第 3 章　串、数组和广义表

3.1　内 容 要 点

3.1.1　串

串是非数值处理中的主要对象，在信息检索、文本编辑、符号处理等许多领域，得到越来越广泛的应用。串是特殊的线性表，它可以利用顺序存储结构也可以利用链式存储结构进行存储。串的顺序存储结构简称为顺序串，若用单链表方式存储串值，这种链式存储结构简称为链串。

1. 串的基本概念

串(String)是零个或多个字符组成的有限序列。一般记为 S="$a_1 a_2 \cdots\cdots a_n$"。其中，S 是串名。双引号括起的字符序列是串值；将串值括起来的双引号本身不属于串，它的作用是避免串与常数或标识符混淆。长度为零的串称为空串(Empty String)，它不包含任何字符。仅由一个或多个空格组成的串称为空白串(Blank String)。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。

2. 串的基本运算

串的基本运算如下：

(1) 串赋值 ASSIGN(s, t)和 CREAT(s, cs)，其中，t 为一个串的名字，cs 是一个字符序列。ASSIGN(s, t)的操作结果是将 t 的值赋给了 s；CREAT(s, cs)的操作结果为设定了一个串 s，其值为字符序列 cs。

(2) 求串长 LENGTH(s)，返回串 s 的长度。

(3) 串连接 CONCAT(to, from)，将串 from 复制到 to 串的末尾，并返回串 to。

(4) 串比较 Equal(s1, s2) 比较串 s1 和串 s2 是否相等，如果相等，则返回 1(或 TRUE)，否则返回 0(或 FALSE)。

(5) 串定位 INDEX(s, t) 如果主串 s 中存在和 t 相等的子串，则返回 s 中第一个这样的子串在主串 s 中的位置，否则返回 0。

(6) 求子串 SUBSTR(s, start, len)，如果 1≤start≤LENGTH(S)+1 且 0≤len≤LENGTH(S)−start+1，则返回由串 s 中第 start 个字符，长度为 len 的字符串，否则返回一个特殊的串常量。

(7) 串替换操作 REPLACE(s, t, v) 操作结果是以串 v 替换串 s 中出现的和非空串 t 相等的不重叠的子串。

(8) 插入操作 INSERT(s, pos, t),当 1≤pos≤LENGTH(S)+1 时,在串 s 的第 pos 个字符之间插入串 s。

(9) 删除操作 DELETE(s, pos, len),当 1≤pos≤LENGTH(S)且当 0≤len≤LENGTH(S)−pos+1 时,从串 s 中删去从第 pos 个字符起长度为 len 的子串。

3. 串的主要存储结构

与线性表一样,串的存储亦分为顺序存储及链式存储两种存储方式,但由于串的特殊性,对于串的存储一般由如下几种方式:

1) 定长字符串的存储表示

为每一个串分配一个固定大小的存储空间。其使用 C 语言表示如下:

```
#define    STRING_LEN    80              /*此为字符串的最大长度,暂设为 80*/
typedef struct{
    char datas[STRING_LEN];
    int len;
}BuffString;
```

为每个串分配一个固定大小的(STRING_LEN)的存储空间,实际串长度由变量 len 记录。如此存储的串具有如下特点:

(1) 存储的串长有限制,不得超过 STRING_LEN;

(2) 当串较短时,则浪费存储空间;

(3) 对串链接、子串替换等操作需要注意,当结果串超过规定的最大长度,对超长的串进行截断处理。

使用这样的方式存储串时,也可以不使用 len 存储串的长度,将所有串皆视为长度为 STRING_LEN 的串,当实际长度小于 STRING_LEN 时,尾部使用空格补齐,如果实际串的长度超过 STRING_LEN 时,超出部分被截断。当 STRING_LEN≤255 时,亦可不使用 len 记录串的长度。而将串的长度存储在第一个字符存储单元中,即 datas[0]为串的长度。

串的顺序存储结构简称为顺序串。与顺序表类似,顺序串是用一组地址连续的存储单元来存储串中的字符序列。因此可用高级语言的字符数组来实现。

2) 变长字符串的存储表示

在实际应用中,串的长度的变化较大,使用定长方式处理是不合适的,特别是对信息处理类的应用程序,大多使用动态方式存储可变长度的串。根据需要分配其存储空间,当使用完毕后,释放其占用的存储空间。将处理串中的符号存储在一个称为堆的公共存储空间中,串中需要记录其第一个字符在堆中的位置。ANSI C 在串结束时使用一个空字符('\0')表示串结束,其存储表示定义如下:

```
typedef struct{
    char *datas;
}HeapString;
```

