

IPL

电子信息学院

武汉大学

数据结构与算法

(C#语言版)

DATA STRUCTURE & ALGORITHM IN C#

第一章 绪论

王文伟 Wang Wenwei, Dr.-Ing.

Tel: 189-71562600

Email: wwwang@aliyun.com

Web: <http://ipl.whu.edu.cn/sites/ced/st>

电子信息学院

Table of Contents

武汉大学

第1章 绪论

第2章 线性表

第3章 栈与队列

第4章 串

第5章 数组和广义表

第6章 树和二叉树

第7章 图

第8章 查找

第9章 排序

本章位置

本章是整个课程的绪论，将讨论数据结构与算法分析中重要的基本概念，如数据、数据类型、数据结构、算法等，以及算法分析的基本方法。此外还将介绍C#程序设计语言的最基本的内容。

IPL

第一章 绪论

2

电子信息学院

Table of Contents

武汉大学

1.0 简介

1.1 数据结构的基本概念

1.2 算法与算法分析

1.3 C#语言简介

IPL

第一章 绪论

3

1.0 Introduction

◆ 软件设计是计算机学科多个领域的核心。软件设计时要考虑的首要问题是数据的表示、组织和处理方法。**数据结构设计和算法设计是软件系统的核心。**

◆ “数据结构+算法=程序”。

Niklaus Wirth:


Data structure

+

Algorithm

=

Program



IPL

第一章 绪论

4

软件开发的过程

系统分析

↓

系统设计

↓

系统实现

↓

系统维护

确定系统所要实现的功能和达到的目标

确定实现并生成系统的方案

编码、调试

系统修整完善

IPL

第一章 绪论

5

数据结构与算法的角色

Niklaus Wirth:

Algorithm

+

Data structure

=

Program

程序设计

• 为计算机处理问题编制一组指令集

算法

• 处理问题的策略

数据结构

• 问题的数学模型

IPL

第一章 绪论

6

1.1 数据结构的基本概念

- 1.1.1 数据类型与数据结构
- 1.1.2 数据的逻辑结构
- 1.1.3 数据的存储结构
- 1.1.4 数据的操作

- ◆ 数据结构是一门讨论“描述现实世界实体的数学模型及其操作在计算机中如何表示和实现”的学科。

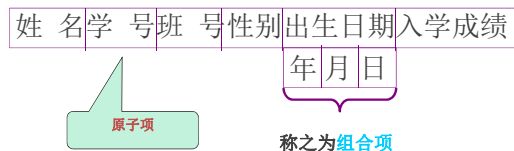
1.1.1 数据类型与数据结构

- ◆ **数据 (data)** :
数据是计算机程序的处理对象, 它可以是任何能输入到计算机的符号集合 (数据是以多种形式呈现的信息), 例如描述客观事物数量特征的数值以及名称特性的字符。
- ◆ **数据元素 (data element)** :
数据的基本单位是数据元素, 它是表示一个事物的一组数据, 有时又称为数据结点。
- ◆ **数据项 (data item/field)** :
构成数据元素的某个成分的数据称作该数据元素的数据项, 数据项是数据元素的基本组成单位。有时又称为数据域 (data field)

数据元素与数据项

描述一个学生的**数据元素**由多个域构成, 其中每个域称为一个“**数据项**”。

数据项是数据结构中讨论的**最小单位**。



高级程序语言中的数据类型

在用高级程序语言编写的程序中, 必须对程序中出现每个变量、常量或表达式, **明确说明**它们所属的**数据类型**。

例如, C语言中的**基本数据类型**有:

字符型 char、**整型** int 和 **实型**(浮点型 float和双精度型 double)

数据类型的定义

- ◆ **数据类型 (data type)** 是一个“**值**”的集合和定义在此集合上的“**一组操作**”的总称。它定义了数据的性质、取值范围以及对数据所能进行的各种操作。

例如, C#语言中整数类型int的值域是 $\{-2^{31}, \dots, -2, -1, 0, 1, 2, \dots, 2^{31}-1\}$, 对这些值进行的操作包括加、减、乘、除、求模、相等或不等比较操作。

