第三章 线性表-Part2

刘进超

人工智能学院

南開大學
Nankai University

目录

线性表的应用—— 多项式

一元m阶多项式

$$A(x) = a_{m-1}x^{e_{m-1}} + \cdots + a_0 x^{e_0}$$

由m个系数唯一确定。其中，a_i是非零系数，e_i非负整数，$e_{m-1} > e_{m-2} > \cdots > e_1 > e_0$。

该多项式可用线性表 $(a_0, a_1, a_2, \ldots, a_{m-1})$ 表示。其数据元素包含系数和指数两项。

一元多项式的相加

不失一般性，设有两个一元多项式：

$$P(x) = p_0 + p_1x + p_2x^2 + \ \dots + p_nx^n\ ,$$

$$Q(x) = q_0 + q_1x + q_2x^2 + \ \dots + q_mx^m \ \ (m < n)$$

$$R(x) = P(x) + \ Q(x)$$

$R(x)$由线性表$R((p_0 + q_0), (p_1 + q_1), \dots, (p_m + q_m), \dots, p_n)$唯一表示。

顺序存储表示的相加

```
typedef struct
{
    float  coef; //系数部分
    int    expn; //指数部分
} ElemType;
```

```
typedef  struct
{   ElemType  a[MAX_SIZE] ;
    int    length;
} Sqlist;
```

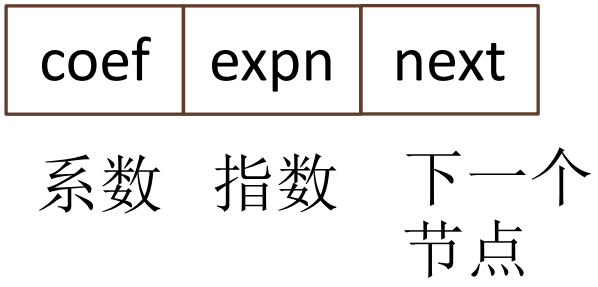
用顺序表示的相加非常简单。访问第5项可直接访问：

L.a[4].coef ， L.a[4].expn

可以直接使用expn值相同的coef值相加来得出多项式对应的系数。

多项式的链式存储表示

```
typedef struct poly
{
    float coef;     /*系数部分*/
    int   expn;     /*指数部分*/
    struct poly  *next;
} Poly,*PolyPointer;
```



$a = 3x^{14} + 2x^{8}+1$

$b = 8x^{14}-3x^{10}+10x^{6}$

链式存储表示的相加

当采用链式存储表示时，根据结点类型定义，凡是系数为0的项不在链表中出现，从而可以大大减少链表的长度。

一元多项式相加的实质是：

- 指数不同：**链表的合并**。
- 指数相同：**系数相加，和为0，去掉结点，和不为0，修改结点的系数域**。

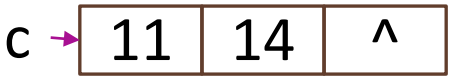
链式存储表示的相加算法

- 多项式a、b相加得c。从a和b头开始遍历链表
  - 若指数相同则系数相加，构造新节点接在c上；之后a和b同时移动
  - 若a当前系数小于b当前系数，则复制b当前项到c，b指针后移
  - 若a当前系数大于b当前系数，则复制a当前项到c，a指针后移
- 每次生成新节点，coef和expn赋值，接在c后

