第二章 数组

2-1 设 n 个人围坐在一个圆桌周围，现在从第 s 个人开始报数，数到第 m 个人，让他出局；然后从出局的下一个人重新开始报数，数到第 m 个人，再让他出局，……，如此反复直到所有的人全部出局为止。下面要解决的 Josephus 问题是：对于任意给定的 n, s 和 m，求出这 n 个人的出局序列。请以 $n = 9$, $s = 1$, $m = 5$ 为例，人工模拟 Josephus 的求解过程以求得问题的解。

【解答】

出局人的顺序为 5, 1, 7, 4, 3, 6, 9, 2, 8。

2-2 试编写一个求解 Josephus 问题的函数。用整数序列 1, 2, 3, ……, n 表示顺序围坐在圆桌周围的人，并采用数组表示作为求解过程中使用的数据结构。然后使用 $n = 9$, $s = 1$, $m = 5$，以及 $n = 9$, $s = 1$, $m = 0$，或者 $n = 9$, $s = 1$, $m = 10$ 作为输入数据，检查你的程序的正确性和健壮性。最后分析所完成算法的时间复杂度。

【解答】函数源程序清单如下：

```
void Josephus( int A[ ], int n, s, m ) {
    int i, j, k, tmp;
    if ( m == 0 ) {
        cout << "m = 0 是无效的参数！" << endl;
        return;
    }
    for ( i = 0; i < n; i++ ) A[i] = i + 1;          /*初始化，执行 n 次*/
    i = s - 1;                                        /*报名起始位置*/
    for ( k = n; k > 1; k−− ) {                       /*逐个出局，执行 n−1 次*/
        if ( i == k ) i = 0;
        i = ( i + m − 1 ) % k;                        /*寻找出局位置*/
        if ( i != k−1 ) {
            tmp = A[i];                               /*出局者交换到第 k−1 位置*/
            for ( j = i; j < k−1; j++ ) A[j] = A[j+1];
            A[k−1] = tmp;
        }
    }
    for ( k = 0; k < n / 2; k++ ) {                   /*全部逆置，得到出局序列*/
        tmp = A[k]; A[k] = A[n−k+1]; A[n−k+1] = tmp;
    }
}
```

例：$n = 9$, $s = 1$, $m = 5$

	0	1	2	3	4	5	6	7	8	
$k = 9$	1	2	3	4	5	6	7	8	9	第 5 人出局, $i = 4$
$k = 8$	1	2	3	4	6	7	8	9	5	第 1 人出局, $i = 0$
$k = 7$	2	3	4	6	7	8	9	1	5	第 7 人出局, $i = 4$
$k = 6$	2	3	4	6	8	9	7	1	5	第 4 人出局, $i = 2$
$k = 5$	2	3	6	8	9	4	7	1	5	第 3 人出局, $i = 1$
$k = 4$	2	6	8	9	3	4	7	1	5	第 6 人出局, $i = 1$
$k = 3$	2	8	9	6	3	4	7	1	5	第 9 人出局, $i = 2$

$k = 2$	2	8	9	6	3	4	7	1	5	第 2 人出局, $i = 0$
	8	2	9	6	3	4	7	1	5	第 8 人出局, $i = 0$
逆置	5	1	7	4	3	6	9	2	8	最终出局顺序

例：$n = 9, s = 1, m = 0$

报错信息　$m = 0$ 是无效的参数！

例：$n = 9, s = 1, m = 10$

	0	1	2	3	4	5	6	7	8	
$k = 9$	1	2	3	4	5	6	7	8	9	第 1 人出局, $i = 0$
$k = 8$	2	3	4	5	6	7	8	9	1	第 3 人出局, $i = 1$
$k = 7$	2	4	5	6	7	8	9	3	1	第 6 人出局, $i = 3$
$k = 6$	2	4	5	7	8	9	6	3	1	第 2 人出局, $i = 0$
$k = 5$	4	5	7	8	9	2	6	3	1	第 9 人出局, $i = 4$
$k = 4$	4	5	7	8	9	2	6	3	1	第 5 人出局, $i = 1$
$k = 3$	4	7	8	5	9	2	6	3	1	第 7 人出局, $i = 1$
$k = 2$	4	8	7	5	9	2	6	3	1	第 4 人出局, $i = 0$
	8	4	7	5	9	2	6	3	1	第 8 人出局, $i = 0$
逆置	1	3	6	2	9	5	7	4	8	最终出局顺序