数据类型与抽象数据类型

数据类型是一个“**值**”的集合和定义在此集合上的“**一组操作**”的总称。

对程序员而言, 各种语言中的整数类型都是一样的, 因为它们数学特性相同。

从这个意义上可称“**整数**”是一个抽象数据类型。

抽象数据类型和**数据类型**的实质相同, 不局限于编程语言中的数据类型, 范畴更广。

抽象数据类型

数据类型指的是程序设计语言直接支持的数据类型；而**抽象数据类型**是数据与算法在较高层次的描述中用到的概念，是指一个概念意义上的类型和这个类型上的逻辑操作集合。最终演变为在常规数据类型支持下用户新设计的高层次数据类型。

(Abstract Data Type 简称**ADT**)

是指一个**数学模型**以及定义在该模型上的一组操作。

抽象数据类型有两个重要特征：

“数据抽象” 和 “数据封装”

数据抽象与数据封装

用**ADT**描述程序处理的实体时，强调的是数据的**本质特征**、**其所能完成的功能**以及它和**外部世界的接口**（即外界使用它的方法）。- **数据抽象**

将实体的**外部特性**和其**内部实现细节**分离，并且对外部用户**隐藏其内部实现细节**。- **数据封装**

抽象数据类型的描述方法

ADT = (D, S, P)

其中：**D** 是数据对象，

S 是 **D** 上的关系集，

P 是对 **D** 的基本操作集。

数据对象是个特定的数据子集，
(一起)用来表示实体的特性。**字段**

抽象数据类型的描述方法(II)

ADT 抽象数据类型名称{

数据对象：〈数据对象的定义〉

数据关系：〈数据关系的定义〉

基本操作：〈基本操作的定义〉

} ADT 抽象数据类型名称

其中基本**操作**的定义格式为：

基本操作名 (参数表)

初始条件：〈初始条件描述〉

操作结果：〈操作结果描述〉

数据结构

往往要处理很多的数据，而且众多的数据间存在着**内在的联系**。

对一个**数据（元素）的集合**来说，如果在数据元素之间存在一种或多种特定的关系，则称为**数据结构**（data structure）。因此，“结构”就是指数据元素之间存在的**关系**。

数据结构可以看成是关于数据集合的数据类型，是一种特殊的**抽象数据类型ADT**。

数据结构的形式定义

数据结构是关于数据集合的抽象数据类型，它关注三个方面的内容：数据元素的特性、数据元素之间的关系以及由这些数据元素组成的数据集合所允许进行的操作。

数据结构是一个三元组

Data_Structures = (D, S, P)

其中：**D** 是数据元素的有限集，

S 是 **D** 上关系的有限集。

P 是数据集合所允许进行的操作

数据结构两个方面(层次)

逻辑结构 是对数据元素之间的逻辑关系的描述，它可以用一个数据元素的集合和定义在此集合上的若干关系来表示。**侧重于数据集合的抽象特性**

物理结构 是数据结构在计算机中的表示和实现，故又称“**存储结构**”。数据的存储结构则依赖于计算机，它是逻辑结构在计算机中的实现。

TPL

第一章 绪论

19

例1.1：学生和学生信息表

学生类（型）定义

```
class Student{
    string studentID;
    string name;
    string gender;
    int age;
}
```

TPL

第一章 绪论

20

学生信息表

学号	姓名	性别	年龄
200518001	王兵	男	18
200518002	李霞	女	19
200518003	张飞	男	19

学生信息表

逻辑结构：一个班级的学生按学号排列就是**顺序关系**；按班长、组长、组员排列具有**层次关系**；

存储结构：不同学生既可按顺序存储于一个数组(**顺序结构**)，也可分散存储在不同位置但保持链接(**链式结构**)；

TPL

第一章 绪论

21

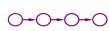
学生信息表是由**Student**类型的数据元素组成的、能够进行特定操作的数据集合，即学生信息表是一种特定类型的数据结构。

```
class StudentInfoTable{
    Student[] studentList;//学生信息表内部存储块
    public int Add(Student st);
        //将新学生添加到表的结尾处
    public bool Contains(Student st);
        //确定某个学生是否在表中
    public void Sort(); //对表中元素进行排序
}
```

1.1.2 数据的逻辑结构

按照数据集中数据元素之间存在的不同特性的逻辑关系，数据结构可以分为三种基本类型：

线性结构



1 to 1

树形结构



1 to n

图状结构



n to n

集合结构



TPL

第一章 绪论

23

定义

◆ **线性结构**：每个数据元素只有一个前驱数据元素和一个后继数据元素，**逻辑上的顺序关系**。



1 to 1

◆ **树结构**：每个数据元素只有一个前驱数据元素，可有零个或若干个后继数据元素，**层次关系**。



1 to n

图结构：每个数据元素可有零个或若干个前驱数据元素，零个或若干个后继数据元素，**网状关系**。



n to n

TPL

第一章 绪论

24

1.1.3 数据的存储结构

◆ 数据的逻辑结构是软件设计人员从逻辑关系的角度观察和描述数据，而为了在计算机中实现对数据的操作，还需要按某种方式在计算机中表示和存储这些数据，这指的是数据的存储结构。

