第四章　　串

计算机上的非数值处理的对象基本上都是字符串数据。

例，学生管理系统中，学生学号、姓名、性别、院系等；

图书管理系统中，图书编号、书名、作者、简介等；

4.1 串类型的定义

1. 基本概念

串： 由零个或多个字符组成的有限序列。

$$s = \text{‘}a_1a_2 \cdots a_n\text{’} \qquad (n \geq 0)$$

s— 串名

$a_1a_2 \cdots a_n$— 串值

a_i 为字符

$|s|$— 串的长度，即串中字符的数目。

零个字符的串称为空串，计作 φ = ‘’ 。

$|\phi| = 0$

串中任意个连续的字符组成的子序列称为该串的子串。

包含子串的串相应的称为主串。

例，串 ‘eij’是串 ‘beijing’的子串， ‘beijing’称为主串。

字符在序列中的序号称为该字符在串中的位置。

子串在主串中的位置定义为子串的第一个字符在主串中的位置。

例，字符 ‘n’在串 ‘beijing’中的位置为 6 。

例，子串 ‘eij’在串 ‘beijing’中的位置为 2 。

两个串相等，当且仅当这两个串的值相等。

例，串 ‘bei jing’ 与串 ‘beijing’ 不相等 。

串值必须用一对单引号括起来，但单引号本身不属于串，只起界定作用。

由一个或多个空格组成的串称为空格串 .‘ ␣␣␣’

$\phi \neq$ ‘␣’

串的逻辑结构和线性表相似，故看作一种线性表。

$$s = \text{`}a_1 a_2 \cdots a_n\text{'} \qquad (n \geq 0)$$

串的基本操作和线性表区别很大。

线性表： 大多以“单个元素”为操作对象

例，查找某个元素；插入某个元素；删除某个元素

串： 通常以“串的整体”为操作对象

例，查找某个子串；截取某个子串；

在某个位置插入、删除某个子串

4.1 串的抽象数据类型的定义如下：

ADT String {

串是有限长的字符序列，由一对单引号相括，如：‘ a string ’

数据对象：

$$D = \{ a_i \mid a_i \in CharacterSet, \quad i=1,2,...,n, \quad n \geq 0 \}$$

数据关系：

$$R_1 = \{ < a_{i-1}, a_i > \mid a_{i-1}, a_i \in D, \quad i=2,...,n \}$$

StrAssign (String &T, char *s)

初始条件： char *s 是字符串常量。

操作结果：把 s 数组内容 赋为 T 的值。

StrCopy (String &T, String S)

初始条件：串 S 存在。

操作结果：由串 S 复制得串 T。

DestroyString (String &S)

初始条件：串 S 存在。

操作结果：串 S 被销毁。

StrEmpty (String S)

初始条件：串 S 存在。

操作结果：若 S 为空串，则返回 true，
否则返回 false。

" "表示空串，空串的长度为零。

StrCompare (String S, String T)

初始条件：串 S 和 T 存在。

操作结果：若 S > T，则返回值 > 0；

若 S = T，则返回值 = 0；

若 S < T，则返回值 < 0。

例如：StrCompare('data', 'state') < 0

StrCompare('cat', 'case') > 0

StrLength (String S)

初始条件：串 S 存在。

操作结果：返回 S 的元素个数，称为串的长度。

Concat (String &T, String S1, String S2)

初始条件：串 S1 和 S2 存在。

操作结果：用 T 返回由 S1 和 S2 联接而成的新串。

例如：Concate(T, "man", "kind")

求得 T =

SubString (&Sub, S, pos, len)

初始条件:

串 S 存在，**1≤pos≤StrLength(S)**

且 **0≤len≤StrLength(S)-pos+1**。

操作结果:

用 Sub 返回串 S 的第 pos 个字符起

长度为 len 的子串。

子串为“串”中的一个字符子序列

例如：

SubString(sub, ″commander ″, 4, 3)

求得 sub = ″man ″ ;

SubString(sub, ″commander ″ , 1, 9)

求得 sub = ″commander ″

SubString(sub, ″commander ″, 9, 1)

求得 sub = ″r ″

SubString(sub, 'commander', 4, 7)

sub = ?

SubString(sub, 'beijing', 7, 2) = ?

sub = ?