当 $m = 1$ 时，时间代价最大。达到$(n-1) + (n-2) + \cdots\cdots + 1 = n(n-1)/2 \approx O(n^2)$。

2-3 设有一个线性表 $(e_0, e_1, \cdots, e_{n-2}, e_{n-1})$ 存放在一个一维数组 $A[arraySize]$中的前 n 个数组元素位置。请编写一个函数将这个线性表原地逆置，即将数组的前 n 个原址内容置换为 $(e_{n-1}, e_{n-2}, \cdots, e_1, e_0)$。

【解答】

```
template<class Type> void inverse ( Type A[ ], int n ) {
    Type tmp;
    for ( int i = 0; i <= ( n−1 ) / 2; i++ ) {
        tmp = A[i];   A[i] = A[n−i−1];   A[n−i−1] = tmp;
    }
}
```

2-7 设有一个二维数组 $A[m][n]$，假设 $A[0][0]$存放位置在 $644_{(10)}$，$A[2][2]$存放位置在 $676_{(10)}$，每个元素占一个空间，问 $A[3][3]_{(10)}$存放在什么位置？脚注$_{(10)}$表示用 10 进制表示。

【解答】

设数组元素 A[i][j]存放在起始地址为 Loc (i, j) 的存储单元中。

∵ Loc (2, 2) = Loc (0, 0) + 2 * n + 2 = 644 + 2 * n + 2 = 676.

∴ n = (676 − 2 − 644) / 2 = 15

∴ Loc (3, 3) = Loc (0, 0) + 3 * 15 + 3 = 644 + 45 + 3 = 692.

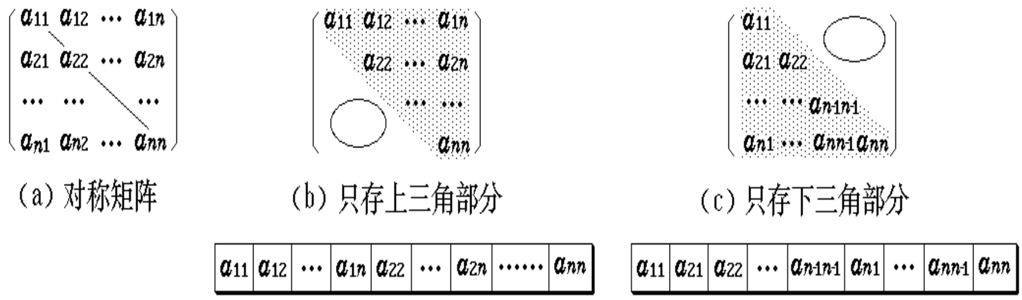
2-9 设有一个 $n \times n$ 的对称矩阵 A，如图(a)所示。为了节约存储，可以只存对角线及对角线以上的元素，或者只存对角线或对角线以下的元素。前者称为上三角矩阵，后者称为下三角矩阵。我们把它们按行存放于一个一维数组 B 中，如图(b)和图(c)所示。并称之为对称矩阵 A 的压缩存储方式。试问：

(1) 存放对称矩阵 A 上三角部分或下三角部分的一维数组 B 有多少元素？

(2) 若在一维数组 B 中从 0 号位置开始存放，则如图(a)所示的对称矩阵中的任一元素 a_{ij} 在只存上三角部分的情形下(图(b))应存于一维数组的什么下标位置？给出计算公式。

