58529d1e0bd5f7

（2）计算机概论：
https://www.bilibili.com/video/BV1ks411G7LE?spm_id_from=333.337.search-card.all.click&vd_source=61e0d65010300db5ea58529d1e0bd5f7

学习建议

- 课上：听讲+**上机练习**
- 课下：看视频/书+**上机练习**

实践、实践、实践！！！

授课内容

一、C#基础语法

1、Visual Studio 2019

2、基本数据类型

3、程序的控制结构

4、批量数据处理

二、面向对象程序设计

1、类和对象

2、继承

3、多态

C#编程基础

任课教师：李娜
邮箱：709233393@qq.com

第一章 程序设计基本方法

1.1 计算机的概念

1.2 程序设计语言

1.3 C#语言概述

1.4 C#语言开发环境配置

1.5 程序的基本编写方法

1.1 计算机的概念

计算机是**根据指令操作数据**的设备。

- 功能性

  数据操作：对数据计算、输入输出处理和结果存储等。

- 可编程性

  根据一系列指令自动地、可预测地、准确完成操作者的意图。

问题1、计算机为什么能计算?

（1）“数”在计算机中是如何表示的？
（2）逻辑上“数”是如何计算？
（3）物理上“数”的计算是如何实现的？

（1）数的表示：十进制、二进制、十六进制

- 十进制：

  计数符号：0、1、2、3、4、5、6、7、8、9

  基数：10

$$256 = 2 \times 10^{2} + 5 \times 10^{1} + 6 \times 10^{0}$$

- 二进制：

  计数符号：0、1

  基数：2

$$10110 = 1 \times 2^{4} + 0 \times 2^{3} + 1 \times 2^{2} + 1 \times 2^{1} + 0 \times 2^{0}$$

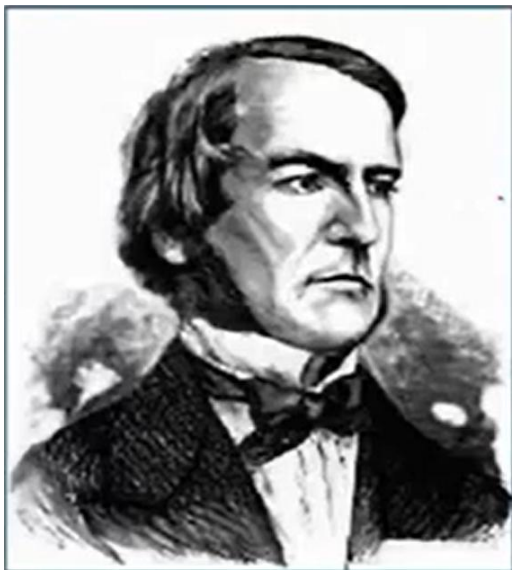
- 十六进制：

  计数符号：0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F

  基数：16

$$(ABCD)_{16} = A \times 16^{3} + B \times 16^{2} + C \times 16^{1} + D \times 16^{0}$$

（2）如何进行计算？

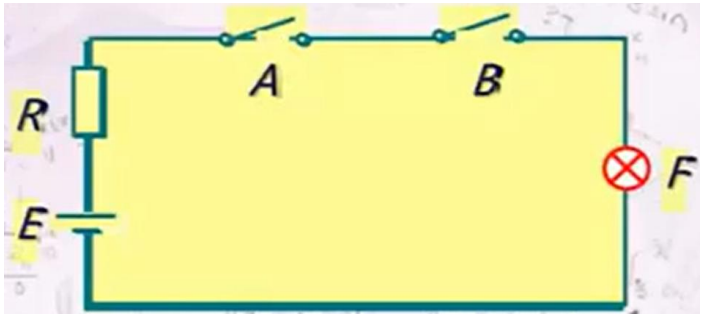


英国数学家布尔（G.Boole）

- 布尔代数
- 1854年：布尔发表《思维规律的研究---逻辑与概率的数学理论基础》，并综合其另外一篇文章《逻辑的数学分析》，创立了一门全新的学科—布尔代数。
- 为计算机的开关电路设计提供了重要的数学方法和理论基础。

- 基本逻辑运算：与、或、非
- 复合逻辑运算：同或、异或、与非、或非、与或非

计算机中数的逻辑运算方法

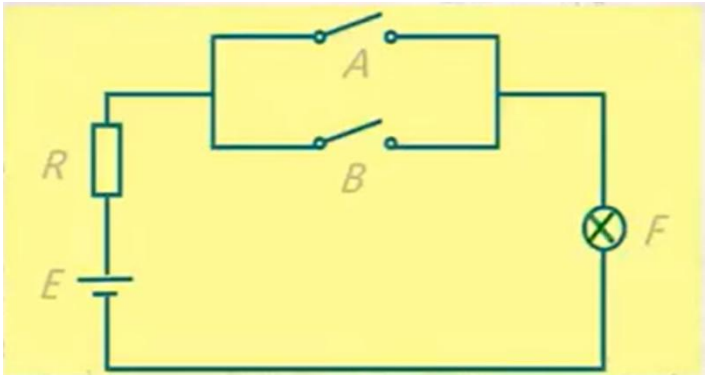


逻辑函数表达式：F=A·B

真值表：

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

计算机中数的逻辑运算方法

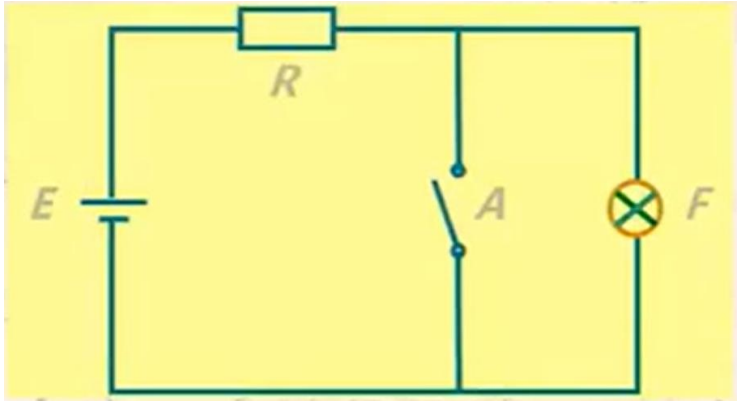


逻辑函数表达式：F=A+B

真值表：

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

计算机中数的逻辑运算方法



真值表：

A	F
0	1
1	0

逻辑函数表达式：$F=\overline{A}$

- 异或：$F = A \oplus B$
  - 两数相同为“0”
  - 两数相异为“1”
- 同或　$F = A \odot B$
  - 两数相同为“1”
  - 两数 相异为“0”

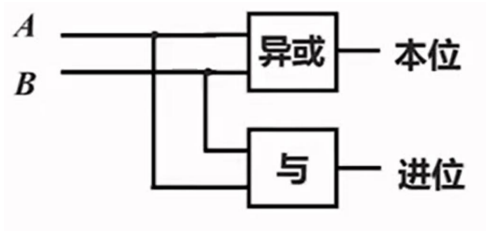
真值表：

A	B	F1	F2
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

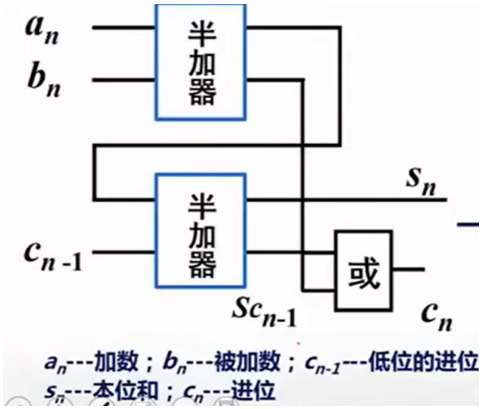
加法的计算

- A=1101，B=1001，求A+B

```
  1 1 0 1
+ 1 0 0 1
  1 0 0 1
-------------
1 0 1 1 0
```




```
    1          1
+   1      +   0
-------    -------
  1 0        0 1

    1          1
+   1      +   0
-------    -------
  1 0        0 1
```

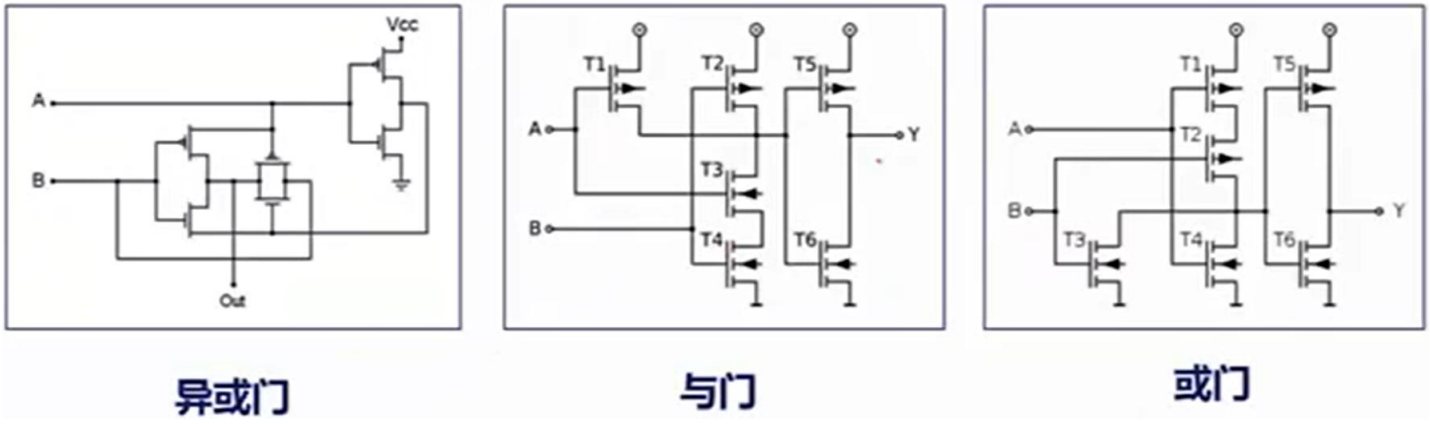


按照布尔逻辑：

- 位数上的运算：“异或”运算的结果。
- 进位运算：“与”运算的结果。

(3) 布尔运算怎么实现?

- 加法的实现

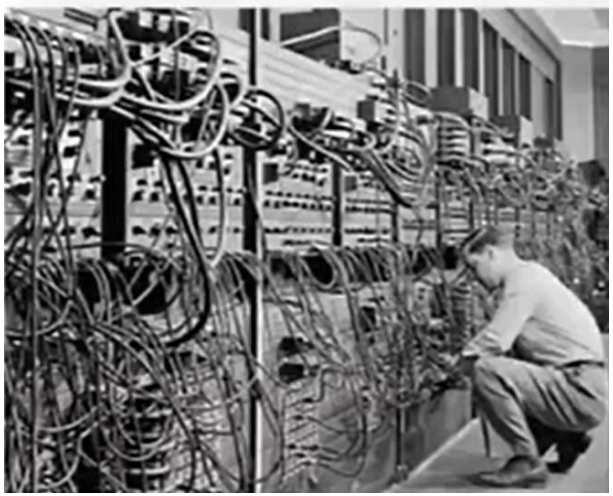


异或门　　与门　　或门

问题：是不是可以这样：

- 需要完成什么计算，就动手设计个相应的电路！
- 设计好很多个原子电路，需要的时候就把它们临时组装在一起？

- **思考：**是不是可以这样：
  - 需要完成什么计算，就动手设计个相应的电路！
  - 设计好很多个原子电路，需要的时候我就把它们临时组装在一起！
- ENIAC



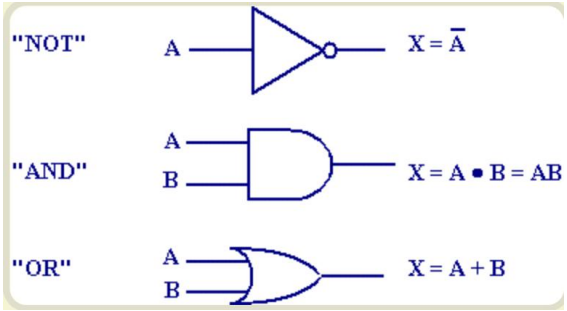
- **不能**：通过重新“组合不同电路”的方式，去完成新的计算任务！

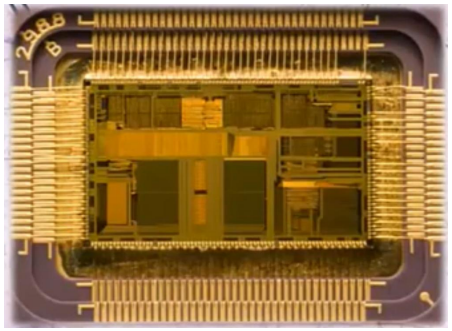
- **应该：**
- 通过某种命令来控制计算机，让计算机按照这种命令来运行，这种命令可以用电信号表示；
- 这种命令不是“临时输入”到计算机，而是存放在某个地方，随时可以更改；
- 命令改了，计算机的功能也就改了！
- 这就是EDVAC！现在的计算机都是EDVAC!

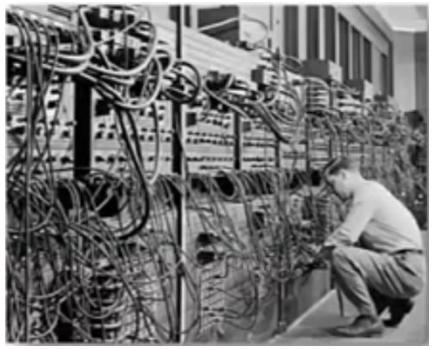
小结

- 用二进制表示数据
- 用布尔代数进行计算
- 用电路实现布尔计算

  --→所以 电路能够进行计算
- 计算机就是由这样的电路构成的。





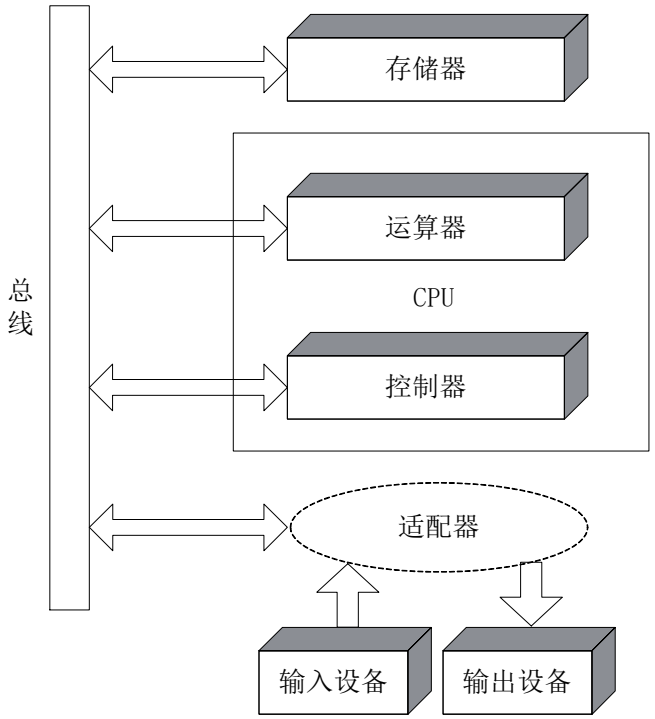


ENIAC

问题2：什么是存储程序式计算机?

计算机系统工作时：

- 输入设备将程序与数据存入存储器；
- 控制器从存储器中逐条取出指令，将其解释成控制命令，去控制各部件的动作；
- 数据在运算器中加工处理，处理后的结果通过输出设备输出。



计算机的发展

计算机的发展参照摩尔定律，表现为指数方式

- 硬件所依赖的集成电路规模参照摩尔定律发展
- 运行速度因此也接近几何级数快速增长
- 高效支撑的各类运算功能不断丰富发展

摩尔定律 Moore's Law

摩尔定律：当今世界，唯一长达**50**年有效且按照指数发展的技术领域

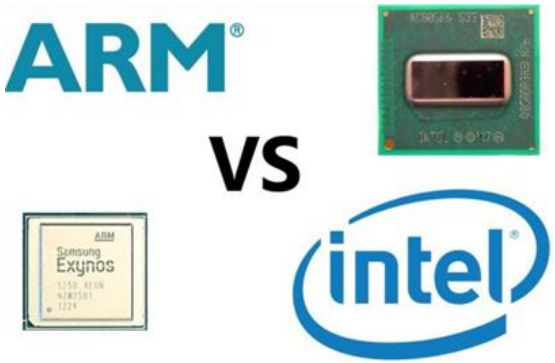
- Intel公司创始人之一 戈登▪摩尔在1965年提出
- 单位面积集成电路上可容纳晶体管的数量约每两年翻一番
- CPU/GPU、内存、硬盘、电子产品价格等都遵循摩尔定律

1971---2015年，44年期间：
理论上：约2^{22}，四百万倍
实际上：四百三十五万倍

问题3：是否不管什么“程序”CPU都能识别并执行?

- No
- 指令集：CPU出厂时候已经指定

指令的种类	助记符	功能
运算指令	ADD A,num	把数值num加到寄存器A的值上
	ADD A,reg	把寄存器reg的值加到寄存器A的值上
	SUB num	从寄存器A的值中减去数值num
	SUB reg	从寄存器A的值中减去寄存器reg的值
	INC reg	将寄存器reg的值加1
	DEC reg	将寄存器reg的值减1
	AND num	计算寄存器A的值和数值num的逻辑积
	AND reg	计算寄存器A的值和寄存器reg的值得逻辑积
	OR num	计算寄存器A的值和数值num的逻辑和
	OR reg	计算寄存器A的值和寄存器reg的值的逻辑和
	XOR num	计算寄存器A的值和数值num的逻辑异或
	XOR reg	计算寄存器A的值和寄存器reg的值的逻辑异或
	SLA reg	对寄存器reg的值进行算数左移运算
	SRA reg	对寄存器reg的值进行算术右移运算
	SRL reg	对寄存器reg的值进行逻辑右移运算
	CP num	比较寄存器A的值和数值num的大小
	CP reg	比较寄存器A的值和寄存器reg的值的大小
内存与CPU之间的输入输出指令	LD reg,num	把数值num写入到寄存器reg中
	LD reg1,reg2	把寄存器reg2的值写入到寄存器reg1中
	LD (num),reg	把寄存器reg的值写入到地址num上
	LD (reg1),reg2	把寄存器reg2的值写入到存放在寄存器reg1中的地址上
	PUSH reg	把寄存器reg的值写入到栈中
	POP reg	把由栈顶读出的数据存放到寄存器reg中
I/O与CPU之间的输入输出指令	IN A,(num)	从地址num中读出数据,存放到寄存器A中
	IN reg,(C)	从存储在寄存器C中地址上读出数据,存放到寄存器reg中
	OUT (num),A	把寄存器A的值写入到地址num中
	OUT (C),reg	把寄存器reg的值写入到存储在寄存器C中的地址上
程序流程控制指令	JP num	使程序的流程跳转到地址num上,接下来从那个地址上的指令开始执行
	CALL num	调用存放在地址num上的子例程
	RET	从子例程中返回
	HALT	中止CPU的运行

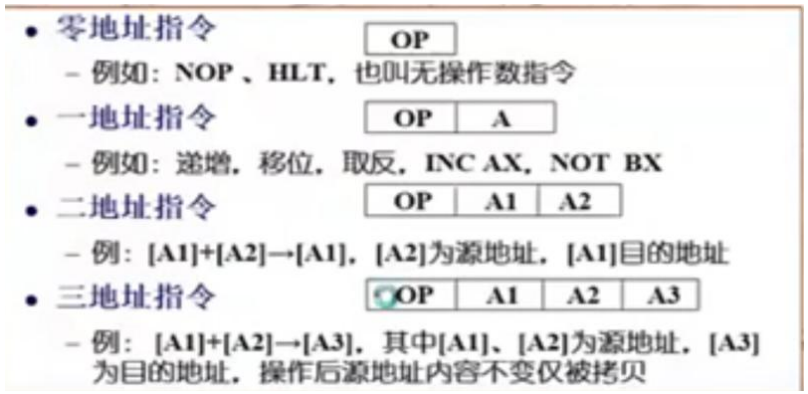


指令

- 最终表现为二进制码
- 其长度随着CPU类型不同而不同
- 包含：**操作码**和**操作数**

  操作码：要执行的动作

  操作数：要操作的数或者地址



CPU指令集—小结

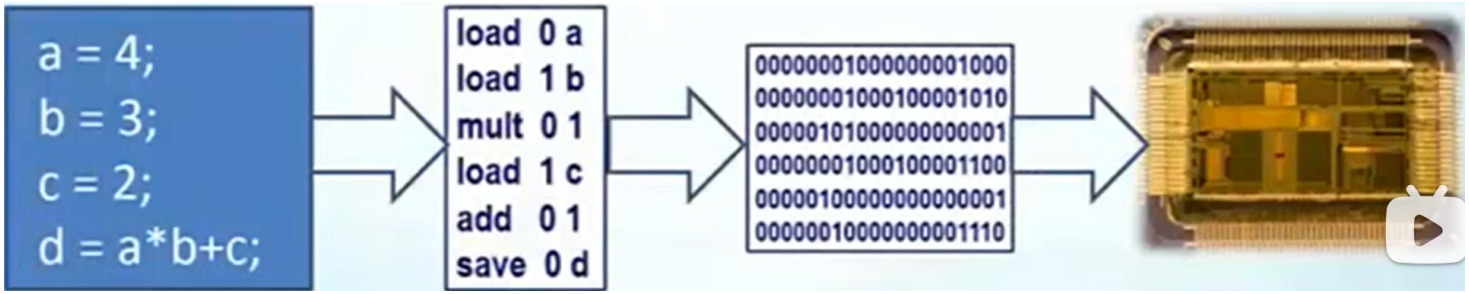
- CPU不是随便什么程序都能执行；
- 它执行规定的指令集中的指令；
- 指令是二进制编码；
- 要把你的命令转换成满足指令集要求的二进制代码，才能在计算机上运行；
- 然后才能在CPU上执行

程序的执行

- 写给计算机的“命令”是长成这样的吗？

```
1001010101010011 00011110
1100011101010001 01111011 10100001
1001010001110111 00111001 10000001
1100010101000101 01100111
1001010101010011 00101010 10000111
```

- 不会的，计算机程序没有那么恐怖！



小结：

- 计算机是能用电路进行计算的！

  数-->二进制数-->布尔运算-->电路实现

- CPU是制造好的电路，它能完成指令集里的运算！

  指令是二进制的码，CPU能看懂，并执行它

- 想让CPU按照我们的想法做事，就应该：

  （1）把我们想做的事情写出来（用某种语言）

  （2） 把这个写出来的东西，翻译成CPU能看懂的指令集

问题4：求出最大数

- 45,67,32,88,32,89,25,77,92,34,26,91,30,69,74 求出最大的数字 。

你是怎么做的?