◆ 数据集合在计算机中的存储表示方式称为数据的存储结构，也称为物理结构。有两种基本的存储结构：

➢ 顺序存储结构

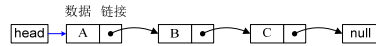
➢ 链式存储结构

用不同方式组合这两种基本存储结构，可以产生复杂的存储结构

◆ 逻辑结构独立于计算机，存储结构则依赖于计算机。

下标 数据

0	A
1	B
2	C



(a) 顺序存储结构

(b) 链式存储结构

顺序存储结构

◆ 顺序存储结构将数据集合的元素存储在一块地址连续的内存空间中，并且逻辑上相邻的元素在物理上也相邻。

以 x 和 y 之间相对的存储位置表示后继关系

例如： y 的存储位置和 x 的存储位置（之间的关系）取决于它们之间的逻辑位置

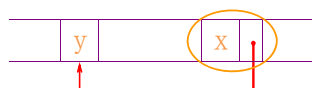


存储结构中只包含元素本身的信息。典型的顺序结构是数组

链式存储结构

链式存储结构使用称为结点（node）的扩展类型存储各个数据元素，结点由数据元素域和指向其他结点的指针域组成，链式存储结构使用指针把相互关联的结点链接起来。逻辑上相邻的数据元素在物理上不一定相邻，数据间的逻辑关系表现在结点的链接关系上。

以附加信息（指针）表示后继关系



数据存储结构的选择

◆ 在软件设计时，既要用正确的逻辑结构描述要解决的问题，还应选择一种合适的存储结构，使得所实现的程序在以下两方面的综合性能最佳：数据操作所花费的时间和程序所占用的存储空间。

◆ 例如，对于线性表的存储，可以按下面两种情况分别处理：

- 当不需要频繁插入和删除时，可以采用顺序存储结构，此时占用的存储空间少。
- 当插入和删除操作很频繁时，需要采用链式存储结构。此时虽然占用的存储空间较多，但操作的时间效率高。这种方案以存储空间为代价换取了时间效率。

1.1.4 数据的操作

◆ 对数据结构进行的某种处理称作数据的操作。

◆ 每种特定的逻辑结构都有一个自身的操作集合，不同的逻辑结构有不同的操作。

- 访问数据元素，更新数据元素值(get/set)。
- 统计数据元素个数(count)。
- 插入数据元素：在数据结构中增加新的结点(insert)。
- 删除数据元素：将指定数据元素从数据结构中删除(remove)。
- 查找：在数据结构中查找满足一定条件的数据元素。插入、删除、更新操作都包括一个查找操作，以确定需要插入、删除、更新数据元素的确切位置(search)。
- 排序：在线性结构中数据元素数目不变的情况下，将数据元素按某种指定的顺序重新排列(sort)。

数据操作的实现与存储结构有关

◆ 操作一般是根据数据的逻辑结构定义的，但操作的实现则与数据的存储结构有关。

◆ 例如对于线性表，选择顺序存储结构还是链式存储结构对于插入或删除操作，都会造成不同的实现方式。

1.2 算法与算法分析

1.2.1 算法

1.2.2 算法设计

1.2.3 算法效率分析

1.2.1 算法

《The Art of Computer Programming》

D.Knuth

算法 (Algorithm) 是对特定问题求解过程的一种描述, 是解决该问题的一个确定的、有限长的操作序列。

一个算法必须满足以下五个重要特性:

1. 确定性
2. 可行性
3. 有穷性
4. 有输入
5. 有输出

算法的特性

确定性 对于每种情况下所应执行的操作, 在算法中都有确切的规定, 算法的执行人或阅读者都能明确其含义, 并且在任何条件下, 算法都只有一条执行路径。

可行性 算法中的所有操作都必须足够基本, 都可以通过已经实现的基本操作运算有限次予以实现。

有穷性 对于任意一组合法输入值, 在执行有穷步骤之后一定能结束。算法中的每个步骤都能在有限时间内完成。

算法的特性(III)