- 起始位置和子串长度之间存在约束关系

SubString("student", 5, 0) = ""

- 长度为 0 的子串为“合法”串

Index (S, T, pos)

初始条件：串 S 和 T 存在， T 是非空串，

1≤pos≤StrLength(S) 。

操作结果： 若主串 S 中存在和串 T 值相同的子串，则返回它在主串 S 中第 pos 个字符之后第一次出现的位置；否则函数值为 0 。

“子串在主串中的位置”意指子串中的第一个字符在主串中的位序。

假设 S = "abcaabcaaabc", T = "bca"

Index(S, T, **1**) = 2;

Index(S, T, **3**) = 6;

Index(S, T, **8**) = 0;

Replace (&S, T, V)

初始条件：串 S, T 和 V 均已存在，且 T 是非空串。

操作结果：用 V 替换主串 S 中出现

例如：

假设　S = "abcabcaabca"，T = "bca "

若　V = "x "，则经置换后得到

S = "axaxaax "

若　V = "bc "，则经置换后得到

S = "abcabcaabc"

StrInsert (&S, pos, T)

初始条件：串 S 和 T 存在，

$1 \le pos \le StrLength(S) + 1$ 。

操作结果：在串 S 的第 pos 个字符之前插入串 T 。

例如：S = "chater"，T = "**rac**"，

则执行 StrInsert(S, 4, T) 之后得到

S = "cha**rac**ter"

StrDelete (&S, pos, len)

始条件：串 S 存在

1≤pos≤StrLength(S)-len+1 。

作结果：从串 S 中删除第 pos 个字符

起长度为 len 的子串

ClearString (&S)

初始条件：串 S 存在。

操作结果：将 S 清为空串。

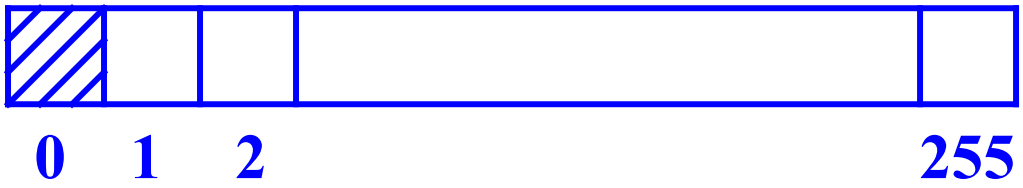
4.2 串的表示和实现

- 定长顺序存储表示——顺序存储
- 堆分配存储表示——顺序存储
- 块链存储表示——链式存储

4.2.1 定长顺序存储表示

用一组地址连续的存储单元存储串值的字符序列；

按照预定义的大小，为每个定义的串变量分配一个固定长度的存储区。



串的实际长度可在此预定义长度内随意，但超过预定义长度的串值则被舍去，称之为“截断”。

串有两种表示方法：

- 下标 0 的分量存放串的长度，其它存放串字符。

```
# define   MAXSTRLEN   255

typedef   unsigned  char  SString[MAXSTRLEN+1]
```

PASCAL 语言

- 在串值后面加一个不计入串长的结束标记字符。

C 语言中以“ \0” 表示串值的终结

算法 4.2　串联接　strcat(&T ， S1 ， S2)