（1）在你的大脑开辟一片存储空间存放输入的数字；

（2）开辟另一个存储空间存放“特别数字”

（3）从存储空间中第一个数字开始，直到最后一个数字，重复以下操作：

a) 比较“存储空间的数字”与“特别数字”：

b) 如果“存储空间的数字”大于“特别数字”：

c) 那么，将“特别数字”换成“存储空间的数字”

（4）说出“特别数字”

计算机是这样做的（C语言为例）

```
#include<iostream>
using namespace std;
int main()
{
int number[14]=[45,67,32,88,32,89,25,77,92,34,26,91,30,69,74 ];
int max=0;
int i=0;
for(i=0;i<15;i++)
{
if(number[i]>max)
max=number[i];
}
cout<<“the max number is:”<<max;
return 0;
}
```

问题5：关于编程语言：

A、是不是 无论我们在程序里写什么“单词”，计算机都能明白？

B、是不是 无论我们在程序里写什么“数”和“计算符号”，计算机都能明白？

C、世界上可能要用“程序来表达的逻辑”纷繁复杂，程序设计语言得多少种“句式”才能够用？

问题A：是不是 无论我们在程序里写什么“单词”，计算机都能明白？

- NO
- 每一种编程语言定义了一些有特定含义的“关键字”，计算机“只能明白”这些“词”的含义。

B、是不是 无论我们在程序里写什么“数”和“计算符号”，计算机都能明白？

- No

- 计算机只能“看懂”某些类似的数据，这些“数据的类型”和相应的“操作符号”也是定义好的。

C、世界上可能要用“程序来表达的逻辑”纷繁复杂，程序设计语言得多少种“句式”才能够用？

- 3种：顺序、分支、循环

1.2 程序设计语言

程序设计语言是一种用于交互（交流）的语言

- 程序设计语言，也称为编程语言，程序设计的具体实现方式
- 编程语言主要用于**人类和计算机之间**的交互
- 编程语言超过600种，绝大部分不再被使用
- C语言诞生于1972年，是第一个被广泛使用的编程语言
- C#微软公司2000年发布，微软公司配合.NET战略推出

问题6：程序有哪些部分组成?

输入（Input）：程序的输入

文件输入、网络输入、控制台输入、交互界面输入、内部参数输入等

处理（Process）：程序的主要逻辑

处理方法称为算法、程序中最重要的部分。

输出（Output）：程序的输出

控制台输出、图形输出、文件输出、网络输出、内部变量输出

上述三个部分（IPO），计算机能做哪个部分？

问题7：计算机能做哪部分工作？

- 计算机只能解决计算问题，即问题的计算部分。

  解决问题的方法：在每位同学的大脑中！！！

- 一个问题可能有多种角度理解，产生不同的计算部分。

编程解决问题的步骤：

- 确定IPO：明确计算部分及功能边界
- 编写程序：将计算求解的设计编程现实
- 调试程序：确保程序按照正确逻辑能够正确运行

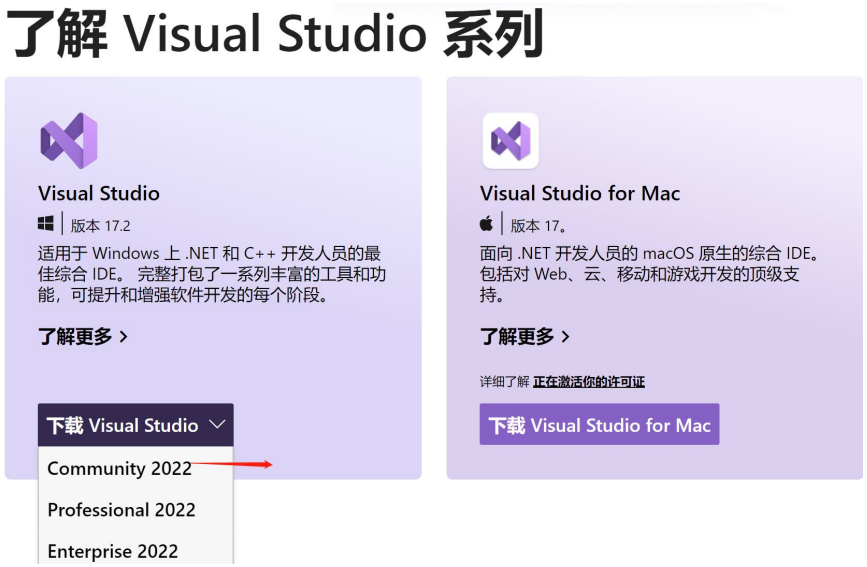
1.3 C#语言概述

- 特点：
  （1）语法简洁，不允许直接操作内存，去掉了指针操作；
  （2）彻底的面向对象程序设计：封装、继承和多态；
  （3）与web紧密结合
  （4）安全、兼容性较好
  （5）完善的错误、异常处理机制
- 应用领域：
  （1）游戏软件开发
  （2）桌面应用系统
  （3）web应用开发
  （4）多媒体系统

安德斯.海尔斯伯格（Anders Hejlsberg）

1.4 基本开发环境

- Visual Studio：
  （1）下载链接：https://visualstudio.microsoft.com/zh-hans/
  （2）下载社区版



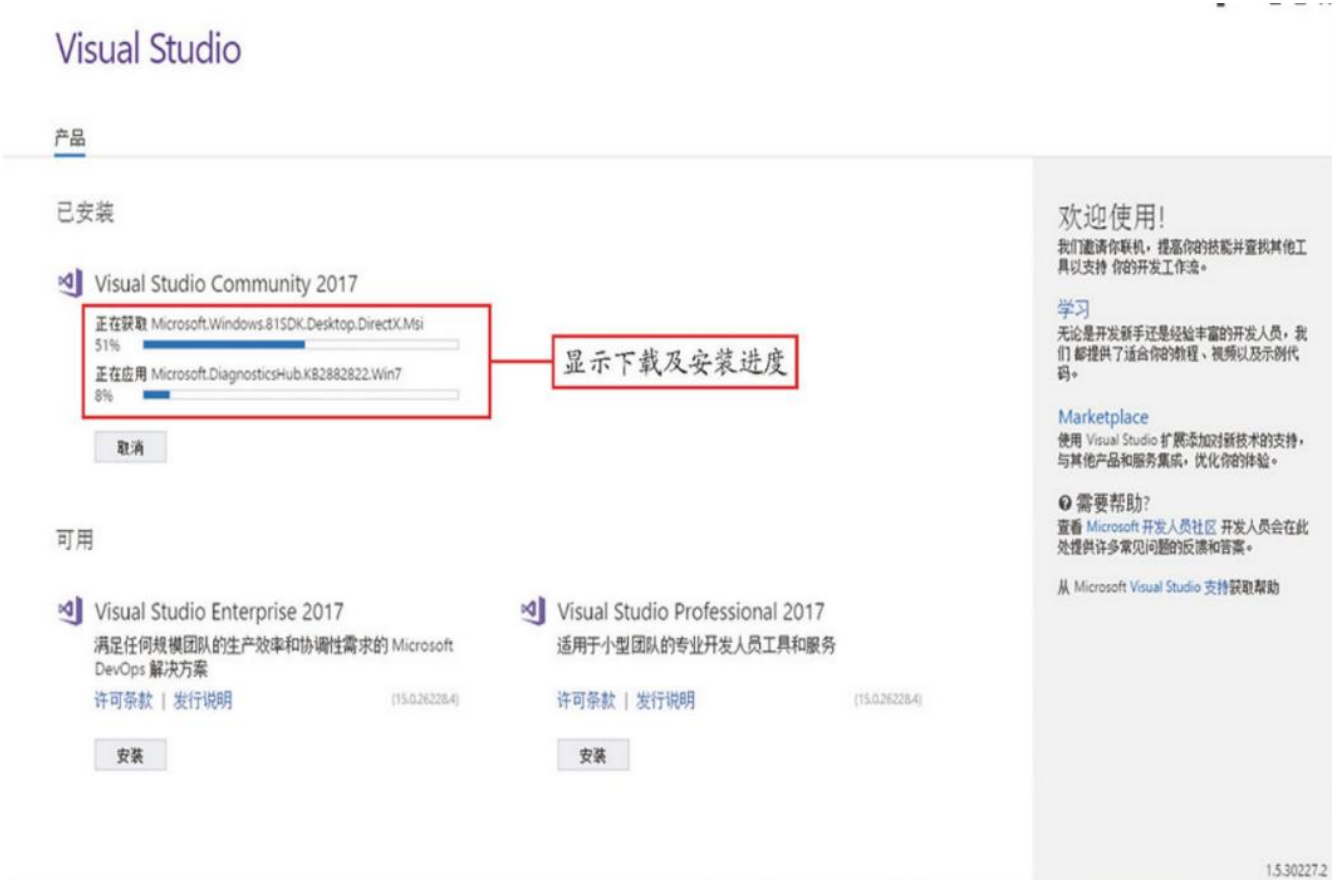
下载和安装

- 安装vs2022分为三个步骤：

（1）下载：vs_community_编译版本号.exe

（2） 双击安装

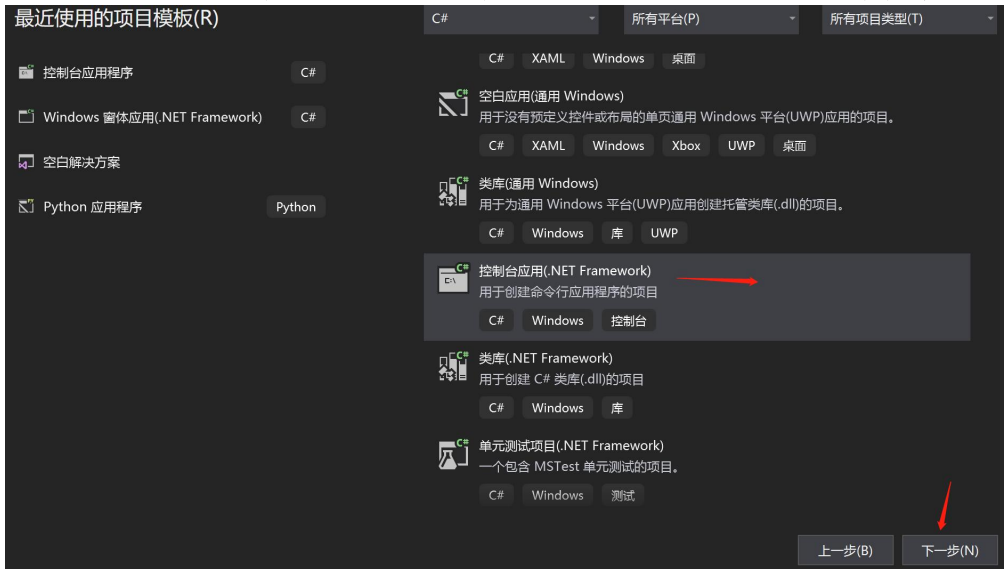
显示安装进度



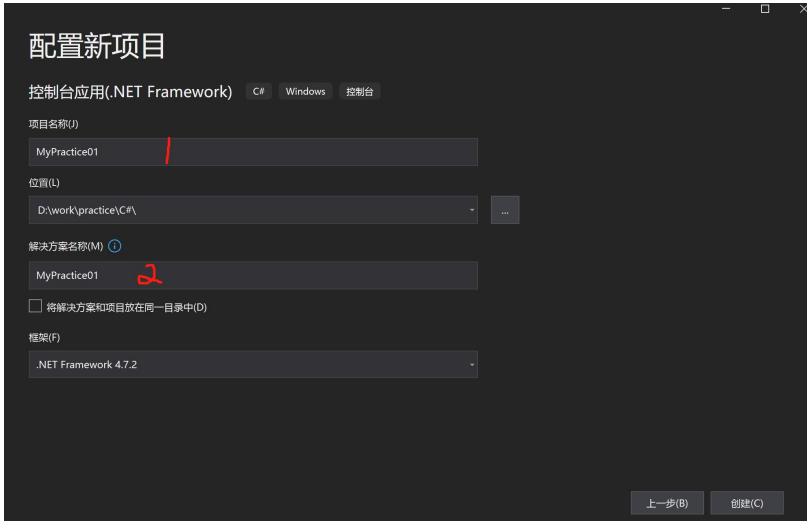
（3）注册登录，启动开发环境

1.5 创建项目

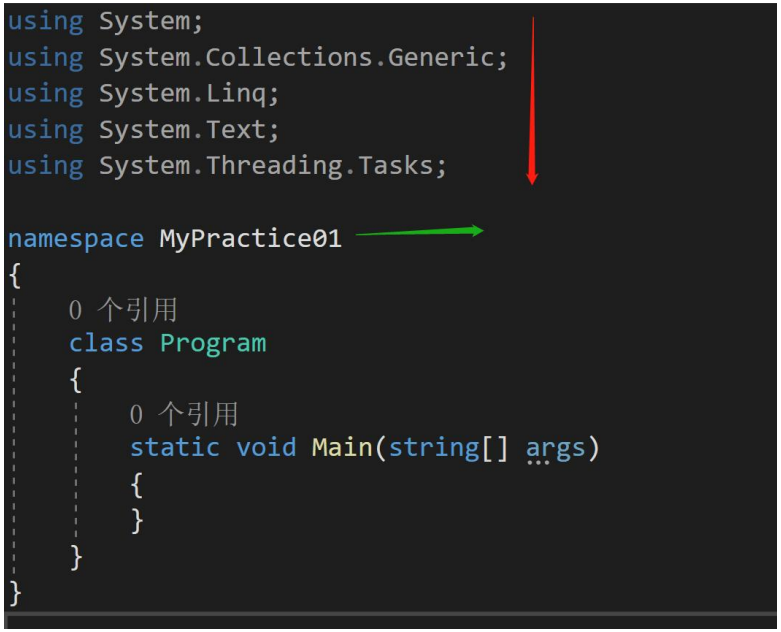
- 方法一：起始页中选择：创建新项目



（2）下图中，解决方案名称与项目名称要统一



（3）自动生成代码



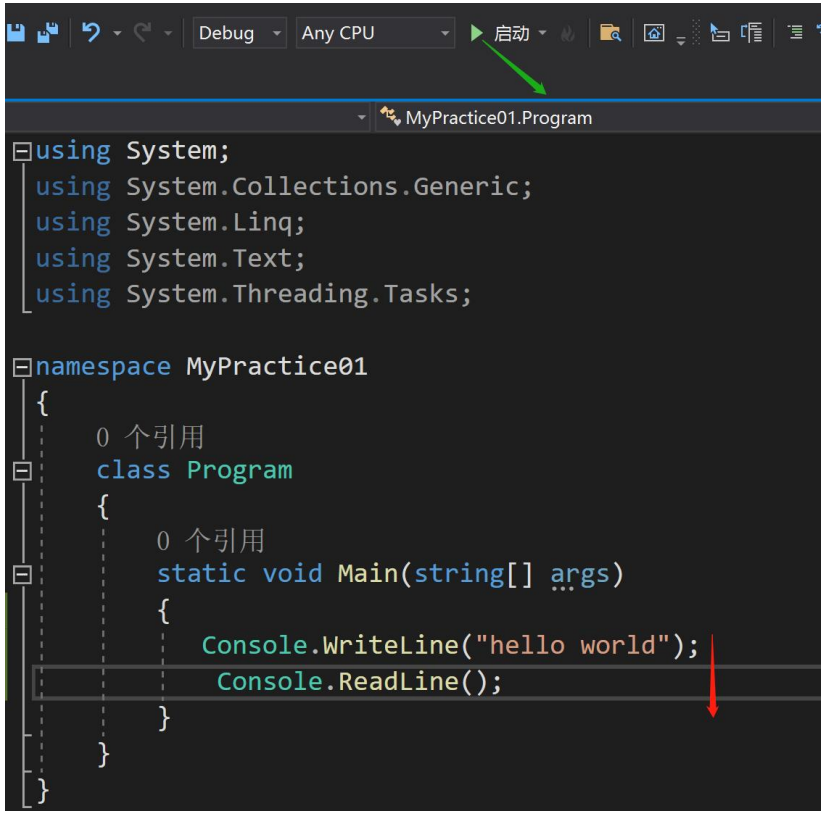
- 红色箭头5行：对命名空间的引用。
- 定义新的命名空间，将本项目的所有元素都组织到该名字空间MyPractice01中，该名字空间是在创建项目时使用的项目名称

- 菜单栏中选择文件→新建→项目

使用Visual Stdio

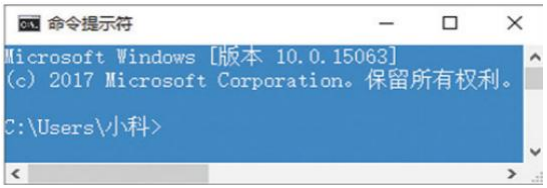
（1）在自动生成的脚本里，添加如下红色箭头所示代码

（2）点击绿色箭头所示：运行



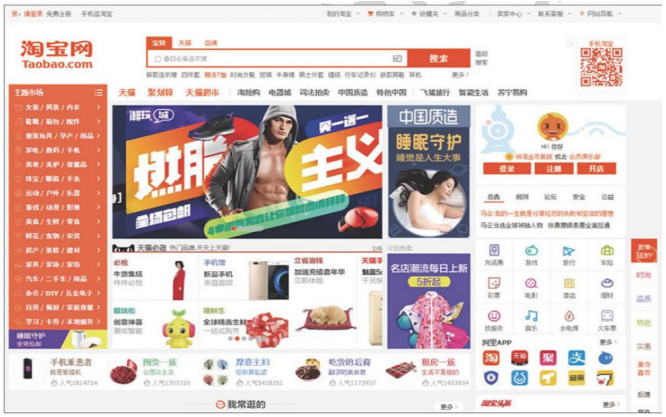
C#中常见项目类型

- 控制台应用程序：没有独立窗体，命令行运行



- Windows窗体应用程序：客户端应用程序

- ASP.NET网站应用程序：web访问



C#编程基础

任课教师：李娜
邮箱：709233393@qq.com

第一章 C#程序结构

1.1 C#程序结构预览

1.2 程序编写规范

1.1 C#程序结构预览

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace MyPractice01
8   {
        0 个引用
9       class Program
10      {
            0 个引用
11          static void Main(string[] args)
12          {
13              Console.WriteLine("hello world");
14              Console.ReadLine();
15          }
16      }
17  }
18
```

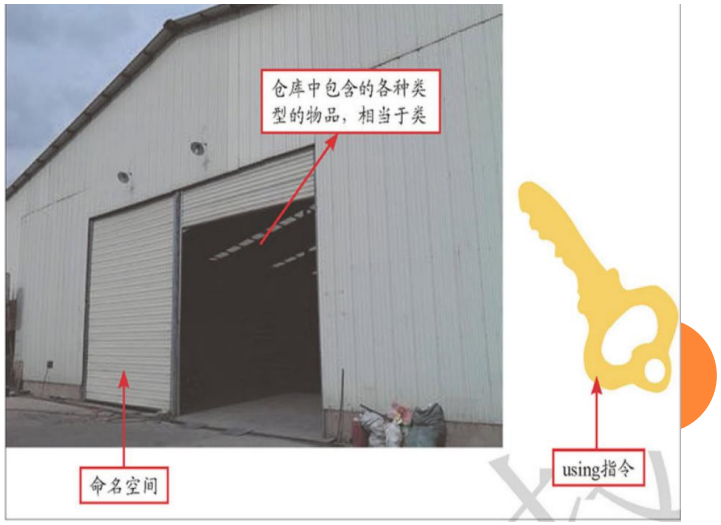
(1) 第11行代码是自动生成的Main方法，用来作为程序的入口方法，每个C#程序都必修有一个Main方法；

(2)第13行中Console.WriteLine方法主要用来向控制台中输出内容；

(3)第14行代码中Console.ReadLine方法主要用来获取控制台中的输出，用来将控制台窗体固定在桌面上。

- C#程序总体分为**命名空间、类、关键字、标识符、Main方法、C#语句**和**注释**等。

- Namespace MyFirstPractice01
  (1)命名空间：组织程序作用,
  (2)namespace 命名空间名字



示例1：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace MyPractice01
{
    0 个引用
    class Program
    {
        0 个引用
        static void Main(string[] args)
        {
            //  Console.WriteLine("hello world");
            //  Console.ReadLine();
            Operation oper = new Operation();
        }
    }
}
namespace Demo
{
0 个引用
class Operation
    {
    }
}
```