链式存储表示的相加

$a = 3x^{14} + 2x^{8}+1$

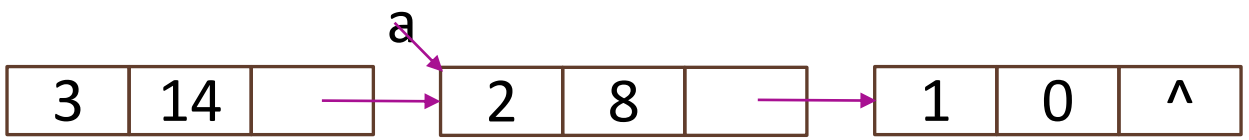
$b = 8x^{14}-3x^{10}+10x^{6}$



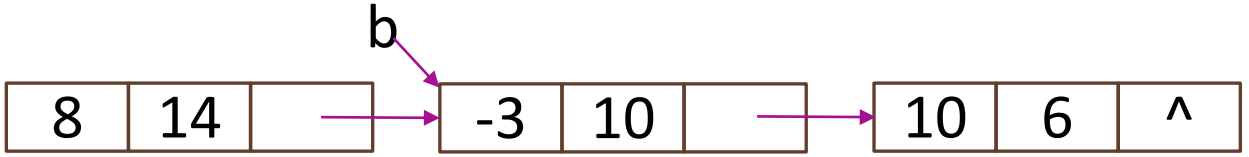
a->expn == b->expn

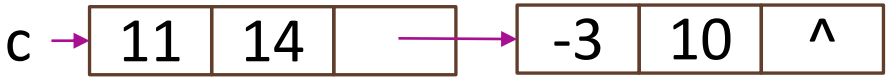
链式存储表示的相加

a = 3x14 + 2x8+1



b = $8x^{14}-3x^{10}+10x^{6}$

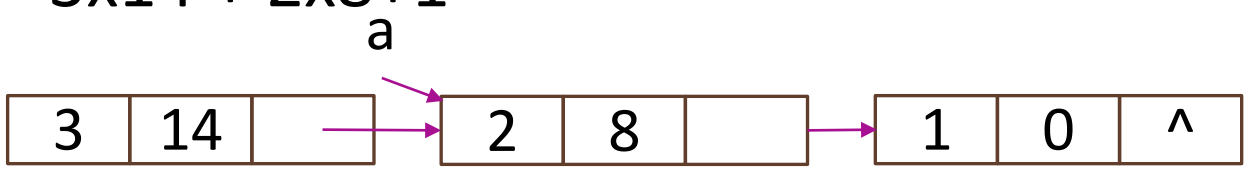




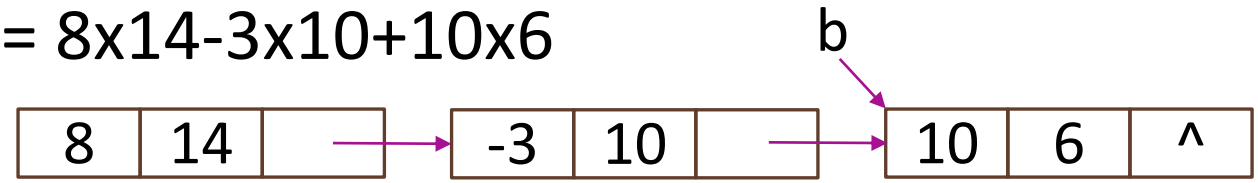
a->expn < b->expn

链式存储表示的相加

a = 3x14 + 2x8+1



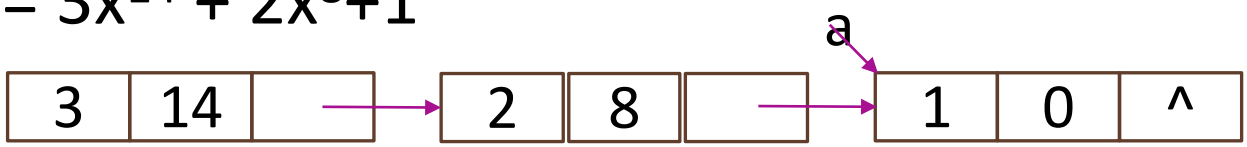
b = 8x14-3x10+10x6



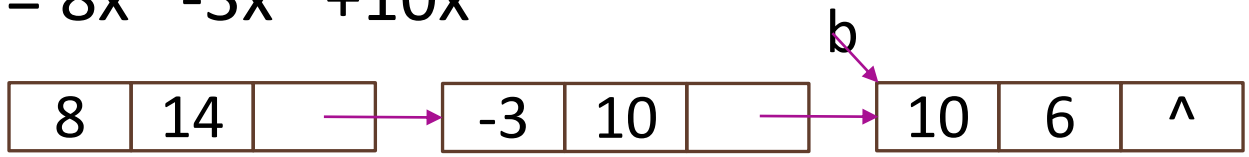
a->expn > b->expn

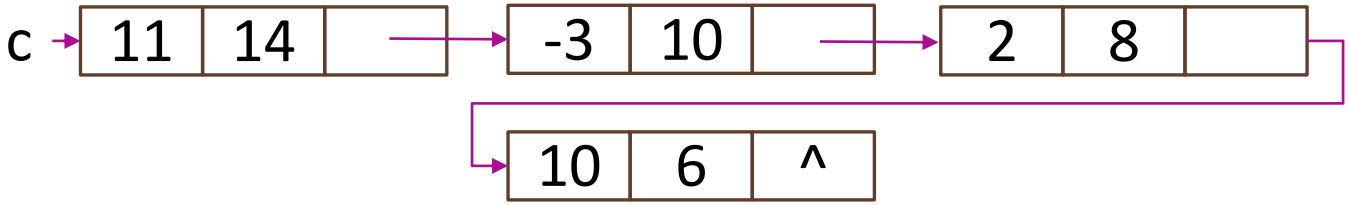
链式存储表示的相加

$a = 3x^{14} + 2x^{8}+1$



$b = 8x^{14}-3x^{10}+10x^{6}$

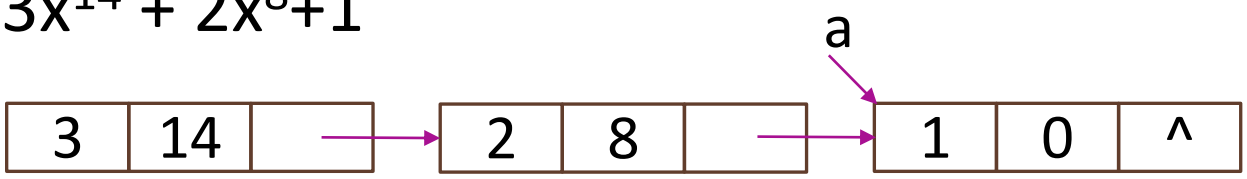




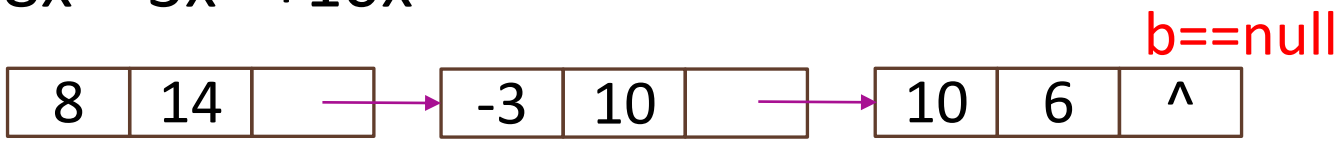
a->expn < b->expn

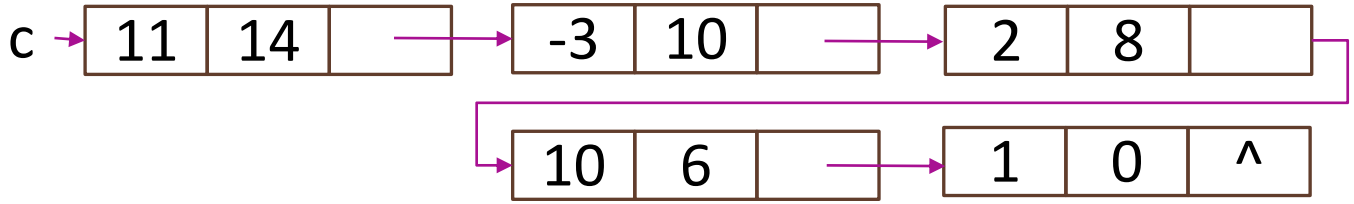
链式存储表示的相加

$a = 3x^{14} + 2x^{8}+1$



$b = 8x^{14}-3x^{10}+10x^{6}$





b==null

```
polyPointer padd(polyPointer a, polyPointer b)
{
    polyPointer c, rear, temp;
    int sum;
    rear = (PolyPointer)malloc(sizeof(Poly));
    c = rear;
    while (a && b) {//结点不为NULL时
        switch (compare(a->expn, b->expn))
        {
        case -1: //a的系数小于b的系数
            attach(b->coef, b->expn, &rear); //在rear后插入
            b = b->next;
            break;
        case 0://系数相等
            sum = a->coef + b->coef;
            if (sum != 0) :
                attach(sum, a->expn, rear);
            a = a->next;
            b = b->next;
            break;
```

```
int compare(int a, int b)
{
if (a == b)
    return 0;
else
    return a > b ? 1 : -1;
}
```

```
            case 1: //a的系数大于b的系数，将a结点的系数和指数拼接到
rear后
                attach(a->coef, a->expn, rear);
                a = a->next;
            }
        }
        //a、b链表有一个已经遍历到最后
        for (; a; a = a->next)
            attach(a->coef, a->expn, rear);
        for (; b; b = b->next)
            attach(b->coef, b->expn, rear);
        rear->next = null;
        temp = c;
        c = c->next;
        free(temp);
        return c;
}
```

```
void attach(float coef, int expn,
PolyPointer& c) {
    PolyPointer temp =
(PolyPointer)malloc(sizeof(Poly));
    temp->coef = coef;
    temp->expn = expn;
    c->next = temp;
    c = c->next;
}
```

