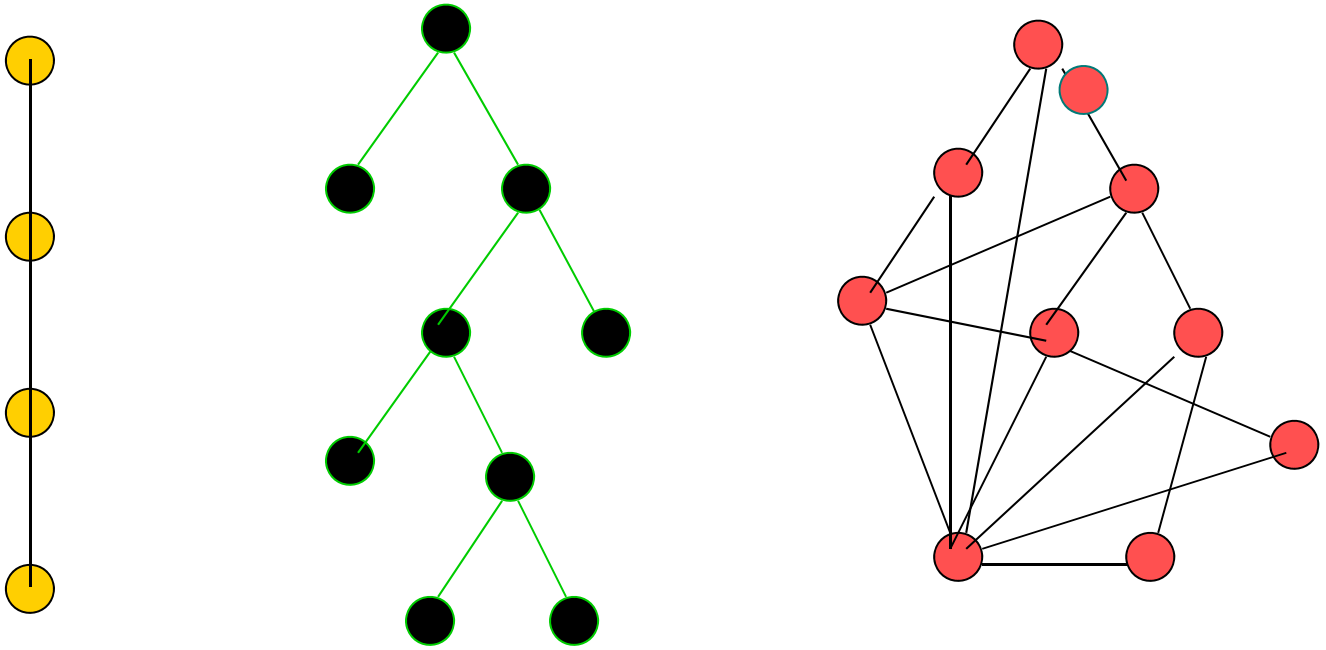
数据结构 华中科技大学计算机学院



第四章 字符串/串(string)

4.1 串的定义与操作

1.术语

(1)串：由零个或多个字符组成的有限序列。

n个字符C1,C2,...,Cn组成的串记作：

s=’$C_1C_2...C_n$’ n>=0

其中:s为串名， $C_1C_2...C_n$为串值 n为串长

例 PASCAL语言:

s1=’data1234’ s2=’123*abc’

C语言:

s1=″data1234″ s2=″123*abc″

’A’为字符 ″A″为字符串

%c为字符格式 %s为字符串格式

(2)空串：不含字符的串/长度为零的串。

PASCAL语言： s=''

C语言： s=""

(3)空格串：仅含空格字符' '的串。

例 s1='Φ' s2='ΦΦ'

s1=' ' s2=' '

(4)子串：串s中任意个连续的字符组成的子序列称为串s的子串。

主串---包含某个子串的串。

例 st="ABC123A123CD"

s1="ABC" s3="123A" s4="ABCA"

s2="ABC12" s5="ABCE" s6="321CBA"

2. 串变量、字符变量的定义与使用

例1 串变量

```
char st[]="abc\'*123";
gets(st);   scanf("%s",st);
strcpy(st,"data");
puts(st);   printf( "st=%s\n" ,st);
```

例2 字符变量

```
char ch='A';
ch=' B' ;   ch=getchar();
scanf("%c",&ch);   printf("ch=%c\n",ch);
```