红色箭头处表面报错；
修改方法是什么？

类

- 类是一种数据结构，可以封装数据成员、方法成员和其他类。
- C#支持自定义类，编写自己的类来描述实际需要解决的问题

（1）命名空间：比作医院；

（2）类：相当于医院的各个科室；各科室中有自己的工作方法，相当于在类中定义的变量、方法等。

类

- **类的声明**

（1）使用类之前必修先进行声明，语法如下：

```
class  [类名]
{
    [类中的代码]
}
```

关键字

- **关键字：**C#语言中已经被赋予特定意义的一些单词，开发程序时，不可以把这些关键字作为命名空间、类、方法或者属性来使用。

int	public	this	finally	boolean	abstract
continue	float	long	short	throw	return
break	for	foreach	static	new	interface
if	goto	default	byte	do	case
void	try	switch	else	catch	private
double	protected	while	char	class	using

标识符

- 标识符可简单理解为一个名字，用来标识类名、变量名、方法名、属性名等。
- C#语言标识符命名规则如下：
  （1）由任意顺序的字母、下划线_和数字组成；
  （2）第一个字符不能是数字
  （3）不能是C#中的保留关键字
  （4）不能包含#、%或者$等特殊字符
  （5）严格区分大小写
- 示例：

  合法：_ID　name　user_age

  非法：4word　string

Main（）方法

- 每个C#程序中都必须包含一个Main方法，类体中的主方法，也叫入口方法；
- Static和void是Main方法的静态修饰符和返回值修饰符；
- Main方法必须声明为static,并且区分大小写。
- 注意：

  （1）Main方法在类或结构内声明，必须是静态的（static）的，不应该是公用（public）的

  （2）Main的返回类型有两种：void或int

  （3）Main方法可包含命令行参数string[] args，也可以不包括

  （4）通常Main方法中不写具体逻辑代码，只作实例化和方法的调用。

```
static void Main ( string[ ] args )  {  }
static void Main ( ) {  }
static int Main ( string[ ] args )  {  }
static int Main ( ) {  }
```

C#语句

- 构造C#程序的基本单位。可以声明变量、常量、调用方法、创建对象或者执行任何逻辑操作，C#语句以分号终止。

```
01  Console.WriteLine("Hello World");          //输出“Hello World”
02  Console.ReadLine();                        //定位控制台窗体
```

（1）上述语句用来在控制台窗口中输出和读取内容。

（2）Console类表示控制台应用程序的标准输入流、输出流和错误流。

（3）在开发控制台应用程序时，经常使用Console.Read方法或者Console.ReadLine方法固定控制台窗体。

注释

- 编写程序时不执行的代码或文字，主要功能是对某行/段代码进行说明；或者调试时，将代码设置为无效代码。
- 行注释和块注释

（1）行注释以符号“//”开头，后面时注释的内容；

```
static void    //错误的注释 Main(string[] args)
```

（2）块注释以符号“/*”标记开始，以“*/”标记结束

```
/*   private string _name;
   public string Name
   {
       get { return _name; }
       set { _name = value; }
   }  */
```

1.2 程序编写规范

- 统一代码缩进的样式
- 编写完一行代码后换行编写下一行；
- 合理使用空格
- 局部变量在最接近使用它的地方声明
- 不要使用goto系列语句，除非是用在跳出深层循环
- 避免在同一个文件中编写多个类
- 对于if语句，应该使用一对大括号{}把语句括起来
- Switch语句要有default语句来处理意外情况
- 避免编写超过5格参数的方法，传递多个参数，使用结构
- ……

命名规范

- 两种命名方法：
  （1）首字母大写
  （2）名称中第一个单词的首字母小写

```
01  public  class  User
02  {
03      public  void  GetInfo()
04      {
05      }
06  }
```

```
public void AverageScore()
{
    Console.WriteLine("平均分是{0}",(this.English+this.Math+this.Chinese)/3);
    Console.ReadKey();
}
```

思考：下述两种类的命名，哪种更好？

(1) class User{
}

(2) class aaa{
}

命名规范

- 接口的名称加前缀“I”

```
01  public  interface  Iconvertible
02  {
03      byte ToByte();
04  }
```

- 类的命名最好能够体现类的功能或操作

```
01  public  class  Operation
02  {
03  }
```

C#编程基础

任课教师：李娜
邮箱：709233393@qq.com

第二章 C#基本语法

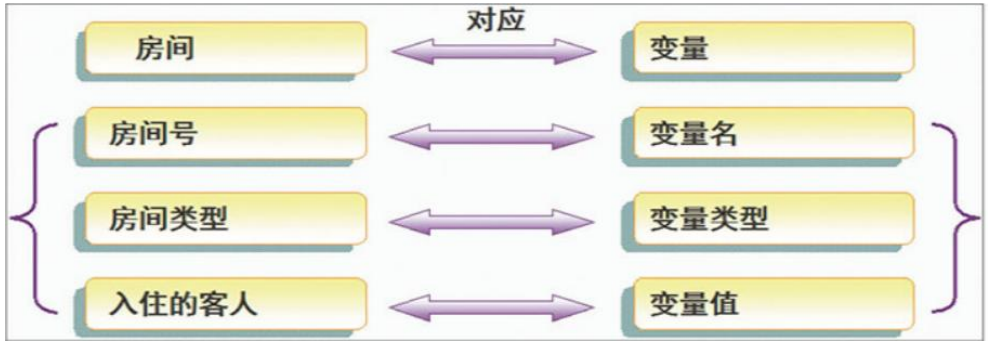
2.1 变量的声明和初始化

2.2 数据类型转换

2.3 运算符

变量的声明和初始化

- 变量：存储特定类型的数据



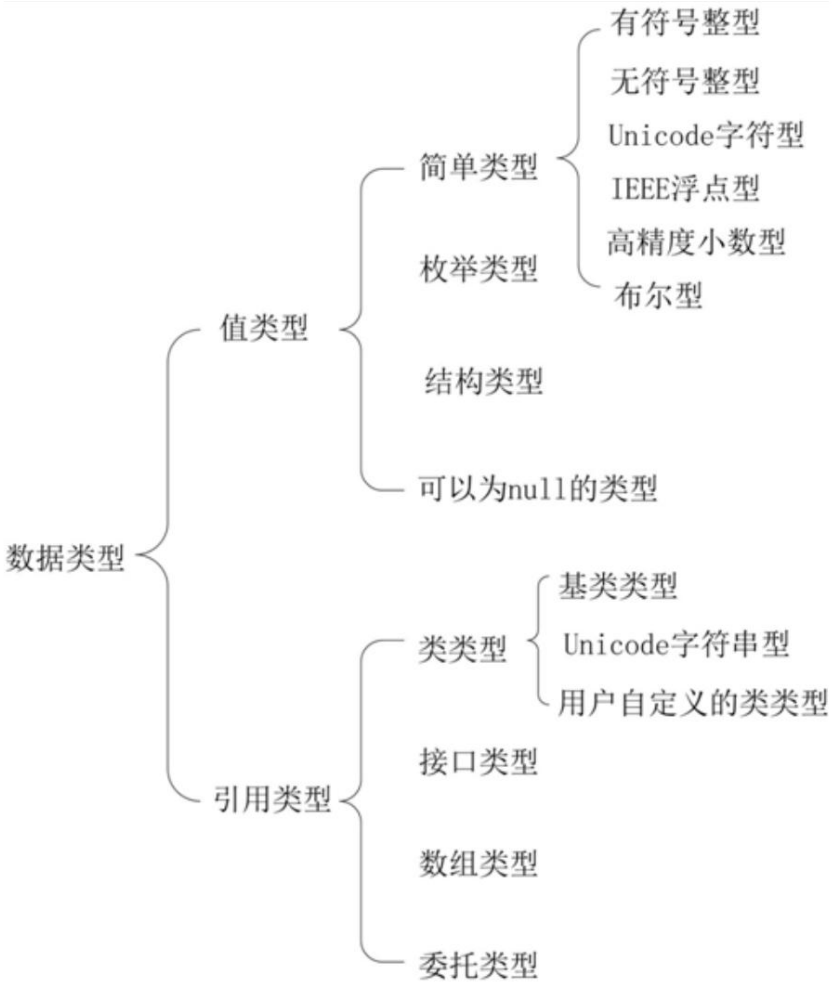
- 变量声明：

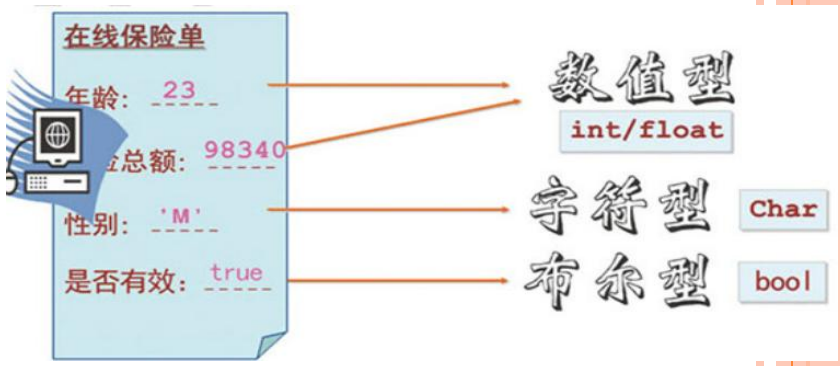
变量类型 变量名1，变量名2；

- 变量命名规则

（1）变量名只能由数字、字母和下划线
（2）变量名的第一个字符只能是字母或下划线，不能是数字
（3）不能使用关键字作为变量名
（4）同一个作用域内，变量不能重名

C#的数据类型


数据类型
值类型
简单类型
有符号整型
无符号整型
Unicode字符型
IEEE浮点型
高精度小数型
布尔型
枚举类型
结构类型
可以为null的类型
引用类型
类类型
基类类型
Unicode字符串型
用户自定义的类类型
接口类型
数组类型
委托类型


在线保险单
年龄: 23
总额: 98340
性别: 'M'
是否有效: true
数值型
int/float
字符型
Char
布尔型
bool

整数

- 十进制
- 八进制

（1）以0开头的数字：例如：0123(83)、-0123(-83)

- 十六进制

（1）以0x或0X开头的数，0x25(37)、0Xb01e(45086)

类别	位数	类型标识符	范围
有符号整型	8	sbyte	–128...127
有符号整型	16	short	–32,768...32,767
有符号整型	32	int	–2,147,483,648...2,147,483,647
有符号整型	64	long	–9,223,372,036,854,775,808... 9,223,372,036,854,775,807
无符号整型	8	byte	0...255
无符号整型	16	ushort	0...65,535
无符号整型	32	uint	0...4,294,967,295
无符号整型	64	ulong	0...18,446,744,073,709,551,615

浮点类型

- 浮点类型：处理含有小数的数据，类型：float、doube、decimal

类别	位数	类型标识符	范围/精度
IEEE浮点型	32	float	-3.4×10^{38} 到 $+3.4 \times 10^{38}$，7位精度
IEEE浮点型	64	double	$\pm 5.0 \times 10^{-324}$ 到 $\pm 1.7 \times 10^{308}$，15 位或16位精度
高精度小数型	128	decimal	$(-7.9 \times 10^{28} - 7.9 \times 10^{28}) / (10^{0-28})$，28 位或29位精度

- 示例1：创建一个控制台应用程序，声明double型变量height记录身高，单位为米；声明int型变量weight记录体重，单位为千克，根据“BMI=体重/（身高*身高）”的公式计算BMI指数

示例1代码：

```
    static void Main(string[] args)
    {
        double height = 1.78;
        int weight = 75;
        double BMIExponent = weight / (height * height);
        Console.WriteLine("身高为：" + height);
        Console.WriteLine("体重为：" + weight);
        Console.WriteLine("你的体重属于：");
        Console.WriteLine("BMI指数为：" + BMIExponent);
            if (BMIExponent < 18.5)
        {
            Console.WriteLine("体重过轻");
        }
        else if (BMIExponent < 24.9 && BMIExponent >= 18.5)
        {
            Console.WriteLine("体重正常");
        }
        else if (BMIExponent < 29.9 && BMIExponent >= 24.9)
        {
            Console.WriteLine("体重过重");
        }
        else {
            Console.WriteLine("肥胖");
        }
        Console.ReadLine();

    }
}
```

字符类型

- 字符类型记为char,可表示中文字符、英文字符或者数字；
- char类型数据在内存中占2个字节；

示例：　char ch1=‘L’;

　　char　ch2=‘s’;

字符串类型

- 记为string,用于描述包含零个、一个或多个Unicode标准字符的字符序列
- 示例：

  string name=“我喜欢吃西瓜”;

  string address=“耿丹学院”;

- “+”号可以连接两个字符，这里的“+”表示连接。

- 占位符：{序号}

练习1：定义两个个变量存储客户的名称，然后屏幕上显示：你好，**，你好，~~

练习1代码：

方法1：

```
static void Main(string[] args)
{
    string name1 = "张三";
    string name2 = "李四";
    Console.WriteLine("你好，" + name1 + "," + "你好，" + name2);
    Console.ReadLine();
```

方法2：

```
static void Main(string[] args)
{
    string name1 = "张三";
    string name2 = "李四";
    Console.WriteLine("你好，{0}；你好，{1}", name1,name2);
    Console.ReadLine();
}
```

- 作业：完成实验报告一中的第2个实验

示例2：

```
static void Main(string[] args)
{
    char a = 'a';                                   //声明字符a
    char b = '8';                                   //声明字符b
    char c = 'L';                                   //声明字符c
    char d = '.';                                   //声明字符d
    char e = '|';                                   //声明字符e
    char f = ' ';                                   //声明字符f
    //使用IsLetter方法判断a是否为字母
    Console.WriteLine("IsLetter方法判断a是否为字母：{0}", Char.IsLetter(a));
    //使用IsDigit方法判断b是否为数字
    Console.WriteLine("IsDigit方法判断b是否为数字：{0}", Char.IsDigit(b));
    //使用IsLetterOrDigit方法判断c是否为字母或数字
    Console.WriteLine("IsLetterOrDigit方法判断c是否为字母或数字：{0}", Char.IsLetterOrDigit(c));
    //使用IsLower方法判断a是否为小写字母
```

练习2：问用户喜欢吃什么水果，假设用户输入“苹果”，则显示“哈哈，这么巧，我也喜欢吃苹果”

布尔型

- C#采用布尔类型描述实际应用中“真”和“假”、“成立”和“不成立”或“存在”和“不存在”的情况；
- 类型标识符为bool，可能值为true或false,其中true用以表示“真”、“成立”或“存在”的情况，而false则表示“假”、“不成立”或“不存在”
- Bool类型数据在内存中占1个字节，不能和整数1与0转换

变量的初始化

- 变量的初始化就是给变量赋值。

（1）单独初始化变量

```
int sum;
sum = 2017;
```

（2）声明时初始化变量

```
01  int mr = 927;                                    //初始化整型变量mr
02  //初始化字符串型变量mr_1、mr_2和mr_3
03  string mr_1 = "零基础学", mr_2 = "项目入门", mr_3 = "实例精粹";
```

（3）同时初始化多个变量

```
01  int a, b, c, d, e;
02  a = b = c = d = e = 0;
```

作业：完成实验报告一中的实验3

变量的作用域

- 变量的作用域：程序代码能够访问该变量的区域。

（1）成员变量：在类体内定义的变量，在整个类内都有效。

```
static void Main(string[] args)
{
    double height = 1.78;
    int weight = 75;
    double BMIExponent = weight / (height * height);
```

（2）局部变量：在类的方法体重定义的变量，只在当前代码块中有效。

（3）方法被调用，为局部变量分配内存空间；方法调用结束，局部变量占用的内存空间被释放，局部变量被销毁

常量

- 常量：值固定不变的量
- 常量的分类：const常量、readonly常量

（1）const常量：静态常量；在声明时就进行初始化，并且之后不可以再进行更改。

```
01  const double PI = 3.1415926;          //正确的声明常量的方法
02  const int MyInt;                      //错误：定义常量时没有初始化
```

（2）readonly常量：动态常量；只能在构造函数中进行赋值

```
class Program
{
    readonly int Price;                   //定义一个reanonly常量
    Program()                             //构造函数
    {
        Price = 368;                      //在构造函数中修改reanonly常量的值
    }
    static void Main(string[] args)
    {
    }
}
```

常量

- const常量和readonly常量的区别：

  (1)const常量必须在声明时初始化，而readonly常量可以延迟到构造函数中初始化；

  （2）const常量在编译时就被解析，即将常量的值替换成了初始化的值，而readonly常量的值需要在运行时确定；

  （3）const常量可以在类中或者方法体中定义，而readonly常量只能在类中定义。

数据类型转换

- 类型转换是将一个值从一种数据类型更改为另外一种数据类型。

（1）低精度数据类型向高精度数据类型转换，不会溢出；

（2）高精度数据类型向低精度数据类型转换，信息丢失或者出错。

- 隐式转换和显示转换

（1）隐式类型转换：不需要声明就能进行的转换。

```
01  int i = 927;              //声明一个整型变量i并初始化为927
02  long j = i;               //隐式转换成long类型
```

（2）显式类型转换：强制类型转换，需要在代码中明确声明要转换的类型。

（3）显式类型转换形式：（类型说明符）表达式

```
int  i ;
i = (int)4.5;
```

- 使用Convert类进行转换

（1）示例1： double x=198.99

int y=Convert.ToInt32(x)

（2）示例2：long l=3000000000;

int i=Convert.ToInt32(l)

- 问题：下述转换能进行吗？

string s="123abc";

double d=Convert.ToDouble(s);

int n=Convert.ToInt32(s)

C#编程基础

任课教师：李娜
邮箱：709233393@qq.com

第二章 C#基本语法

2.1 变量的声明和初始化

2.2数据类型转换

2.3 运算符

运算符

- 运算符：具有运算功能的符号；运算符所操作的数值或表达式称为操作数
  - （1）算术运算符
  - （2）自增自减运算符
  - （3）赋值运算符
  - （4）关系运算符
  - （5）逻辑运算符
  - （6）条件运算符
  - ……

算术运算符

- 包括：+（加）、-（减）、*（乘）、/（除）、%（求余数）

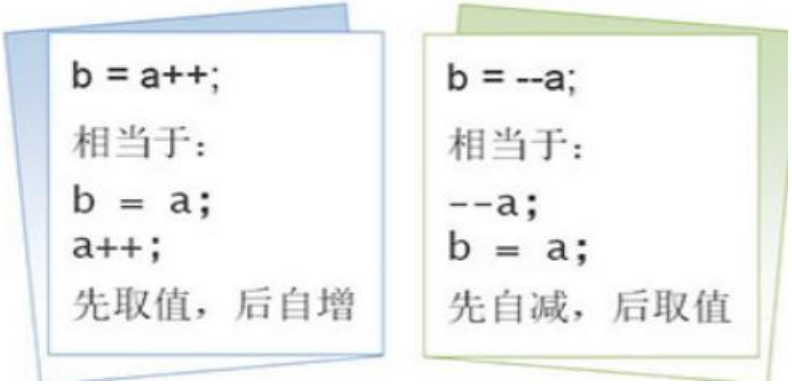
运算符	说明	实例	结果
+	加	12.45f+15	27.45
-	减	4.56-0.16	4.4
*	乘	5l*12.45f	62.25
/	除	7/2	3
%	求余	12%10	2

练习1：用户输入姓名、语文、数学、英语三门课的成绩，给用户显示：**，你的总成绩是**分，平均分是**

练习2：编程实现计算几天（47天）是几周零几天？

自增自减运算符

- 自增运算符：++；自减运算符：--；表示对数值型变量的值进行加1或者减1操作



- 说出下面的运行结果：

```
int i=0,j=0;
int  post_i，pre_j;
 post_i=i++;
Console.WriteLine(i);
Pre_j=++j;
Console.WriteLine(j)
```

- 下面写法是否正确？

```
3++;
(i+j)++;
```

赋值运算符

- 简单赋值运算符

```
int a = 100;
```

- 复合赋值运算符

  +=、-=、、/=、*=、%=、&=、|=、>>=、<<=

```
int a = 3;
a += 2;
```

关系运算符

- 双目运算符，常用于比较，返回一个布尔值表示运算结果。

  （1）常用在条件语句中作为判断的依据。

运 算 符	作 用	举 例	操 作 数 据	结 果
>	大于	a > b	整型、浮点型、字符型	false
<	小于	156 < 456	整型、浮点型、字符型	true
==	等于	c == c	基本数据类型、引用型	true
!=	不等于	y! = t	基本数据类型、引用型	true
>=	大于或等于	479>=426	整型、浮点型、字符型	true
<=	小于或等于	12.45<=45.5	整型、浮点型、字符型	true

