# 第 4 章　串

**教学要点**

字符串是数据处理中常见的一种数据类型，特别是在非数值信息处理中，字符串具有广泛的应用。字符串是由多个字符组成的有限序列，可以看成是由若干个仅包含一个字符的结点组成的特殊线性表。字符串可以用顺序存储结构和链式存储结构存储。

本章首先介绍字符串的基本概念与逻辑结构，然后详细讨论以顺序存储结构实现的字符串和以链式存储结构实现的字符串的类型定义和操作实现，分析和比较字符串不同实现的优缺点。

本章在 Visual Studio 中用名为 strings 的类库型项目实现有关数据结构的程序编码，用名为 stringstest 的应用程序型项目实现串类型数据结构的测试和演示程序。

建议本章授课 2 学时，实验 3 学时。

## 4.1　串的概念及类型定义

字符串一般简称为串（string），它是由多个字符组成的有限序列。字符串是非数值信息处理的基本对象，具有广泛的应用。计算机中央处理器的指令系统一般都包含支持基本的数值操作的指令，而对字符串数据的操作一般需用算法来实现，有的高级程序设计语言提供了某种字符串类型及一定的字符串处理功能。

字符串是由多个字符组成的有限序列，可以看成是由若干个仅包含一个字符的数据结点组成的特殊线性表，字符串可以用顺序存储结构和链式存储结构实现。

### 4.1.1　串的定义及其抽象数据类型

串是由 $n$（$n \geq 0$）个字符 $a_0$，$a_1$，$a_2$，…，$a_{n-1}$ 组成的有限序列，记作：

$$\text{String} = \{ a_0, a_1, a_2, \cdots, a_{n-1} \}$$

其中，$n$ 表示串的字符个数，称为串的长度。若 $n$=0，则称为空串，空串不包含任何字符。

串中所能包含的字符依赖于所使用的字符集，为了能处理包括中文在内的字符，C#采用 16 位 Unicode 编码，而非 8 位的 ASCII 编码。C#中用单引号将字符括起来，而用双引号将字符串括起来。例如，

```
s1 ＝“C#”                          串长度为 2
s2 ＝ “data structure in C#”       串长度为 20
s3 ＝“”                            空串，长度为 0
s4 ＝ “  ”                         两个空格的串，长度为 2
```

在上面的例子中，s1, s2, s3 和 s4 分别是四个字符串变量的名字，简称串名。

字符串类型的数据作为一种特殊的线性结构，可以如同一般线性表一样采用顺序存储结构和链式存储结构来实现，在不同类型的应用中，要根据具体的情况，使用合适的存储结构处理字符串数据。

1. 字符及字符串的比较

每个字符根据所使用的字符集会有一个特定的编码，最常用的字符集编码是 8 位的 ASCII 码。为了能处理包括常用汉字在内的字符，C#采用 16 位的 Unicode 编码。不同的字符在字符集编码中是按顺序排列编码的，字符可以按其编码次序规定它的大小，因此两个字符可以进行比较，例如：

‘A’ < ‘a’　　比较结果为 true

‘9’ > ‘A’　　比较结果为 false

对于两个字符串的比较，则按串中字符的次序，依次比较对应位置字符的大小，从而决定两个串的大小，例如：

“data”<“date”　　比较结果为 true

在 C#语言中，char 类型和 string 类型都是可比较的（comparable）。

2. 子串及子串在主串中的序号

由串的所有字符组成的序列即为串本身，又称为主串，而由串中若干个连续的字符组成的子序列则称为主串的一个子串（substring）。一般作如下规定：空串是任何串的子串；主串 s 也是自身的子串。除主串外，串的其他子串都称为真子串。例如，串 s1“C#”是串 s2“data structure in C#”的真子串。

串 s 中的某个字符 c 的位置可用其在串中的位置序号整数表示，称为字符 c 在串 s 中的序号（index）。串的第一个字符的位置序号为 0。一种特殊情况是，如果串 s 中不包含字符 c，则称 c 在 s 中的序号为 -1。

子串的序号是该子串的第一个字符在主串中的序号。例如，s1 在 s2 中的序号为 19。一种特殊情况是，如果串 sub 不是串 mainstr 的子串，则称 sub 在 mainstr 中的序号为-1。