要求： 顺序联接串　S1　和串　S2　得到新串　T　。

思想：

基于串 S1 和 S2 长度的不同情况，串 T 可能出现 3 种情况：

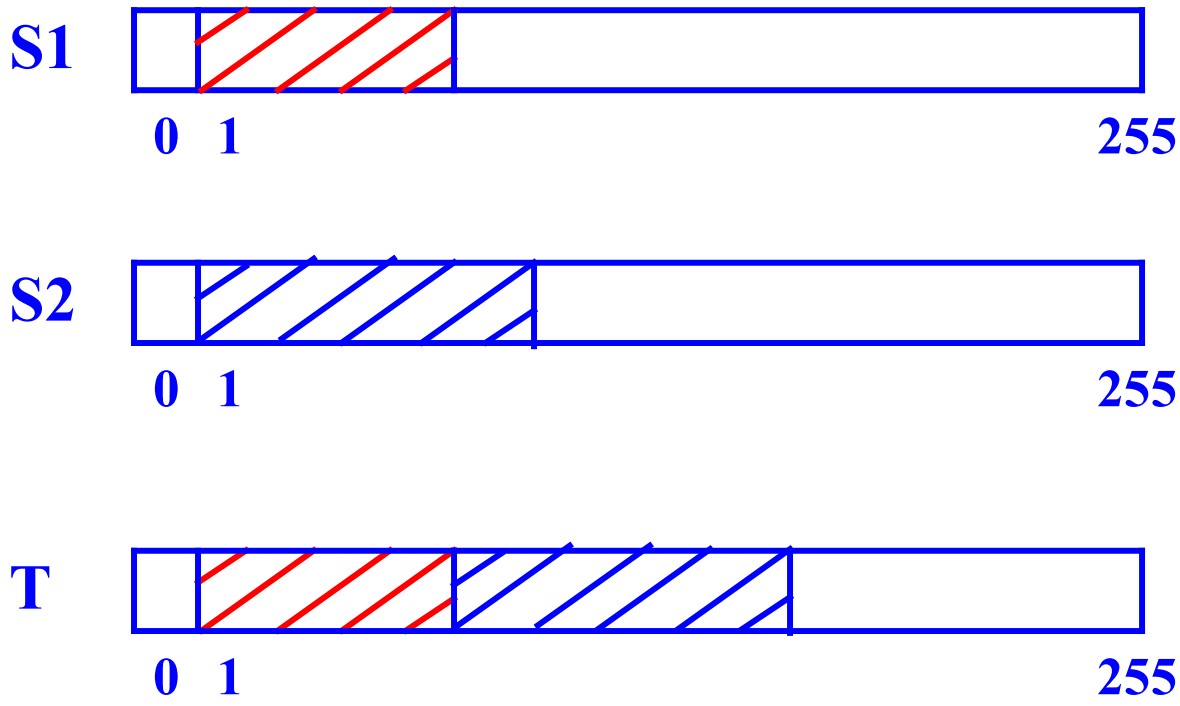
S1[0]+S2[0]≤MAXSTRLEN，直接联接，T[0]≤MAXSTRLEN；