逻辑运算符

- 对真和假两种布尔值进行运算，运算结果仍是一个布尔值。

运算符	含义	用法	结合方向
&&，&	逻辑与	op1&&op2	从左到右
||、|	逻辑或	op1||op2	从左到右
!	逻辑非	！op	从右到左

op1	op2	op1&&op2	op1||op2	!op1
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

- 练习3：

  用户输入张三的语文和数学成绩，输出下面判断是否正确，正确输出True,错误输出False

  判断1：张三的语文和数学成绩都大于90分；

  判断2：语文和数学有一门大于90分；

条件运算符

- 用”?”表示，唯一的三目运算符，需要3个操作数
  - 格式：**<表达式1>?<表达式2>:<表达式3>**
  - 表达式1是一个布尔值，可以为真或假；如果表达式1为**真**，返回**表达式2** 的运算结果；如果表达式1为**假**，则返回**表达式3**的运算结果。
  - 示例：int x=5,y=6,max;

    max=x<y?y:x;

- 注意：**条件运算符不能单独作为语句**，需要一个变量来记录三目运算符计算后的结果。

```
static void Main(string[] args)
{
    int a = 10, b = 5;
    (a > b) ? a : b;
    Console.WriteLine(n);
    Console.ReadLine();
}
```

练习4：

键盘接收用户的姓名和年纪，使用条件运算符输出年龄所处的阶段

- 大于40岁，输出“人到中年了”；
- 小于40岁，输出“正是奋斗的黄金年龄”

运算符优先级与结合性

- C#中运算符优先级由高到低的顺序依次是：
  （1）自增、自减运算符
  （2）算术运算符
  （3）移位运算符
  （4）关系运算符
  （5）逻辑运算符
  （6）条件运算符
  （7）赋值运算符

```
01  !a++;                    等效于:    !(a++);
02  a ? b : c ? d : e;       等效于:    a ? b : (c ? d : e);
03  a = b = c;               等效于:    a = (b = c);
04  a + b - c;               等效于:    (a + b) - c;
```

注意：

- 使用赋值运算符时的注意事项：

（1）左操作数不能是常量

（2）所有表达式都可以作为赋值运算符的右操作数

- 思考：下述语句是否正确？

```
int i=1,j=2,k=3;
const int val=5;
5=k;
i+j=k;
val=i;
```

运算符小结：

- 算术运算符：+、-、*、/、%
- 赋值运算符：=
- 自增/减运算符：++、--
- 逻辑运算符：&&、&、||、！
- 关系运算符：==、>、<、>=、<=、！=
- 条件运算符：<表达式1>?<表达式2>:<表达式3>

C#编程基础

任课教师：李娜
邮箱：709233393@qq.com

第三章 流程控制

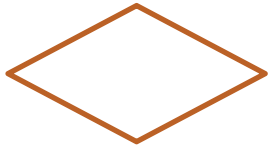
3.1 分支结构

3.2 循环结构

程序的基本结构---程序流程图

用图形、流程线和文字说明描述程序的**基本操作**和**控制流程**。

起止框

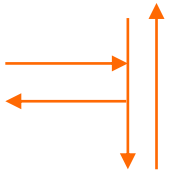


判断框

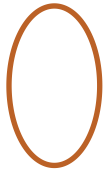
处理框

输入/输出框

注释框



流向线



连接点

实例3.1：

圆面积和周长的计算。

输入(I)：

圆半径R

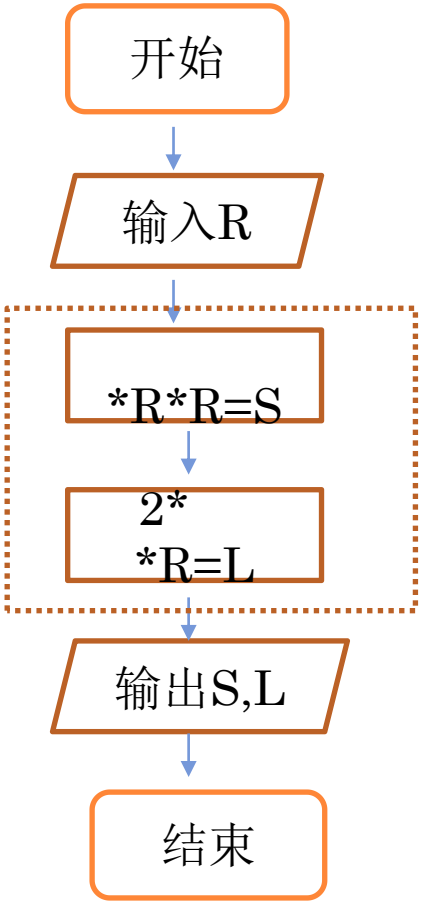
处理(P)：

圆面积：S=　*R*R

圆周长：L=2*　*R

输出(O)：

圆面积S，周长L



实例3.2

- 实数绝对值的计算

输入(I):

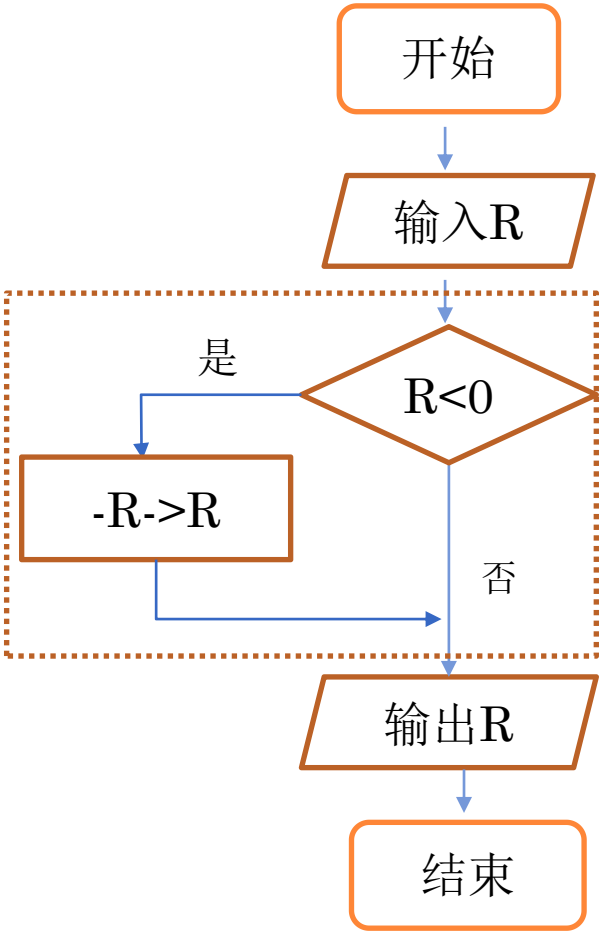
实数R

处理(P):

$$|R| = \begin{cases} R & R \geq 0 \\ -R & R \leq 0 \end{cases}$$

输出(O):

输出|R|



实例3.3

- 整数累加

输入(I)：

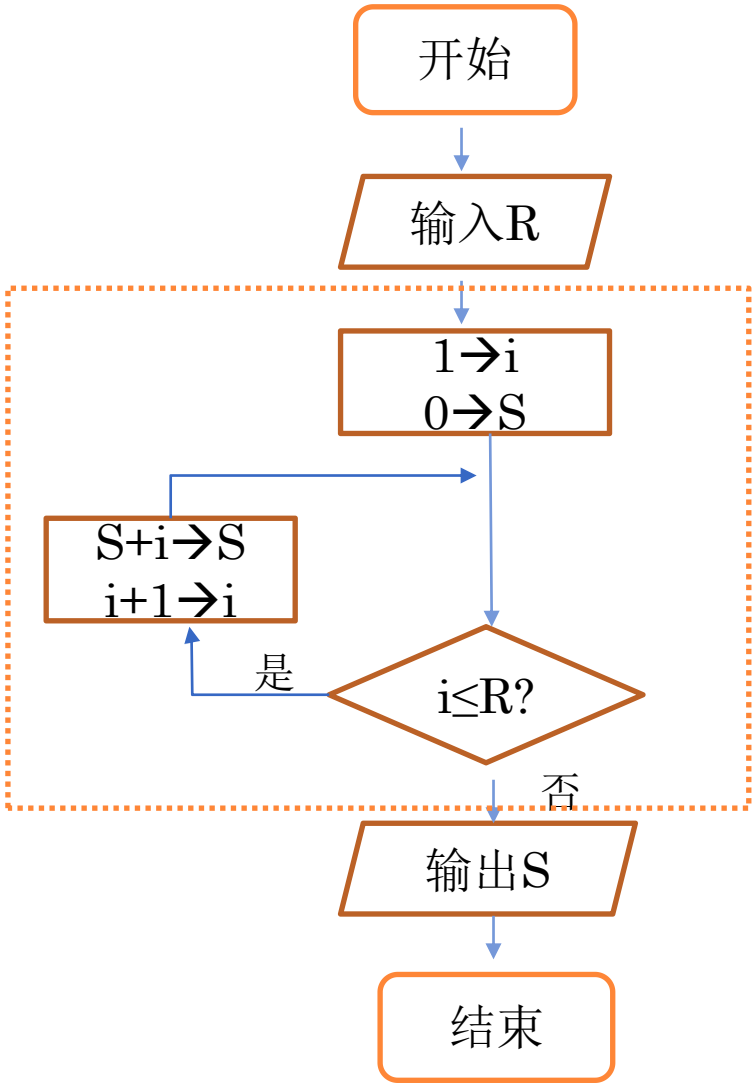
正整数R

处理(P)：

S=1+2+3+….+R

输出(O)：

输出S

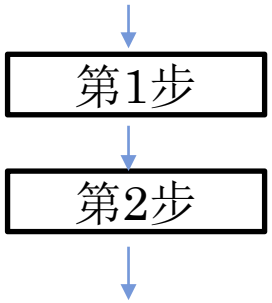


程序的控制结构

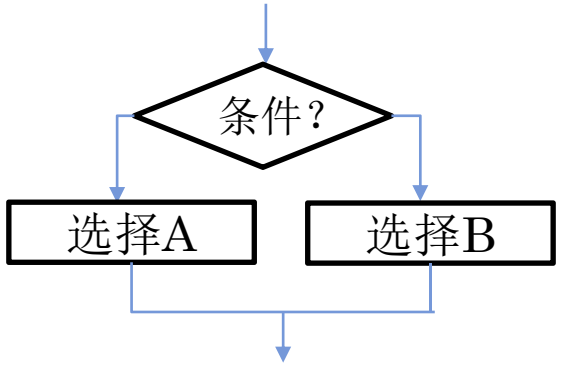
Bohm C., Jacopini G. "Flow diagrams, Turing machines and languages with only two formation rules." Communications of the ACM, Vol.9, pp. 366--371. 1966.

从理论上证明：任何具有**单入口单出口**的程序都可以用三种基本结构表达。

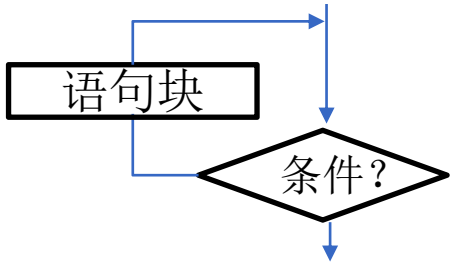
- 顺序



- 分支



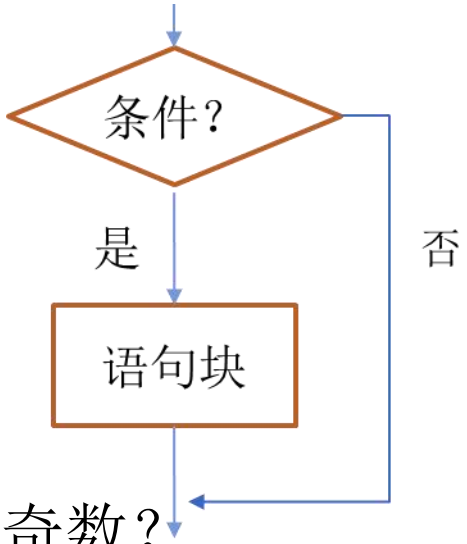
- 循环



分支结构

- if关键字组成选择语句：

```
if(表达式)
{
    语句块
}
```



- 示例1：判断输入的一个数字是否为奇数？

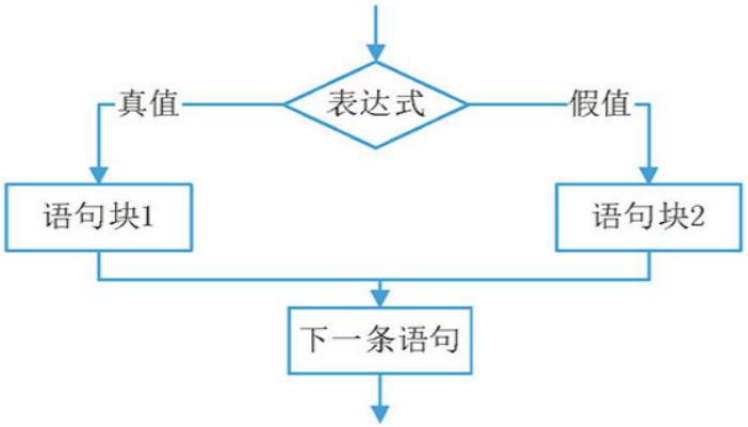
```
static void Main(string[] args)
{
    Console.WriteLine("请输入一个数字：");
    int iInput = Convert.ToInt32(Console.ReadLine());
    if (iInput % 2 != 0)//使用if语句进行判断
    {
        Console.WriteLine(iInput + " 是一个奇数！");
    }
    Console.ReadLine();
}
```

作业：完成实验报告：判断输入的一个数字是奇数还是偶数？

分支结构

- if...else语句

```
if(表达式)
{
    语句块1;
}
else
{
    语句块2;
}
```



示例2：用户输入成绩，大于90分输出“你非常优秀”，否则输出“希望你继续努力”

```
static void Main(string[] args)
{
    Console.WriteLine("请输入你的分数：");
    int score = Convert.ToInt32(Console.ReadLine());
    if (score > 90)
    {
        Console.WriteLine("你非常优秀！");
    }
    else
    {
        Console.WriteLine("希望你继续努力！");
    }
    Console.ReadLine();
}
```

分支结构

- if...else if...else语句

  (1)表达式部分用括号{ }括起来

  （2）else if 和 else都必须跟if一起使用，不能单独使用

```
if(表达式1)
{
    语句1;
}
else if(表达式2)
{
    语句2;
}
else if(表达式3)
{
    语句3
}
…
else if(表达式m)
{
    语句m
}
else
{
    语句n
}
```

- 练习1：

根据用户的输入数字，判断该数是奇数还是偶数

示例2：根据用户输入的年龄，输出以下信息：

- 年龄<18,输出：年龄还小，需要好好学习；
- 年龄在18和30之间，输出：好好工作；
- 年龄在30到55之间，输出：中年人，努力工作；
- 年龄大于55岁，输出：最美不过夕阳红！

```
static void Main(string[] args)
{
    int YouAge = 0;                                  //声明一个int类型的变量YouAge，值为0
    Console.WriteLine("请输入您的年龄：");
    YouAge = int.Parse(Console.ReadLine());          //获取用户输入的数据
    if (YouAge <= 18)                                //调用if语句判断输入的数据是否小于等于18
    {
        Console.WriteLine("您的年龄还小，要努力奋斗哦！");
    }
    else if (YouAge > 18 && YouAge <= 30)            //判断是否大于18岁小于30岁
    {
        Console.WriteLine("您现在的阶段正是努力奋斗的黄金阶段！");
    }
    else if (YouAge > 30 && YouAge <= 50)            //判断输入的年龄是否大于30岁小于等于50岁
    {
        Console.WriteLine("您现在的阶段正是人生的黄金阶段！");
    }
    else
    {
        Console.WriteLine("最美不过夕阳红！");
    }
    Console.ReadLine();
}
```

练习2：根据输入的分数划分等级；90分以上输出：优秀；80到90之间，输出：良好；70~80之间，输出:中等；60~70之间，输出：及格；60分以下：不及格

课下练习：实现一个报销业务流程，判断报销的金额是否小于5000，如果是，输出“正常报销！”；否则，输出“超出报销额度”

分支结构

- if语句的嵌套：上述3种形式的选择语句之间可以进行互相嵌套

```
if(表达式1)
{
    if(表达式2)
        语句1;
    else
        语句2;
}
```

```
if(表达式1)
{
    if(表达式2)
        语句1;
    else
        语句2;
}
else
{
    if(表达式2)
        语句1;
    else
        语句2;
}
```

- **练习4：**

根据用户输入的密码，进行判断：

（1）如果输入正确（888888）,输出：登录成功；

（2）再次输入密码，如果输入错误，输出：密码错误，再输入一遍；如果输入正确，输出：密码输入两次，终于正确了。

（3）如果再次输入仍然错误，输出：密码不对，程序退出

分支结构

- switch多分支语句

（1）switch关键字后面的括号中是要判断的参数，**参数值不可为浮点数**；

（2）case关键字后面有相应的语句块；

（3）case后各**常量值不可以相同**

（4）case后面的语句块可以包含多条语句，不必使用大括号{}括起来；

（5）case语句和default语句的顺序可以改变，不影响执行结果

（6）一个switch语句只能**有一个default语句**，并且default语句可以省略

```
switch(判断参数)
{
case 常量值1:
     语句块1
     break;
case 常量值2:
     语句块2
     break;
…
case 常量值n:
     语句块n
     break;
defaul:
     语句块n+1
     break;
}
```

示例3：使用switch语句实现查询高考分数段的功能

```
static void Main(string[] args)
{
    Console.WriteLine("请输入要查询的录取分数线（比如民办本科、艺术类本科、体育类本科、二本、一本）");
    string strNum = Console.ReadLine();//获取用户输入的数据
    switch (strNum)
    {
        case "民办本科":
            Console.WriteLine("民办本科录取分数线：350");
            break;
        case "艺术类本科":
            Console.WriteLine("艺术类本科录取分数线：290");
            break;
        case "体育类本科":
            Console.WriteLine("体育类本科录取分数线：280");
            break;
        case "二本":
            Console.WriteLine("二本录取分数线：445");
            break;
        case "一本":
            Console.WriteLine("一本录取分数线：555");
            break;
        default:
            Console.WriteLine("您输入的查询信息有误！");
            break;
    }
    Console.ReadLine();
}
```

练习5：

对用户的成绩输出对应的等级，成绩大于90分，输出：优秀；成绩在80和90分之间，输出：良好；成绩在70~80之间，输出中等，成绩在60~70之间，输出及格；否则，输出：不及格

分支结构

- if….else if…else：主要对**布尔表达式、关系表达式**或者**逻辑表达式**进行判断

- switch多分支语句主要对**常量值**进行判断

练习6：

某大型商超为答谢新老顾客，当累计消费金额达到一定数额时，顾客可享受不同的折扣；

（1）尚未超过200元，按照小票价格支付全款；

（2）200到600之间，消费金额可以享8.5折优惠；

（3）600到1000元之间，消费金额可以享7折优惠；

（4）大于1000元，消费金额可以享6折优惠；

顾客输入购物小票上消费金额，在控制台上输出该顾客将享受的折扣与打折后需支付的金额。

C#编程基础

任课教师：李娜
邮箱：709233393@qq.com

第三章 流程控制

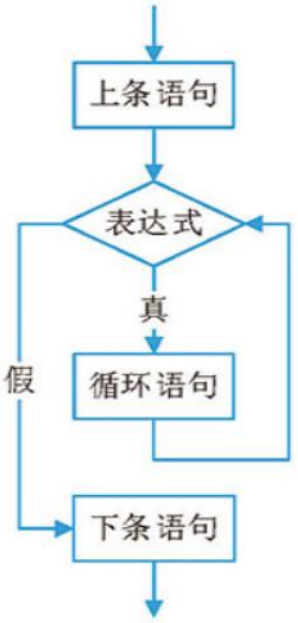
3.1 分支结构

3.2 循环结构

循环结构

- while循环

```
while(表达式)
{
    语句
}
```



- 表达式值为**真**或者**假**；为真，则执行循环；为假，则退出循环。

- 示例1：计算1~100的整数之和。

```
static void Main(string[] args)
{
    int iNum = 1;
    int iSum = 0;
    while (iNum <= 100)
    {
        iSum += iNum;
        iNum++;
    }
    Console.WriteLine("1到100的累加结果是：" + iSum);
    Console.ReadLine();
}
```

注意：

（1）循环体如果是多条语句，需要用大括号括起来；否则，循环体只包含while语句后的第一条语句

（2）循环体内或者表达式中必须有使循环结束的条件

- 练习1

  使用while循环，实现1~100的偶数求和

- 练习2：

  猜数字小游戏，随机生成一个1~20的数字，玩家每次通过键盘输入一个数字，如果输入的数字和基准数相同，则成功过关，否则重新输入；玩家输入-1，表示退出游戏；

  提示：使用下面两条语句生成1~20之间的随机数

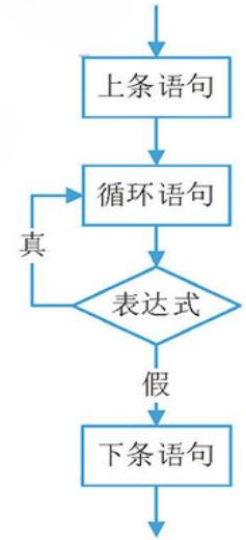
```
Random rand=new Random();
int  num=rand.Next(1,20);
```