字符串

在非数值处理、事务处理等问题常涉及到一系列的字符操作。计算机的硬件结构主要是反映数值计算的要求，因此，字符串的处理比具体数值处理复杂。下面讨论串的存储结构及几种基本的处理。

串类型的定义

串(字符串)：是零个或多个字符组成的有限序列。记作： S="$a_1a_2a_3$…"，其中S是串名，$a_i(1 \le i \le n)$是单个，可以是字母、数字或其它字符。

串值：双引号括起来的字符序列是串值。

串长：串中所包含的字符个数称为该串的长度。

串类型的定义

空串(空的字符串)：长度为零的串称为空串，它不包含任何字符。

空格串(空白串)：构成串的所有字符都是空格的串称为空白串。

注意：空串和空白串的不同，例如“ ”和“”分别表示长度为1的空白串和长度为0的空串。

串类型的定义

子串(substring)：串中任意个连续字符组成的子序列称为该串的子串，包含子串的串相应地称为主串。

子串的序号：将子串在主串中首次出现时的该子串的首字符对应在主串中的序号，称为子串在主串中的序号（或位置）。

串类型的定义

例如，设有串A和B分别是：

A=“xxf2aaa55a10a1xxf2aaa55a10a1xxx”

B=“aaa55a10a1”

则B是A的子串，A为主串。B在A中出现了两次，其中首次出现所对应的主串位置是4。因此，称B在A中的序号为4 。

特别地，空串是任意串的子串，任意串是其自身的子串。

串类型的定义

串相等：如果两个串的串值相等(相同)，称这两个串相等。换言之，只有当两个串的长度相等，且各个对应位置的字符都相同时才相等。

通常在程序中使用的串可分为两种：串变量和串常量。

串常量和整常数、实常数一样，在程序中只能被引用但不能改变其值，即只能读不能写。串变量和其它类型的变量一样，其值是可以改变。

串的ADT定义

ADT String{

- 数据对象：$D = \{ a_i | a_i \in CharacterSet,\ i = 1,2,\dots,n, n \geq 0 \}$
- 数据关系：$R = \{ < a_{i-1}, a_i > | a_{i-1}, a_i \in D,\ i = 2,3,\dots,n \}$
- 基本操作：
- StrAssign(t , chars)
- 初始条件： chars是一个字符串常量。
- 操作结果：生成一个值为chars的串t 。

串的ADT定义

- StrConcat(s, t)
- 初始条件：串s, t 已存在。
- 操作结果：将串t联结到串s后形成新串存放到s中。
- StrLength(t)
- 初始条件：字符串t已存在。
- 操作结果：返回串t中的元素个数，称为串长。

串的ADT定义

- SubString (s, pos, len, sub)
- 初始条件：串s已存在, 满足

  $1 \le pos \le StrLength(s)$且$0 \le len \le StrLength(s) - pos + 1$。
- 操作结果：用sub返回串s的第pos个字符起长度为len的子串。
- ……

} ADT String

串的存储表示与实现

串是一种特殊的线性表，其存储表示和线性表类似，但又不完全相同。串的存储方式取决于将要对串所进行的操作。串在计算机中有3种表示方式：

定长顺序存储表示：将串定义成字符数组，利用串名可以直接访问串值。用这种表示方式，串的存储空间在编译时确定，其大小不能改变。

串的存储表示与实现

堆分配存储方式：仍然用一组地址连续的存储单元来依次存储串中的字符序列，但串的存储空间是在程序运行时根据串的实际长度动态分配的。

块链存储方式：是一种链式存储结构表示。

串的定长顺序存储表示

串的顺序存储是用一组连续的存储单元来存放串中的字符序列。所谓定长顺序存储结构，是直接使用定长的字符数组来定义，数组的上界预先确定。

定长顺序存储结构定义为：

```
#define MAX_STRLEN  256
typedef  struct
{
   char  str[MAX_STRLEN];
   int  length;
} StringType;
```