3. 串的基本操作与串函数

(1) strcpy(t,s)---s的串值复制到t中。

执行：strcpy(t,"data"); 有：t="data"

(2) strlen(s)----求s的长度

strlen("data*123")=8 strlen("")=0

(3) strcat(s1,s2)----s2的值接到s1的后面。

设 s1="data" , s2="123"

执行：strcat(s1,s2);

有： s1="data123" , s2="123"

(4) strcmp(s1,s2)---比较s1和s2的大小

若s1<s2,返回负整数　　如："ABC"<"abc"

若s1=s2,返回0如：　　如："abc"="abc"

若s1>s2,返回正整数　　如："ABCD">"ABC"

(5) strstr(s1,s2)----若s2是s1的子串，则指向s2在s1中第1次出现时的第1个字符的位置；否则返回NULL。

设 s1="ABE123*DE123bcd"

s2="E123"

有 strstr(s1,s2)=3

s1="ABE123*DE123"

(6)Replace(s,t,v)----置换

用v代替主串s中出现的所有与t相等的不重叠的子串

设 s="abc123abc*ABC", t="abc", v="OK"

执行：Replace(s,t,v);

有： s="OK123OK*ABC", t="abc", v="OK"

设 A="abcaaaaaABC", B="aa", C="aaOK"

执行：Replace(A,B,C);

有： A="abcaaOKaaOKaABC", B="aa", C="aaOK"

(7) StrInsert(s,i,t)----将t插入s中第i个字符之前。

设 s="ABC123"

执行: StrInsert(s,4,"**");

有: s="ABC**123"

(8)用Replace(s,t,v)实现删除

设　　s="ABC//123"

执行：Replace(s,"//","")

有：　s="ABC123"

(9)用Replace(s,t,v)实现插入

设　　s="ABC123"

执行：Replace(s,"123","**123")

有：　s="ABC**123"

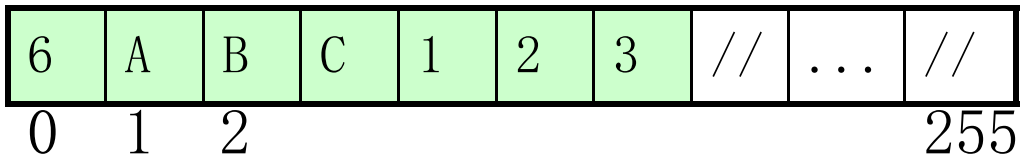
4.2 串的存储表示和实现

4.2.1 串的定长顺序存储表示

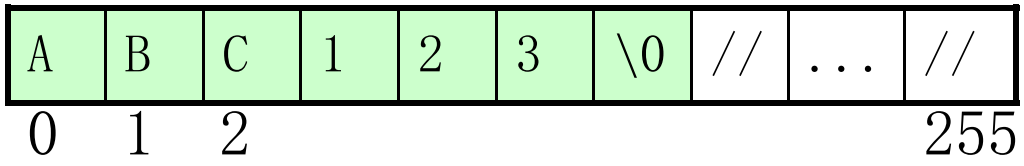
给每个定义的串分配一个固定长度的存储区

```
#define   MAXSTRLEN    255    //用户可在255以内定义最大长度
typedef unsigned char SString[MAXSTRLEN+1]; //0号单元存放
                                            //串的长度
```