有输入 算法有零个或多个输入数据, 即算法的加工对象。有些输入量需要在算法执行过程中输入, 而有的算法表面上可以没有输入, 实际上输入已被嵌入算法之中。

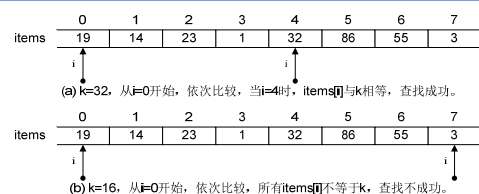
有输出 算法有一个或多个输出数据, 它是算法进行信息加工后得到的结果, 与“输入”有确定关系, 这种关系体现算法的功能。

算法的描述

- ◆ 算法可用文字、流程图、高级程序设计语言或类同于高级程序设计语言的伪码描述。
- ◆ 此时算法是由语义明确的操作步骤组成的有限序列, 它精确地指出怎样从给定的输入信息得到要求的输出信息。
- ◆ 算法的执行人或阅读者都能明确其含义。

【例1.3】线性表的顺序查找算法

在线性表中, 按关键字进行顺序查找的算法思路为: 对于给定值 k , 从线性表的一端开始, 依次与每个元素的关键字进行比较, 如果存在关键字与 k 相同的数据元素, 则查找成功; 否则查找不成功。



算法与数据结构的关系

◆数据的**逻辑结构**、**存储结构**以及对数据所进行的**操作**三者是相互依存的。

- 在研究一种数据结构时，总是离不开研究对这种数据结构所能进行的各种操作，因为，这些操作从不同角度体现了这种数据结构的某种性质，只有通过研究这些操作的算法，才能更清楚地理解这种数据结构的性质。
- 反之，每种算法都是建立在特定的数据结构上的。数据结构和算法之间存在着本质的联系，失去一方，另一方就没有意义。

(1) 同样的逻辑结构因为存储结构的不同而采用不同的算法。

线性表可以用**顺序存储结构**或**链式存储结构**实现，不同存储结构的线性表上的排序算法是不同的。例如，冒泡排序、折半插入排序等算法适用于顺序存储结构的线性表；适用于链式存储结构线性表的排序算法有直接插入排序、简单选择排序等。

(2) 同样的逻辑结构和存储结构，因为要解决问题的要求不同而采用不同的算法。

【例1.4】大规模线性表的**分块查找**（blocking search）算法

顺序查找算法适合于数据量较小的线性表，如学生成绩表。一部按字母顺序排序的**字典**也是一个顺序存储的线性表，具有与学生成绩表相同的逻辑结构和存储结构，但数据量较大，采用顺序查找算法的效率会很低，此时可以采用分块查找算法。

【例1.4】字典的分块查找算法

- ◆字典是由首字母相同、大小不等的若干块（block）所组成的，为使查找方便，每部字典都设计了一个**索引表**，指出每个字母对应单词的起始页码。
- ◆字典分块查找算法的基本思想：将所有单词排序后存放在数组dict中，并为字典设计一个**索引表index**，index的每个数据元素由两部分组成：首字母和下标，它们分别对应于单词的首字母和以该字母为首字母的单词在dict数组中的起始下标。
- ◆使用分块查找算法，在字典dict中查找给定的单词token，必须分两步进行：
 1. 根据token的首字母，查找索引表index，确定token应该在dict中的哪一块。
 2. 在相应数据块中，使用顺序查找算法查找token，得到查找成功与否的信息。

1.2.2 算法设计的要求

正确性

可读性

健壮性

高时间效率

高空间效率

算法的正确性（Correctness）

- ◆**首先**，算法应确切地满足具体问题的需求，这是算法设计的基本目标。
- ◆**其次**，对算法是否“正确”的理解可以有以下四个层次：
 - 不含语法错误；
 - 对于某几组输入数据能够得出满足要求的结果；
 - 程序对于精心选择的、典型、苛刻且带有刁难性的几组输入数据能够得出满足要求的结果；
 - 程序对于一切合法的输入数据都能得出满足要求的结果；

算法的可读性 (Readability)

- ◆ 算法既是为了计算机执行，也是为了人的**阅读与交流**。因此算法应该**易于人的理解**，这既有利于程序的调试和维护，也有利于算法的交流和移植。
- ◆ 相反，晦涩难读的程序易于隐藏较多错误而难以调试；
- ◆ 算法的可读性主要体现在两方面：一是被描述算法中的类名、对象名、方法名等的**命名要见名知意**；二是要有足够多的清晰**注释**。