采用此方式存储字符串,必须注意使用串前需要对其初始化,并且串使用完毕后需要

释放串占用的存储空间。采用上面的方式处理串(即以空字符表示串结束)时,求其串长需要遍历整个串,对串链接或串替换等操作需要释放串的已分配存储空间,并为其分配新的存储空间(见 3.2 节基础实验之实验二)。

3) 链串

与线性表一样,亦可使用链式存储结构存储串。但由于每个字符在内存中占一个字节,而一个指针在 16 位系统中占两个字节,在 32 位系统中占四个字节,在 64 位系统中占 8 个字节,当使用链式存储结构存储串时,链表的有效存储空间分别占 1/3、1/5 及1/9,显然存储空间利用率较低。为了提高存储空间的利用率,将若干个符号组成一个块,构成分块链式结构,其使用 C 语言描述如下:

```
#define    BLOCK_SIZE 100                    /* 可根据需要取值 */
struct BlocoStringNode{
    char buff[BLOCK_SIZE];
    struct BlockStringNode *pNext;
};
```

使用块链式存储时,对于每个块(特别是最后一个块)来说,可能未填满字符,对于未填满字符的块其最后一个字符后面增加一个空字符('\0'),更一般地,每个块中的字符皆以空字符结尾。当块大小为 1 时,即为普通链式存储表示,当块的大小充分大(大于所处理的串的长度)时,其转换为固定长度串。块的大小根据实际情况及空间利用率确定。

3.1.2 数组

数组和广义表是一种复杂的非线性结构,它们的逻辑特征是:一个数据元素可能有多个直接前驱和多个直接后继。

1. 数组的定义

数组是由相同类型的元素组成的一个有限序列。数组按维数来划分,对于 $m(m\geqslant 1)$ 维数组,每个元素受 m 个线性关系的约束,所以数组是线性表的推广,一维数组可以看成是一个线性表;二维数组可以看成元素是线性表的线性表,依次类推。n 维数组可以看成是元素为 $n-1$ 维数组的线性表。

2. 数组的顺序存储方式

由于数组一般不作插入与删除操作,只是作访问数据元素操作,因此,使用顺序存储结构较为合适。

由于计算机内存是一维的,多维数组的元素应排成线性序列后存入在计算机的内存中。对于一个 $m\times n$ 的二维数组,可以将按照行的顺序排列,即将二维数组的所有元素排列成如下序列:

$$a_{11},\ a_{12},\ \cdots,a_{1n},\ a_{21},\ a_{22},\cdots,a_{2n},\ \cdots,\ a_{m1},\ a_{m2},\cdots,\ a_{mn}$$

以这种方式存储的二维数组称为行主序存储(或行主序排列)。显然亦可将二维数组按照列的顺序排列,即将二维数组的所有元素排列成如下序列:

$$a_{11},\ a_{21},\ \cdots,a_{m1},\ a_{12},\ a_{22},\cdots,a_{m2},\ \cdots,\ a_{1n},\ a_{2n},\cdots,\ a_{mn}$$

这种方式存储的二维数组称为列主序存储(或列主序排列)。以 Loc(a_{ij})表示数组元素 a_{ij}的存储地址,l 表示数组元素占有的字节数,则对于行主序存储的数组来说,其存储地址 Loc(a_{ij})满足

$$Loc(a_{ij}) = Loc(a_{11}) + ((i-1)*n+j-1)*l$$

而对列主序存储的数组,Loc(a_{ij})满足

$$Loc(a_{ij}) = Loc(a_{11}) + ((j-1)*m+i-1)*l$$

对 n 维数组可采用类似的方式进行处理(略)。

3. 特殊矩阵的压缩存储

二维数组亦称矩阵,对矩阵一般采用二维数组方式来存储。但当矩阵中的非 0 元素值分布有规律时,该矩阵称为特殊矩阵。为了节省存储空间,利用非 0 元素的规律对于其进行压缩存储。以下讨论行主序存储特殊矩阵的压缩存储方法及地址计算公式。

1) 对称矩阵的压缩存储

所谓对称矩阵是一个 $n \times n$ 矩阵,其元素满足如下关系:

$$a_{ij} = a_{ji} \quad 1 \leqslant i \leqslant n, \quad 1 \leqslant j \leqslant n$$

对于对称矩阵,只需存储其对角线及对角线以下元素即可。因此只需 $n \times (n+1)/2$ 个元素的存储空间,而非 $n \times n$ 个元素的存储空间。其地址计算如下:

$$Loc(a_{ij}) = Loc(a_{11}) + (i*(i-1)/2+j-1)*l \quad \text{其中 } 1 \leqslant j \leqslant i \leqslant n$$