3. 串的基本操作

串的基本操作有以下几种：

- Initialize: 初始化。预分配一定的存储空间，建立一个空串。
- Length: 求长度。返回串的长度，即串包含的字符的个数。
- Empty/Full: 判断串状态是否为空或已满。
- Get/Set: 获得或设置串中指定位置的字符值。
- Concat: 连接两个串。
- Substring: 求满足特定条件的子串。
- IndexOf: 查找字符或子串。

还可以为串定义许多其他操作，如插入，删除和替换等，这些操作都可以用上述基本操作来实现。

### 4.1.2 C#中的串类

为了支持字符串数据类型的处理，.NET Framework 的类库中定义了两个串类：类 String 和类 StringBuilder。String 类定义在 System 命名空间中，用于一般的文本表示，它提供了字符串的定义和操作。C#预定义的关键字 string 类型是 System.String 类的简化的别名，使用 string 与使用 System.String 是相同的，反之亦然。

一个 String 对象一旦创建就不能再修改其内容，所以称 String 对象是恒定的。.NET Framework 的类库在 System.Text 命名空间中还定义了一个串类 StringBuilder，此类表示值为可变字符序列的对象，即 StringBuilder 类型的对象在创建后可以通过追加、移除、替换或插入字符而对它的内容进行修改。

String 类具有如下成员（属性和方法）实现串的各种操作：

**公共构造函数**

| | |
|---|---|
| String(char[ ]); | 初始化 String 类的新实例 |
| String(char[] cs, int startIndex, int length); | |
| String(char c, int count); | |

**公共属性**

| | |
|---|---|
| int Length {get;} | 获取串中的字符个数 |
| char this[int index] {get;} | 获取串中位于指定位置的字符 |

**公共方法**

| | |
|---|---|
| static int Compare(string strA, string strB); | 比较两个指定的 String 对象 |
| int CompareTo(string strB); | 将此实例与指定的 String 对象进行比较 |
| static string Concat(object a); | 创建指定对象的 String 表示形式 |
| int IndexOf(char c); | 返回指定字符在串中的第一个匹配项的索引 |
| int IndexOf(string str); | 返回指定串在此实例中的第一个匹配项的索引 |
| string Insert(int startIndex, string s); | 在指定位置插入指定的 String 实例 |
| string Remove(int startIndex, int count); | 从指定位置开始删除指定数目的字符 |
| string Replace(string oldstr, string newstr); | 将指定子串的所有匹配项替换为指定子串 |
| string Substring(int startIndex, int length); | 返回从指定位置开始且具有指定长度的子字符串 |

String 类还有一些其他的公共方法实现串的各种操作，请参见 C#相关手册中关于 String 的说明。StringBuilder 类所具有的属性和方法与 String 类类似，其使用也请参见 C#相关手册。本章设计的字符串类的特性类似于 StringBuilder 类。

【例 4.1】 从身份证号码中提取出生年月日信息

```
using System;
namespace stringstest {
    public class SubStringTest {
        public static void Main(string[] args){
          string id = "420100199012311234";
          int y = int.Parse(id.Substring(6, 4));
          int m = int.Parse(id.Substring(10,2));
          int d = int.Parse(id.Substring(12,2));
          DateTime dt = new DateTime(y,m,d);
          Console.WriteLine(dt.Year);
          Console.WriteLine(dt.Month);
          Console.WriteLine(dt.Day);
        }
    }
}
```

程序运行结果如下：

```
1990
12
31
```

# 4.2 串的顺序存储结构及其实现

## 4.2.1 串的顺序存储结构的定义

字符串的顺序存储结构是指用一个占据连续存储空间的数组来存储字符串的内容，串中的字符依次存储在数组的相邻单元中。用顺序存储结构实现的字符串称作顺序串，顺序串结构如图 4.1 所示。

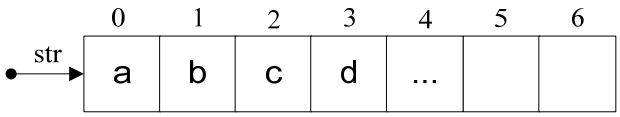



图 4.1 串的顺序存储结构

在图 4.1 中，串 str = “abcd”，它的长度为 4，但这个串预分配的存储空间有 7 个单元。所以在顺序串的实现中，需要一个计数器变量记载实际存入字符数组中的字符个数，即串的实际长度。

串的顺序存储结构与第二章介绍的顺序表相似，只是数据元素的类型不同而已，因此其优缺点亦与顺序表相似。优点是数据结点存储密度高，缺点是必须为数组预分配一定容量的存储空间。如果预分配的容量不够，则在执行过程中需重新分配更大空间的数组；反之，如果预分配的容量过大，则可能造成内存资源的浪费。