PASCAL:　下标为0的分量存放串的实际长度

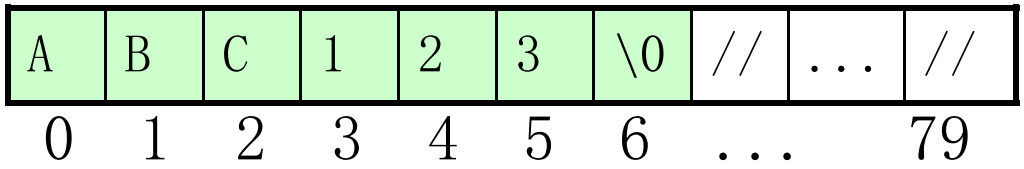


C:　　在串值后加串结束标记‘\0’,串长为隐含值

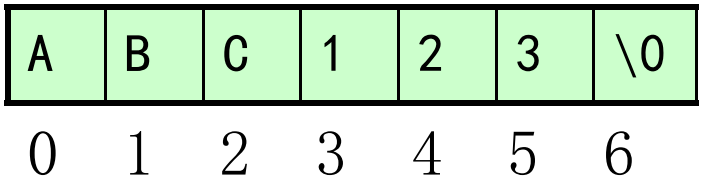


1. 顺序存储----用一维字符数组表示一个串的值。

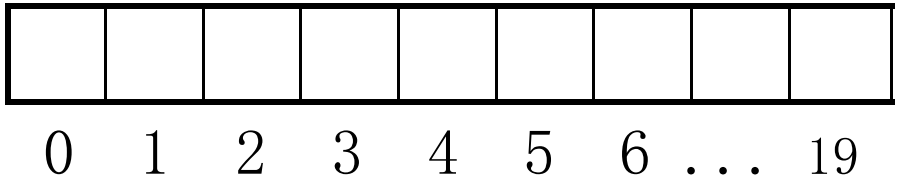
例1 char st1[80]="ABC123";



例2 char st2[]="ABC123";



例3 char st[20];



2. 串运算实现举例

例 联接运算：给定串s1,s2,将s1,s2联接为串t,记作：

Concat(t,s1,s2)

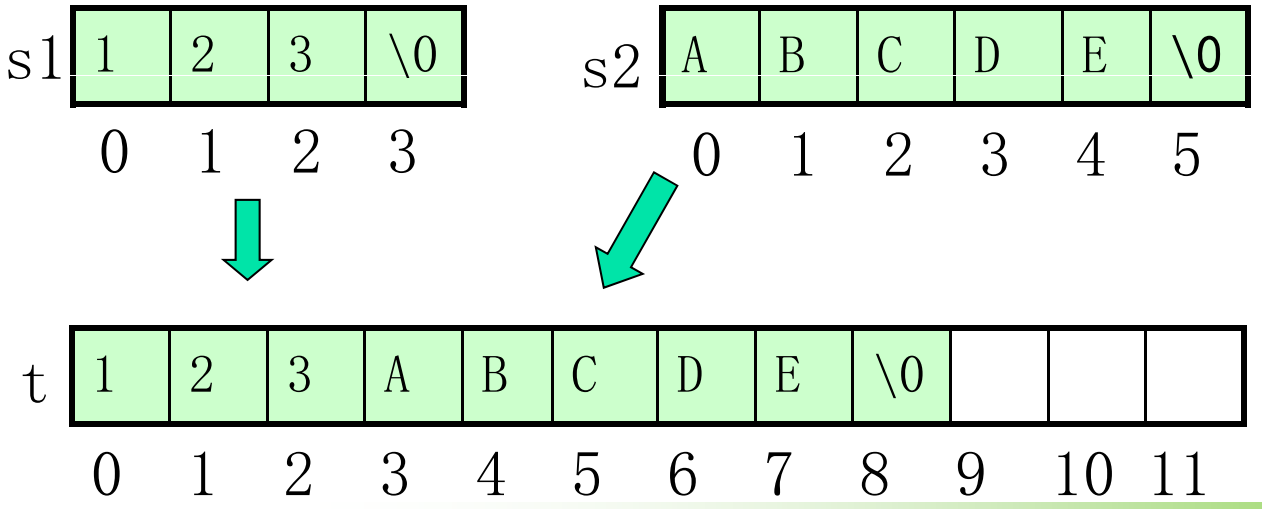
设 s1=”123”, s2=”ABCDE”