循环结构

- do...while循环
  （1）先执行循环体，再判断循环条件；
  （2）do是关键字，必须与while配对使用；
  （3）do与while之间语句为循环体
  （4）while语句后一定有分号”;”

```
do
{
    语句
}
while(表达式);
```

示例2：使用do....while语句实现1~100的数据求和。




```
static void Main(string[] args)
{
    int iNum = 1;
    int iSum = 0;
    do
    {
        iSum += iNum;
        iNum++;
    } while (iNum <= 100);
    Console.WriteLine("1到100的累加结果是：" + iSum);
    Console.ReadLine();
}
```

- 练习3：

(1)使用do…while语句计算n的阶乘（1*2*3…*n），要求输入n的值，输出n的阶乘。

- 思考：

(1) do…while循环和while循环有什么不同？

（2）下面两段代码分别执行几次循环？

```
int iNum = 1;
while (iNum < 1)
{
    Console.WriteLine(iNum);
    iNum++;
}
```

```
int iNum = 1;
do
{
    Console.WriteLine(iNum);
    iNum++;
} while (iNum < 1);
```

循环结构

- for循环执行过程如下：

（1）求解表达式1；

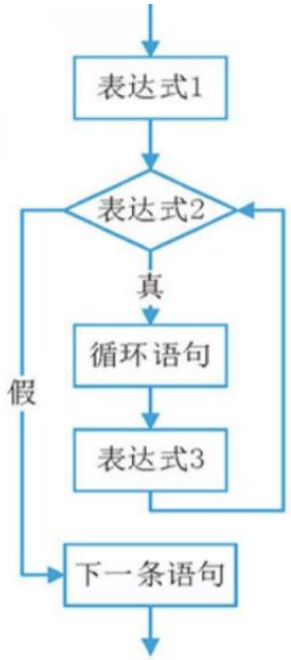
（2）求解表达式2，若表达式2的值为“真”，则执行循环体内的语句组，然后执行下面第（3）步，若值为“假”，则执行下面第（5）步；

（3）求解表达式3；

（4）转回到第(2)步执行；

（5）循环结束，执行for循环接下来的语句。

```
for(表达式1;表达式2;表达式3)
{
    语句组
}
```



循环结构

- for循环常用格式:

```
for(循环变量赋初值;循环条件;循环变量增值)
{
    语句组
}
```

示例3：使用for循环实现1到100之间的整数求加

```
static void Main(string[] args)
{
    int iSum = 0;
    for (int iNum = 1; iNum <= 100; iNum++)
    {
        iSum += iNum;
    }
    Console.WriteLine("1到100的累加结果是: " + iSum);
    Console.ReadLine();
}
```

for循环的常用形式：

(1)**表达式1可以省略**；for循环中“表达式1”一般用于为循环变量赋初值，若省略“表达式1”，则需要在for循环的前面为循环条件赋初值。

```
for(;iNum <= 100; iNum++)
{
    sum += iNum;
}
```

（2）**省略“表达式2”**：如果省略了“表达式2”，则循环没有终止条件，会无限循环下去，针对该情况，会配合break语句来结束循环。

```
for(iNum = 1;;iNum++)
{
    iSum += iNum;
}
```

循环结构

（3）3个表达式都省略：需要配合使用break语句来结束循环，否则，会造成死循环。

```
int i = 100;
for(;;)
{
    Console.WriteLine(i);
}
```

- for循环中逗号的应用

（1）在for循环中，表达式1和表达式3处都可以使用逗号表达式。

```
for(iSum=0,iNum=1;iNum<=100;iNum++)
{
   iSum+=iNum;
}
```

- 实验4：

使用for循环实现1到100之间的奇数累加

循环的嵌套

- 在一个循环里又包含另一个循环，构成循环的嵌套。
  （1）while循环中嵌套while循环
  （2）do…while循环嵌套
  （3）for循环嵌套

```
while（表达式）
{
    语句组
    while（表达式）
    {
        语句组
    }
}
```

```
do
{
    语句组
    do
    {
        语句组
    }
    while（表达式）;
}while（表达式）;
```

```
for(表达式;表达式;表达式)
{
    语句组
    for（表达式;表达式;表达式）
    {
        语句组
    }
}
```

循环的嵌套

（4）while 循环中嵌套do…while循环

（5）while循环中嵌套for循环；

（6）for循环中嵌套while循环

```
while(表达式)
{
    语句组
    do
    {
        语句组
    }
    while(表达式) ;
}
```

```
while(表达式)
{
    语句组
    for(表达式;表达式;表达式)
    {
        语句组
    }
}
```

```
for(表达式;表达式;表达式)
{
    语句组
    while(表达式)
    {
        语句组
    }
}
```

循环的嵌套

- 示例4：

  使用嵌套的for循环打印九九乘法表

```
static void Main(string[] args)
{
    int iRow, iColumn;                                  //定义行数和列数
    for (iRow = 1; iRow < 10; iRow++)                       //行数循环
    {
        for (iColumn = 1; iColumn <= iRow; iColumn++)       //列数循环
        {
            //输出每一行的数据
            Console.Write("{0}*{1}={2} ", iColumn, iRow, iRow * iColumn);
        }
        Console.WriteLine();                            //换行
    }
    Console.ReadLine();
}
```

```
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
```

练习5：使用循环嵌套输出金字塔形状，需要注意：控制三角形输出的行数、控制三角形的空白位置；类似如下显示：

跳转语句

break语句和continue语句

- break语句：**跳出循环体**，执行循环体外的语句；

  (1)通常用在switch\while\do..while或for语句中

  （2）存在循环相互嵌套时，break应用于最里层的语句；

  （3）一般会与if语句进行搭配使用

```
break;
```

- 示例5：1~100的正整数求和运算，计算到50时，退出循环；

```
static void Main(string[] args)
{
    int iNum = 1;
    int iSum = 0;
    while (iNum <= 100)
    {
        iSum += iNum;
        iNum++;
        if (iNum == 50)
            break;
    }
    Console.WriteLine("1到49的累加结果是：" + iSum);
    Console.ReadLine();
}
```

```
1到49的累加结果是：1225
```

- 练习6：通过键盘输入一个整数，判断这个数是否是素数（只能被1和其自身整除的数为素数）

跳转语句

- continue语句

  (1)**结束本次循环**，常用于while、do..while或for语句中，用于忽略循环语句内位于它后面的代码而直接开始下一次的循环。

  （2）一般会与if语句进行搭配使用，表示在某种条件下不执行后面的够，直接开始下一次的循环。

```
continue;
```

- 示例6：在for循环中使用continue语句实现1到100之间的偶数和。

```
static void Main(string[] args)
{
    int iSum = 0;
    int iNum = 1;
    for (; iNum <= 100; iNum++)
    {
        if (iNum % 2 == 1)
            continue;
        iSum += iNum;
    }
    Console.WriteLine("1到100之间的偶数的和：" + iSum);
    Console.ReadLine();
}
```

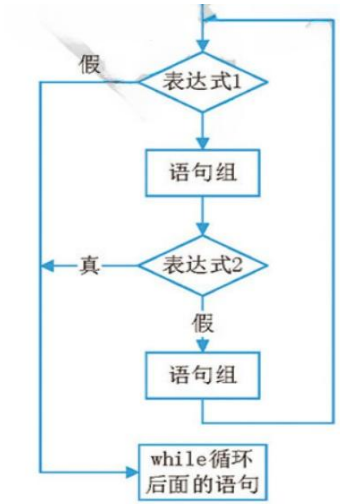
跳转语句

- Continue和break语句的区别：

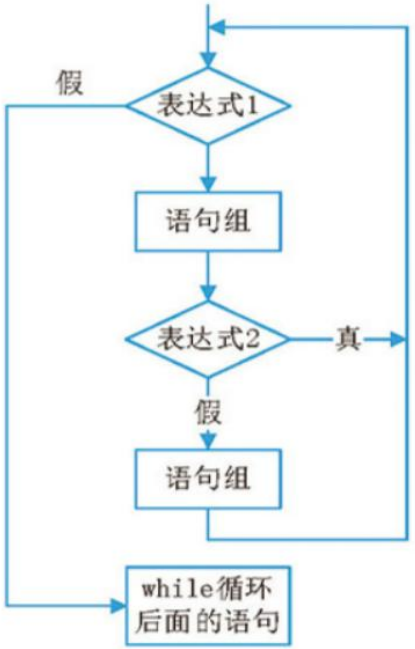
  (1)continue只结束本次循环，而不是终止整个循环；

  （2）break结束整个循环后，开始执行循环之后的语句

```
while（表达式 1）
{
    if（表达式 2）
        break;
}
```




```
while（表达式 1）
{
    if（表达式 2）
        continue;
}
```

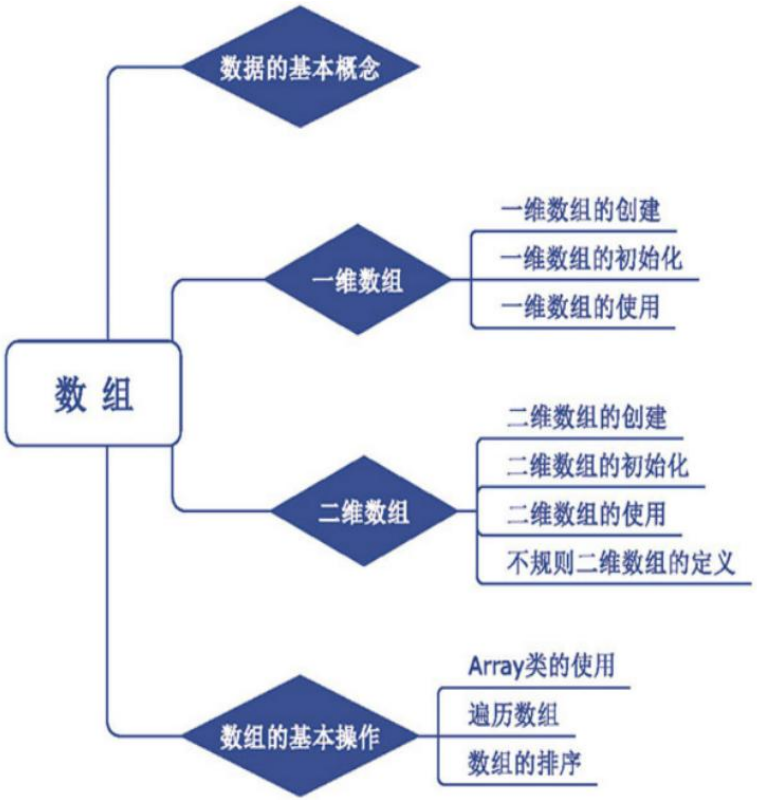


C#编程基础

任课教师：李娜
邮箱：709233393@qq.com

第四章 数组—批量数据处理

4.1 一维数组
4.2 二维数组
4.3 数组的基本操作



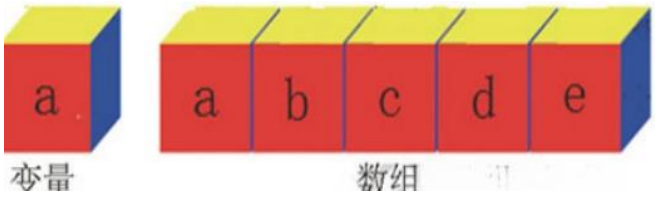
数组

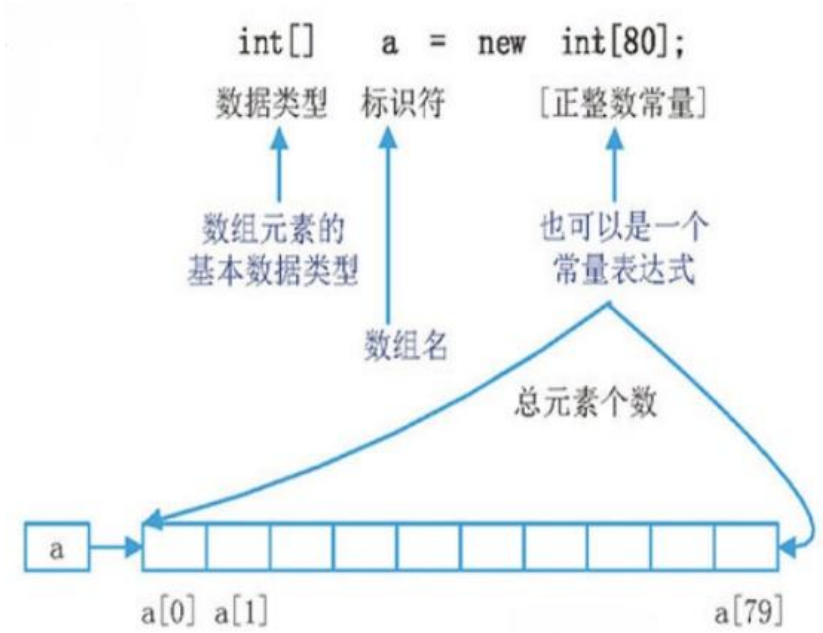
- 数组：相同类型的数据集合

  （1）数组中每一个变量称为数组的元素，数组能够容纳元素的数量称为数组的长度。

  （2）数组中的每个元素都具有唯一的索引与其相对应，数组的索引从零开始。

  （3）数组的定义需要包含：元素类型、数组的维数、每个维数的上下限





一维数组

- 一维数组：一组相同类型数据的线性集合。
- 创建：

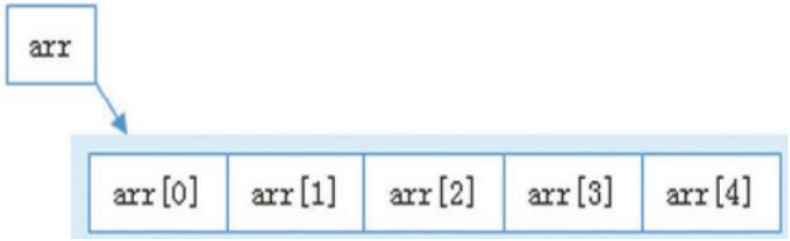
（1）先声明，再用new关键字进行内存分配：

- **声明： 数组元素类型[] 数组名字；**

  例如： int[] arr;　　string[] str;

- **内存分配：数组名字=new 数组元素类型[数组元素的个数]**

  例如：arr=new int[5];



- 注意：

使用new关键字为数组分配内存时，整型数组中各个元素的初始值都为0.

一维数组

- 创建：

（2）声明的同时为数组分配内存：

- **语法：**

数组元素类型[] 数组名=new 数组元素类型[数组元素的个数]

- **示例：**

int[ ] arr=new arr[12];

一维数组

- 一维数组的初始化：单个数组元素赋值和整个数组赋值。

（1）为单个数组元素赋值

```
01  int[] arr = new int[5];          //定义一个int类型的一维数组
02  arr[0] = 1;                      //为数组的第1个元素赋值
03  arr[1] = 2;                      //为数组的第2个元素赋值
04  arr[2] = 3;                      //为数组的第3个元素赋值
05  arr[3] = 4;                      //为数组的第4个元素赋值
06  arr[4] = 5;                      //为数组的第5个元素赋值
```

或者循环实现：

```
int[] arr = new int[5];              //定义一个int类型的一维数组
for (int i = 0; i < arr.Length; i++) //遍历数组
{
    arr[i] = i + 1;                  //为遍历到的数组元素赋值
}
```

（2）同时为整个数组赋值

- 同时为整个数组赋值需要使用**大括号**，将要赋值的数据包含在大括号中，并用“,”隔开

- string[] arrStr=new string[7]
  {“Sun”, ”mon”, ”Tue”, ”Wed”, ”Thu”, ”Fri”, ”Sat”}
- string[] arrStr=new string[]
  {“Sun”,”mon”,”Tue”,”Wed”,”Thu”,”Fri”,”Sat”}
- string[] arrStr= {“Sun”,”mon”,”Tue”,”Wed”,”Thu”,”Fri”,”Sat”}

- 说明：实现效果相同，定义一个长度为7的string类型数组，并进行初始化。

一维数组的使用

- 示例1：输出一年中每个月的天数

```
namespace ArrayPractice01
{
    0 个引用
    class Program
    {
        0 个引用
        static void Main(string[] args)
        {
            int[] day = new int[] { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
            for (int i = 0; i < 12; i++)
            {
                Console.WriteLine((i + 1) + "月有" + day[i] + "天");
            }
            Console.ReadLine();
        }
    }
}
```

```
1月有31天
2月有28天
3月有31天
4月有30天
5月有31天
6月有30天
7月有31天
8月有31天
9月有30天
10月有31天
11月有30天
12月有31天
```

练习1：定义一个数组，保存学生的成绩，

（1）输出学生成绩，

（2）输出该组学生的平均分；

（3）计算最高分，最低分；

二维数组

- 多维数组：可以用多个索引访问的数组。用多个中括号或者中括号内加逗号，表明是多维数组。
- 二维数组的创建
  - **声明二维数组**

```
type[,] arrayName;
type[][] arrayName;
```

（1）type:二维数组的数据类型； arrayName:二维数组的名称；

（2）例如： int[,] myarr; int[][] myarr;

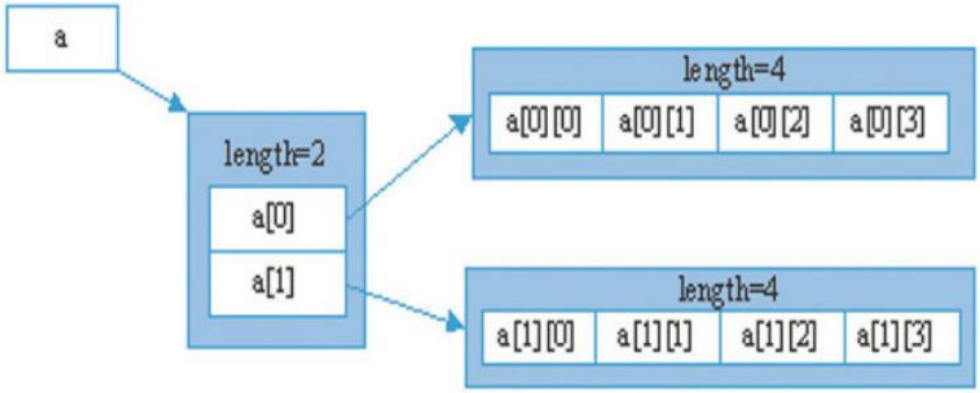
（3）注意：二维数组声明时也没有分配内存空间，

二维数组的创建

- 二维数组的创建
  - **内存分配方式一：**

```
int[,] a = new int[2, 4];     //定义一个2行4列的int类型二维数组
```

内存分配示意图:

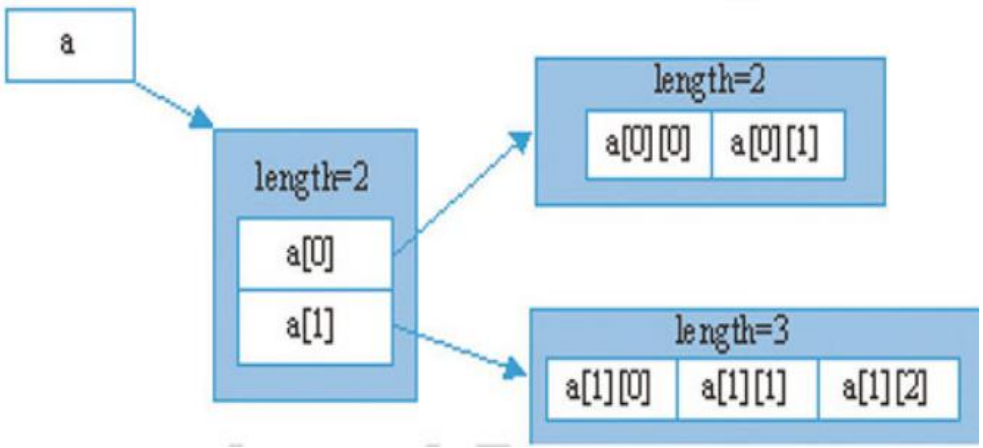


二维数组的创建

- 二维数组的创建
  - **内存分配方式二**

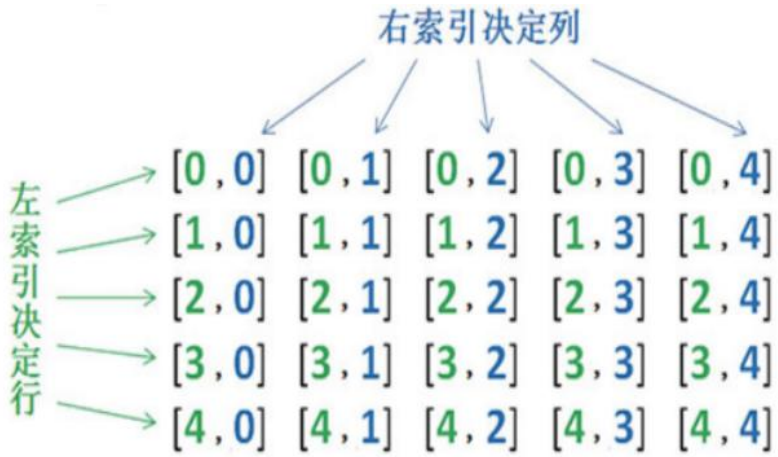
int[ ][ ] a=new int[2][ ]; //定义一个2行的int类型二维数组
a[0]=new int[2]; //初始化二维数组的第0行，有2个元素
a[1]=new int[3]; //初始化二维数组的第1行，有3个元素