算法的健壮性 (Robustness)

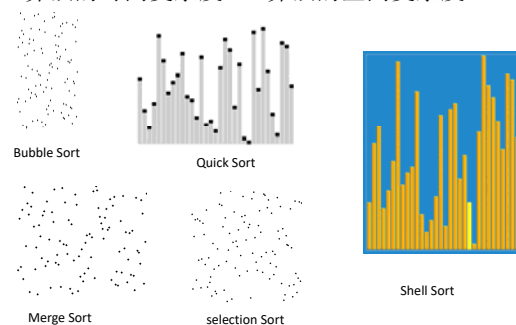
- ◆ 当输入的数据**非法**时，算法应当恰当地作出反应或进行相应处理（**错误或异常处理**），而不是产生莫名其妙的输出结果。
- ◆ **错误或异常处理**：处理出错的方法不应是中断程序的执行，而应是**返回一个表示错误或错误性质的值**，以便在更高的抽象层次上进行处理。

高效性 (Efficiency)

- ◆ 算法的执行时间应满足问题的需求。执行时间短的算法称为**高时间效率**的算法。
- ◆ 算法在执行时一般要求额外的内存空间。内存要求低的算法称为**高空间效率**的算法。
- ◆ 算法应满足高时间效率与低存储量需求的目标，对于同一个问题，如果有多个算法可供选择，应尽可能选择执行时间短和内存要求低的算法。
- ◆ 但算法的高时间效率和高空间效率通常是矛盾的，在很多情况下，首先考虑算法的时间效率目标。

1. 2. 3 算法效率分析

1. 算法的时间复杂度 2. 算法的空间复杂度



1. 算法的时间复杂度

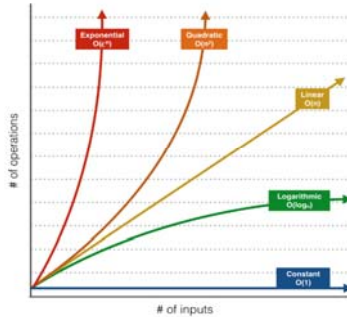
算法的执行时间等于所有语句执行时间的总和，它取决于**控制结构**和**原操作**两者的综合效果。通常选取一种对于所研究的问题来说是基本操作的原操作，以该**基本操作**重复执行的次数作为算法的**时间度量**。

算法的执行时间等于所有语句执行时间的总和，是算法所处理的**数据个数 n** 的函数，表示为 **$O(f(n))$** ，称为该**算法的时间复杂度**（time complexity）。 $O(1)$ 表示时间是一个常数，不依赖于 n ； $O(n)$ 表示时间与 n 成正比，是线性关系， $O(n^2)$ 、 $O(n^3)$ 、 $O(2^n)$ 分别称为平方阶、立方阶和指数阶； $O(\log_2 n)$ 为对数阶。

时间复杂度随 n 变化情况的比较

时间复杂度	$n=8(2^3)$	$N=10$	$n=100$	$n=1000$
$O(1)$	1	1	1	1
$O(\log_2 n)$	3	3.322	6.644	9.966
$O(n)$	8	10	100	1000
$O(n \log_2 n)$	24	33.22	664.4	9966
$O(n^2)$	64	100	10 000	10^6

Big-O Complexity Plot



【例1.5】分析算法片段的时间复杂度

- ◆ 时间复杂度为 $O(1)$ 的简单语句。

```
sum = 0;
```

- ◆ 时间复杂度为 $O(n)$ 的单重循环。

```
int n = 100, sum = 0;
for(int i=0; i<n; i++)
    sum += a[i];
```

【例1.5】分析算法片段的时间复杂度（2）

- ◆ 时间复杂度为 $O(n^2)$ 的二重循环。

```
int n = 100;
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
        Console.WriteLine(i*j);
```

二重循环中的循环体语句
被执行 $n \times n$ 次

```
int n = 100;
for(int i=0; i<n; i++)
    for(int j=0; j<i; j++)
        Console.WriteLine(i*j);
```

二重循环的执行次数为

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$

【例1.5】分析算法片段的时间复杂度（3）

- ◆ 时间复杂度为 $O(n \log_2 n)$ 的二重循环。

```
int n = 64;
for(int i=1; i<=n; i*=2)
    for(int j=1; j<=n; j++)
        Console.WriteLine(i*j);
```

