作为抽象数据类型数组的类声明。

```
#include <iostream.h>                    //在头文件“array.h”中
#include <stdlib.h>
const int DefaultSize = 30;

 template <class Type> class Array {
//数组是数据类型相同的 n（size）个元素的一个集合，下标范围从 0 到 n−1。对数组中元素
//可按下标所指示位置直接访问。
private:
   Type *elements;                                    //数组
   int ArraySize;                                     //元素个数
public:
   Array ( int Size = DefaultSize );                  //构造函数
   Array ( const Array<Type> & x );                   //复制构造函数
   ~Array ( ) { delete [ ] elements; }                //析构函数
   Array<Type> & operator = ( const Array<Type> & A );//数组整体赋值 (复制)
   Type& operator [ ] ( int i );                      //按下标访问数组元素
   int Length ( ) const { return ArraySize; }         //取数组长度
   void ReSize ( int sz );                            //修改数组长度
}
```

顺序表的类定义

```
#include < iostream.h>                  //定义在头文件“seqlist.h”中
 #include <stdlib.h>
 template <class Type> class SeqList {
private:
    Type *data;                        //顺序表的存放数组
    int MaxSize;                       //顺序表的最大可容纳项数
    int last;                          //顺序表当前已存表项的最后位置
    int current;                       //顺序表的当前指针（最近处理的表项）
 public:
    SeqList ( int MaxSize );           //构造函数
    ~SeqList ( ) { delete [ ] data; }  //析构函数
    int Length ( ) const { return last+1; }  //计算表长度
    int Find ( Type& x ) const;        //定位函数：找 x 在表中位置，置为当前表项
    int IsIn ( Type& x );              //判断 x 是否在表中，不置为当前表项
    Type *GetData ( ) { return current == −1？ NULL : data[current]; } //取当前表项的值
    int Insert ( Type& x );            //插入 x 在表中当前表项之后，置为当前表项
    int Append ( Type& x );            //追加 x 到表尾，置为当前表项
    Type * Remove ( Type& x );         //删除 x，置下一表项为当前表项
    Type * First ( );                  //取表中第一个表项的值，置为当前表项
    Type * Next ( ) { return current < last ? &data[++current] : NULL; }
                                       //取当前表项的后继表项的值，置为当前表项
    Type * Prior ( ) { return current > 0 ? &data[−−current] : NULL; }
```

```
                                        //取当前表项的前驱表项的值，置为当前表项
    int IsEmpty ( ) { return last == -1; }          //判断顺序表空否，空则返回 1；否则返回 0
    int IsFull ( ) { return last == MaxSize-1; }    //判断顺序表满否，满则返回 1；否则返回 0
}
```

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; i-- ) {                         /*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) + …… + 1 = n(n−1)/2 ≈ $O(n^2)$。

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;
    }
}
```

2-4 假定数组 A[arraySize]中有多个零元素，试写出一个函数，将 A 中所有的非零元素依次移到数组 A 的前端 A[i]（0≤ i ≤ arraySize）。

【解答】

因为数组是一种直接存取的数据结构，在数组中元素不是像顺序表那样集中存放于表的前端，而是根据元素下标直接存放于数组某个位置，所以将非零元素前移时必须检测整个数组空间，并将后面变成零元素的空间清零。函数中设置一个辅助指针 free，指示当前可存放的位置，初值为 0。

```
template<class Type> void Array<Type> :: compact( ) {
  int free = 0;
  for ( int i = 0; i < ArraySize; i++ )                    //检测整个数组
```

```
    if ( elements[I] != 0 )                                    //发现非零元素
      { elements[free] = elements[i];   free++;   elements[i] = 0; }    //前移
}
```

2-5 顺序表的插入和删除要求仍然保持各个元素原来的次序。设在等概率情形下，对有 127 个元素的顺序表进行插入，平均需要移动多少个元素？删除一个元素，又平均需要移动多少个元素?

【解答】

若设顺序表中已有 n = last+1 个元素，last 是顺序表的数据成员，表明最后表项的位置。又设插入或删除表中各个元素的概率相等，则在插入时因有 n+1 个插入位置(可以在表中最后一个表项后面追加)，每个元素位置插入的概率为 1/(n+1)，但在删除时只能在已有 n 个表项范围内删除，所以每个元素位置删除的概率为 1/n。

插入时平均移动元素个数 AMN(Averagy Moving Number)为

$$AMN=\frac{1}{n}\sum_{i=0}^{n-1}(n-i-1)=\frac{1}{n}((n-1)+(n-2)+\cdots+1+0)=\frac{1}{n}\frac{(n-1)n}{2}=\frac{n-1}{2}$$

删除时平均移动元素个数 AMN 为

$$AMN=\frac{1}{n+1}\sum_{i=0}^{n}(n-i)=\frac{1}{n+1}(n+(n-1)+\cdots+1+0)=\frac{1}{n+1}\frac{n(n+1)}{2}=\frac{n}{2}$$

2-6 若矩阵 $A_{m\times n}$ 中的某一元素 A[i][j]是第 i 行中的最小值，同时又是第 j 列中的最大值，则称此元素为该矩阵的一个鞍点。假设以二维数组存放矩阵，试编写一个函数，确定鞍点在数组中的位置（若鞍点存在时），并分析该函数的时间复杂度。

【解答】