我们用如下的 SequencedString 类来实现串的顺序存储结构，该类型声明如下：

```
public class SequencedString {
    private char[] items;                    //用字符数组存储串
    private int count = 0;                   //记载串的长度
    ……
}
```

类中的成员变量有 items 和 count。成员变量 items 为一字符数组，将用以存储串的内容。成员变量 count 记录串的长度。用 SequencedString 类定义的对象就是一个字符串的实例。

## 4.2.2 串的基本操作的实现

串的基本操作将作为 SequencedString 类的属性和方法予以实现，下面分别描述实现这些操作的算法。

1) 串的初始化

使用构造方法创建并初始化一个串对象：它为 items 数组申请指定大小的存储空间，将用来存放字符串的数据；设置串的初始长度为零。多种形式的构造方法编码如下：

```
// 构造n个存储单元的空串
public SequencedString(int n) {
    items = new char[n];
    count = 0;
}
```

```
// 构造16个存储单元的空串
public SequencedString (): this(16){    }

//构造包含一个指定字符的串
public SequencedString (char c): this(16){
    items[0] = c;
    count++;
}

//以一个字符数组构造串
public SequencedString (char[] c): this(c.Length*2){
    Array.Copy(c,items,c.Length);                    //复制数组
    count = c.Length;
}
```

2) 获取串的长度

该操作告知串实例中所包含的字符的个数。将该操作以类的属性成员 Length 来实现，相对于将它定义为成员方法显得更简洁。编码如下：

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

3) 判断串状态是否为空或已满

将这两个测试操作分别用相应的属性（Empty/Full）来实现。当 count 等于 0 时，表明串为空状态，Empty 属性应返回 true 值，编码如下：

```
public bool Empty{
    get{ return count==0;}
}
```

当 count 等于 items.Length 时，表明串为满状态，Full 属性应返回 true 值，编码如下：

```
public bool Full{
      get{ return count == items.Length; }
}
```

4) 获得或设置串的第 i 个字符值

将这两个操作通过定义一个读写型索引器成员予以实现，它提供对串实例进行类似于数组的访问方式。就像 C#的数组下标从 0 开始一样，我们用从 0 开始的索引参数 *i* 来指示串中字符的位置。

```
  public char this[int i]{
      get{
          if(i>=0 && i<count)
              return items[i];
          else
              throw new IndexOutOfRangeException(
                  "Index Out Of Range Exception in " + this.GetType() );
      }
```

```
        set{
            if(i>=0 && i<count) {
                items[i] = value;
            } else{
                throw new IndexOutOfRangeException(
                    "Index Out Of Range Exception in " + this.GetType() );
            }
        }
    }
```

5) 连接一个串与一个字符

方法 Concat(char c) 将指定的字符 c 加入串对象的尾部。当串内部的数组 items 预分配的空间还未满时，将数组单元 items[count]的内容设置为字符 c，计数器 count 自加 1。如果串当前分配的存储空间已装满，在进行后续的操作前，需要调用 DoubleCapacity 方法重新分配更大的存储空间，并将原数组中的字符数据逐个拷贝到新数组。相应的编码如下：

```
public void Concat(char c) {
    if (Full)                              //串满扩容
        DoubleCapacity();
    this.items[count] = c;
    count++;
}

private void DoubleCapacity() {
    int len = Length;
    int capacity = 2 * items.Length;
    char[] copy = new char[capacity];
    for (int i = 0; i < len; i++)
        copy[i] = items[i];
    items = copy;
}
```

6) 连接两个串

方法 Concat(StringObject) 连接两个串实例的内容，依次将参数指定的串的每个字符连接到当前串对象。也可通过对运算符‘+’关于类 SequencedString 重载，连接两个串 str1 和 str2。这样两个串的连接操作可以表示为：

str1.Concat(str2);

或者表示为：

str3 = str1+ str2;

相应的实现编码如下：

```
public void Concat(SequencedString s2) {
    if (s2 != null) {
        for (int i = 0; i < s2.Length; i++)
            this.Concat(s2[i]);
```

```
        }
    }

    public static SequencedString operator +(SequencedString s1, SequencedString s2) {
        SequencedString newstr = new SequencedString(s1.Length + s2.Length + 8);
        newstr.Concat(s1);
        newstr.Concat(s2);
        return newstr;
    }
```