串的联结操作

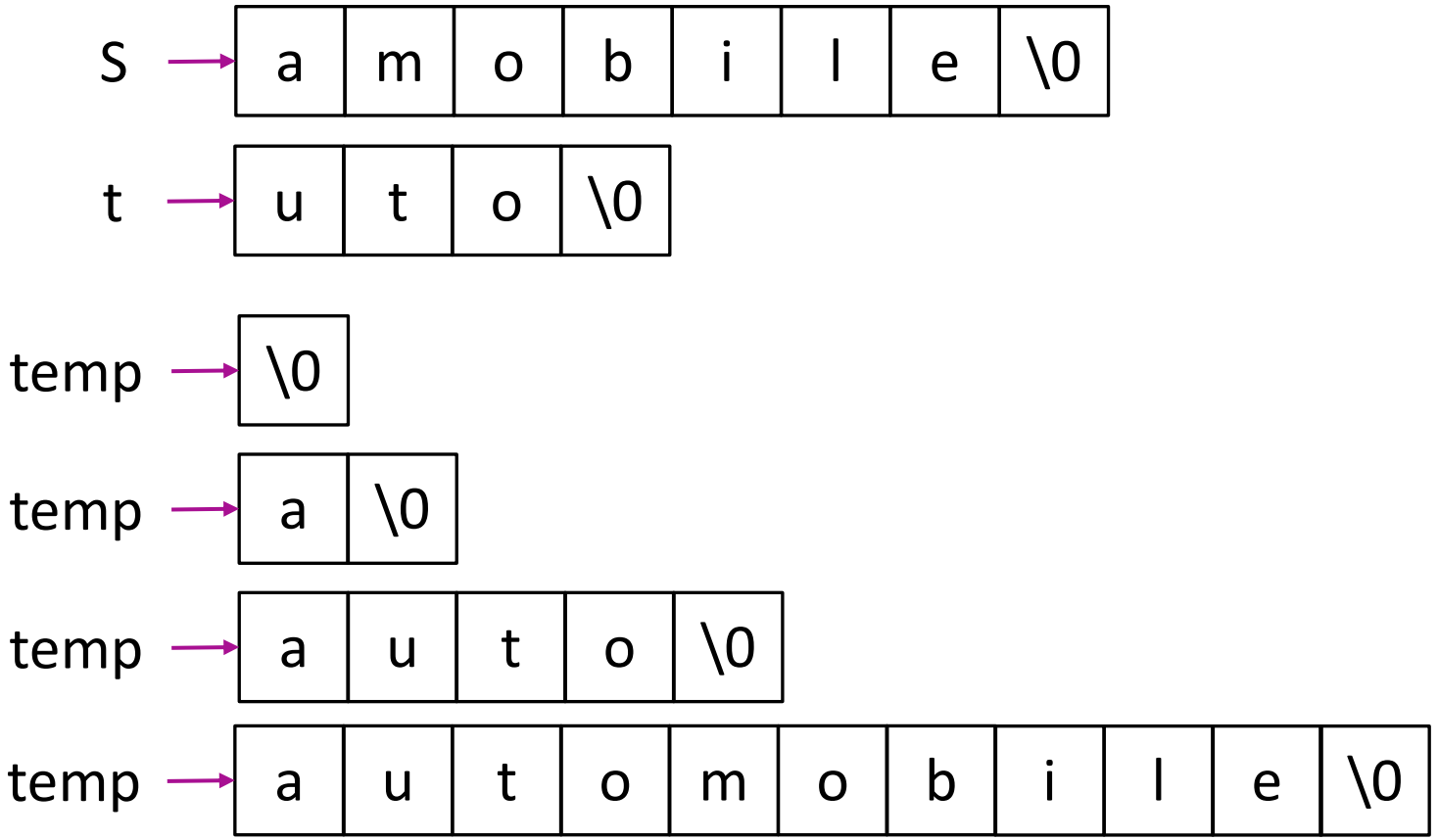
```
bool  StrConcat(StringType  &s, StringType t)
/*  将串t联结到串s之后，结果仍然保存在s中  */
{
    int i, j;
    if ((s.length + t.length)>MAX_STRLEN)
        Return 0; /*联结后长度超出范围 */
    for (i = 0; i<t.length; i++)
        s.str[s.length + i] = t.str[i];   /*  串t联结到串s
之后  */
    s.length = s.length + t.length;  /* 修改联结后的串长度
*/
    return 1;
}
```

求子串操作

```
//求子串操作
bool SubString(StringType s, int pos, int len,
StringType &sub)
{
    int k, j;
    if (pos<0 || pos>s.length || len<0 || len>(s.length
- pos + 1))
        return 0;    /*  参数非法  */
    sub->length = len;
    for (j = 0, k = pos; k <= pos + len - 1; k++, j++)
        sub->str[j] = s.str[k];    /*  逐个字符复制求得子
串  */
    return 1;
}
```