2) 三对角矩阵的压缩存储

一个 $n \times n$ 阶矩阵称为三对角矩阵,如果其元素满足如下条件:

$$a_{ij} = 0 \quad \text{如果 } |i-j| > 1$$

显然,三对角矩阵仅含有 $3 \times (n-2)+4$ 个非 0 元素。其非 0 元素地址计算公式如下:

$$Loc(a_{ij}) = Loc(a_{11}) + ((i-2)*3+3+j-i)*l \quad \text{其中 } |i-j| \leqslant 1$$

4. 稀疏矩阵的压缩存储

当矩阵的元素分布没有规律,但其非零元素的个数远远小于矩阵元素总数,即为稀疏矩阵时,为了节省存储单元,可只存储非零元素。在存储非零元素的同时,还必须存储非零元素所在的行号、列号,才能确定一个非零元素在矩阵中的位置。稀疏矩阵的压缩存储会失去随机存取功能。其中每一个非零元素所在的行号、列号和值组成一个三元组(i,j,aij),并由此三元组唯一确定。将表示稀疏矩阵的非零元素的三元组按行优先(或列优先)的顺序排列(跳过零元素),并依次存放在向量中,这种稀疏矩阵的顺序存储结构称为三元组表。

3.1.3　广义表

广义表是线性表的推广。线性表是指元素个数为 $n(n \geqslant 0)$的有限序列,线性表中的元素均具有相同的数据类型。线性表中的元素一般是固定类型,它可以是一个基本数据类型或一个结构体类型。若放松对表元素的这种限制,容许它们具有其自身结构,这样就产生了广义表的概念。

1. 广义表的概念

广义表 GList 定义如下:

GList 是空表(没有任何元素)

或

GList=(H,T)

其中 H 称为广义表的表头,是一个原子元素或是一个广义表元素;T 称为表尾,是一个广义表。

由此可见,广义表是 $n(n\geqslant 0)$个元素 $a_1,a_2,\cdots,a_n$ 的有限序列,其中 a_i 或者是原子元素,或者是一个广义表。

2. 广义表的存储结构

由于广义表中的元素本身又可以具有结构,是一种带有层次的非线性结构,因此难以用顺序存储结构表示。通常采用链式存储结构,每个元素可用一个结点表示。结点的结构如下:

tag	data/slink	link

每个结点有三个域构成,其中 tag 是一个标志位,用来区分当前结点是原子还是子表。当 tag 为 0 时,该结点是子表,第二个域为 slink,用以存放子表的地址;当 tag 为 1 时,该结点是原子结点,第二个域为 data,用于存放原子元素的值。link 域用来存放与本元素同一层的下一个元素对应结点的地址,当该元素是所在层的最后一个元素时,link 的值为 NULL。

```
typedef enum{atom,list}NodeTag;
typedef char DataType;                    /* 可用实际类型替代 */
typedef struct GLnode{
  NodeTag tag;
  union{
      DataType *dlink;
      struct GLnode *slink;
  };
  struct GLnode *link;
}GLNode,*GList;
```

3. 广义表的基本运算

广义表是线性表和树的推广。具有共享和递归特性的广义表可以和有向图建立对应关系,因此,广义表的大部分运算与这些数据结构上的运算类似。

广义表的两个特殊的基本运算：取表头 head(GL)和取表尾 tail(GL)。

根据表头、表尾的定义可知,任何一个非空广义表的表头是表中第一个元素,它可以是原子,也可以是子表,而其表尾必定是子表。

3.2 基 础 实 验

3.2.1 实验目的

(1) 掌握串的顺序存储方法、链式存储方法。

(2) 掌握基于顺序存储结构的串的模式匹配算法。
(3) 掌握基于链式存储结构的串的模式匹配算法。
(4) 掌握基于块链式存储结构的串的模式匹配算法。
(5) 掌握矩阵的三元组存储方法及有关的算法。
(6) 培养学生利用串和数组的知识编写应用程序的能力。
(7) 掌握广义表的存储方法。
(8) 掌握广义表的有关算法。

3.2.2　实验内容

实验一　固定长度的顺序字符串基本操作实验

【实验内容与要求】

为字符串分配一个固定长度的存储空间,完成有关字符串的基本操作:字符串链接、字符串相等比较、取字符串长度、删除子串、字符串替换、定位子串及取子串。

【实验目的】

通过实验深入理解为字符串分配固定长度的优缺点,并掌握相应的基本操作。

【实现提示】

为字符串分配固定长度的存储空间,需要有一个充分长的字符数组,除此之外还要有一种办法知道字符串的长度或者说字符串何时结束。标识字符串结束或字符串的长度有3种方法:使用一个整数标识字符串长度;在字符串尾部添加一个特殊字符(空字符'\0')或者所有字符串长度皆相同,不足部分用空格补齐,本示例程序中使用记录字符串长度的方法标识字符串。对固定长度的字符串进行操作需要注意的是字符串越界问题,这里采取截断方法,即如果串链接等操作结果超过分配的长度,则截断超过的部分,只保留规定长度部分的子串。