```
int minmax ( int A[ ][ ], const int m, const int n ) {
//在二维数组 A[m][n]中求所有鞍点，它们满足在行中最小同时在列中最大
   int *row = new int[m];   int * col = new int[n];
   int i, j;
   for ( i = 0; i < m; i++ ) {                                 //在各行中选最小数组元素，存于 row[i]
      row[i] = A[i][0];
      for ( j = 1; j < n; j++ )
         if ( A[i][j] < row[i] ) row[i] = A[i][j];
   }
   for ( j = 0; j < n; j++ ) {                                 //在各列中选最大数组元素，存于 col[j]
      col[i] = A[0][j];
      for ( i = 1; i < m; i++ )
         if ( A[i][j] > col[j] ) col[j] = A[i][j];
   }
   for ( i = 0; i < m; i++ ) {                                 //检测矩阵，寻找鞍点并输出其位置
      for ( j = 0; j < n; j++ )
         if ( row[i] == col[j] ) cout << "The saddle point is : (" << i << ", " << j << ")" << endl;
   delete [ ] row;   delete [ ] col;
}
```

此算法有 3 个并列二重循环，其时间复杂度为 O(m×n)。

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.

2-8 利用顺序表的操作，实现以下的函数。

(1) 从顺序表中删除具有最小值的元素并由函数返回被删元素的值。空出的位置由最后一个元素填补，若顺序表为空则显示出错信息并退出运行。

(2) 从顺序表中删除第 i 个元素并由函数返回被删元素的值。如果 i 不合理或顺序表为空则显示出错信息并退出运行。

(3) 向顺序表中第 i 个位置插入一个新的元素 x。如果 i 不合理则显示出错信息并退出运行。

(4) 从顺序表中删除具有给定值 x 的所有元素。

(5) 从顺序表中删除其值在给定值 s 与 t 之间（要求 s 小于 t）的所有元素，如果 s 或 t 不合理或顺序表为空则显示出错信息并退出运行。

(6) 从有序顺序表中删除其值在给定值 s 与 t 之间（要求 s 小于 t）的所有元素，如果 s 或 t 不合理或顺序表为空则显示出错信息并退出运行。

(7) 将两个有序顺序表合并成一个新的有序顺序表并由函数返回结果顺序表。

(8) 从顺序表中删除所有其值重复的元素，使表中所有元素的值均不相同。

【解答】

(1) 实现删除具有最小值元素的函数如下：

```
template<Type> Type SeqList<Type> :: DelMin ( ) {
    if ( last == -1 )                                   //表空，中止操作返回
        { cerr << " List is Empty! " << endl;    exit(1); }
    int m = 0;                                          //假定 0 号元素的值最小
    for ( int i = 1; i <= last; i++ ) {                 //循环，寻找具有最小值的元素
        if ( data[i] < data[m] ) m = i;                 //让 m 指向当前具最小值的元素
    Type temp = data[m];
    data[m] = data[last];    last--;                    //空出位置由最后元素填补，表最后元素位置减 1
    return temp;
}
```

(2) 实现删除第 i 个元素的函数如下(设第 i 个元素在 data[i], i=0,1,…,last)：

```
template<Type> Type SeqList<Type> :: DelNo#i ( int i ) {
    if ( last == -1 ||  i < 0 ||  i > last )            //表空，或 i 不合理，中止操作返回
        { cerr << " List is Empty or Parameter is out range! " << endl;    exit(1); }
    Type temp = data[i];                                //暂存第 i 个元素的值
    for ( int j = i; j < last; j++ )                    //空出位置由后续元素顺次填补
        data[j] = data[j+1];
    last--;                                             //表最后元素位置减 1
    return temp;
}
```

(3) 实现向第 i 个位置插入一个新的元素 x 的函数如下(设第 i 个元素在 data[i], i=0,1,…,last)：