内存分配示意图:



二维数组的初始化

- 二维数组有两个索引，构成由行列组成的一个矩阵



- 为单个二维数组元素赋值

（1）首先声明一个二维数组，并指明行数和列数，然后为二维数组中每个元素进行赋值。

```
int[,] myarr = new int[2, 2];
myarr[0, 0] = 0;
myarr[0, 1] = 1;
myarr[1, 0] = 1;
myarr[1, 1] = 2;
```

```
int[,] myarr = new int[2, 2];
for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 2; j++)
    {
        myarr[i, j] = i + j;
    }
}
```

二维数组的初始化

- 为每一维数组元素赋值

（1）使用数组[ ][ ]形式声明一个数组，并指定数组的行数，后再分别为每一维数组元素赋值

（2）例如： int[ ][ ] myarr=new int[2][ ];

myarr[0]=new int[ ] {0,1};

myarr[1]=new int[ ] {1,2};

- 同时为整个二维数组赋值

（1）使用嵌套大括号，将要赋值的数据包含在里层大括号中，每个大括号中间用,隔开。

```
int[,] myarr = new int[2,2] { { 12, 0 }, { 45, 10 } };
```

```
int[,] myarr = new int[,]{ { 12, 0 }, { 45, 10 } };
```

```
int[,] myarr = {{12,0},{45,10}};
```

二维数组的使用

示例2：创建一个控制台应用程序，模拟制作一个简单的客车售票系统，假定客车的座位数是9行4列，使用一个二维数组记录客车售票系统中的所有座位号，并在每个座位号上显示“有票”，然后用户输入一个坐标位置，按下“Enter”键，即可将该座位号显示为“已售”。

实现步骤：

（1）创建二维数组zuo[,]

（2）二维数组初始化为：有票

（3）输出初始化后的二维数组；

（4）创建字符串变量s，表示：行、列

（5）s初值，即为收键盘输入

（6）将（5）中输入数值对应的二维数组修改为：已售

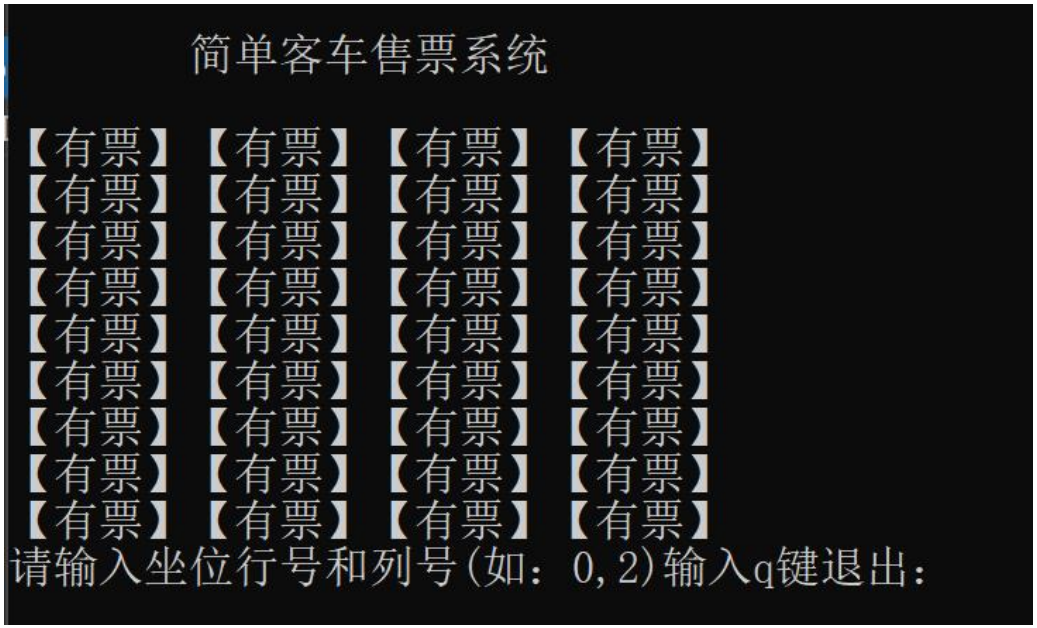
（7）如果输入为”q”,退出

```
static void Main(string[] args)
{
    Console.Title = "简单客车售票系统";                    //设置控制台标题
    string[,] zuo = new string[9, 4];                      //定义二维数组
    for (int i = 0; i < 9; i++)                            //for循环开始
    {
        for (int j = 0; j < 4; j++)                        //for循环开始
        {
            zuo[i, j] = "【有票】";                        //初始化二维数组
        }
    }
```

```
string s = string.Empty;                                   //定义字符串变量
while (true)                                               //开始售票
{
    Console.Clear();                                       //清空控制台信息
    Console.WriteLine("\n          简单客车售票系统" + "\n"); //输出字符串
    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            System.Console.Write(zuo[i, j]);               //输出售票信息
        }
        Console.WriteLine();                               //输出换行符
    }
```

```
    Console.Write("请输入坐位行号和列号(如：0,2)输入q键退出：");
    s = Console.ReadLine();                                    //售票信息输入
    if (s == "q") break;                                              //输入字符串"q"退出系统
    string[] ss = s.Split(',');                                       //拆分字符串
    int one = int.Parse(ss[0]);                                     //得到坐位行数
    int two = int.Parse(ss[1]);                                     //得到坐位列数
    zuo[one, two] = "【已售】";                                     //标记售出票状态
```

运行效果：



数值与ARRAY类

- System.Array类派生而来
- 可以用Array类中的各种属性或者方法对数组进行各种操作。Array类的常用方法如下：

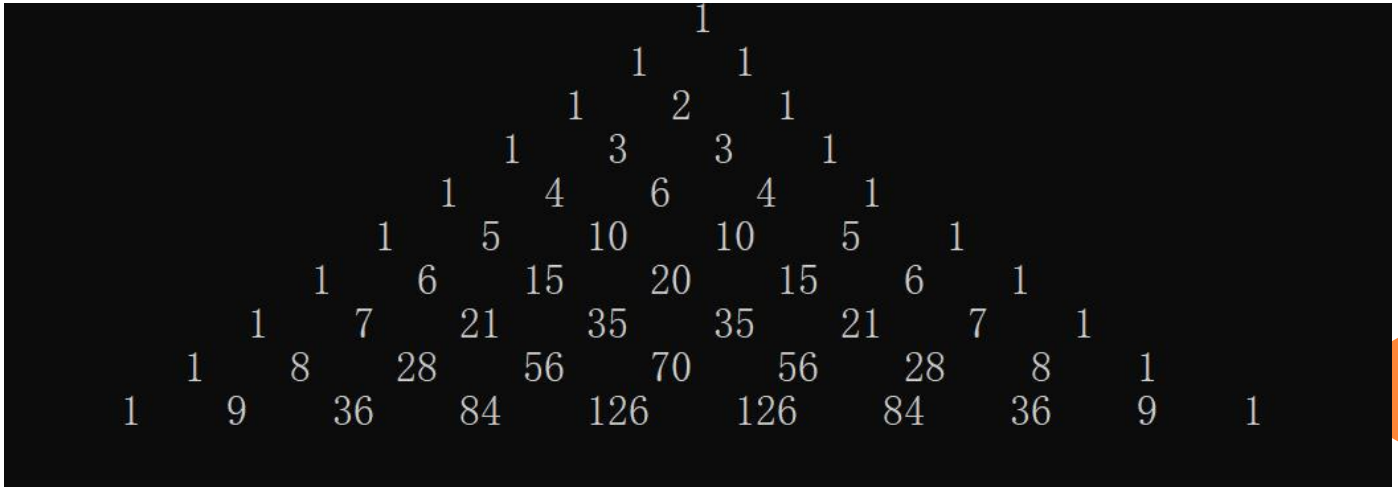
方　法	说　明
Copy	将数组中的指定元素复制到另一个 Array 中
CopyTo	从指定的目标数组索引处开始，将当前一维数组中的所有元素复制到另一个一维数组中
Exists	判断数组中是否包含指定的元素
GetLength	获取 Array 的指定维中的元素数
GetLowerBound	获取 Array 中指定维度的下限
GetUpperBound	获取 Array 中指定维度的上限
GetValue	获取 Array 中指定位置的值
Reverse	反转一维 Array 中元素的顺序
SetValue	设置 Array 中指定位置的元素
Sort	对一维 Array 数组元素进行排序

- 示例3：打印杨辉三角

```
static void Main(string[] args)
{
    int[][] Array_int = new int[10][];                  //定义一个10行的二维数组
    //向数组中记录杨辉三角形的值
    for (int i = 0; i < Array_int.Length; i++)              //遍历行数
    {
        Array_int[i] = new int[i + 1];                  //定义二维数组的列数
        for (int j = 0; j < Array_int[i].Length; j++)           //遍历二维数组的列数
        {
            if (i <= 1)                                 //如果是数组的前两行
            {
                Array_int[i][j] = 1;                    //将其设置为1
                continue;
            }
            else
            {
                if (j == 0 || j == Array_int[i].Length - 1)     //如果是行首或行尾
                    Array_int[i][j] = 1;                        //将其设置为1
                else                                            //根据杨辉算法进行计算
                    Array_int[i][j] = Array_int[i - 1][j - 1] + Array_int[i - 1][j];
            }
        }
    }
```

```
for (int i = 0; i <= Array_int.Length-1; i++)              //输出杨辉三角
{
    //循环控制每行前面打印的空格数
    for (int k = 0; k <= Array_int.Length - i; k++)
    {
        Console.Write("   ");
    }
    //循环控制每行打印的数据
    for (int j = 0; j < Array_int[i].Length; j++)
    {
        Console.Write("{0}    ", Array_int[i][j]);
    }
    Console.WriteLine();
}
Console.ReadLine();
```

运行效果：



数组的基本操作

- **遍历**

（1）使用foreach语句遍历数组

```
foreach(【类型】 【迭代变量名】 in 【集合】)
{
    语句
}
```

（2）示例：输出狼人杀游戏的主要角色

```
static void Main(string[] args)
{
    Console.WriteLine("狼人杀游戏主要身份：");
    string[] roles = { "狼人", "预言家", "村民", "女巫", "丘比特", "猎人", "守卫" };
    foreach(string role in roles)
    {
        Console.Write(role + "");
    }
    Console.ReadLine();
}
```

数组的基本操作

- 排序：**Array.Sort**

(1)Array.Sort：对一维Array中的元素进行排序。常用的两种形式：

```
public static void Sort(Array array)
public static void Sort(Array array,int index,int length)
```

（2）array:要排序的一维Array; index:排序范围的起始索引；length:排序范围内的元素数。

```
int[] arr = new int[] { 3, 9, 27, 6, 18, 12, 21, 15 };
Array.Sort(arr);                                    //对数组元素排序
```

排序：Array.Reverse：

（1）反转一维Array中的元素的顺序。该方法有两种形式：

```
public static void Reverse(Array array)
public static void Reverse(Array array,int index,int length)
```

（2）array:要反转的一维Array; index:反转部分的起始索引；length:要反转的部分中的元素数。

```
int[] arr = new int[] { 3, 9, 27, 6, 18, 12, 21, 15 };
Array.Reverse(arr);                                    //对数组元素反转
```

C#编程基础

任课教师：李娜
邮箱：709233393@qq.com

第五章 字符串

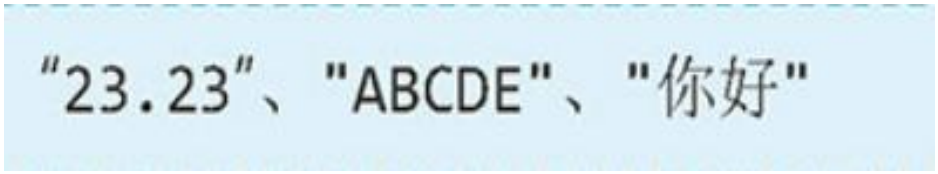
5.1 字符串的声明与初始化

5.2 提取字符串信息

5.3 字符串操作

字符串的声明和初始化

- string类创建字符串，字符串包含在一对双引号“”内



- 字符串声明：string str=[null]

  (1)string：指定该变量为字符串类型；

  (2)str:有效的标识符，表示字符串变量的名称；

  (3)null：省略null，表示str变量是初始化状态，否则，表示字符串的值就等于null

  (4)例如：string strName

字符串的初始化

```
string str;
```

```
string a = "时间就是金钱，我的朋友 ";
string b = "锄禾日当午";
string str1, str2;
str1 = "We are students";
str2 = "We are students";
```

提取字符串信息

- 获取字符串的长度：使用string类的Length属性，语法格式：

```
public int Length { get; }
```

（1）示例：

```
string num = "12345 67890";
int size = num.Length;
```

- 获取指定位置的字符：使用string类的Char属性

```
public char this[
    int index
] { get; }
```

（1）index：当前的字符串中的位置；
（2）属性值：位于index位置的字符

示例：

```
string str = "努力工作是人生最好的投资";          //创建字符串对象str
char chr = str[5];                                //将字符串str中索引位置为5的字符赋值给chr
Console.WriteLine("字符串中索引位置为5的字符是：" + chr); //输出chr
```

```
字符串中索引位置为5的字符是：人
```

- 获取子字符串索引位置：IndexOf和LastIndexOf

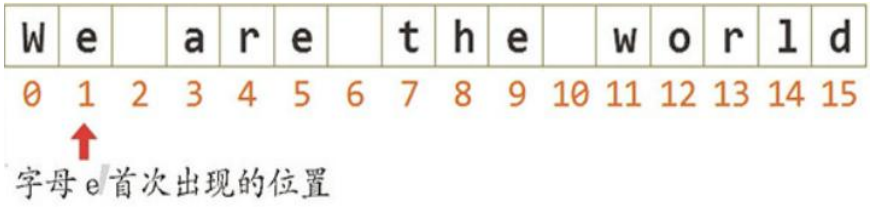
(1)**IndexOf方法**：返回搜索的字符或字符串首次出现的索引位置

```
public int IndexOf(char value)
public int IndexOf(string value)
public int IndexOf(char value,int startIndex)
public int IndexOf(string value,int startIndex)
public int IndexOf(char value,int startIndex,int count)
public int IndexOf(string value,int startIndex,int count)
```

（2）Value：要搜索的字符或字符串；startIndex:搜索起始位置；count:要检查的字符位置数；返回值：找到，返回索引位置；未找到，返回-1

（3）例如：string str="we are the world";

int size=str.IndexOf('e');



(1)Last**IndexOf方法：**返回搜索的字符或字符串最后一次出现的索引位置

```
public int LastIndexOf(char value)
public int LastIndexOf(string value)
public int LastIndexOf(char value,int startIndex)
public int LastIndexOf(string value,int startIndex)
public int LastIndexOf(char value,int startIndex,int count)
public int LastIndexOf(string value,int startIndex,int count)
```

（2）Value：要搜索的字符或字符串；startIndex:搜索起始位置；count:要检查的字符位置数；返回值：找到，返回索引位置；未找到，返回-1

（3）例如：string str="we are the world";

int size=str.LastIndexOf('e');

字符串操作

- 字符串的拼接：使用“+”运算符完成对多个字符串的拼接，生成一个string对象

```
string s1 = "hello";                    //声明string对象s1
string s2 = "world";                    //声明string对象s2
string s = s1 + " " + s2;               //将对象s1和s2连接后的结果赋值给s
```

```
Console.WriteLine("I like
C#");
```

```
Console.WriteLine("I like" +
"C#");
```

比较字符串

- “==”

```
string str1 = "mingrikeji";
string str2 = "mingrikeji";
Console.WriteLine((str1 == str2));
```

- Equal

```
public bool Equals (string value)
public static bool Equals (string a,string b)
```

(1)value:与此 字符串比较的字符串；
(2)a和b：进行比较的两个字符串
(3)返回值：相同，返回true;否则为false

- 练习：

（1）加入某网站密码是你名字的汉语拼音，密码为学号，编程实现用户输入的用户名和密码是否正确

- 字符串的大小写转换： ToUpper方法和ToLower方法

```
string str = "Learn and live";
Console.WriteLine(str.ToUpper());
Console.WriteLine(str.ToLower());
```

- 截取字符串：Substring方法，截取字符串中指定位置和指定长度的字符串。

```
public string Substring(int startIndex)
public string Substring (int startIndex,int length)
```

（1）startIndex：子字符串的起始位置的索引；length:子字符串中的字符数；返回值：截取的子字符串

- 练习：

```
static void Main(string[] args)
{
    string strFile = "Program.cs";                          //定义字符串
    Console.WriteLine("文件完整名称：" + strFile);           //输出文件完整名称
    string strFileName = strFile.Substring(0, strFile.IndexOf('.'));  //获取文件名
    string strExtension = strFile.Substring(strFile.IndexOf('.'));   //获取扩展名
    Console.WriteLine("文件名：" + strFileName);             //输出文件名
    Console.WriteLine("扩展名：" + strExtension);            //输出扩展名
    Console.ReadLine();
}
```

C#编程基础

任课教师：李娜
邮箱：709233393@qq.com

第六章 面向对象程序设计

6.1 概述
6.2 类
6.3 方法
6.4 对象的创建及使用
6.5 继承
6.6多态

概述

- 面向对象（Object Oriented）：英文缩写：OO
- 对象（Object）：指客观世界中存在的对象，该对象具有唯一性，对象之间各不相同,可相互联系。
- 现实世界中对象具有的特征：

（1）每一个对象必须有一个名字以区别其他对象；

（2）用属性来描述对象的某些特征；

（3）有一组操作，每个操作决定对象的一种行为；

（4）对象的操作：自身承受的操作；施加于其他对象的操作。

概述

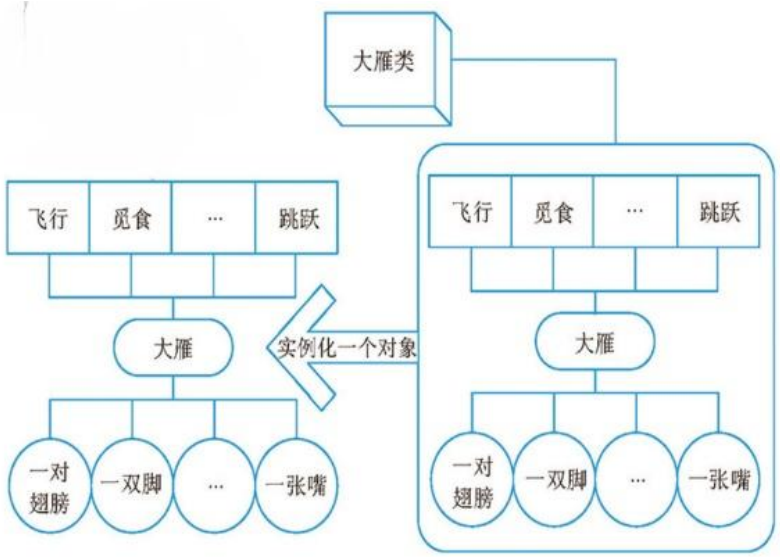
如何用面向对象的思想来解决“大雁从北方飞往南方”的问题？

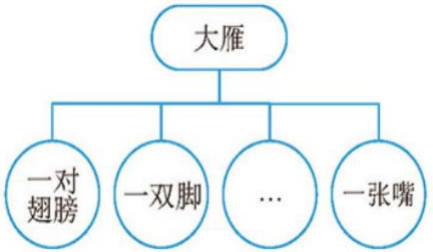
（1）抽象出对象，大雁

（2）确认该对象的静态属性：翅膀、羽毛、嘴巴等

（3）确认对象的动态行为，例如：飞行、觅食等

（4）对象定义完成，封装起来，描述大雁这一类动物





练习

找出下列对象共性：

1、张三（学生）\杨老师\邻居售货员张阿姨\隔壁主播李四

2、门口停放的S320汽车\操场上的大货车\我的凤凰自行车

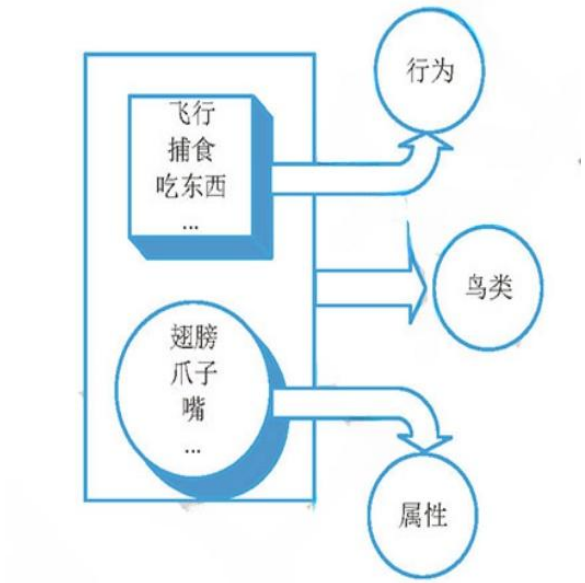
类是模子，确定对象将会拥有的特性(属性)和行为（方法）

类

- 具有相同属性和行为的一类实体被称为类。

（1）在类中，对象的行为以方法的形式定义；对象的属性是以成员变量的形式定义。

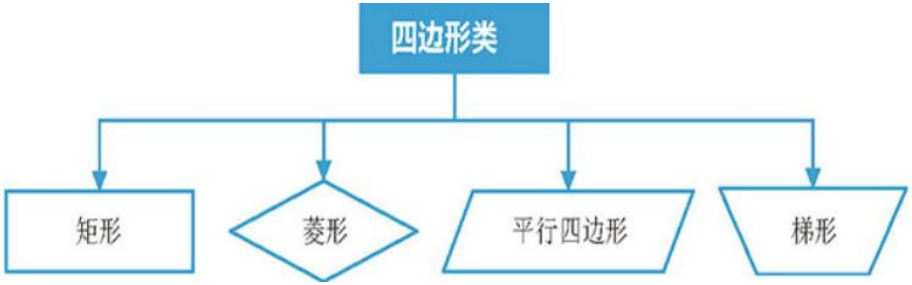
（2）类包括对象的属性和方法。



- 面向对象程序设计的基本特征：封装、继承和多态。

（1）**封装**：将对象的属性和行为封装为类，对用户隐藏实现细节。

（2）**继承**：实现重复利用的手段。子类通过继承复用父类的属性和行为，又添加子类特有的属性和行为。



（3）**多态**：将父类对象应用于子类。

类

- 类的声明：class关键词

```
class 类名
{
}
```

```
class Car
{
}
```

- 语法：

  [public] **class** 类名

  {

  字段；

  属性；

  方法

  }

- 属性的**作用**：保护字段，对字段的赋值和取值进行限定。
- 属性的**本质**：两个方法：get()和set();