执行： Concat(t,s1,s2)；

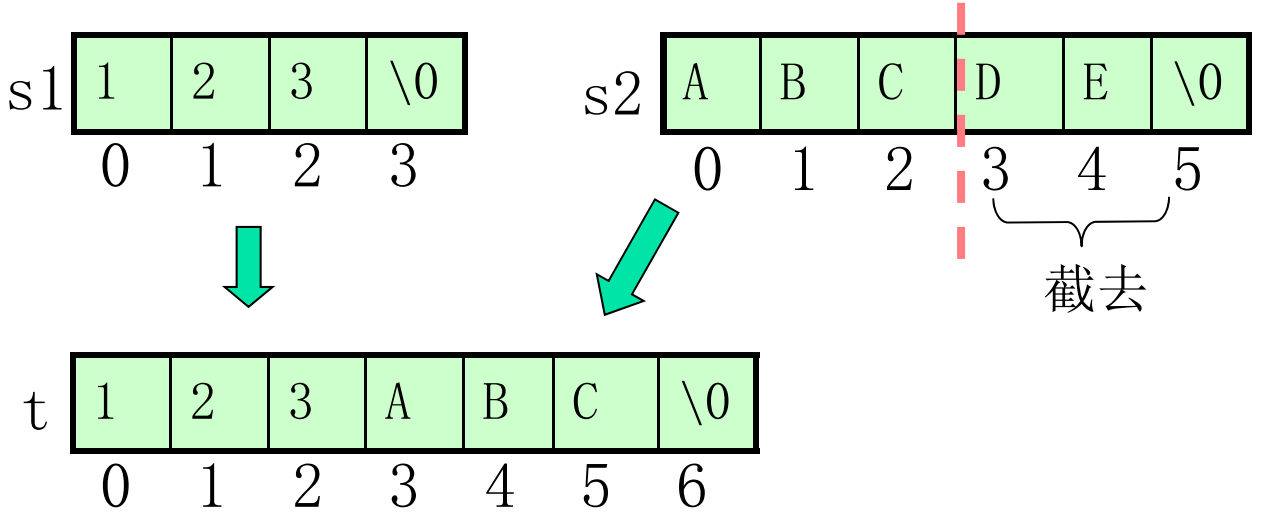
有： t=”123ABCDE”