7) 获取串的子串

Substring 方法返回当前串实例中从序号 *i* 开始的长度为 *n* 的子串。当前串对象的长度为 this.Length，*i* 与 *n* 应满足 0≤*i*<*i*+*n*≤this. Length，否则返回空串。

```
    public SequencedString Substring(int i, int n) {
        int j = 0;
        if (i >= 0 && n > 0 && (i + n <= this.Length)) {
            SequencedString sub = new SequencedString(n * 2);
            while (j < n) {
                sub.items[j] = this.items[i + j];
                j++;
            }
            sub.count = j;
            return sub;
        }
        else {
            return null;
        }
    }
```

8) 查找子串

IndexOf 方法在当前串实例中查找与参数 sub 指定的串有相同内容的子串，若查找成功，返回子串的序号，即子串在主串中首次出现时第一个字符的序号；如果当前串不包含子串 sub，则返回-1。

子串的查找算法描述如图 4.2 所示。

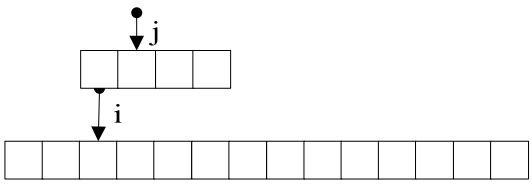



图 4.2 子串的查找

相应的编码如下：

```
    public int IndexOf(SequencedString sub) {
        int i = 0, j; bool found = false;
```

```
        if (sub.Length == 0) return 0;
        while (i <= count - sub.Length) {
            j = 0;
            while (j < sub.Length && this.items[i + j] == sub[j]) j++;
            if (j == sub.Length) { found = true; break; }
            else i++;
        }
        if (found)
            return i;
        else
            return -1;
    }
```

9) 输出串

Show 方法将字符串对象的内容显示在控制台，ToCharArray 方法将串对象的内容转化为一个字符数组，而重写（override）的 ToString 方法将这里定义的字符串类型转换为 C#内在的 string 类型。这两个辅助方法对于一个完整的类型定义是非常有用的。相应的编码如下：

```
    public void Show() {
        for (int i = 0; i < count; i++) {
            Console.Write(items[i]);
        }
        Console.WriteLine();
    }

    public char[] ToCharArray() {
        char[] temp = new char[count];
        for (int i = 0; i < count; i++)
            temp[i] = items[i];
        return temp;
    }

    public override string ToString() {
        string s = new string(ToCharArray());
        return s;
    }
```

### 4.2.3 串的其他操作的实现

对字符串的处理，除了需要前面实现的几种基本操作外，经常还需要插入、删除、替换、逆转等其他操作，这些操作都建立在基本操作之上，因此可以通过组合调用前面的基本操作来实现。

1) 串的插入

在字符串的指定位置插入另一个串，方法签名如下：

```
    public SequencedString Insert(int i, SequencedString s2);
```

它将参数 s2 代表的串插入到当前串实例的位置 *i* 处，*i* 应满足条件 0≤*i*≤this.Length。该方法的实现算法描述如下：

- 用 Substring 操作将当前串分成两个子串，前 *i* 个字符组成子串 sub1，后 Length–i 个字符组成子串 sub2。
- 再用 Concat 操作将 sub1、s2 和 sub2 依次连接起来构成一个新串 newstr。

串的插入算法描述如图 4.3 所示。

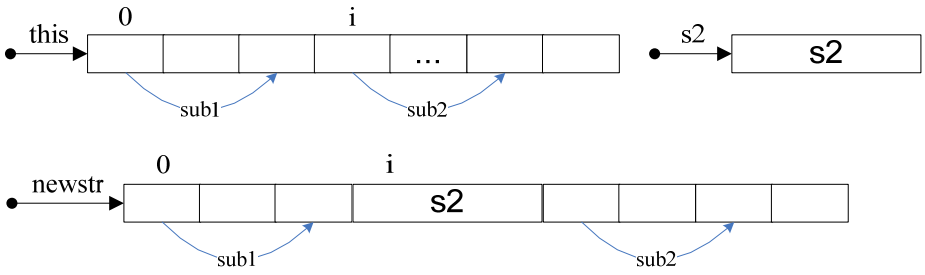



图 4.3 串的插入

串的插入算法的完整实现代码如下：

```
public SequencedString Insert(int i, SequencedString s2) {
    SequencedString sub1, sub2;
    sub1 = this.Substring(0, i);
    sub2 = this.Substring(i, this.Length - i);
    SequencedString newstr = new SequencedString(items.Length + s2.Length + 8);
    newstr.Concat(sub1);
    newstr.Concat(s2);
    newstr.Concat(sub2);
    return newstr;
}

// 将c插入到主串第i位置处
public SequencedString Insert(int i, char c) {
    SequencedString sub1, sub2;
    sub1 = this.Substring(0, i);
    sub2 = this.Substring(i, this.Length - i);
    SequencedString newstr = new SequencedString(items.Length + 8);
    newstr.Concat(sub1);
    newstr.Concat(c);
    newstr.Concat(sub2);
    return newstr;
}
```