- 类的成员：包括字段、属性、方法、构造函数等等

（1）字段：变量或者常量

```
class Person
{
    public string _name;
    public int _age;
    public char _gender;
    1 个引用
    public void Behavior()
    {

        Console.WriteLine("我叫{0},我今年{1}岁了，我是{2}生，我能干很多事情", _name, _age, _gender);
        Console.ReadKey();
    }
}
```

（2）类没法运行，需要创建对象

```
class Program
{
    0 个引用
    static void Main(string[] args)
    {
        Person person = new Person();
        person._name = "张三";
        person._age = 23;
        person._gender = '男';
        person.Behavior();
    }
}
```

- 练习1：

定义一个员工类，在该类中定义员工的姓名及年龄字段，并输出。

类

- 属性：对现实实体特征的抽象，提供对类或对象的访问。 属性的声明语法如下：

```
【权限修饰符】【类型】【属性名】
{
    get  {get访问器体}
    set  {set访问器体}
}
```

（1）修饰符：指定属性的访问级别；

（2）类型：指定属性的类型，可以是任何的预定义或自定义类型；

（3）属性名：一种标识符，命名规则与变量相同，属性的第一个字母经常大写。

（4）get 访问器：相当于一个具有属性类型返回值的无参数方法，**需要用return语句来返回**；

（5）set访问器：相当于一个具有单个属性类型值参数和**void返回类型**的方法。

方法

- 定义类可执行的操作，包含一系列语句的代码块。
- 方法的声明：

  （1）在类或者结构中声明，需要指定**访问级别**、**返回值**、**方法名称**及方法**参数**，方法参数用逗号隔开。

  （2）基本格式：

```
修饰符 返回值类型 方法名(参数列表)
{
    //方法的具体实现;
}
```

  - ✓ 修饰符可以是**private**、**public**、**protected**、**internal**4个中的任意一个；
  - ✓ 返回值类型：指定方法返回数据的类型，可以是任何类型；如果方法不需要返回值，则使用void关键字；
  - ✓ 参数列表：用逗号分割的类型、标识符；如果方法没有参数，则参数列表为空

权限修饰符

- private、protected、internal、protected internal和public

权限修饰符	应用于	访问范围
private	所有类或者成员	只能在本类中访问
protected	类和内嵌类的所有成员	在本类和其子类中访问
internal	类和内嵌类的所有成员	在同一程序集中访问
protected internal	类和内嵌类的所有成员	在统一程序集和子类中访问
public	所有类成员	任何代码都可以访问

- 示例2：通过属性控制年龄的输入范围

类的创建：

```
class Student
{
    private string _name;    //姓名:字段
    3 个引用
    public string Name     //属性
    {
        get {
            return _name;
        }
        set {
            _name = value;
        }

    }
```

```
    private    int _age;       //年龄:字段
    3 个引用
    public int Age      //属性
    {
        get { return _age; }
        set {

            if (value < 0 || value > 100)
            {
                Console.WriteLine("赋值出错，请重新输入");

            }
            else
            {
                _age = value;
            }

        }

    }
```

```
    private    char _gender; //性别: 字段
    3 个引用
    public char Gender    //属性
    {
        get {
            if (_gender != '女' || _gender != '男')
            {
                _gender = '男';
            }
            return _gender;
        }
        set {
            _gender = value;
        }

    }
```

```
    public void Behavior()
    {

        Console.WriteLine("我叫{0},我今年{1}岁了，我是{2}生，我能干很多事情", this.Name,this.Age, this.Gender);
    }
```

红色箭头：字段

绿色箭头：属性

蓝色箭头：方法

对象的创建：

```
namespace MyClassPractice02
{
    0 个引用
    class Program
    {
        0 个引用
        static void Main(string[] args)
        {
            Student per1 = new Student();   //创建一个对象per1
            per1.Name = "Zhagsan";   //给对象属性赋初始
            per1.Age = 120;
            per1.Gender= '女';
            per1.Behavior();   //对象名.方法名


            Student per2 = new Student();   //创建一个对象per2
            per2.Name = "lisi";   //给对象属性赋初始
            per2.Age = 20;
            per2.Gender = '中';
            per2.Behavior();   //对象名.方法名
        }
    }
}
```

创建两个对象，通过属性给对象赋值。

- 练习2：

补充示例中的Student类

（1）添加数学、英语、语文字段、属性；

（2）添加计算平均成绩的方法；

（3）创建对象，输出对象的平均成绩。

静态和非静态---静态方法和非静态方法

方法前面有static关键字，该方法为静态方法。

```
class Person
{
    1 个引用
    public void M1()
    {
        Console.WriteLine("我是非静态的方法");
    }
    1 个引用
    public static void M2()
    {
        Console.WriteLine("我是静态的方法");
    }
}
```

```
static void Main(string[] args)
{
    Person per1 = new Person();    //创建对象    1
    per1.M1();  //实例对象名.方法名    2
    Person.M2();    //类名.方法名    3
```

```
我是非静态的方法
我是静态的方法
```

静态方法：Console.Writeline();Console.ReadLine()等

1、在调用实例成员的时候，需要使用对象名.实例成员；

2、在调用静态成员的时候，需要使用类名.静态成员名

- 示例：

```
class Student02
{
    private static string _name;    //静态字段
    0 个引用
    public static string Name      //静态属性
    {
        get { return Student02._name; }       //使用类名.字段
        set { Student02._name = value; }            //使用类名.字段

    }
    private char _gender;      //字段
    0 个引用
    public char Gender        //属性
    {
        get { return _gender; }
        set { _gender = value; }

    }
    0 个引用
    public void M1()                              //普通方法
    {
        Console.WriteLine("我是非静态方法");
    }
    0 个引用
    public static void M2()                       //静态方法
    {
        Console.WriteLine("我是静态方法");
    }
}
```

静态成员必须使用类名去调用，实例成员使用对象名调用；

普通方法中，既可以使用静态成员，也可以使用实例成员；

静态方法中，只能访问静态成员，不允许访问实例成员；

```
namespace MyClassPractice02
{
    0 个引用
    class Program
    {
        0 个引用
        static void Main(string[] args)
        {

            Student02 per3 = new Student02();      //创建对象per3
            per3.M1();                  //对象名.方法名   （普通方法）
            Student02.M2();             //类名.方法名     （静态方法）
```

入口函数中执行

静态和非静态

- 静态类中只允许有静态成员，不允许出现实例成员。

```
namespace MyClassPractice02
{
    2 个引用
    public static class Student03
    {
        private static string _name;   //静态字段
        0 个引用
        public static string Name     //静态属性
        {
            get { return Student03._name; }      //使用类名.字段
            set { Student03._name = value; }           //使用类名.字段

        }
        0 个引用
        public static void M1()                          //静态方法
        {
            Console.WriteLine("我是非静态方法");
        }
        0 个引用
        public static void M2()                   //静态方法
        {
            Console.WriteLine("我是静态方法");
        }
    }
```

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

            Student03.M1();         //静态类不创建对象，直接使用类名.方法名
```

使用：

- 如果类要充当“工具类”使用，则可以考虑将该类设置为静态类，添加static
- 静态类中在整个项目中资源共享

构造函数

- 创建对象时执行的方法，用来初始化对象的数据成员。构造函数特点如下：
  （1）没有返回值，不能使用void;
  （2）构造函数的名称要与本类的名称相同。
- 定义语法

```
public class Book
{
    public Book()                   //无参数构造方法
    {
    }
    public Book(int args)           //有参数构造方法
    {
        args = 2 + 3;
    }
}
```

（1）public：构造函数修饰符
（2）Book:构造函数的名称；
（3）int args:构造函数的参数

- 创建类：

```
class Student
{
    private string _name;   //姓名:字段
    3 个引用
    public string Name    //属性...
    private    int _age;      //年龄:字段
    3 个引用
    public int Age     //属性...
    private    char _gender; //性别：字段
    3 个引用
    public char Gender   //属性...
    private int _math;  //字段
    2 个引用
    public int Math          //属性...
    private int _english;  //字段
      2 个引用
      public int English...    //属性
      private int _chinese;   //字段
      3 个引用
      public int Chinese...    //属性
```

```
public void Behavior()   //普通方法Behavior()
{
    Console.WriteLine("我叫{0},我{1}了，我是{2}生,我能做很多事情", this.Name, this.Age, this.Gender);
    Console.ReadKey();
}
2 个引用
public Student()              //构造函数
{
    Console.WriteLine("我被调用啦！");
}
0 个引用
public void AverageScore()   //普通方法AverageScore()
{
    Console.WriteLine("平均分是{0}", (this.Math + this.Chinese + this.English) / 3);
    Console.ReadKey();
}
```

创建实例：

```
class Program
{
    0 个引用
    static void Main(string[] args)
    {
        Student stu = new Student();
        stu.Name = "张三";
        stu.Age = 20;
        stu.Gender = '男';
        stu.Math = 90;
        stu.Chinese = 85;
        stu.English = 90;
        stu.Behavior();
        stu.AverageScore();
    }
}
```

Main()函数中，创建对象

执行结果：

```
我被调用啦
我叫张三,我今年20岁了，我是男生，我能干很多事情
平均分是88
```

示例中：

```
Student stu = new Student("张三",20,'女',89,90,92);
```

New关键字的作用：

（1）在内存中开辟一块空间；

（2）在开辟的空间中创建对象；

（3）调用构造函数进行初始化对象。

默认构造函数和有参构造函数

- 定义类时，如果没有定义构造函数，编译器会自动创建一个不带参数的默认构造函数。示例：

```
class Book
{
}
```

- 创建对象：Book book=new Book();

构造函数的重载：

```
public Student(string name,int age,char gender,int chinese,int math,int english)  //构造函数
{
    this.Name = name;
    this.Age = age;
    this.Gender = gender;
    this.Chinese = 0;
    this.English = 0;
    this.Math = 0;
}
1 个引用
public Student(string name, int age, char gender)   //构造函数
{
    this.Name = name;
    this.Age = age;
    this.Gender = gender;

}
```

```
static void Main(string[] args)
{
    Student stu = new Student("张三",20,'男',89,90,92);   1
    Student stu1 = new Student("李四", 21, '女');      2
```

1和2创建2个对象，调用不同的构造函数。

- 讨论：下面代码是否正确?

```
class Book
{
    public string Name { get; set; }
    public Book(string name)
    {
        Name = name;
    }
    void ShowInfo()
    {
        Book book = new Book();
    }
}
```

不正确。类中定义了一个新的构造函数（不论是否有参数），系统自动创建的构造函数都失效。

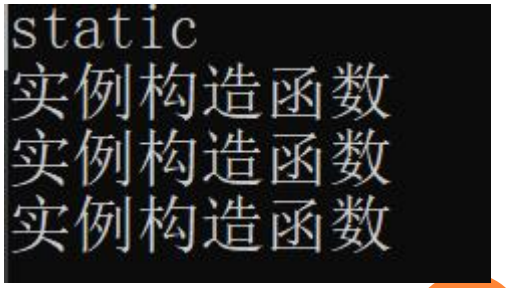
静态构造函数

- 类中有一些静态字段或者属性，需要在第一次使用类之前，从外部初始化这些静态字段和属性。
- 定义静态构造函数时，不能设置访问修饰符，因为其他C#代码不会调用它，只在引用类之前执行一次；
- 不带任何参数，一个类中只能有一个静态构造函数；
- 只能访问类的静态成员

```
static Program()
{
    Console.WriteLine("static");
}
```

示例4:

```
class Program
{
    0 个引用
    static Program()//静态构造函数
    {
        Console.WriteLine("static");
    }
    3 个引用
    private Program()//实例构造函数
    {
        Console.WriteLine("实例构造函数");
    }
    0 个引用
    static void Main(string[] args)
    {
        Program p1 = new Program();      //创建类的对象p1
        Program p2 = new Program();      //创建类的对象p2
        Program p3 = new Program();      //创建类的对象p3
        Console.ReadLine();
    }
}
```



构造函数小结：

- 构造函数用来创建对象，并且可以在构造函数中对对象进行初始化；
- 构造函数是用来创建对象的特殊方法，方法名和类名一样，没有返回值，不使用void；
- 构造函数可以有参数，new对象的时候传递函数参数即可；
- 如果不指定构造函数，则类有一个默认的无参构造函数；
- 如果指定了构造函数，则不再有默认的无参构造函数，如果需要无参构造函数，则需要自己来写；
- 构造函数可以重载，也就是有多个参数不同的构造函数。

析构函数

- 析构函数用来释放对象资源
- 类名加前缀 ~ 来命名

```
~Program()                                    //析构函数
{
    Console.WriteLine("析构函数自动调用");
}
```

- 注意：

  （1）析构函数是自动调用，不需要开发人员显式定义；

  （2）一个类中只能定义一个析构函数

对象的创建及使用

- 面向对象的程序设计语言，所有问题都通过对象来处理。
- 对象可通过操作类的属性和方法解决问题。

对象的创建

- 语法：

```
Test test=new Test();
Test test=new Test("a");
```

- 说明：

参　数	说　明
Test	类名
test	创建 Test 类对象
new	创建对象操作符
"a"	构造函数的参数

- 获取对象的属性和行为：对象名.类成员
- 示例5要求：

（1）创建一个控制台程序，在程序中创建一个cStockInfo类，表示库存商品类；

（2）在该类中定义一个FullName属性和ShowGoods方法；

（3）然后在Program类中创建cStockInfor类的对象，并使用该对象调用其中的属性和方法。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

引入

```
namespace MyClassPractice02
{
    0 个引用
    class cStockInfo
    {
        private string _fullName;
        1 个引用
        public string FullName
        {
            get { return _fullName; }
            set { _fullName = value; }
        }
        0 个引用
        public void ShowGoods()
        {
            Console.WriteLine("库存商品名称：");
            Console.WriteLine(FullName);
        }
```

1、命名空间；
2、类名：cStockInfo
3、属性：FullName
4、类定义的方法：ShowGoods()

```
0 个引用
class Program
{
    0 个引用
    static void Main(string[] args)
    {
        cStockInfo stockInfo = new cStockInfo();
        stockInfo.FullName = "笔记本电脑";
        stockInfo.ShowGoods();
        Console.ReadLine();
    }
}
```

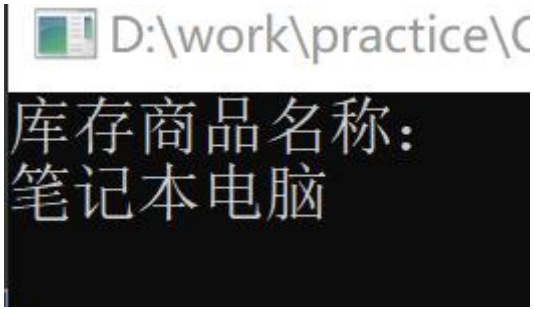
5、创建类：Program

6、程序入口函数

7、创建对象

8、对象的FullName属性赋值

9、调用对象的ShowGoods()方法



- 练习4：

一家商场的促销如下：

满500可享受9折优惠；满1000可享受8折优惠；满2000可享受7折优惠；满3000可享受6折优惠。

使用程序实现计算顾客优惠后的金额。

C# 编程基础

任课教师：李娜
邮箱：709233393@qq.com

- 方法的重载

方法

- 定义类可执行的操作，包含一系列语句的代码块。
- 方法的声明：

  （1）在类或者结构中声明，需要指定**访问级别**、**返回值**、**方法名称**及方法**参数**，方法参数用逗号隔开。

  （2）基本格式：

```
修饰符 返回值类型 方法名(参数列表)
{
    //方法的具体实现;
}
```

方法的参数

- 调用方法时给该方法传递一个或多个值，传给方法的值叫做实参；
- 方法内部，接收实参的变量叫做形参；形参只在方法内部有效。
- C#中方法的参数有4种：值参数、ref参数、out参数和params参数。

  （1）**值参数**：声明时不加修饰的参数，表明实参与形参之间按**值传递**。

  （2）out参数：侧重于在一个方法中可以返回多个不同类型的值

  （3）ref参数：能够将一个变量带入一个方法中进行改变，改变完成后，再将更改后的值传递出去。

  （4）params可变参数：将实参列表中跟可变参数数组类型一致的元素都当作数组的元素去处理。

示例01：

问题：能否只使用一个方法，只是改变参数的个数或者类型？

```
class FunctionOverLoad
{
    1 个引用
    public int M1(int n1, int n2)
    {
        return  n1 + n2;
    }
    1 个引用
    public double M2(double d1, double d2)
    {
        return  d1 + d2;
    }
    1 个引用
    public int M3(int n1, int n2, int n3)
    {
        return n1 + n2 + n3;
    }
    1 个引用
    public double M4(double d1, double d2,double d3)
    {
        return  d1 + d2+d3;
    }
}
```

类中方法的定义
这四个方法中，只有参数个数或类型不同，函数体实现功能相似

```
class Program
{
    0 个引用
    static void Main(string[] args)
    {
        FunctionOverLoad fun = new FunctionOverLoad();
        Console.WriteLine(fun.M1(23, 78));
        Console.WriteLine(fun.M2(23.9, 90.8));
        Console.WriteLine(fun.M3(23, 78, 99));
        Console.WriteLine(fun.M4(23.9, 90.8,1.09));
        Console.ReadKey();
    }
```

对象的创建和方法使用

方法的重载

- 方法名相同，但参数的数据类型、个数或顺序不同的方法。
- 只要类中有两个以上的同名方法，但是使用的参数类型、个数或顺序不同，调用时，编译器即可判断在哪种情况下调用哪种方法。
- 重载方法不能仅仅根据参数是否声明为ref\out或者params来区分。
- 重载方法不能仅在返回值类型上不同。

修改示例01

```
class FunctionOverLoad
{
    1 个引用
    public int M(int n1, int n2)
    {
      return  n1 + n2;
    }
    1 个引用
    public double M(double d1, double d2)
    {
      return  d1 + d2;
    }
    1 个引用
    public int M(int n1, int n2, int n3)
    {
     return n1 + n2 + n3;
    }
    1 个引用
    public double M(double d1, double d2,double d3)
    {
     return  d1 + d2+d3;
    }
```

方法的名称相同，参数的个数或者类型不同

```
class Program
{
    0 个引用
    static void Main(string[] args)
    {
        FunctionOverLoad fun = new FunctionOverLoad();
        Console.WriteLine(fun.M(23, 78));
        Console.WriteLine(fun.M(23.9, 90.8));
        Console.WriteLine(fun.M(23, 78, 99));
        Console.WriteLine(fun.M(23.9, 90.8,1.09));
        Console.ReadKey();
    }
}
```

使用相同的函数名，传递的参数个数或者类型不同

1、
```
static void SayHello(string name)
{
Console.WriteLine("我是{0}",name);
}
static void SayHello(string name)
{
Console.WriteLine("我是{0}",name);
}
```

2、
```
static void SayHello(string name)
{
Console.WriteLine("我是{0}",name);
}
static void SayHello(int age)
{
Console.WriteLine("我的年龄{0},age);
}
```

3、
```
static void SayHello(string name)
{
Console.WriteLine("我是{0}",name);
}
static int SayHello(string name)
{
return 10;
}
```

4
```
static void SayHello(string name)
{
Console.WriteLine("我是{0}",name);
}
static void SayHello(string name0,string name1)
{
Console.WriteLine("我是{0},小名是{1}",name0,name1);
}
```

上面4段示例代码，哪几个属于函数的重载？

示例02：

创建一个控制台应用程序，定义一个Add方法，该方法有3种重载形式，

（1）计算两个int数据的和、

（2）计算一个int和一个double数据的和、

（3）计算3个int数据的和；

（4）在Main方法中分别调用Add方法的3种重载形式，并输出计算结果。

```
class Program
{
    1 个引用
    public static int Add(int x, int y) //定义方法Add，返回值为int类型，有两个int类型参数
    {
        return x + y;
    }
    1 个引用
    public double Add(int x, double y)  //重新方法Add，它与第一个的参数类型不同
    {
        return x + y;
    }
    1 个引用
    public int Add(int x, int y, int z) //重新方法Add，它与第一个的参数个数不同
    {
        return x + y + z;
    }
```

入口函数

```
static void Main(string[] args)
{
    Program program = new Program();    //创建类对象
    int x = 3;
    int y = 5;
    int z = 7;
    double y2 = 5.5;
    //根据传入的参数类型及参数个数的不同调用不同的Add重载方法
    Console.WriteLine("第1种重载形式: " + x + "+" + y + "=" + Program.Add(x, y));
    Console.WriteLine("第2种重载形式: " + x + "+" + y2 + "=" + program.Add(x, y2));
    Console.WriteLine("第3种重载形式: " + x + "+" + y + "+" + z + "=" + program.Add(x, y, z));
    Console.ReadLine();
}
```

练习1:

1、在示例5的基础上，计算两个虚数的和，在Main方法中调用该Add方法，输出计算结果。

C# 编程基础

任课教师：李娜
邮箱：709233393@qq.com

示例1：Student类

```
class Student
{
    public string _name;
    3 个引用
    public string Name...
    public int _age;
    3 个引用
    public int Age...
    public char _gender;
    3 个引用
    public char Gender...
    private int _math;
    2 个引用
    public int Math ...
    private int _english;
    2 个引用
    public int English ...
    private int _chinese;
    2 个引用
    public int Chinese...
    1 个引用
    public Student(string name,int age,char gender,int chinese,int math,int english)  //构造函数...
    1 个引用
    public Student(string name, int age, char gender)   //构造函数...
```

字段和属性

构造函数

```
public void Behavior()
{

    Console.WriteLine("我叫{0},我今年{1}岁了，我是{2}生，我能干很多事情", this.Name,this.Age, this.Gender);
}
1 个引用
public void AverageScore()
{
    Console.WriteLine("平均分是{0}",(this.English+this.Math+this.Chinese)/3);
    Console.ReadKey();
}
```

- Teacher类

```
class Teacher
{
    public string _name;
    1 个引用
    public string Name...
    public int _age;
    1 个引用
    public int Age...
    public char _gender;
    1 个引用
    public char Gender...
    private double _salary;
    0 个引用
    public double Salary...
    0 个引用
    public void Teaching()
    {
        Console.WriteLine("我是老师");
    }
    0 个引用
    public void Behavior()
    {

        Console.WriteLine("我叫{0},我今年{1}岁了，我是{2}生，我能干很多事情", this.Name, this.Age, this.Gender)
    }
}
```

红色大括号：字段和属性
绿色大括号：方法

- Driver类：

```
class Driver
{
    public string _name;
    1 个引用
    public string Name...
    public int _age;
    1 个引用
    public int Age...
    public char _gender;
    1 个引用
    public char Gender...

    private int _driveTime;
    1 个引用
    public int DriveTime...