```
template<Type> void SeqList<Type> :: InsNo#i ( int i, Type& x ) {
```

```
    if ( last == MaxSize-1|| i < 0 || i > last+1 )          //表满或参数 i 不合理，中止操作返回
        { cerr << " List is Full or Parameter is out range! " << endl;   exit(1); }
    for ( int j = last; j >= i; j-- )                       //空出位置以便插入，若 i=last+1，此循环不做
        data[j+1] = data[j];
    data[i] = x;                                            //插入
    last++;                                                 //表最后元素位置加 1
}
```

(4) 从顺序表中删除具有给定值 x 的所有元素。

```
template<Type> void SeqList<Type> :: DelValue ( Type& x ) {
    int i = 0, j;
    while ( i <= last )                                     //循环，寻找具有值 x 的元素并删除它
        if ( data[i] == x ) {                               //删除具有值 x 的元素，后续元素前移
            for ( j = i; j < last; j++ ) data[j] = data[j+1];
            last--;                                         //表最后元素位置减 1
        }
        else i++;
}
```

(5) 实现删除其值在给定值 s 与 t 之间（要求 s 小于 t）的所有元素的函数如下：

```
template<Type> void SeqList<Type> :: DelNo#sto#t ( Type& s, Type& t ) {
    if ( last == -1 || s >= t )
       { cerr << "List is empty or parameters are illegal!" << endl;   exit(1); }
    int i = 0, j;
    while ( i <= last )                                     //循环，寻找具有值 x 的元素并删除它
        if ( data[i] >= s && data[i] <= t ) {               //删除满足条件的元素，后续元素前移
            for ( j = i; j < last; j++ ) data[j] = data[j+1];
            last--;                                         //表最后元素位置减 1
        }
        else i++;
}
```

(6) 实现从有序顺序表中删除其值在给定值 s 与 t 之间的所有元素的函数如下：

```
template<Type> void SeqList<Type> :: DelNo#sto#t1 ( Type& s, Type& t ) {
    if ( last == -1 || s >= t )
       { cerr << "List is empty or parameters are illegal!" << endl;   exit(1); }
    for ( int i = 0; i <= last; i++ )                       //循环，寻找值 ≥s 的第一个元素
        if ( data[i] >= s ) break;                          //退出循环时, i 指向该元素
    if ( i <= last ) {
        for ( int j = 1; i + j <= last; j++ )               //循环，寻找值 > t 的第一个元素
            if ( data[i+j] > t ) break;                     //退出循环时, i+j 指向该元素
        for ( int k = i+j; k <= last; k++ )                 //删除满足条件的元素，后续元素前移
            data[k-j] = data[k];
        last-= j;                                           //表最后元素位置减 j
    }
}
```

(7) 实现将两个有序顺序表合并成一个新的有序顺序表的函数如下：

```
template<Type> SeqList<Type>& SeqList<Type> :: Merge ( SeqList<Type>& A, SeqList<Type>& B ) {
//合并有序顺序表 A 与 B 成为一个新的有序顺序表并由函数返回
    if ( A.Length() + B.Length() > MaxSize )
       { cerr << "The summary of The length of Lists is out MaxSize!" << endl;    exit(1); }
    Type value1 = A.Fist ( ), value2 = B.Fisrt ( );
    int i = 0, j = 0, k = 0;
    while ( i < A.length ( ) && j < B.length ( ) ) {              //循环, 两两比较, 小者存入结果表
        if ( value1 <= value2 )
           { data[k] = value1;    value1 = A.Next ( );    i++; }
        else { data[k] = value2;    value2 = B.Next ( );    j++; }
        k++;
    }
    while ( i < A.Length ( ) )                                    //当 A 表未检测完, 继续向结果表传送
        { data[k] = value1;    value1 = A.Next ( );    i++;    k++; }
    while ( j < B.Length ( ) )                                    //当 B 表未检测完, 继续向结果表传送
        { data[k] = value2;    value2 = B.Next ( );    j++;    k++; }
    last = k – 1;
    return *this;
}
```

(8) 实现从表中删除所有其值重复的元素的函数如下：