实现Concat(t,s1,s2)的方法：

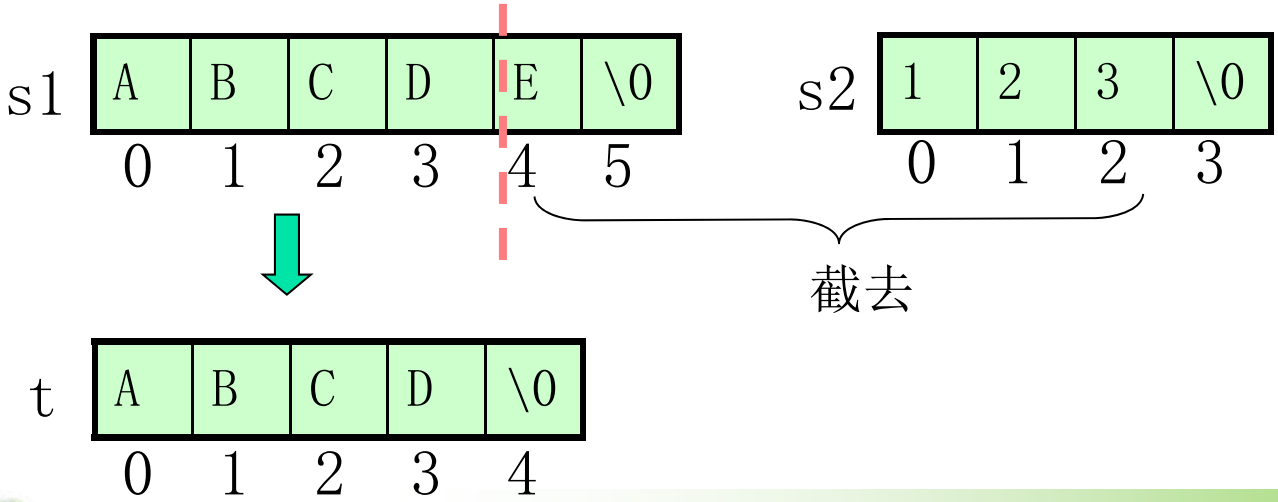
(1)s1的长度+s2的长度≤t的最大长度：



(2)s1的长度≤t的最大长度≤s1的长度+s2的长度：

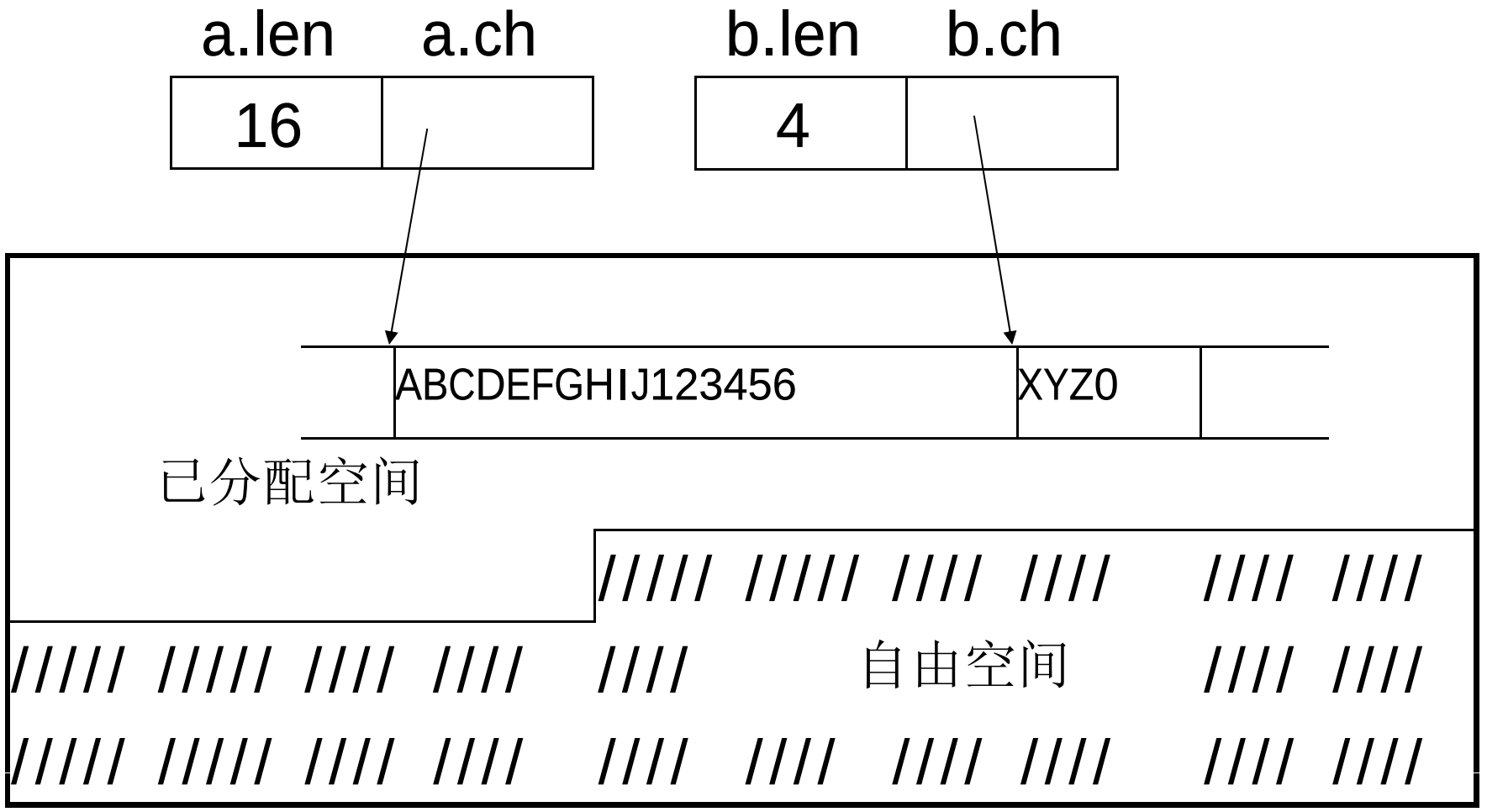


(3) t的最大长度<s1的长度：



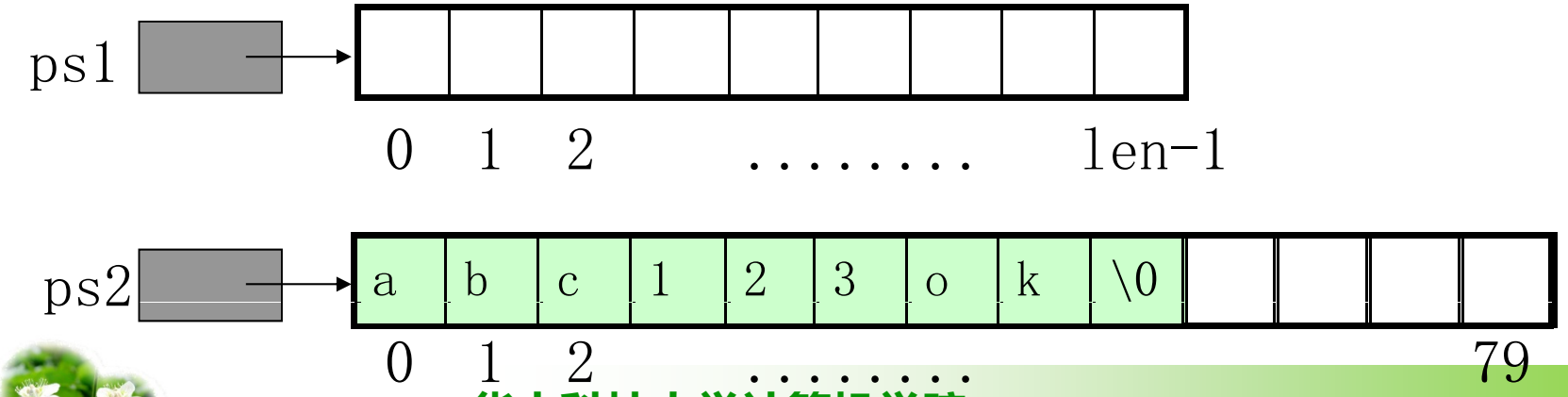
4.2.2 串的堆分配存储表示

提供一个足够大的连续存储空间，存放字符串的值。



例1：C语言中利用动态分配使用系统堆。

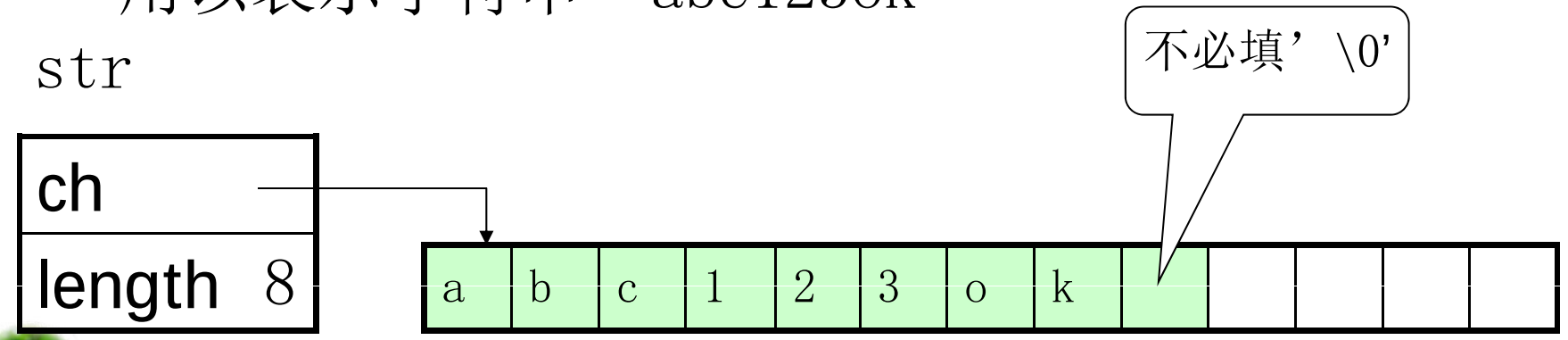
```
{ char *ps1,*ps2; int len;
  scanf("%d",&len);                    //输入长度值
  ps1=(char *)malloc(len);             //ps1指向分配的存储空间
  gets(ps1);  puts(ps1);               //输入一个串，再输出
  ps2=(char *)malloc(80);              //ps2指向分配的存储空间
  strcpy(ps2,"abc123ok") ;             //赋值，再输出
  puts(ps2);
  free(ps1); free(ps2);                //释放存储空间
 }
```



堆存储结构描述,定义将串长作为存储结构的一部分:

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

用以表示字符串“abc123ok”



例2：字符串赋值操作

```
int StrAssign( HString *T,  char *chars)
{  char *c;  int i;
   if (T->ch) free(T->ch);              /*释放T原有空间*/
   for(i=0,c=chars; *c; i++,++c);       /*求chars的串长i*/
   if (!i)
      {T->ch=NULL;T->length=0;}         /*当chars为空串时*/
   else {
         if (!(T->ch=(char *) malloc(i*sizeof(char))))
                  return OVERFLOW;
         T->length=i;
         for(; i>0; i--)                /*复制chars串值到串T*/
              T->ch[i-1]=chars[i-1];
         }
   return OK;
}
```