【参考程序】

```
/*////////////////////////////////////////////////
//     有关固定长度字符串的结构定义及操作说明
//     文件名是:StaticString.h
////////////////////////////////////////////////*/
#ifndef _STATIC_STRING_H__
#define _STATIC_STRING_H__
#define MAX_SIZE    15                  /*处理字符串的最大长度*/
typedef enum {FALSE,TRUE} BOOL;
typedef struct {
    char buff[MAX_SIZE];
    int len;
}StaticString;

/*/////////////////有关字符串的操作///////////////*/
```

```
BOOL CreateStaticString(StaticString *pstr,char *cstr);
void AssignStaticString(StaticString *ptStr,StaticString *sStr);
int LengOfStaticString(StaticString *pStr);
int IndexInStaticString(StaticString *pStr,StaticString *sStr);
int IndexInStaticStringA(StaticString *pStr,StaticString *sStr,int pos);
void CopyStaticString(StaticString *ptStr,StaticString *psStr);
BOOL ConcatStaticString(StaticString *tStr,StaticString *ptStr);
BOOL ReplaceStaticString(StaticString *ptStr,int pos,int len,StaticString *
psStr);
void DeleteStaticString(StaticString *ptStr,int pos,int len);
BOOL EqualStaticString(StaticString *pStr,StaticString *qStr);
void DisplayStaticString(StaticString *str);
BOOL SubstringStaticString(StaticString *sStr,int pos,int len,StaticString *
subStr);
#endif
/*////////////////////////////////////////////////
//      文件名 StaticString.h 结束
////////////////////////////////////////////////*/

/*////////////////////////////////////////////////////
//    有关固定长度字符串的操作实现
//    文件名是:StaticString.c
////////////////////////////////////////////////////*/
#include "StaticString.h"
#include<stdio.h>
BOOL CreateStaticString(StaticString *pstr,char *cstr)
{
    int i;
    char *p=pstr->buff;
    pstr->len=0;
    for(i=0;i<MAX_SIZE && *cstr!='\0';i++,pstr->len++)
        *p++=*cstr++;
    if(*cstr=='\0')
        return TRUE;
    return FALSE;
}
void AssignStaticString(StaticString *ptStr,StaticString *sStr)
{
    int i;
    for(i=0;i<sStr->len;i++)
        ptStr->buff[i]=sStr->buff[i];
    ptStr->len=sStr->len;
}
int LengOfStaticString(StaticString *pStr)
```

```
{
    return pStr->len;
}
int IndexInStaticString(StaticString *pStr,StaticString *sStr)
{
    return IndexInStaticStringA(pStr,sStr,1);
}
int IndexInStaticStringA(StaticString *pStr,StaticString *sStr,int pos)
{
    char *p,*q;
    pos--;
    while(pos<pStr->len)
    {
        for(p=pStr->buff+pos,q=sStr->buff;q<sStr->buff+sStr->len;p++,q++)
            if(*p!=*q) break;
        if(q==sStr->buff+sStr->len) break;
        pos++;
    }
    if(pos==pStr->len)
        return 0;
    return pos+1;
}
void CopyStaticString(StaticString *ptStr,StaticString *psStr)
{
    char *p,*q;
    for(p=ptStr->buff,q=psStr->buff;q<psStr->buff+psStr->len;p++,q++)
        *p=*q;
    ptStr->len=psStr->len;
}
BOOL ConcatStaticString(StaticString *ptStr,StaticString *psStr)
{
    int b;
    char *p=ptStr->buff+ptStr->len,*q=psStr->buff;
    int len=(b=ptStr->len+psStr->len>MAX_SIZE)?MAX_SIZE:ptStr->len+psStr->len;
    while(p<ptStr->buff+len)
    {
        *p++=*q++;
    }
    ptStr->len=len;
    return b==1?FALSE:TRUE;
}
BOOL ReplaceStaticString(StaticString *ptStr,int pos,int len,StaticString *
psStr)
{
```

```
    BOOL b;
    int newLen;
    char *p,*q;

    if(pos-1+psStr->len>MAX_SIZE)
    {
        for(p=ptStr->buff+pos-1,q=psStr->buff;p<ptStr->buff+MAX_SIZE;p++,q++)
            *p=*q;
        ptStr->len=MAX_SIZE;
        b=FALSE;
    }
    else
    {
        if(len+pos-1>ptStr->len)len=ptStr->len-pos+1;
                                                    /*计算实际替换的子串长度*/