2) 串的删除

删除串中指定位置开始的一段子串，方法签名如下：

```
public SequencedString Remove(int i, int n);
```

它删除当前串实例中从位置 *i* 开始的长度为 *n* 的子串，*i* 和 *n* 应满足条件 0≤*i*≤*i*+*n*≤this.Length。该方法的实现算法描述如下：

- 用 Substring 操作将当前串分成三个子串 sub1、sub2 和 sub3，前 *i* 个字符组成 sub1，从第 *i* 个字符开始的长度为 *n* 的子串 sub2，后 this.Length – *i* - *n* 个字符组成 sub3。
- 用 Concat 操作将 sub1 和 sub3 依次连接起来构成一个新串 newstr。

串的删除算法描述如图 4.4 所示。

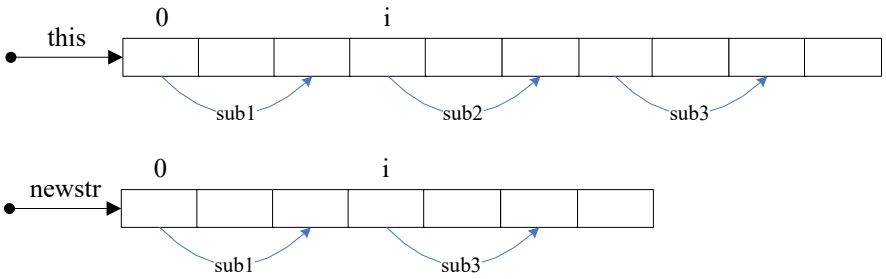



图 4.4 串的删除

串的删除算法的完整实现代码如下：

```
public SequencedString Remove(int i, int n) {
    SequencedString sub1, sub2, sub3;
    sub1 = this.Substring(0, i);
    sub2 = this.Substring(i, n);
    sub3 = this.Substring(i + n, this.Length - i - n);
    SequencedString newstr = new SequencedString(items.Length);
    newstr.Concat(sub1);
    newstr.Concat(sub3);
    return newstr;
}
```

3) 串的替换

将串中指定的子串（它在主串中的首次出现）替换成新的子串，方法签名如下：

```
public SequencedString Replace(SequencedString oldsub, SequencedString newsub);
```

它将当前串实例中 oldsub 子串的首次出现替换成 newsub 子串。该方法的实现算法描述如下：

- 用 IndexOf 操作找到 oldsub 子串在当前串实例中的位序 *i*。
- 用 Substring 操作将当前串实例分成三个子串 sub1、sub2 和 sub3，前 *i* 个字符组成子串 sub1，中间的子串 sub2 与参数 oldsub 串相同，它之后的子串组成 sub3。
- 用 Concat 操作将 sub1，newsub 和 sub3 依次连接起来构成一个新串 newstr。

串的替换算法描述如图 4.5 所示。

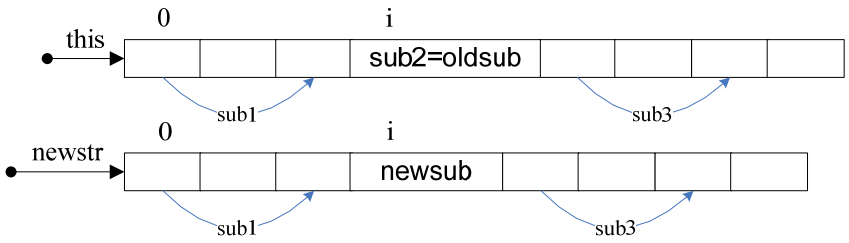



图 4.5 串的替换

串的替换算法的完整实现代码如下：

```
public SequencedString Replace(SequencedString oldsub, SequencedString newsub) {
    int i, n;
    SequencedString sub1, sub3;
    SequencedString newstr = new SequencedString(items.Length + newsub.Length);
    i = this.IndexOf(oldsub);
    if (i != -1) {
        sub1 = this.Substring(0, i);
        n = oldsub.Length;
        sub3 = this.Substring(i + n, this.Length - i - n);
        newstr.Concat(sub1);
        newstr.Concat(newsub);
        newstr.Concat(sub3);
    }
    return newstr;
}
```