例3：输出字符串

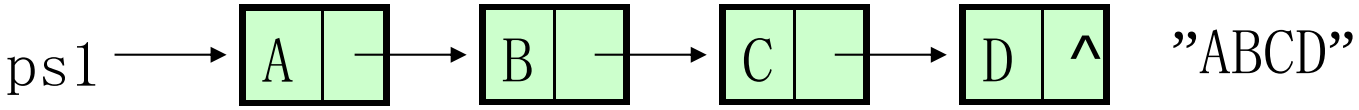
```
void StrPrint(HString T)
{
 int i;
 for(i=0;i<T.length;i++)
     putchar(T.ch[i]);
 }

void  main(void)
{
  HString str;
  StrAssign(&str,”abcd123”);
  StrPrint(str);
 }
```

4.2.3 串的单链表表示

例1 一个结点只放1个字符

```
struct node1
 { char data;              //为一个字符
   struct node1 *next;     //为指针
  }*ps1;
```

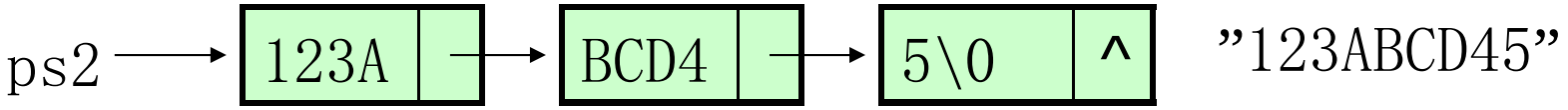


存储密度=串值所占存储位/实际分配存储位

存储密度为 0.33

例2 一个结点放4个字符

```
struct node4
{ char data[4];          //为4个字符的串
  struct node4 *next;    //为指针
 }*ps2;
```



存储密度为 0.67

思考题

使用链串，实现判断一个输入的字符串是否是回文。所谓回文是指中间对称的字符串，如 ”abccba”, “aabbkbbaa” 等是回文，而 “abcda” 不是回文。

模式匹配

- 什么是模式匹配
- 朴素匹配算法
- KMP算法
- 更多模式匹配算法

模式匹配

- 什么是模式匹配
- 朴素匹配算法
- KMP算法
- 更多模式匹配算法

模式匹配

- 给定一个子串(也称模式串)，要求在主串中找出与该子串相同的所有子串，也称为子串定位

- 例如：主串‘acbcdabc‘ 子串'abc‘

模式匹配

- 什么是模式匹配
- 朴素匹配算法
- KMP算法
- 复杂度分析
- 更多模式匹配算法

朴素匹配算法

- 主串：s[] = ‘ababcabcacbab’
- 子串：t[] = ‘abcac’

- How does it work?

第一次匹配 a b a b c a b c a c b a b

a b c a c

第二次匹配 a b a b c a b c a c b a b

a b c a c

第三次匹配 a b a b c a b c a c b a b

a b c a c

第四次匹配 a b a b c a b c a c b a b

a b c a c

第五次匹配 a b a b c a b c a c b a b

a b c a c

第六次匹配 a b a b c a b c a c b a b

a b c a c

朴素匹配算法：约定S[0],T[0] 存放串长

```
int Normal(String S, String T, int pos)
{
        int i,j;
        i=pos;j=1;
        while(i<=S[0] &&j<=T[0])
        {
                if(S[i]==T[j]){i++;j++;}
                else{i=i-j+2; j=1;}
        }
        if(j>T[0])
                return i-T[0];
        else
                return -1;
}
```

复杂度分析

- 一般情况

  效率可以近似认为O(m+n)

- 极端特殊情况

  算法时间复杂度为O((n-m+1)*m)

模式匹配

- 什么是模式匹配
- 朴素匹配算法
- KMP算法
- 更多模式匹配算法

Knuth-Morris-Pratt

Professor Emeritus of The Art of Computer Programming at Stanford University, welcomes you to his home page.

Donald E. Knuth，1938年出生于Wisconsin。1960年，当他毕业于Case Institute of Technology数学系时，因为成绩过于出色，被校方打破历史惯例，同时授予学士和硕士学位。他随即进入大名鼎鼎的加州理工学院数学系，仅用三年时间便取得博士学位，此时年仅25岁。 毕业后留校任助理教授，28岁时升为副教授。30岁时，加盟斯坦福大学计算机系，任正教授。从31岁那年起，他开始出版他的历史性经典巨著：The Art of Computer Programming。他计划共写7卷，然而仅仅出版三卷之后，已经震惊世界，使他获得计算机科学界的最高荣誉Turing Award!此时，他年仅36岁！后来，此书与牛顿的“自然哲学的数学原理”等一起，被评为“世界历史上最伟大的十种科学著作”之一。

KMP算法

- 主串：　abcdefgabcdab
- 模式串：　abcdex

- How does it work?