(3) 若在一维数组 B 中从 0 号位置开始存放，则如图(a)所示的对称矩阵中的任一元素 a_{ij} 在只

存下三角部分的情形下(图(c))应存于一维数组的什么下标位置？给出计算公式。



【解答】

(1) 数组 B 共有 n + (n−1) +⋯⋯ + 1= n * (n+1) / 2 个元素。

(2) 只存上三角部分时，若 i ≤ j，则数组元素 A[i][j]前面有 i−1 行（1~i−1，第 0 行第 0 列不算)，第 1 行有 n 个元素，第 2 行有 n−1 个元素，⋯⋯，第 i−1 行有 n−i+2 个元素。在第 i 行中，从对角线算起，第 j 号元素排在第 j−i+1 个元素位置（从 1 开始)，因此，数组元素 A[i][j]在数组 B 中的存放位置为

n + (n−1) + (n−2) + ⋯⋯ + (n−i+2) + j−i+1
= (2n−i+2) * (i−1) / 2 + j−i+1
= (2n−i) * (i−1) / 2 + j

若 i > j，数组元素 A[i][j]在数组 B 中没有存放，可以找它的对称元素 A[j][i]。在数组 B 的第 (2n−j) * (j−1) / 2 + i 位置中找到。

如果第 0 行第 0 列也计入，数组 B 从 0 号位置开始存放，则数组元素 A[i][j]在数组 B 中的存放位置可以改为

当 i ≤ j 时，= (2n−i+1) * i / 2 + j − i = (2n − i − 1) * i / 2 + j
当 i > j 时，= (2n − j − 1) * j / 2 + i

(3) 只存下三角部分时，若 i ≥ j，则数组元素 A[i][j]前面有 i−1 行（1~i−1，第 0 行第 0 列不算)，第 1 行有 1 个元素，第 2 行有 2 个元素，⋯⋯，第 i−1 行有 i−1 个元素。在第 i 行中，第 j 号元素排在第 j 个元素位置，因此，数组元素 A[i][j]在数组 B 中的存放位置为

1 + 2 + ⋯⋯ + (i−1) + j = (i−1)*i / 2 + j

若 i < j，数组元素 A[i][j]在数组 B 中没有存放，可以找它的对称元素 A[j][i]。在数组 B 的第 (j−1)*j / 2 + i 位置中找到。

如果第 0 行第 0 列也计入，数组 B 从 0 号位置开始存放，则数组元素 A[i][j]在数组 B 中的存放位置可以改为

当 i ≥ j 时，= i*(i+1) / 2 + j
当 i < j 时，= j*(j+1) / 2 + i

2-10 设 A 和 B 均为下三角矩阵，每一个都有 n 行。因此在下三角区域中各有 $n(n+1)/2$ 个元素。另设有一个二维数组 C，它有 n 行 $n+1$ 列。试设计一个方案，将两个矩阵 A 和 B 中的下三角区域元素存放于同一个 C 中。要求将 A 的下三角区域中的元素存放于 C 的下三角区域中，B 的下三角区域中的元素转置后存放于 C 的上三角区域中。并给出计算 A 的矩阵元素 a_{ij} 和 B 的矩阵元素 b_{ij} 在 C 中的存放位置下标的公式。

【解答】

$$A = \begin{pmatrix} a_{00} & & & \\ a_{10} & a_{11} & & \\ \cdots & \cdots & \ddots & \\ a_{n-10} & a_{n-11} & \cdots & a_{n-1n-1} \end{pmatrix} \qquad B = \begin{pmatrix} b_{00} & & & \\ b_{10} & b_{11} & & \\ \cdots & \cdots & \ddots & \\ b_{n-10} & b_{n-11} & \cdots & b_{n-1n-1} \end{pmatrix}$$

$$C = \begin{pmatrix} a_{00} & b_{00} & b_{10} & \cdots & b_{n-20} & b_{n-10} \\ a_{10} & a_{11} & b_{11} & \cdots & b_{n-21} & b_{n-11} \\ a_{20} & a_{21} & a_{22} & & b_{n-22} & b_{n-12} \\ \cdots & \cdots & \cdots & & \cdots & \cdots \\ a_{n-10} & a_{n-11} & a_{n-12} & \cdots & a_{n-1n-1} & b_{n-1n-1} \end{pmatrix}$$