4) 串的逆转

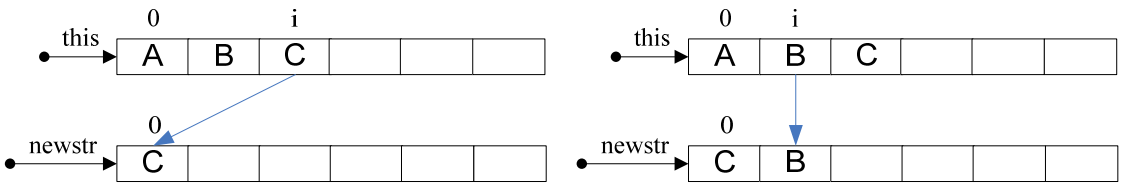
将串中字符序列逆转，方法签名如下：

```
public SequencedString Reverse();
```

逆转算法描述如下：

- 初始化 newstr 为空串。
- 初始设 *i* 为原串最后一个字符的位置
- 进入循环，循环次数为串的长度。
  - 取得串中的第 *i* 个字符 *c*。
  - 用 Concat 操作将字符 *c* 连接到串 newstr 之后，*i* 自减 1。

串的逆转算法描述如图 4.6 所示。




(a) 第一次循环　　(b) 第二次循环

图 4.6 串的替换

串的逆转算法的完整实现代码如下：

```
public SequencedString Reverse() {
    int i;
    SequencedString newstr = new SequencedString(items.Length);
    for (i = this.Length - 1; i >= 0; i--) {
        newstr.Concat(this.items[i]);
    }
    return newstr;
}
```

## 4.3. 串的链式存储结构及其实现

串的链式存储结构就是用一个链表的方式来存储串的内容。链式串的一种简单的实现方式是，链表的每个结点容纳一个字符，并指向后一个字符结点。在建立链式串时，按实际需要分配存储，即在运行过程中动态地分配结点，每个结点的值是一个字符，串的链式存储结构如图 4.7 所示。

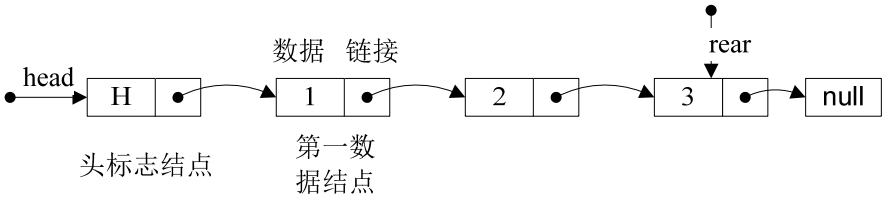



图 4.7 串的链式存储结构

图 4.7 中，串 s =“123”，其内容用单向链表存储，其长度为 3，相应的链表有 1 个头结点和 3 个数据结点，每个数据结点容纳一个字符。

### 4.3.1 串的链式存储结构的定义

实现串的链式存储结构，需要定义一个结点类，它与一般单向链表的结点类相似，只是数据元素的类型已确定为字符型。链式串的结点类 StringNode 声明如下：

```
public class StringNode {
    private char item;
    private StringNode next;
```

```
    public char Item {
        get { return item; }
        set { item = value; }
    }

    public StringNode Next {
        get { return next; }
        set { next = value; }
    }

    public StringNode(char c) {
        item = c;
        next = null;
    }
}
```

我们定义如下的 LinkedString 类来实现串的链式存储结构，该类声明如下：

```
public class LinkedString{
    private StringNode head, rear;        //单向链表的头结点、尾结点引用
    private int count = 0;                //记载串的实际长度
}
```

LinkedString 类用单向链表的方式实现串的链式存储结构，成员变量有 head，rear 和 count。count 表示串实例的长度，rear 指向单向链表的最后一个结点，head 指向单向链表的仅作为标志的头结点。头结点的数据域可以不存储任何信息，而该结点的链域存储对第一个数据结点的引用。若字符串为空，则头结点的链域为 null。