        if(psStr->len+ptStr->len-len>MAX_SIZE)   /*计算替换后字符串的剩余长度*/
        {
            b=FALSE;
            newLen=MAX_SIZE-psStr->len-pos+1;

        }
        else{
            b=TRUE;
            newLen=ptStr->len-pos-len+1;

        }
    for(p=ptStr->buff+pos+len+newLen-1,q=ptStr->buff+pos+psStr->len+
    newLen-1;p>=ptStr->buff+pos+len-1;p--,q--)
            *q=*p;
        for(p=ptStr->buff+pos-1,q=psStr->buff;q<psStr->buff+psStr->len;p++,q++)
            *p=*q;
    }
    if(b==TRUE)
        ptStr->len=ptStr->len+psStr->len-len;
    else
        ptStr->len=MAX_SIZE;
    return b;
}
void DeleteStaticString(StaticString *ptStr,int pos,int len)
{
    char *p,*q;
    if(pos+len-1>ptStr->len)
        ptStr->len=pos-1;
    else{
```

```
        for(p=ptStr->buff+pos-1,q=ptStr->buff+pos+len-1;p<ptStr->buff+
        ptStr->len;p++,q++)
            *p=*q;
        ptStr->len-=len;
    }
}
BOOL EqualStaticString(StaticString *pStr,StaticString *qStr)
{
    char *p,*q;
    if(pStr->len==qStr->len)
        for(p=pStr->buff,q=qStr->buff;p<pStr->buff+pStr->len;p++,q++)
            if(*p!=*q) break;
    if(p==pStr->buff+pStr->len)
        return TRUE;
    return FALSE;
}
void DisplayStaticString(StaticString *str)
{
    int i;
    for(i=0;i<str->len;i++)
        putchar(str->buff[i]);
}
BOOL SubstringStaticString(StaticString *sStr,int pos,int len,StaticString *
substr)
{
    char *p,*q;
    if(pos<1 && pos>sStr->len) return FALSE;      /*字符串开始位置错误*/
    if(len<0) len=0;
    len=len+pos-1>sStr->len?sStr->len-pos+1:len;
    substr->len=len;
    for(p=sStr->buff+pos-1,q=substr->buff;p<sStr->buff+pos-1+len;p++,q++)
        *q=*p;
    return TRUE;
}
/*////////////////////////////////////////////
//      文件名 StaticString.c 结束
////////////////////////////////////////////////*/

/*////////////////////////////////////////////////
//     实验一主程序
//     文件名是:Ch03S1Exp1.c
////////////////////////////////////////////////*/
#include "StaticString.h"
#include<stdio.h>
```

```
#include<conio.h>
#include<windows.h>

void Link();
void StringEqual();
void Length();
void Delete();
void Replace();
void Index();
void substring();
void main_3_1_1()
{
    int idx;
    char str[81]="按任意键继续...";

    while(1)
    {
        system("CLS");
        printf("      固定长度顺序字符串基本操作实验\n");
        printf("1. 字符串链接          2. 字符串相等比较\n");
        printf("3. 取字符串长度        4. 删除子串\n");
        printf("5. 字符串替换          6. 定位子串\n");
        printf("7. 取子串              0. 退出\n");
        printf("请选择(0~7):");
        idx=getch();
        printf("\n");
        switch(idx)
        {
        case '1':
            Link();
            break;
        case '2':
            StringEqual();
            break;
        case '3':
            Length();
            break;
        case '4':
            Delete();
            break;
        case '5':
            Replace();
            break;
        case '6':
```

```
            Index();
            break;
        case '7':
            substring();
            break;
        case '0':
            exit(0);
        default:
            strcpy(str,"非法输入,请输入正确的选择(0~6),按任意键继续...");
        }
        printf("%s",str);
        getch();
    }
}
static void Link()
{
    StaticString str1,str2;              /*定义两个固定长度顺序字符串*/
    char buff[81];
    printf("请输入一个长度小于80的字符串(以回车结束):");
    gets(buff);
    CreateStaticString(&str1,buff);      /*根据一个C字符串建立一个固定顺序串*/
    printf("请输入另一个长度小于80的字符串(以回车结束):");
    gets(buff);
    CreateStaticString(&str2,buff);      /*根据一个C字符串建立一个固定顺序串*/
