

第4章 串

4.1 串的基本概念

4.2 串的存储结构

4.3 串的模式匹配

4.1 串的基本概念

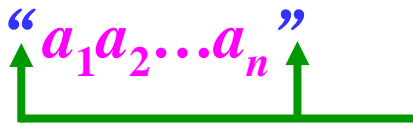
串（或字符串）是由零个或多个**字符**组成的**有限序列**。

串 \subset 线性表

串中所含字符的个数称为该**串的长度**（或串长），含零个字符的串称为空串，用 Φ 表示。

串的逻辑表示， a_i ($1 \leq i \leq n$) 代表一个字符：

“ $a_1 a_2 \dots a_n$ ”



双引号不是串的内容，起标识作用

串相等：当且仅当两个串的长度相等并且各个对应位置上的字符都相同时，这两个串才是相等的。

如：

“*abcd*” \neq “*abc*”

“*abcd*” \neq “*abcde*”

所有空串是相等的。

子串：一个串中任意个连续字符组成的子序列（含空串）称为该串的子串。

例如，“*abcde*”的子串有：

“”、“*a*”、“*ab*”、“*abc*”、“*abcd*”和“*abcde*”等

真子串是指不包含自身的所有子串。

串抽象数据类型=逻辑结构+基本运算（运算描述）

串的基本运算如下：

- ① **StrAssign(&s,cstr)**: 将字符串常量cstr赋给串s, 即生成其值等于cstr的串s。
- ② **StrCopy(&s,t)**: 串复制。将串t赋给串s。
- ③ **StrEqual(s,t)**: 判串相等。若两个串s与t相等则返回真；否则返回假。
- ④ **StrLength(s)**: 求串长。返回串s中字符个数。
- ⑤ **Concat(s,t)**: 串连接:返回由两个串s和t连接在一起形成的新串。
- ⑥ **SubStr(s,i,j)**: 求子串。返回串s中从第 i ($1 \leq i \leq n$) 个字符开始的、由连续 j 个字符组成的子串。

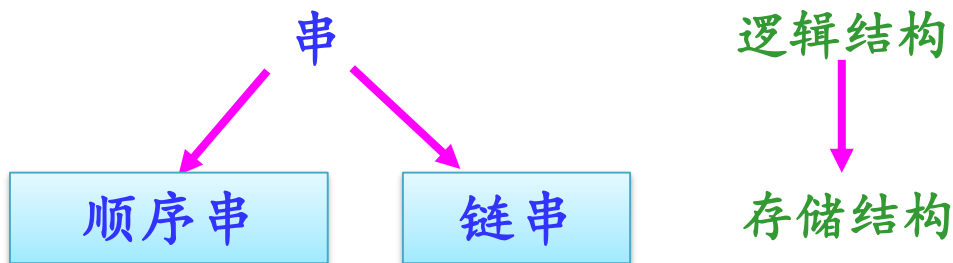
- ⑦ **InsStr(s1,i,s2):** 插入。将串s2插入到串s1的第 i ($1 \leq i \leq n+1$) 个字符中, 即将s2的第一个字符作为s1的第 i 个字符, 并返回产生的新串。
- ⑧ **DelStr(s,i,j):** 删除。从串s中删去从第 i ($1 \leq i \leq n$) 个字符开始的长度为 j 的子串, 并返回产生的新串。
- ⑨ **RepStr(s,i,j,t):** 替换。在串s中, 将第 i ($1 \leq i \leq n$) 个字符开始的 j 个字符构成的子串用串t替换, 并返回产生的新串。
- ⑩ **DispStr(s):** 串输出。输出串s的所有元素值。

思考题

串和线性表有什么异同？

4.2 串的存储结构

串中元素逻辑关系与线性表的相同，串可以采用与线性表相同的存储结构。



4.2.1 串的顺序存储及其基本操作实现

串的顺序存储（顺序串）有两种方法：

- 每个单元(如4个字节)只存一个字符，称为**非紧缩格式**（其存储密度小）。
- 每个单元存放多个字符，称为**紧缩格式**（其存储密度大）。

1001	A			
1002	B			
1003	C			
1004	D			
1005	E			
1006	F			
1007	G			
1008	H			
1009	I			
100a	J			
100b	K			
100c	L			
100d	M			
100e	N			

非紧缩格式示例

1001	A	B	C	D
1002	E	F	G	H
1003	I	J	K	L
1004	M	N		

紧缩格式示例

一个单元

对于非紧缩格式的顺序串，其类型定义如下：

```
#define MaxSize 100
```

```
typedef struct
```

```
{   char data[MaxSize];
```

```
    int length;
```

```
} SqString;
```

用来存储字符串

用来存储字符串长度

顺序串中实现串的基本运算与顺序表的基本运算类似。详细算法实现参见教材。

【例4-1】 设计顺序串上实现串比较运算Strcmp(s,t)的算法。

例如：

"ab" < "abcd"

"abcd" < "abd"

解： 算法思路如下：

(1) 比较s和t两个串**共同长度范围内**的对应字符：

① 若s的字符 > t的字符，返回1；

② 若s的字符 < t的字符，返回-1；

③ 若s的字符 = t的字符，按上述规则继续比较。

(2) 当(1)中对应字符均相同时，比较s和t的长度：

① 两者相等时，返回0；

② s的长度 > t的长度，返回1；

③ s的长度 < t的长度，返回-1。

```
int Strcmp(SqString s, SqString t)
{
    int i, comlen;
    if (s.length < t.length) comlen = s.length; //求s和t的共同长度
    else comlen = t.length;
    for (i = 0; i < comlen; i++) //在共同长度内逐个字符比较
    {
        if (s.data[i] > t.data[i])
            return 1;
        else if (s.data[i] < t.data[i])
            return -1;
    }
    if (s.length == t.length) //s==t
        return 0; //所有共同长度内的字符相同，哪个长哪个大
    else if (s.length > t.length) //s>t
        return 1;
    else //s<t
        return -1;
}
```