插入串

在字符串s的第一位后面插入字符串t：



插入串的实现

```
//插入串
bool insertString(StringType t1, StringType t2, int I,
StringType &S)
{
    StringType temp;
    if (i<0 && i>StrLength(t1))
        return 0;
    if (!StrLength(t1))
        StrConcat(t1, t2);
    else if (StrLength(t2))
    {
        SubString(t1, 1, i, &S);
        SubString(t1, i + 1,StrLength(t1) - i, &temp);
        StrConcat(S, t2);
        StrConcat(S, temp);
    }
    return 1;
}
```

串的堆分配存储表示

实现方法：系统提供一个空间足够大且地址连续的存储空间(称为“堆”)供串使用。可使用C语言的动态存储分配函数malloc()和free()来管理。

特点是：仍然以一组地址连续的存储空间来存储字符串值，但其所需的存储空间是在程序执行过程中动态分配，故是动态的，变长的。

串的堆分配存储表示

串的堆分配存储结构定义

```
typedef  struct
{
   char *ch;      /*  若非空，按长度分配，否则为
NULL */
   int length;          /*  串的长度  */
} HString;
```

向系统请求存储空间：

```
T.ch = (char *)malloc(sizeof(char)*T.length)
```

堆存储表示串的联结实现

```
bool StrConcat(HString  &T, HString *s1, HString *s2)
/*  用T返回由s1和s2联结而成的串  */
{
    int k, j, t_len;
    if (T.ch)  free(T);      /*  释放旧空间   */
    t_len = s1->length + s2->length;
    if (!(T.ch = (char *)malloc(sizeof(char)*t_len)))
    {
        printf("系统空间不够，申请空间失败！\n");
        return 0;
    }
    for (j = 0; j<s->length; j++)
        T->ch[j] = s1->ch[j];     /*  将串s复制到串T中 */
    for (k = s1->length, j = 0; j<s2->length; k++, j++)
        T->ch[k] = s2->ch[j];     /*  将串s2复制到串T中 */
    free(s1->ch);
    free(s2->ch);
    return 1;
}
```

串的链式存储表示

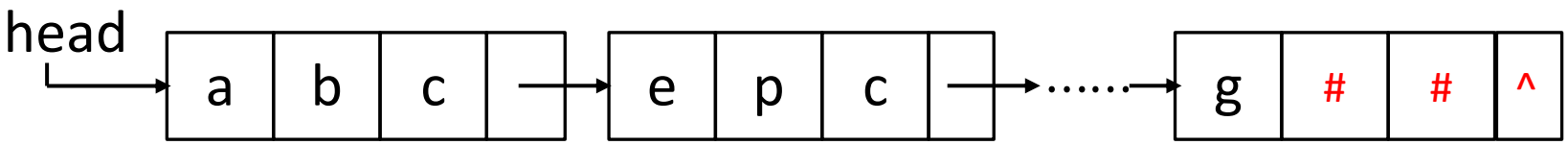
串的链式存储结构和线性表的串的链式存储结构类似，采用单链表来存储串，结点的构成是：

data域：存放字符，data域可存放的字符个数称为结点的大小；

next域：存放指向下一结点的指针。

串的链式存储表示

然而，若每个结点仅存放一个字符，则结点的指针域就非常多，造成系统空间浪费，为节省存储空间，考虑串结构的特殊性，使每个结点存放若干个字符，这种结构称为块链结构。如图所示，是块大小为3的串的块链式存储结构示意图。



块大小为3的串的块链式存储结构示意图

串的链式存储表示

串的块链式存储的类型定义包括：

```
//(1) 块结点的类型定义
#define CHUNK_SIZE  4
typedef  struct  Chunk
{
    char  data[CHUNK_SIZE];
    struct  Chunk *next;
}Chunk;

//(2)  块链串的类型定义
typedef  struct
{
    Chunk  head;    //头指针
    int  Strlen;    //当前长度
} Blstring;
```

在这种存储结构下，结点的分配总以完整的结点为单位，因此，为使一个串能存放在整数个结点中，在串的末尾填上不属于串值的特殊字符表示串的终结。

当一个块内存放多个字符时，往往会使操作过程变得较为复杂，如在串中插入或删除字符操作时通常需要在块间移动字符。