    if(ConcatStaticString(&str1,&str2)==TRUE)
        printf("链接成功,链接后的字符串是:");
    else
        printf("链接后的字符串超过长度,字符串被截断,截断后的字符串如下:");
    DisplayStaticString(&str1);
    printf("\n");
}
static void StringEqual()
{
    StaticString str1,str2;              /*定义两个固定长度顺序字符串*/
    char buff[81];
    printf("请输入一个长度小于80的字符串(以回车结束):");
    gets(buff);
    CreateStaticString(&str1,buff);      /*根据一个C字符串建立一个固定顺序串*/
    printf("请输入另一个长度小于80的字符串(以回车结束):");
    gets(buff);
    CreateStaticString(&str2,buff);
    if(EqualStaticString(&str1,&str2)==TRUE)
        printf("输入的两个字符串相等。\n");
    else
```

```
        printf("输入的两个字符串不相等。\n");
}
static void Length()
{
    StaticString str1;                    /*定义一个固定长度顺序字符串*/
    char buff[81];
    printf("请输入一个长度小于80的字符串(以回车结束):");
    gets(buff);
    CreateStaticString(&str1,buff);
    printf("字符串  ");
    DisplayStaticString(&str1);
    printf(" 的长度是:%d\n",LengOfStaticString(&str1));
}
static void Delete()
{
    StaticString str1;                    /*定义一个固定长度顺序字符串*/
    int pos,len;
    char buff[81];
    printf("请输入一个长度小于80的字符串(以回车结束):");
    gets(buff);
    CreateStaticString(&str1,buff);
    printf("请输入删除字符串的开始位置(以1开始)及长度:");
    scanf("%d%d",&pos,&len);
    if(pos<0 || len<0)
        printf("删除位置或长度错误。\n");
    else{
        DeleteStaticString(&str1,pos,len);
        printf("删除后的字符串是:");
        DisplayStaticString(&str1);
        printf("\n");
    }
    gets(buff);
}
static void Replace()
{
    StaticString str1,str2;               /*定义两个固定长度顺序字符串*/
    int pos,len;
    char buff[81];
    printf("请输入一个长度小于80的字符串(以回车结束):");
    gets(buff);
    CreateStaticString(&str1,buff);       /*根据一个C字符串建立一个固定顺序串*/
    printf("请输入替换字符串及替换位置、长度:");
    scanf("%s%d%d",buff,&pos,&len);
    if(pos<0 || len<0){
```

```
        printf("替换位置、替换长度不能小于 0。\n");
    }else{
        CreateStaticString(&str2,buff);
        if(ReplaceStaticString(&str1,pos,len,&str2)==TRUE)
            printf("替换成功,替换后的字符串是:");
        else
            printf("替换后字符串超长,截断后的字符串是:");
        DisplayStaticString(&str1);
    }
    gets(buff);

}
static void Index()
{
    StaticString str1,str2;              /*定义两个固定长度顺序字符串*/
    int pos;
    char buff[81];
    printf("请输入一个长度小于 80 的字符串(以回车结束):");
    gets(buff);
    CreateStaticString(&str1,buff);      /*根据一个 C 字符串建立一个固定顺序串*/
    printf("请输查找的子串及开始位置:");
    scanf("%s%d",buff,&pos);
    if(pos<0 ){
        printf("查找位置不能小于 0。\n");
    }else{
        CreateStaticString(&str2,buff);
        printf("字符串<");
        DisplayStaticString(&str1);
        printf(">在字符串<");
        DisplayStaticString(&str2);
        printf(">中的位置是%d\n",IndexInStaticStringA(&str1,&str2,pos));
    }
    gets(buff);
}

static void substring()
{
    StaticString str1,str2;              /*定义两个固定长度顺序字符串*/
    int pos,len;
    char buff[81];
    printf("请输入一个长度小于 80 的字符串(以回车结束):");
    gets(buff);
    CreateStaticString(&str1,buff);      /*根据一个 C 字符串建立一个固定顺序串*/
```

```
        printf("请输入子串的开始位置及长度:");
        scanf("%d%d",&pos,&len);
        if(pos<0 || len<0){
            printf("替换位置、替换长度不能小于 0。\n");
        }else{
            SubstringStaticString(&str1,pos,len,&str2);
            printf("字符串<");
            DisplayStaticString(&str1);
            printf(">位于%d,长度为%d 的子串是<",pos,len);
            DisplayStaticString(&str2);
            printf(">\n");
        }
    }
/*//////////////////////////////////////////
//      文件名 Ch03S1Exp1.c 结束
//////////////////////////////////////////*/
```