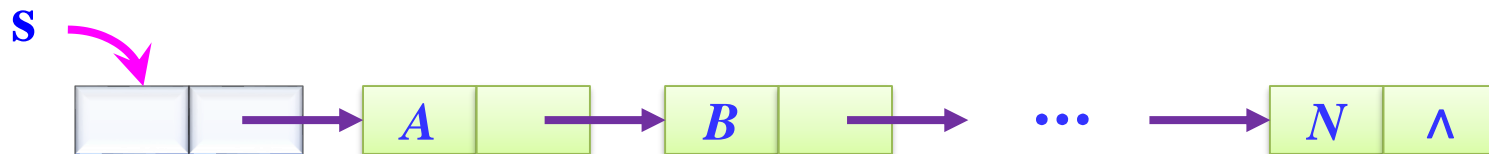
4.2.2 串的链式存储及其基本操作实现

链串的组织形式与一般的链表类似。

链串中的一个节点可以存储多个字符。通常将链串中每个节点所存储的字符个数称为节点大小。



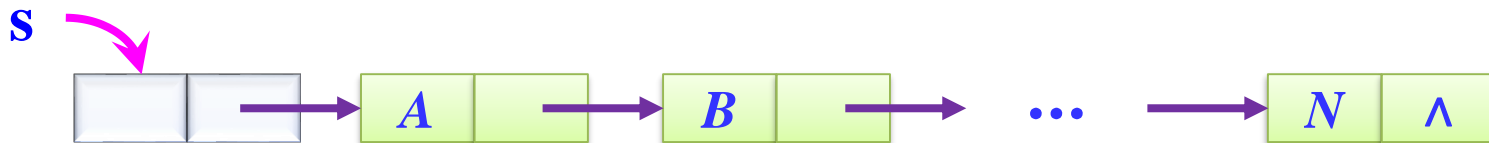
节点大小为4的链串



节点大小为1的链串

链串节点大小1时，链串的节点类型定义如下：

```
typedef struct snode
{
    char data;
    struct snode *next;
} LiString;
```



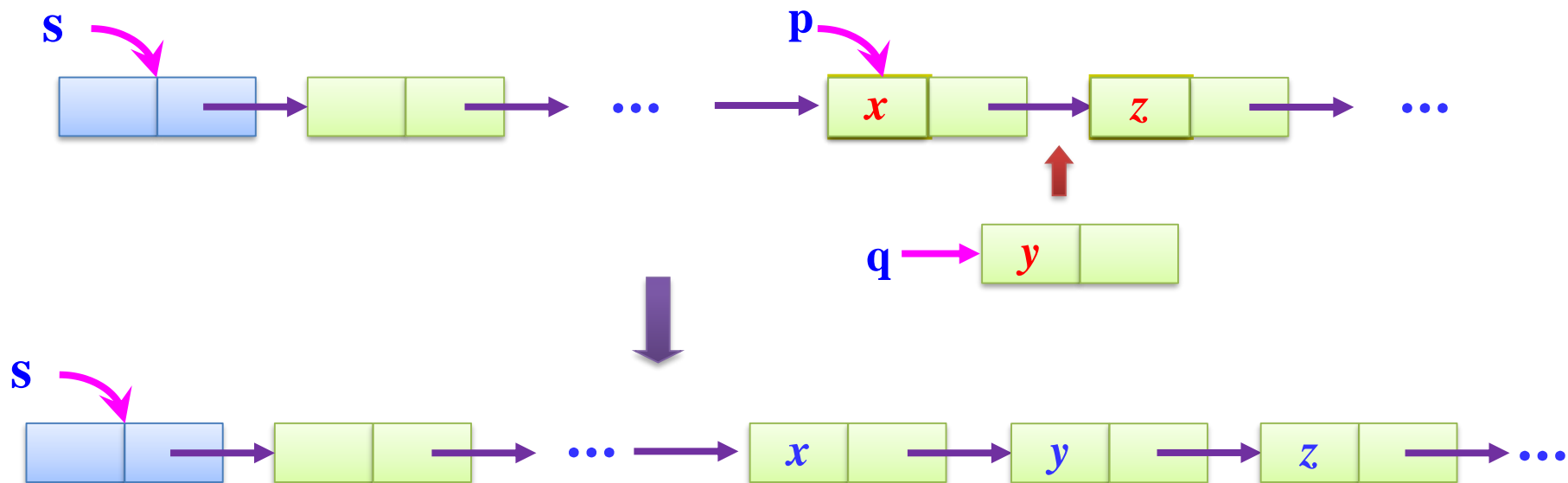
链串中实现串的基本运算与单链表的基本运算类似。详细算法实现参见教材。

【例4-2】 在链串中，设计一个算法把最先出现的子串“*ab*”
改为“*xyz*”。

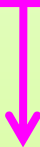
❶ 查找： $p \rightarrow \text{data} = 'a' \ \&\& \ p \rightarrow \text{next} \rightarrow \text{data} = 'b'$



② 替换



```
void Repl(LiString *&s)
{
    LiString *p=s->next,*q;
    int find=0;
    while (p->next!=NULL && find==0)           //查找ab子串
    {
        if (p->data==' a' && p->next->data=='b')
        {
            p->data='x'; p->next->data='z';
            q=(LiString *)malloc(sizeof(LiString));
            q->data='y'; q->next=p->next; p->next=q;
            find=1;
        }
        else p=p->next;
    }
}
```



替换为xyz

算法的时间复杂度为 $O(n)$ 。

——本讲完——