第一次匹配 a b c d e f g a b c d a b

a b c d e x

第二次匹配 a b c d e f g a b c d a b

a b c d e x

第三次匹配 a b c d e f g a b c d a b

a b c d e x

第四次匹配 a b c d e f g a b c d a b

a b c d e x

第五次匹配 a b c d e f g a b c d a b

a b c d e x

第六次匹配 a b c d e f g a b c d a b

a b c d e x

观察：

1、T串中首字符a 与后面字符都是不相等的

2、T串中的a与S串后面的b, c, d, e也都可以在第一次匹配之后就确定是不相等的

结论：第二、三、四、五次匹配没有必要，只需保留第一、六次匹配，也即 i 的值没有必要回溯

问题：T串后面含有首字符a怎么办？

- 主串：　abcababcabc
- 模式串：　abcabx

第一次匹配 a b c a b a b c a b c

a b c a b x

第二次匹配 a b c a b a b c a b c

a b c a b x

第三次匹配 a b c a b a b c a b c

a b c a b x

第四次匹配

a	b	c	a	b	a	b	c	a	b	c	
			a	b	c	a	b	x			

结论：i的值不可以变小，变化的是j的值，j值多少取决于当前字符之前的串的前后缀的相似程度

KMP算法原理：

- 假设现在文本串S匹配到 i 位置，模式串T匹配到 j 位置
  - 如果当前字符匹配成功（即S[i] == T[j]），则令i++，j++，继续匹配下一个字符；
  - 如果当前字符匹配失败（即S[i] != T[j]），则令 i 不变，j = next[j] (next[j] <= j – 1)。此举意味着失配时，模式串T相对于文本串S向右移动了j - next [j] (至少为1) 位。
  - 关键：如何求next[j]

next[j]：

$$next[j]=\begin{cases}0, & \text{当}j=1\text{时}\\ Max\{k|1<k<j, \text{且}p_1\cdots p_{k-1}=p_{j-k+1}\cdots p_{j-1}\} & \text{当此集合不空}\\ 1, & \text{其他情况}\end{cases}$$

j	1 2 3 4 5 6 7 8 9
模式串	a b a b a a a b a
next[j]	0 1 1 2 3 4 2 2 3

J=1时，next[1]=0

J=2时，只有字符a,属其他情况,next[2]=1

J=3时，j由1到j-1的字符串是ab,a与b不等，属其他情况,next[3]=1

J=4时，j由1到j-1的字符串是aba,前缀字符a与后缀字符a相等，可得到k=2,故next[4]=2

J=5时，j由1到j-1的字符串是abab,前缀字符ab与后缀字符ab相等，可得到k=3,故next[5]=3

J=6时，j由1到j-1的字符串是ababa,前缀字符aba与后缀字符aba相等，可得到k=4,故next[4]=4

…

a b a b c a b c a c b a b

a b c a c

a b c a c

a b c a c

j	1 2 3 4 5
模式串	a b c a c
next[j]	0 1 1 1 2

KMP算法：

```
int KMP(String S, String T, int pos)
{
        int i=pos,j=1;
        int next[255];
        Get_Next(T, next);
        while(i<=S[0]&&j<=T[0])
        {
                if(j==0||S[i]==T[j]){i++;j++;}
                else{j=next[j];}
        }
        if(j>T[0])
                return i-T[0];
        else
                return -1;
}
```

next[]算法：

```
void Get_Next(String T, int *next)
{
        int i,j;
        i=1; j=0;
        next[1]=0;
        while(i<T[0])
        {
                if(j==0||T[i]==T[j])
                    {i++; j++; next[i]=j;}
                else{j=next[j];}  //若字符不相同，则j值回溯
        }
}
```

复杂度分析

1、Get_Next时间复杂度为O(m)

2、KMP内部while循环时间复杂度O(n)

3、故而整个算法的时间复杂度为O(n+m)

模式匹配

- 什么是模式匹配
- 朴素匹配算法
- KMP算法
- 更多模式匹配算法

更多模式匹配算法

- Boyer-Moore算法

  这个算法KMP算法的不同点是在作s[k+1..k+m]与t[1..m]的匹配测试时是从右到左，而不是从左到右。

- Rabin-Karp算法

  这个算法用到数论中诸如两个整数关于第三个整数取模的等价性等初等概念。

- Sunday算法