S1[0]+S2[0] ＞ MAXSTRLEN

S1[0] ＜ MAXSTRLEN，截断 S2，联接，T[0] ＝ MAXSTRLEN；

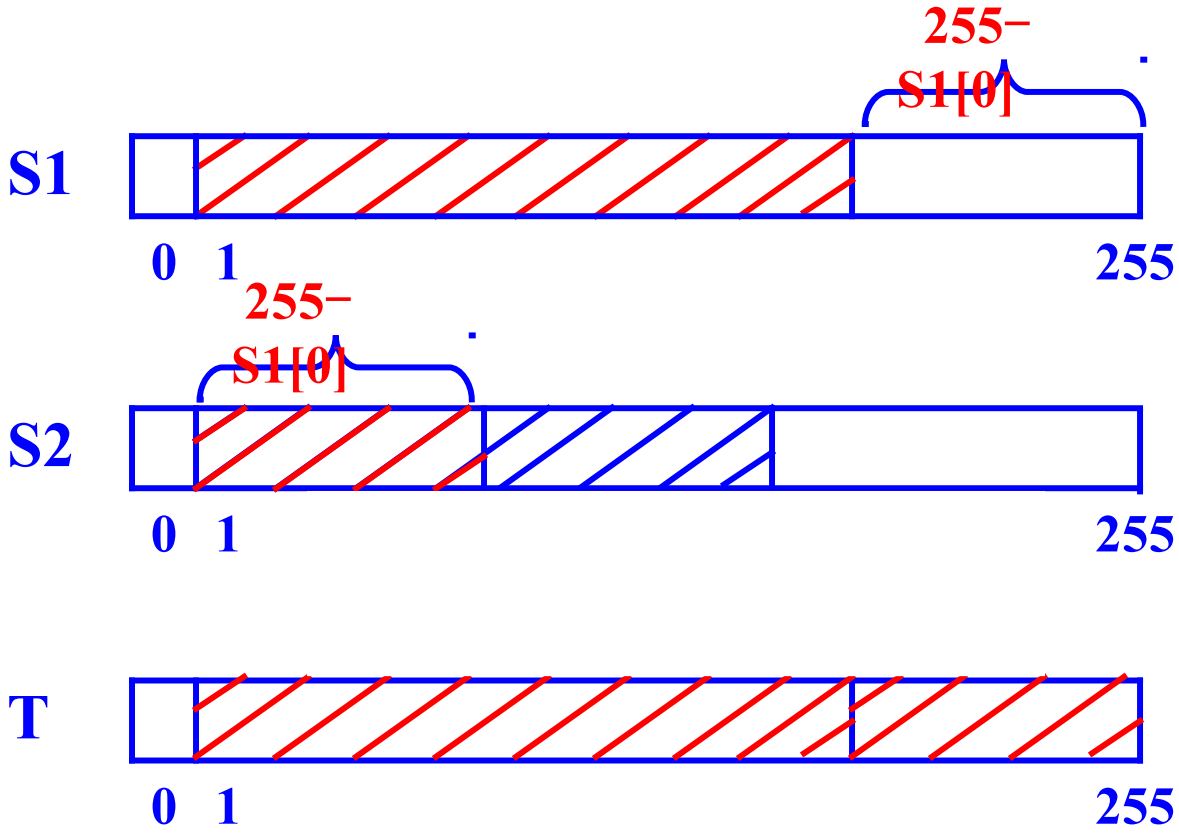
S1[0] ＝ MAXSTRLEN，　T　＝　S1；

S1[0]+S2[0]≤MAXSTRLEN



T[0] = S1[0] + S2[0]

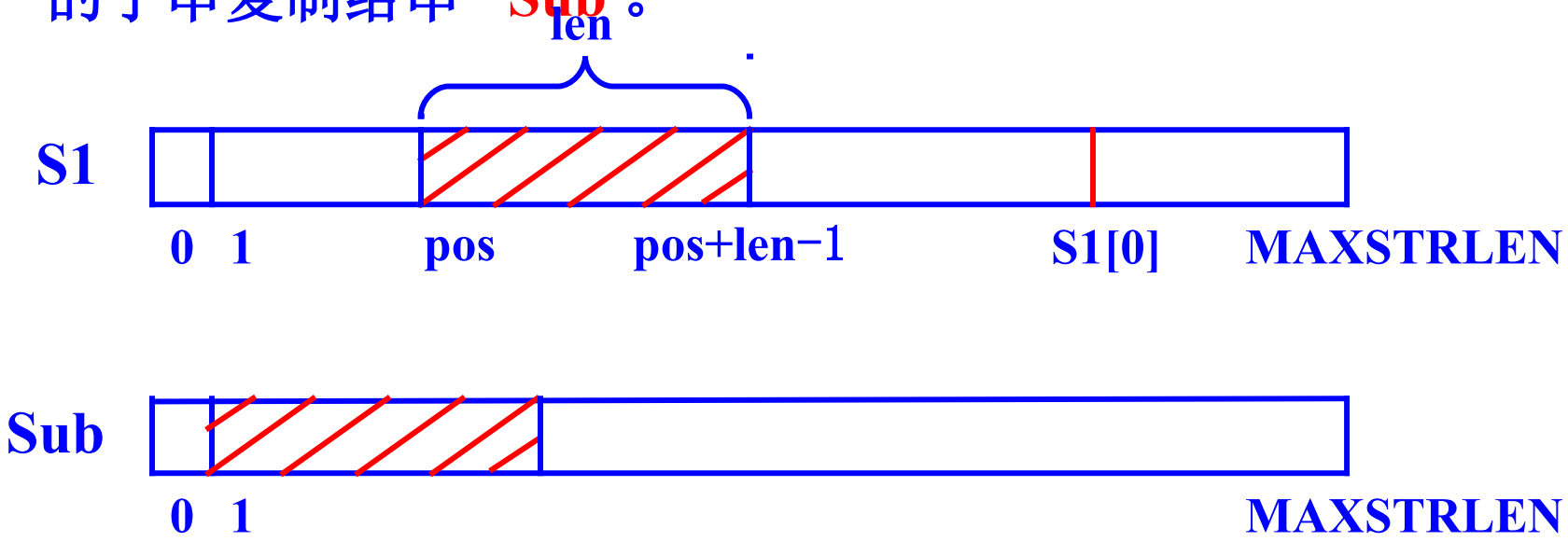
S1[0]+S2[0] ＞ MAXSTRLEN ， S1[0] ＜ MAXSTRLEN