计算公式

2-14 字符串的替换操作 *replace* (*String* **&***s*, *String* **&***t*, *String* **&***v*)是指：若 *t* 是 *s* 的子串，则用串 *v* 替换串 *t* 在串 *s* 中的所有出现；若 *t* 不是 *s* 的子串，则串 *s* 不变。例如，若串 *s* 为“aabbabcbaabaaacbab”，

$$B[i][j] = \begin{cases} C[j][i+1], & \text{当 } i \geq j \text{ 时} \\ C[i][j+1], & \text{当 } i < j \text{ 时} \end{cases}$$
$$A[i][j] = \begin{cases} C[i][j], & \text{当 } i \geq j \text{ 时} \\ C[j][i], & \text{当 } i < j \text{ 时} \end{cases}$$

串 *t* 为“bab”，串 *v* 为“abdc”，则执行 *replace* 操作后，串 *s* 中的结果为“aababdccbaabaaacabdc”。试利用字符串的基本运算实现这个替换操作。

【解答】

```
String & String :: Replace ( String & t, String &v) {
  if ( ( int id = Find ( t ) ) == −1 )        //没有找到，当前字符串不改，返回
     { cout << "The (replace) operation failed." << endl;    return *this; }
  String temp( ch );                     //用当前串建立一个空的临时字符串
  ch[0] = '\0';    curLen = 0;                //当前串作为结果串，初始为空
  int j, k = 0, l;                        //存放结果串的指针
  while ( id != −1 ) {
     for ( j = 0; j < id; j++) ch[k++] = temp.ch[j];
            //摘取 temp.ch 中匹配位置 id 前面的元素到结果串 ch。
     curLen += id + v.curLen;                    //修改结果串连接后的长度
     if ( curLen <= maxLen ) l = v.curLen;         //确定替换串 v 传送字符数 l
     else { l = curLen −maxLen;    curLen = maxLen; }
     for ( j = 0; j < l; j++ ) ch[k++] = v.ch[j];
            //连接替换串 v 到结果串 ch 后面
     if ( curLen == maxLen ) break;                  //字符串超出范围
     for ( j = id + t.curLen; j < temp.curLen; j++ )
        temp.ch[j− id − t.curLen] = temp.ch[j];         //删改原来的字符串
     temp.curLen −= ( id + t.curLen );
     id = temp.Find ( t );
  }
  return *this;
}
```

2-15 编写一个算法 *frequency*，统计在一个输入字符串中各个不同字符出现的频度。用适当的测试数据来验证这个算法。

【解答】

统计算法

```
include <iostream.h>
include "string.h"
void frequency( String& s, char& A[ ], int& C[ ], int &k ) {
// s 是输入字符串，数组 A[ ]中记录字符串中有多少种不同的字符，C[ ]中记录每
//一种字符的出现次数。这两个数组都应在调用程序中定义。k 返回不同字符数。
    int i, j, len = s.length( );
    if ( !len ) { cout << "The string is empty. " << endl;   k = 0;   return; }
    else {   A[0] = s[0];   C[0] = 1;   k = 1;       /*语句 s[i]是串的重载操作*/
          for ( i = 1; i < len; i++ ) C[i] = 0;      /*初始化*/
          for ( i = 1; i < len; i++ ) {              /*检测串中所有字符*/
                    j = 0;
          while ( j < k && A[j] != s[i] ) j++;       /*检查 s[i]是否已在 A[ ]中*/
              if ( j == k ) { A[k] = s[i]; C[k]++; k++ }        /*s[i]从未检测过*/
              else C[j]++;                                   /*s[i]已经检测过*/
          }
       }
}
```

测试数据 s = "cast cast sat at a tasa\0"

测试结果

A	c	a	s	t	ƀ
C	2	7	4	5	5

【另一解答】

```
include <iostream.h>
include "string.h"
const int charnumber = 128;                  /*ASCII 码字符集的大小*/
void frequency( String& s, int& C[ ] ) {
// s 是输入字符串，数组 C[ ]中记录每一种字符的出现次数。
    for ( int i = 0; i < charnumber; i++ ) C[i] = 0;     /*初始化*/
    for ( i = 0; i < s.length ( ); i++ )         /*检测串中所有字符*/
        C[ atoi (s[i]) ]++;                  /*出现次数累加*/
    for ( i = 0; i < charnumber; i++ )           /*输出出现字符的出现次数*/
        if ( C[i] > 0 ) cout << "( " << i << " ) : \t" << C[i] << "\t";
}
```