- ◆ 外层循环每执行一次， i 就乘以 2，直至 $i > n$ ，外层循环执行 $\log_2 n$ 次。内层循环执行次数恒为 n 。则时间复杂度为 $O(n \log_2 n)$ 。

【例1.5】分析算法片段的时间复杂度（4）

- ◆ 时间复杂度为 $O(n)$ 的二重循环。

```
int n = 64;
for(int i=1; i<=n; i*=2)
    for(int j=1; j<=i; j++)
        Console.WriteLine(i*j);
```

- ◆ 外层循环执行 $\log_2 n$ 次。内层循环执行 i 次，随着外层循环的增长而成倍递增。此时时间复杂度为 $O(n)$ 。

$$\sum_{i=1}^{\log_2 n} 2^i = O(n)$$

2. 算法的空间复杂度

算法的执行除了需要存储空间来寄存本身所用指令、变量和输入数据外，也需要一些对数据进行操作的工作单元和存储一些为实现算法所需的辅助空间。与算法的时间复杂度概念类似，算法在执行时要求的额外内存空间称为算法的**空间复杂度**（space complexity），也用 $O(f(n))$ 的形式表示。

在冒泡排序过程中，需要一个辅助存储空间来交换两个数据元素，这与序列的长度无关，故冒泡排序算法的空间复杂度为 $O(1)$ 。归并排序算法在运行过程中需要与存储原数据序列的空间相同大小的辅助空间，所以它的空间复杂度为 $O(n)$ 。

1.3 C#语言简介

- ◆ 用单独的课件归纳C#的要点。
- ◆ 用一个多媒体教程学习C#程序设计基础。

“编程之道-新编C#程序设计入门”（洪恩）

- ◆ C#语言是掌握面向对象编程技术的捷径。
- ◆ 深入理解面向对象技术就会拥有站在巨人肩上的阶梯。

TPL

第一章 绪论

55

面向对象技术对数据结构与算法描述的优势

- ◆ 面向过程的程序设计以分离方式描述数据和对数据的操作，所设计的代码往往具有重用性差、可移植性差、数据维护困难等缺点。
- ◆ 数据的逻辑结构、存储结构以及对数据所进行的操作三者实际上是相互依存、互为一体的，所以用封装、继承和多态等面向对象的特性能够更深入地刻画数据结构。
- ◆ 例如，将字符串声明为String类，而串连接、串比较等操作则声明为该类的方法，String类的设计者用面向对象思想设计这个类并实现其中的方法。此时数据的描述和对数据的操作都封装在同一个以类为单位的模块中，因此增强了代码的重用性、可移植性，使数据与代码易于维护。String类的使用者，只需要知道该类对外的接口（即类中的公共方法和属性）即可方便地应用字符串这样一种数据结构。

TPL

第一章 绪论

56

通过实例快速入门C#编程

- ◆ 编程语言本身一般都很简单，但掌握编程却比较繁杂，原因：1）面对的问题非常广泛；2）作为程序运行基础的系统非常繁杂；3）编程思想相对抽象。
- ◆ 通过实例快速体验C#语言的便捷和面向对象编程的威力，开始构建站在巨人肩上的阶梯。

```
// A "Hello World!" program in C#
class HelloWorld{
    static void Main(){
        string[] s = {"C#", "says", "Hello", "World", "!"};
        for (int i = 0; i < s.Length; i++)
            Console.WriteLine("{0}", s[i]);
        Console.WriteLine("at {0}", DateTime.Now);
    }
}
```

TPL

第一章 绪论

59

实例中的几个要点

1. 与C/C++/Java等语言相似的语句、语法；
2. **Main 方法**：C# 程序必须包含一个 Main 方法，程序控制在该方法中开始和结束。在 Main 方法中创建对象和执行其他方法。
3. **类**：所有内容都在某个类中，类的设计和使用是编程的中心问题。用到类Console、String、DateTime和Array：**找到合适的对象做正确的事情**；自定义类HelloWorld：**为现实世界实体和问题建立合适的模型**。
4. **输入和输出**：（Console）WriteLine, Write
5. **注释**：单行注释//, 多行注释/* */, 文档注释///
6. **编译compile和执行execute**