【思考与提高】

(1) 如果以空格补齐字符串，如何完成本实验？

(2) 字符串的顺序存储可以视为线性表的顺序存储表示，比较固定长度字符串与顺序表的相关操作的异同。

实验二　动态顺序字符串基本操作实验

【实验内容与要求】

根据字符串的长度为字符串分配一个动态存储空间，完成字符串的基本操作：字符串链接、字符串相等比较、取字符串长度、删除子串、字符串替换、定位子串及取子串。

【实验目的】

通过实验深入理解 C 字符串的特点及操作，掌握 C 语言动态内存管理的应用。

【实现提示】

为了封装，定义动态顺序字符串的结构如下：

```
typedef struct{
    char *pdata;
} DynamicString;
```

pdata 是指向串的指针，字符串以空字符('\0')结束。求串长需要扫描整个串到空字符结束，完成串链接等操作，需要计算出新串的长度，然后为新串分配存储空间，将相关字符复制到新串中后，释放旧串占用的存储空间。

【参考程序】

```
/*//////////////////////////////////////////////////
//    有关可变长度字符串的结构定义及操作说明
```

```
//     文件名是:DynamicString.h
////////////////////////////////////////////////////*/
#ifndef _DYNAMIC_STRING_H__
#define _DYNAMIC_STRING_H__
#define    NULL 0
typedef enum {FALSE,TRUE} BOOL;
typedef struct{
    char *pdata;
} DynamicString;
/*//////////////////有关字符串的操作///////////////*/
void InitDynamicString(DynamicString *pstr);
BOOL CreateDynamicString(DynamicString *pstr,char *cstr);
BOOL AssignDynamicString(DynamicString *ptStr,DynamicString str);
int LengOfDynamicString(DynamicString str);
int IndexInDynamicString(DynamicString str,DynamicString substr);
int IndexInDynamicStringA(DynamicString str,DynamicString substr,int pos);
BOOL ConcatDynamicString(DynamicString *tStr,DynamicString str);
BOOL ReplaceDynamicString(DynamicString *pStr,int pos,int len,DynamicString
str);
BOOL DeleteDynamicString(DynamicString str,int pos,int len);
BOOL EqualDynamicString(DynamicString str1,DynamicString str2);
void DisplayDynamicString(DynamicString str);
void DestroyDynamicString(DynamicString *str);
DynamicString SubstringDynamicString(DynamicString str,int pos,int len);
#endif
/*////////////////////////////////////////////////
//      文件名 DynamicString.h 结束
////////////////////////////////////////////////////*/

/*////////////////////////////////////////////////
//     有关可变长度字符串的操作实现
//     文件名是:DynamicString.c
//       示例程序只给出部分函数的实现
////////////////////////////////////////////////////*/
/*初始化可变长度字符串,在串使用前必须初始化*/
void InitDynamicString(DynamicString *pstr)
{
    pstr->pdata=NULL;
}
/*求串的长度*/
int LengOfDynamicString(DynamicString str)
{
    int len;
    for(len=0;*(str.pdata+len)!='\0';len++);
```

```
    return len;
}
/*串相等差别*/
BOOL EqualDynamicString(DynamicString str1,DynamicString str2)
{
    char *p,*q;
    for(p=str1.pdata,q=str2.pdata;*p!='\0';p++,q++)
        if(*p!=*q) return FALSE;
    if(*p!=*q)
        return FALSE;
    return TRUE;
}
/*串替换操作,每一个字符的位置为1*/
BOOL ReplaceDynamicString(DynamicString *pStr,int pos,int len,DynamicString
str)
{
    int len1=LengOfDynamicString(*pStr),len2=LengOfDynamicString(str);
    char *p,*q,*t;
    pos--;
    if(pos<0 && pos>=len1)              /*替换位置错误*/
        return FALSE;
    len=pos+len<=len1?len:len1-pos;    /*计算被替换串的长度*/
    t=(char *)malloc(sizeof(char)*(len1+len2-len+1));
    if(t==NULL)                         /*溢出*/
        return FALSE;
    for(p=t,q=pStr->pdata;q<pStr->pdata+pos;p++,q++)
        *p=*q;
    for(q=str.pdata;*q!='\0';p++,q++)
        *p=*q;
    for(q=pStr->pdata+pos+len;*q!='\0';q++,p++)
        *p=*q;
    *p='\0';
    free(pStr->pdata);
    pStr->pdata=t;
    return TRUE;
}
/*其他操作省略*/
/*//////////////////////////////////////////////////////////
//      文件名DynamicString.c结束
//////////////////////////////////////////////////////////*/
主程序省略
```