T[0] ＝ MAXSTRLEN

算法 4.3　求子串　substr (&Sub ， S ， pos ， len)

要求：将串 S 从第 pos 个字符开始长度为 len 的子串复制给串 Sub。



注意：　1≤pos≤S1[0]

len≥0

pos+len−1≤S1[0]　即　len≤S1[0]−pos+1

特点：

- **串**的实际长度可在这个预定义长度的范围内随意设定，超过预定义长度的串值则被舍去，称之为“**截断**”
- 按这种串的表示方法实现的串的运算时，其基本操作为“**字符序列的复制**”

二、串的堆分配存储表示

```
typedef struct {
  char *ch;
    // 若是非空串，则按串使用长度分配
    // 存储区，否则 ch 为 NULL
  int  length;  // 串长度
} HString;
```

通常，C 语言中提供的串类型就是以这种存储方式实现的。系统利用函数 malloc( ) 和 free( ) 进行串值空间的动态管理，为每一个新产生的串分配一个存储区，称串值共享的存储空间为“**堆**”。

C 语言中的串以一个空字符为结束符，串长是一个隐含值。

这类串操作实现的算法为：

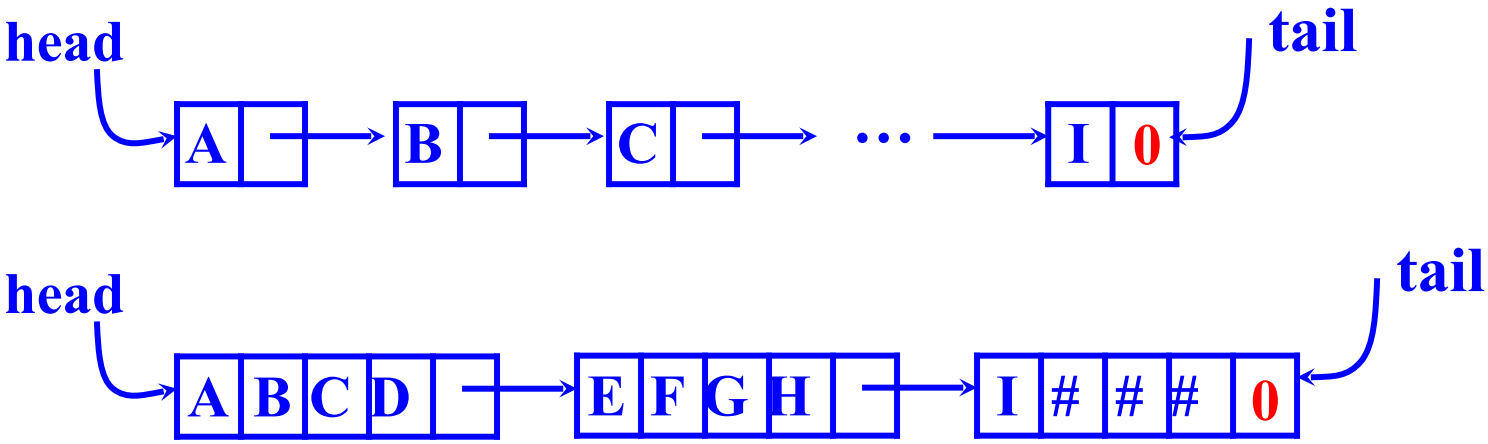
先为新生成的串分配一个存储空间，然后进行串值的复制。

```
Status Concat(HString &T, HString S1, HString S2)
{
  // 用 T 返回由 S1 和 S2 联接而成的新串
  if (T.ch)  free(T.ch);        // 释放旧空间
  if (!(T.ch =new char[S1.length+S2.length]))
      exit (OVERFLOW);
  T.ch[0..S1.length-1] = S1.ch[0..S1.length-1];
  T.length = S1.length + S2.length;
  T.ch[S1.length..T.length-1] = S2.ch[0..S2.length-1];
  return OK;
} // Concat
```