### 4.3.2　串的链式存储结构基本操作的实现

串的基本操作作为 LinkedString 类的属性和方法予以实现，下面分别描述实现这些操作的算法。

1) 串的初始化

用缺省的构造方法创建并初始化一个串对象，它创建一个仅包含头结点的空串。重载的带参数的构造方法可以构造含一个字符的串或以一个字符数组构造串。多种形式的构造方法编码如下：

```
    public LinkedString(){
        head = new StringNode('>');
        rear = head;
        count = 0;
    }

    //构造一个字符的串
    public LinkedString (char c): this(){
        StringNode q = new StringNode(c);
        this.rear.Next = q;
```

```
        this.rear = q;
        this.count = 1;
    }

    //以一个字符数组构造串
    public LinkedString(char[] c): this(){
        StringNode p,q;
        p = this.rear;
        for(int i=0; i<c.Length; i++){
            q = new StringNode(c[i]);
            p.Next = q;
            p = q;
        }
        rear = p;
        count = c.Length;
    }
```

2) 获取串的长度

该操作告知串包含的字符的个数。将该操作以类的属性成员 Length 来实现，相对于将它定义为成员方法显得更简洁。编码如下：

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

3) 判断串状态是否为空

将这个测试操作定义为属性 Empty。当 count 等于 0 时，串为空状态，属性 Empty 返回值 true。相应的编码如下：

```
    public bool Empty{
        get{ return count==0;}
    }
```

LinkedString 类采用动态分配方式为每个结点分配内存空间，程序中可以认为系统所提供的可用空间足够的大，因此不必判断基于链表的串是否已满。如果系统空间已用完，无法分配新的存储单元，则产生运行时异常。

4) 获得或设置串的第 i 个字符值

将这两个操作通过定义一个读写型索引器成员予以实现，它提供对串对象进行类似于数组的访问。就像 C#的数组下标从 0 开始一样，我们用从 0 开始的索引参数 *i* 来指示串中字符的位置。

```
    public char this[int i]{
        get{
            if(i>=0 && i<count){
                StringNode q = head.Next;
                int j = 0;
                while( j<i ){
```

```
                    j++;
                    q = q.Next;
                }
                return q.Item;
            }
            else
                throw new IndexOutOfRangeException(
                    "Index Out Of Range Exception in " + this.GetType() );
        }
        set{
            if(i>=0 && i<count){
                StringNode q = head.Next;
                int j = 0;
                while(j<i){
                    j++;
                    q = q.Next;
                }
                q.Item = value;
            }
            else
                throw new IndexOutOfRangeException(
                    "Index Out Of Range Exception in " + this.GetType() );
        }
    }
```

5) 连接一个串与一个字符

方法 Concat(c) 将指定的字符加入当前串对象的尾部。首先构造一个包含字符 c 的结点，加入串表尾，再更新表尾 rear 指向新结点，count 加 1。相应的编码如下：

```
    public void Concat(char c){
        StringNode q = new StringNode(c);
        rear.Next = q;
        rear = q;
        count++;
    }
```

6) 连接两个串

方法 Concat(s2) 连接两个串对象，依次将参数 s2 指定的串实例的每个字符连接到本串实例上。也可通过对运算符‘+’ 重载，连接两个串 s1 和 s2。这样两个串的连接可以表示为 s1. Concat(s2)，或者表示为：LinkedString s3 = s1+ s2;

相应的编码如下：

```
    public void Concat(LinkedString s2){
        if(s2!=null){
            StringNode q = s2.head.Next;
```

```
            while(q!=null) {
                this.Concat(q.Item);
                q = q.Next;
            }
        }
    }
    //重载运算符'+'，连接两个串s1和s2
    public static LinkedString operator +(LinkedString s1, LinkedString s2){
        LinkedString newstr = new LinkedString();
        newstr.Concat(s1); newstr.Concat(s2);
        return newstr;
    }
```

7) 获取串的子串

Substring 方法返回串中从序号 *i* 开始的长度为 *n* 的子串。本串的长度为 this.Length，*i* 与 *n* 应满足 0≤*i*<*i*+*n*≤this. Length，否则返回空串。

```
    public LinkedString Substring(int i,int n) {
        int j;
        StringNode q;
        if( i>=0 && n>0 && (i+n<=this.Length) ) {
            LinkedString sub = new LinkedString();
            j = 0;
            q = GetNode(i);
            while(j<n) {
                sub.Concat(q.Item);
                j++;
                q = q.Next;
            }
            return sub;
        }else
            return null;
    }
```