    0 个引用
    public void Behavior()
    {

        Console.WriteLine("我叫{0},我今年{1}岁了，我是{2}生，我能干很多事情", this.Name, this.Age, this.Gender)
    }
    0 个引用
    public void Driving()
    {
        Console.WriteLine("我已经开车{0}年了",this.DriveTime);
    }
```

红色大括号：字段和属性
绿色大括号：方法

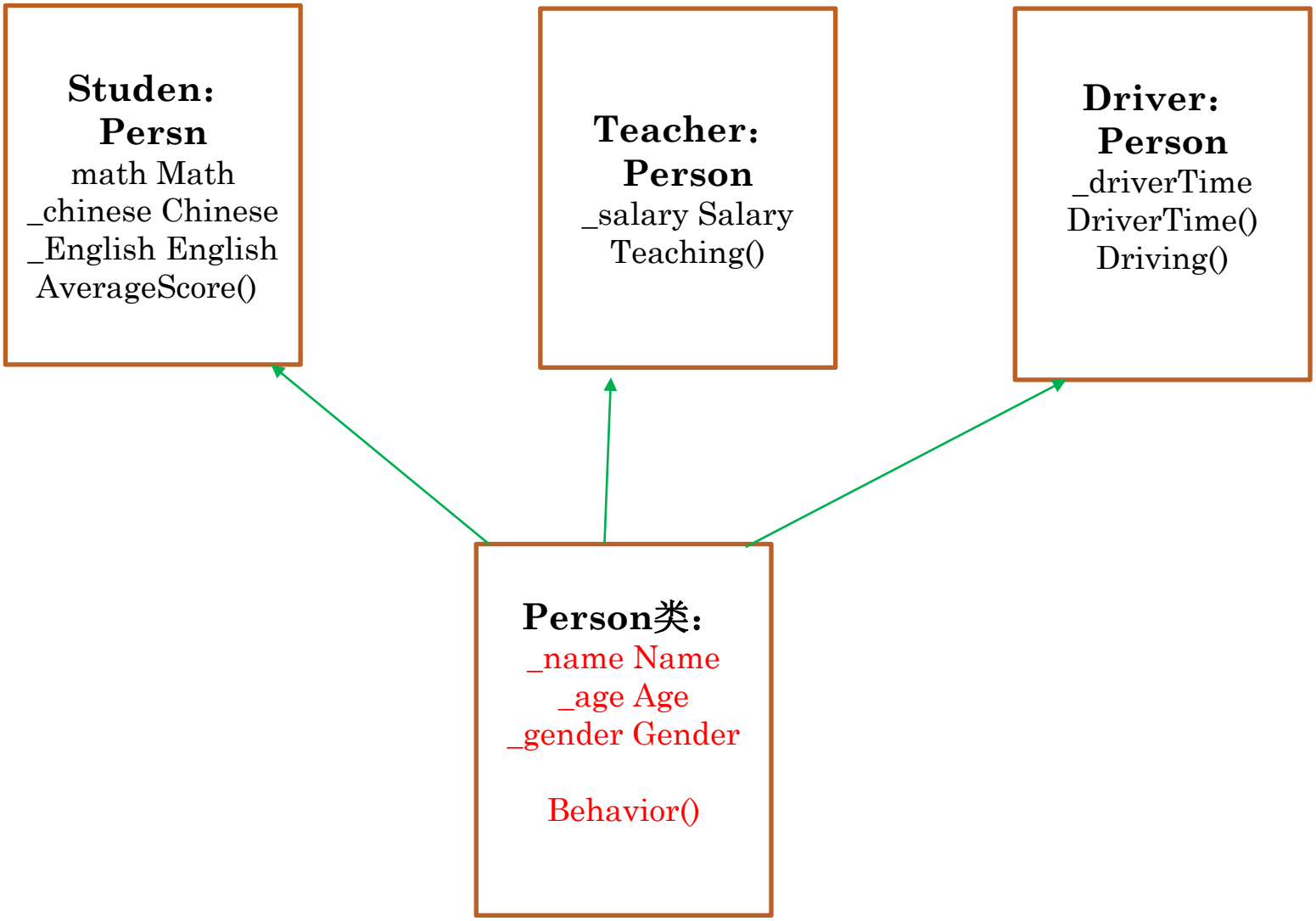
- 比较

Studen类：
_name Name
_age Age
_gender Gender
_math Math
_chinese Chinese
_English English
Behavior()
AverageScore()

Teacher类：
_name Name
_age Age
_gender Gender
_salary Salary
Behavior()
Teaching()

Driver类：
_name Name
_age Age
_gender Gender
_driverTime
DriverTime()
Behavior()
Driving()

- 修改后各个类


Studen:
Persn
math Math
_chinese Chinese
_English English
AverageScore()
Teacher:
Person
_salary Salary
Teaching()
Driver:
Person
_driverTime
DriverTime()
Driving()
Person类:
_name Name
_age Age
_gender Gender
Behavior()

- 创建父类：Person类

```
class Person
{
    public string _name;
    1 个引用
    public string Name...
    public int _age;
    1 个引用
    public int Age...
    public char _gender;
    1 个引用
    public char Gender...
    0 个引用
    public void Behavior()
    {

        Console.WriteLine("我叫{0},我今年{1}岁了，我是{2}生，我能干很多事情", this.Name, this.Age, this.Gender)
    }
}
```

- 修改子类：Student类

```
class Student:Person
{
    private int _math;
    2 个引用
    public int Math ...
    private int _english;
    2 个引用
    public int English ...
    private int _chinese;
    2 个引用
    public int Chinese...
    1 个引用
    public Student(string name,int age,char gender,int chinese,int math,int english)  //构造函数...
    1 个引用
    public Student(string name, int age, char gender)   //构造函数...

    1 个引用
    public void AverageScore()
    {
        Console.WriteLine("平均分是{0}",(this.English+this.Math+this.Chinese)/3);
        Console.ReadKey();
    }
}
```

➢ 由于可以继承父类中的字段、属性和方法，所以将父类中已经存在的该部分代码删掉。

采用类似方法，修改Teacher类和Driver类。

继承

- 在程序设计中实现继承，表示这个类拥有它继承的类的所有公有成员或者受保护成员。

- 在面向对象编程中，被继承的类称为父类或基类，实现继承的类称为子类或派生类。

Student、Teacher、Driver　子类　派生类

Person　父类　基类

继承的实现

- 基类、派生类

  （1）派生类可以继承基类原有的属性和方法，也可增加原基类不具备的属性和方法，或者重写基类中的某些方法。

  （2）：表示继承关系

  （3）public 、protected：父类和派生类能访问public、protected成员，外部代码不能访问protected成员

  （4）子类继承了父类的属性和方法，但是子类并没有继承父类的私有字段

  （5）子类没有继承父类的构造函数，但是子类会默认的调用父类无参数的构造函数，创建父类对象，让子类可以使用父类中的成员。

示例01：

- 创建一个控制台应用程序，模拟实现进销存管理系统的进货信息并输出。
- 提示：

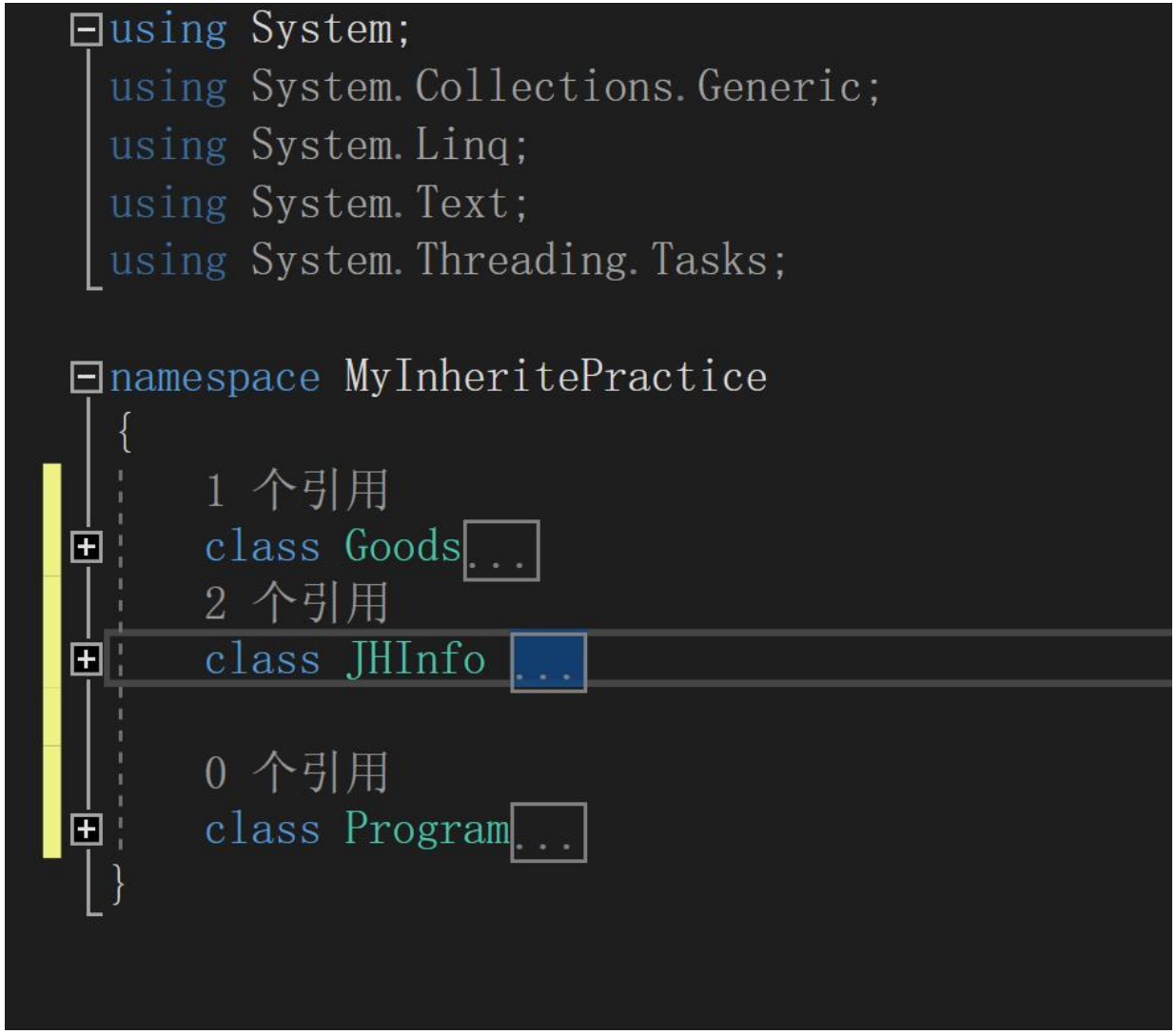
  （1）自定义一个Goods类，该类中定义两个公有属性，表示商品编号和名称；

  （2）定义JHInfo类，继承自Goods类，在该类中定义进货编号属性，以及输出进货信息的方法

  （3）在Program类的Main方法中创建派生类JHInfo的对象，使用该对象调用基类Goods中定义的公有书信那个。

代码实现

- 程序框架：


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace MyInheritePractice
{
1 个引用
class Goods...
2 个引用
class JHInfo ...
0 个引用
class Program...
}

- Goods类：

```
class Goods
{
    2 个引用
    public string TradeCode { get; set; }
    2 个引用
    public string FullName { get; set; }
}
```

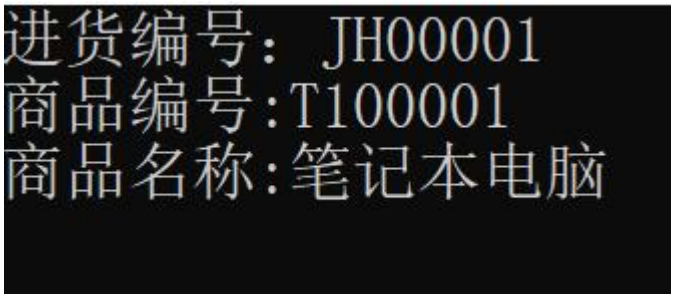
- JHInfo类：

```
class JHInfo : Goods {

    2 个引用
    public string JHID { get; set; }
    1 个引用
    public void showInfo()
    {
        Console.WriteLine("进货编号：{0}\n商品编号:{1}\n商品名称:{2}",JHID,TradeCode,FullName);
    }

}
```

- Program类：

```
class Program
{
    0 个引用
    static void Main(string[] args)
    {
    JHInfo jh = new JHInfo();
    jh.TradeCode = "T100001";
    jh.FullName = "笔记本电脑";
    jh.JHID = "JH00001";
    jh.showInfo();
    Console.ReadLine();
    }
}
```

- 运行结果：



- 练习：定义一个销售类，继承自Goods类，并输出销售信息

- 注意：

（1）C#只支持单继承，不支持多重继承，即在C#中一次只允许继承一个类，不能同时继承多个类。

```
01  class Goods
02  {
03  }
04  class JHInfo : Goods                    //正确：继承单个类
05  {
06  }
07  class Program : Goods, JHInfo           //错误：继承多个类
08  {
09  }
```

错误

- 注意:

（2）实现类的继承时，子类的可访问性一定要低于或者等于父类的可访问性

```
class Goods
{
}
public class JHInfo : Goods
{
}
```

分析：

- ✓ 父类Goods声明时，没有指定访问修饰符，则默认访问级别为private;
- ✓ 子类JHInfo的可访问性public要高于父类Goods的可访问性

- base关键字
  - ✓ 目的：子类的方法中实现父类原有的方法；
  - ✓ this关键字代表本类对象，base关键字代表父类对象；

```
base.property;                    //调用父类的属性
base.method();                    //调用父类的方法
```

注意：

✓ 要在子类中使用base关键字调用父类的属性或者方法，父类的属性和方法必须定义为public 或者protected类型。

- 构造函数的继承问题：

（1）子类没有继承父类的构造函数，但是子类会默认的调用父类无参数的构造函数，创建父类对象，让子类可以使用父类中的成员。

带来的问题：如果在父类中重新写了一个有参数的构造函数之后，无参数的构造函数就没用了，子类就调用不到，此时子类会报错。

- 解决方法：

（1）在父类中重新写一个无参数的构造函数

（2）在子类中显式的调用父类的构造函数，使用关键字：base()

- 父类：Person

```
class Person
{
    private string _name;
    2 个引用
    public string Name...
    private int _age;
    2 个引用
    public int Age...
    private char _gender;
    2 个引用
    public char Gender...
    0 个引用
    public void Behavior()
    {

        Console.WriteLine("我叫{0},我今年{1}岁了，我是{2}生，我能干很多事情", this.Name, this.Age, this.Gender);
    }


    0 个引用
    public Person(string name, int age, char gender)
    {
        this.Name = name;
        this.Age = age;
        this.Gender = gender;
    }
}
```

- 子类：

```
class Student:Person
{
    private int _math;
    2 个引用
    public int Math ...
    private int _english;
    2 个引用
    public int English ...
    private int _chinese;
    2 个引用
    public int Chinese...
    2 个引用
    public Student(string name,int age,char gender,int chinese,int math,int english):base(name,age,gender)
    {
        // this.Name = name;
        //this.Age = age;
        // this.Gender = gender;
        this.Chinese = 0;
        this.English = 0;
        this.Math = 0;
    }
    1 个引用
    public void AverageScore()
    {
        Console.WriteLine("平均分是{0}",(this.English+this.Math+this.Chinese)/3);
        Console.ReadKey();
    }
```

截图(Alt + A)

base():调用父类的构造函数

- 入口Main函数

```
namespace MyClassPractice01
{
    0 个引用
    class Program
    {
        0 个引用
        static void Main(string[] args)
        {
            Student stu = new Student("张三",120,'男',89,90,92);
            stu.Behavior();
            stu.AverageScore();
        }
    }
}
```

- 示例02：

创建一个Reporter类，一个Programmer类，Driver类，他们都有一个SayHellow()函数，

（1）Report的SayHellow()函数体：

大家好，我叫**，我的爱好是**，我的年龄是**，我是*性；

（2）Programmer的SayHellow()函数体：

大家好，我叫**，我今年**岁了，我是*性；已经工作**年了，我的合同期是**年；

（3）Driver的SayHellow()函数体：

我叫**，我的年龄是*，我是*性，我的驾龄是*年。

- 父类：Person

```
class Person
{
    private string _name;
    2 个引用
    public string Name...
    private int _age;
    2 个引用
    public int Age...
    private char _gender;
    2 个引用
    public char Gender...
    0 个引用
    public void Behavior()
    {

        Console.WriteLine("我叫{0},我今年{1}岁了，我是{2}生，我能干很多事情", this.Name, this.Age, this.Gender);
    }


    0 个引用
    public Person(string name, int age, char gender)
    {
        this.Name = name;
        this.Age = age;
        this.Gender = gender;
    }
}
```

- 子类：Reporter

```
class Reporter:Person
{
    private string _hobby;
    2 个引用
    public string Hobby
    {
        get;
        set;
    }
    1 个引用
    public void ReporterSayHello()
    {
        Console.WriteLine("我叫{0}，我是一名记者，我的爱好是{1}，我是{2}生，我今年{3}岁了，",
            this.Name,this.Hobby,this.Gender,this.Age);
    }
    1 个引用
    public Reporter(string name, int age, char gender, string hobby) : base(name, age, gender)
    {
        this.Hobby = hobby;
    }
```

- 子类：Programmer

```
class Programmer:Person
{
    private int _workYear;
    2 个引用
    public int WorkYear {
        get;
        set;
    }
    0 个引用
    public void SayHello()
    {
        Console.WriteLine("我叫{0}，我是一名程序员，我是{1}生，我今年{2}岁了，我的" +
            "合同期是{3}年",this.Name, this.Gender, this.Age, this.WorkYear);
    }
    0 个引用
    public Programmer(string name, int age, char sex, int workYear) : base(name, age, sex)
    {
        this.WorkYear = workYear;
    }
}
```

- 子类：Driver

  自己补充。。。。

- 入口Main()函数：

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

    Reporter rep = new Reporter("zhangsan",34,'男',"摄影");
    rep.ReporterSayHello();
    Console.ReadKey();
```

补充语句：

（1）创建Programmer对象和Driver对象；

（2）调用对应的SayHellow()函数进行输出