4.2.3 串的块链存储表示

采用链表方式存储串值。

每个结点可以存放一个字符，也可以存放多个字符。



结点大小的选择直接影响串处理的效率。

$$\text{存储密度} = \frac{\text{串值所占的存储位}}{\text{实际分配的存储位}}$$

在如下定义的链串结点中，每个字符占 1 个字节，指针占 4 个字节，

求该链串的存储密度。

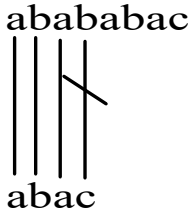
```
typedef struct node
 {
        char data[8];
        struct node *next;
} LinkStrNode;
```

$$\text{存储密度} = \frac{8}{12}$$

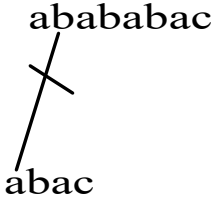
4.3 串的模式匹配算法

1. 顺序串上 的子串定位运算 index(S , T)

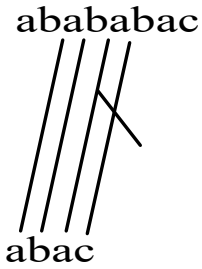
子串的定位运算通常称为串的模式匹配，是串处理中最重要的运算之一。设串 s="$a_1a_2\ldots a_n$"，串 T="$b_1b_2\ldots b_m$"($m \le n$)，子串定位是要在主串 S 中找出一个与子串 T 相同的子串。通常把主串 S 称为目标，把子串 T 称为模式，把从目标 S 中查找模式为 T 的子串的过程称为“模式匹配”。匹配有两种结果出现：若 S 中有模式为 T 的子串，就返回该子串在 S 中的位置，当 S 中有多个模式为 T 的子串，通常只要找出第一个子串即可，这种情况称为匹配成功，若 S 中无模式为 T 的子串，返回值为零，称为匹配失败。模式匹配过程如下图所示，假设 S="abababac"，



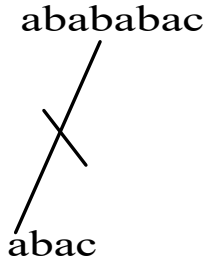
(a) 第一趟匹配 s4≠t4



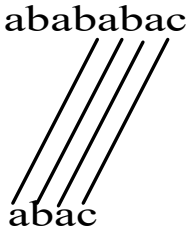
(b) 第二趟匹配 s2≠t1



(c) 第三趟匹配 s6≠t4



(d) 第四趟匹配 s4≠t1



(e) 第五趟匹配成功

图 4-12　　模式匹配过程

模式匹配算法如下：

```
int    Index  (    seqstring  S, seqstring T,int pos )
{  int   i=pos ,  j=1;
   while    ( i<=S.curlen )&&(j<=T.curlen)
        if  (S.ch[i]==T.ch [j])
        {     i++;   j++ ; }
        else
        { i=i-j+2 ;  j=1;}                    // 将 i 指针回溯
   if  ( j>T.curlen )
     return  (i-T.curlen);
   else
     return (0);   }                          // 匹配失败
```

该算法中，将 i 指针回溯语句 i=i-j+2，可以这样理解：

在本趟匹配中，有 si≠tj，但前面的字符都匹配，即有

$s_{i-1}=t_{j-1}, s_{i-2}=t_{j-2}, \cdots, s_{i-j+1}=t_1$，因此，下一趟匹配时，i 应从 i-j+1 的下一位置开始，即有 i=i-j+1+1，就是算法中的 i=i-j+2。该算法的最好时间复杂度为 **O(n+m)**，最坏时间复杂度为 **O(n×m)**。

⌘ 若一个串非空，子串的定位操作通常称为（　）。

A. 串的长度　　B. 原串的子串

C. 串的模式匹配　　D. 串的连接

- 若 n 为主串长， m 为子串长（ m<n ），则用简单模式匹配算法最坏情况下，需要比较字符总数是（　　）。

A．　m　　B．　m(n-m+1)

C．　n*m　　D．　(n-m)*(m-1)

⌘ 若一个串非空，子串的定位操作通常称为(　　)。

A. 串的长度　　B. 原串的子串

C. 串的模式匹配　　D. 串的连接