【思考与提高】

(1) 使用封装可变长度串有何优缺点？

(2) 在主函数中定义一个充分长的字符数组作为一个堆,实现本实验。

实验三　计算矩阵转置运算

【实验内容与要求】

应用 C 语言编写输入矩阵、输出矩阵及转置矩阵的函数实现本实验。

【实验目的】

通过实验掌握矩阵(或多维数组)的存储方式、矩阵(或多数组)的 C 函数处理方法。理解矩阵(或多维数组)的行列存储方式。

【实现提示】

由于矩阵的结构是固定的,对其主要操作是访问指定元素,矩阵中主要的属性包括矩阵的行数、列数及指定位置的数据元素。将矩阵中的所有元素按照行序的方式存储到在一个连续的存储空间中,采用如下方式定义矩阵结构:

```
typedef struct{
    int row;                    /*行数*/
    int col;                    /*列数*/
    DataType *pdata;            /*数据存储区,DataType为矩阵元素的数据类型*/
} Matrix;
```

将矩阵中的元素按照行序存储到 pdata 指向的数据区中,因此,矩阵元素 a_{ij}($1 \leqslant i \leqslant row$,$1 \leqslant j \leqslant col$)的存储在 pdata[(i−1) * col+j−1]元素中。可以使用指向 Matrix 指针传递矩阵参数。

【参考程序】

```
/*////////////////////////////////////////////////
//     有关矩阵结构定义及操作说明
//     文件名是:matrix.h
////////////////////////////////////////////////*/
#ifndef    _MATRIX_H__
#define    _MATRIX_H__
typedef enum {FALSE,TRUE} BOOL;
typedef struct{
    int row, col;
    int *pdata;
}Matrix;

/*按行输入一个矩阵*/
void EnterMatrix(Matrix *pmtr);
/*输出一个矩阵*/
void OutputMatrix(Matrix *pmtr);
/*求由 psmtr 指向的矩阵的转置矩阵,存储于 ptmtr 中*/
BOOL Transform(Matrix *psmtr,Matrix *ptmtr);
```

```
#endif
/*////////////////////////////////////////////////////////
//      文件名 matrix.h 结束
////////////////////////////////////////////////////////*/

/*////////////////////////////////////////////////
//    有关矩阵操作的实现
//    文件名是:matrix.c
////////////////////////////////////////////////*/
#include "matrix.h"
#include<stdio.h>

void EnterMatrix(Matrix *pmtr)
{
    int i,j;
    int idx;
    printf("按行输入一个%dx%d矩阵\n",pmtr->row,pmtr->col);
    for(i=0;i<pmtr->row;i++)
        for(j=0;j<pmtr->col;j++)
        {
            idx=i*pmtr->col+j;
            scanf("%d",&(pmtr->pdata[idx]));
        }
}

void OutputMatrix(Matrix *pmtr)
{
    int i,j;
    printf("%dx%d矩阵如下:\n",pmtr->row,pmtr->col);
    for(i=0;i<pmtr->row;i++)
    {
        for(j=0;j<pmtr->col;j++)
            printf("%5d",pmtr->pdata[i*pmtr->col+j]);
        printf("\n");
    }
}

BOOL Transform(Matrix *psmtr,Matrix *ptmtr)
{
    int i,j;
    if(psmtr->row!=ptmtr->col || psmtr->col!=ptmtr->row)
        return FALSE;                    /*参数错误,目标矩阵不符合要求*/
    for(i=0;i<psmtr->row;i++)
        for(j=0;j<psmtr->col;j++)
```

```
            ptmtr->pdata[j*ptmtr->col+i]=psmtr->pdata[i*psmtr->col+j];
    return TRUE;
}
/*////////////////////////////////////////////////////////
//      文件名 matrix.c 结束
////////////////////////////////////////////////////////*/

/*////////////////////////////////////////////////////////
//    实验三主程序
//    文件名是:Ch03S1Exp3.c
////////////////////////////////////////////////////////*/
#include "matric.h"

void main()
{
    int m1[4][5],m2[5][4];       /*定义两个二维数组存储一个 4×5 矩阵及 5×4 矩阵*/
    Matrix mtr1,mtr2;
    /*用 mtr1 存储一个 4×5 矩阵、mtr2 存储一个 5×4 矩阵*/
    mtr1.row=4;
    mtr1.col=5;
    mtr1.pdata=&m1[0][0];

    mtr2.row=5;
    mtr2.col=4;
    mtr2.pdata=&m2[0][0];

    EnterMatrix(&mtr1);
    Transform(&mtr1,&mtr2);
    OutputMatrix(&mtr2);
}
/*////////////////////////////////////////////
//      文件名 Ch03S1Exp3.c 结束
////////////////////////////////////////////*/
```

【思考与提高】

(1) 如果以列序存储矩阵,如何实现本实验?

(2) 不定义矩阵的结构体,如何在 C 语言中传递数组参数?

实验四　三元组稀疏矩阵运算

【实验内容与要求】

使用三元组存储一个稀疏矩阵,编写一个函数实现稀疏矩阵的输入与输出,编写函数求稀疏矩阵的转置矩阵及两个矩阵的加法。