TPL

第一章 绪论

58

IO操作和文件系统-认识 .NET提供的IO类

```
StreamReader infile = File.OpenText("Input.txt");
StreamWriter outfile = File.CreateText("Output.txt");
string tmpstr = null;
Stopwatch timer = new Stopwatch();
timer.Start();
while ((tmpstr = infile.ReadLine()) != null) {
    outfile.WriteLine(tmpstr);
}
timer.Stop();
Console.WriteLine("用时 {0} Ticks", timer.ElapsedTicks);
infile.Close(); outfile.Close();
```

- ◆ 注释： 1) File: StreamReader, StreamWriter。
- 2) using System.IO;
- 3) Stopwatch: using System.Diagnostics;

TPL

第一章 绪论

59

提前看【例3.2】利用栈进行数制转换

$$N = a_n \times d^n + a_{n-1} \times d^{n-1} + \dots + a_1 \times d^1 + a_0$$

数制转换就是要确定序列

$\{a_0, a_1, a_2, \dots, a_n\}$

$$N = (N / d) \times d + N \% d$$

例如：(2468)₁₀ 转换成 (4644)₈ 的运算过程如下：

	N	N / 8	N % 8
计算顺序 ↓	2468	308	4
	308	38	4
	38	4	6
	4	0	4
			输出顺序 ↑

TPL

第一章 绪论

60

利用栈进行数制转换

```
using System; using System.Collections.Generic;
public static void Main(string[] args) {
    int n = 2468;
    Stack<int> s = new Stack<int>(20);
    Console.WriteLine("十进制数: {0} -> 八进制:", n);
    while (n != 0) {
        s.Push(n % 8);
        n = n / 8;
    }
    int i = s.Count;
    while (i > 0) {
        Console.Write(s.Pop()); i--;
    }
    Console.WriteLine();
}
```

要点: 1. 找到合适的对象Stack; 2. 构建实例: new 构造函数(); 3. using指令

泛型(泛型方法和泛型类)

- ◆ 泛型通常与集合以及作用于集合的方法一起使用, 如 List<T>泛型类, Array.Sort<T>(array)泛型方法。
- ◆ C#语言中泛型的优越性在下面的一段例子中应能较好的显示出来。对于同样的运算逻辑(例子中是交换两个变量的内容), 但仅是数据的类型不一样, 可能就需要定义一堆相似的方法或重载的方法; 而应用泛型特性则可仅定义一个泛型方法(例子中是 swap<T>)。

泛型方法

```
static void Main(string[] args) {
    int a = 3; int b = 7;
    swapint(ref a, ref b);
    Console.WriteLine("a={0}, b={1}", a, b);
    double ad = 3.5; double bd = 7.5;
    swapdouble(ref ad, ref bd);
    Console.WriteLine("ad={0}, bd={1}", ad, bd);
    swap<int>(ref a, ref b);
    swap<double>(ref ad, ref bd);
}
static void swapint(ref int a, ref int b) {
    int x = a; a = b; b = x;
}
static void swapdouble(ref double a, ref double b) {
    double x = a; a = b; b = x;
}
static void swap<T>(ref T a, ref T b) {
    T x = a; a = b; b = x;
}
```

声明并构造特定类型的列表举例

```
List<int> a = new List<int> ();
// 声明并构造int型数列表
a.Add(86); a.Add(100);
// 向列表中添加整型元素
List<int> nums = new List<int> {0,1,2,3};
List<string> s = new List<string> ();
// 声明并构造字符串列表
s.Add("Hello"); s.Add("C# 2.0");
var st = new List<Student>();
// 声明并构造学生列表
st.Add(new Student("200518001", "王兵" ));
```

通过实验1 熟悉C#语言 (数组和类的定义)

◆实验目的:

理解C#的基本概念及其基本操作, 重点是数组的处理和新的类型的定义。

◆题意:

1. 定义和初始化数组, 在数组中查找特定数据, 对数组中的数据进行排序。
2. 设计复数类, 实现复数的基本操作。

本章学习要点

- ◆熟悉各名词、术语的含义, 掌握基本概念。
- ◆理解算法五个要素的确切含义。
- ◆掌握计算语句频度和估算算法时间复杂度的方法。

几点要求

- ◆ 课前预习，了解本堂课内容;
- ◆ 课后复习，在理解教学内容的基础上做练习题;
- ◆ 独立完成作业;
- ◆ 加强编程实验：重要的话重复多遍
you haven't really learned something well
until you've taught it to a computer. (Don
Knuth)