GetNode 方法获得串的第 i 个结点。

```
    public StringNode GetNode(int i){
        if(i>=0 && i<count){
            StringNode q = head.Next;
            int j = 0;
            while( j<i ){
                j++;
                q = q.Next;
            }
            return q;
        }
```

```
        else
            throw new IndexOutOfRangeException(
                "Index Out Of Range Exception in " + this.GetType() );
    }
```

8) 查找子串

IndexOf(sub)方法在串中查找与串 sub 内容相同的子串，若查找成功，返回子串的序号，即子串在主串中首次出现时第一个字符的序号；若查找不成功，则返回-1。子串的查找算法描述参见前面图 4.2 所示，相应的编码如下：

```
    public int IndexOf(LinkedString sub){
        int i=0,j;
        bool found = false;
        StringNode q;
        if(sub.Length==0)
            return 0;
        while(i<=count-sub.Length) {
            j = 0;
            q = GetNode(i);
            while(j<sub.Length && q.Item==sub[j]){
                j++;
                q = q.Next;
            }
            if(j==sub.Length){
                found = true;
                break;
            }
            else
                i++;
        }
        if(found)
            return i;
        else
            return -1;
    }
```

9) 输出串

Show 方法将字符串对象的内容显示在控制台，编码如下：

```
    public void Show(){
        StringNode q = head.Next;
        while(q!=null){
            Console.Write(q.Item);
            q = q.Next;
        }
```

```
            Console.WriteLine();
        }
```

ToCharArray 方法将字符串对象的内容转化为一个字符数组，而重写（override）的 ToString 方法将这里定义的字符串类型转换为 C#内在的 string 类型。这两个辅助方法对于一个完整的串类型是非常有用的，读者可以参考前面的 SequencedString 类来完成这两个方法的编码。

对串的插入、删除、替换、逆转等其他操作，都可以调用前面的操作予以实现，读者可以参考 SequencedString 类实现这些操作的方法，在 LinkedString 类中实现相应的操作。

【例 4.2】 LinkedString 串类的应用。

```
using System;
using DSAGL;
namespace stringstest {
    public class LinkedStringTest {
        public static void Main(string[] args){
            char[] a = {'H','e','l','l','o'};
            LinkedString s1 = new LinkedString(a);
            s1.Show();
            s1.Reverse().Show();
            char[] b = {'W','o','r','l','d'};
            LinkedString s2 = new LinkedString (b);
            LinkedString s0 = new LinkedString ();
            s0.Concat(s1);
            s0.Concat(' ');
            s0.Concat(s2);
            s0.Show();
            (s1 + s2).Show();
            Console.WriteLine("{0} at {1}, {2} at {3} of {4}",
                s1, s0.IndexOf(s1),s2,s0.IndexOf(s2),s0);
            LinkedString s3 = s0.Substring(s0.IndexOf(s2), s2.Length);
            s3.Show();
            char[] c = {'C','h','i','n','a'};
            LinkedString s4 = new String2(c);
            LinkedString s5 = s0.Replace(s2,s4);
            s5.Show();
            LinkedString s6 = s0.Remove(s1.Length+1,s4.Length);
            s6.Show();
        }
    }
}
```

程序运行结果如下：

```
Hello
olleH
Hello World
```

```
HelloWorld
Hello at 0, World at 6 of Hello World
World
Hello China
Hello
```

# 习题 4

4.1 写出 LinkedString 类中的构造方法以一个字符数组构造串：public LinkedString(char[] c)；

4.2 写出 LinkedString 类中实现查找字符操作的方法：public int IndexOf(char c)；

4.3 写出 LinkedString 类中实现插入操作的方法：public LinkedString Insert(int i, LinkedString s2)；

4.4 写出 LinkedString 类中实现删除操作的方法：public LinkedString Remove(int i, int n)；

4.5 写出 LinkedString 类中实现替换操作的方法：public LinkedString Replace(LinkedString oldsub, LinkedString newsub)；

4.6 写出 LinkedString 类中实现替换操作的方法：public LinkedString Replace(char oldc,char newc)；

4.7 编程实现寻找两个字符串中的最长公共子串的操作。

4.8 分别在 SequencedString 和 LinkedString 类中编程实现（重写）基类 Object 中定义的虚方法“ToString()”的操作：public override string ToString()；

4.9 设 string s=“datastructure”，则用表达式_____可以返回串中字符的个数，其结果等于_____，用 IndexOf 定位字符‘t’的下标的表达式是_____，其结果等于_____。表达式 s.Substring(4, 9)的值为_____。s 的非空子串的数目是_____。