```
template<Type> void SeqList<Type> :: DelDouble ( ) {
    if ( last == −1 )
       { cerr << "List is empty!" << endl;    exit(1); }
    int i = 0, j, k;    Type temp;
    while ( i <= last ) {                                         //循环检测
        j = i + 1;    temp = data[i];
        while ( j <= last ) {                                     //对于每一个 i, 重复检测一遍后续元素
            if ( temp == data[j] ) {                              //如果相等, 后续元素前移
                for ( k = j+1; k <= last; k++ ) data[k−1] = data[k];
                last−−;                                           //表最后元素位置减 1
            }
            else j++;
        }
        i++;                                                      //检测完 data[i], 检测下一个
    }
}
```

2−9 设有一个 n×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？

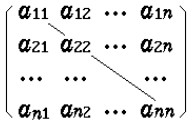
(2) 若在一维数组 B 中从 0 号位置开始存放，则如图(a)所示的对称矩阵中的任一元素 a_{ij} 在只存上三角部分的情形下(图(b))应存于一维数组的什么下标位置？给出计算公式。

(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

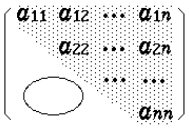
三角部分的情形下(图(c))应存于一维数组的什么下标位置？给出计算公式。

【解答】

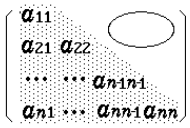
(1) 数组 B 共有 n + (n−1) +…… + 1= n * (n+1) / 2 个元素。



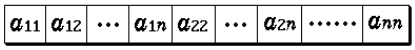
（a）对称矩阵

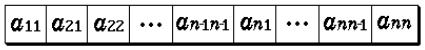


（b）只存上三角部分



（c）只存下三角部分





(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

$$n + (n-1) + (n-2) + \cdots\cdots + (n-i+2) + j-i+1$$
$$= (2n-i+2) * (i-1) / 2 + j-i+1$$
$$= (2n-i) * (i-1) / 2 + j$$

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

$$1 + 2 + \cdots\cdots + (i-1) + j = (i-1)*i / 2 + j$$

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】

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

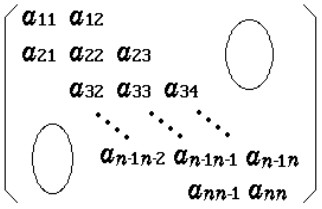
$$C=\begin{pmatrix} a_{00} & b_{00} & b_{10} & \cdots & b_{n-20} & b_{n-10} \\ a_{10} & a_{11} & b_{11} & \cdots & b_{n-21} & b_{n-11} \\ a_{20} & a_{21} & a_{22} & & b_{n-22} & b_{n-12} \\ \cdots & \cdots & \cdots & & \cdots & \cdots \\ a_{n-10} & a_{n-11} & a_{n-12} & \cdots & a_{n-1n-1} & b_{n-1n-1}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

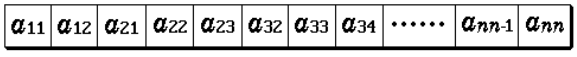
计算公式

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

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

2-11 在实际应用中经常遇到的稀疏矩阵是三对角矩阵，如右图所示。在该矩阵中除主对角线及在主对角线上下最临近的两条对角线上的元素外，所有其它元素均为 0。现在要将三对角矩阵 A 中三条对角线上的元素按行存放在一维数组 B 中，且 a_{11} 存放于 B[0]。试给出计算 A 在三条对角线上的元素 a_{ij} （$1 \le i \le n, i-1 \le j \le i+1$）在一维数组 B 中的存放位置的计算公式。





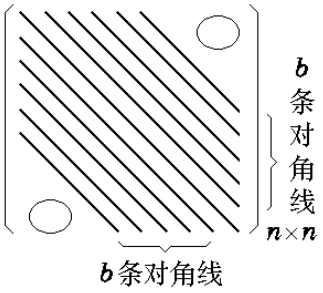
【解答】

在 B 中的存放顺序为 [$a_{11}, a_{12}, a_{21}, a_{22}, a_{23}, a_{32}, a_{33}, a_{34}, \cdots, a_{n\,n-1}, a_{nn}$]，总共有 3n-2 个非零元素。元素 a_{ij} 在第 i 行，它前面有 3(i-1)-1 个非零元素，而在本行中第 j 列前面有 j-i+1 个，所以元素 a_{ij} 在 B 中位置为 2*i+j-3。

2-12 设带状矩阵是 n×n 阶的方阵，其中所有的非零元素都在由主对角线及主对角线上下各 b 条对角线构成的带状区域内，其它都为零元素。试问：

(1) 该带状矩阵中有多少个非零元素？

(2) 若用一个一维数组 B 按行顺序存放各行的非零元素，且设 a_{11} 存放在 B[0]中，请给出一个公式，计算任一非零元素 a_{ij} 在一维数组 B 中的存放位置。



【解答】

(1) 主对角线包含 n 个非零元素，其上下各有一条包含 n-1 个非零元素的次对角线，再向外，由各有一条包含 n-2 个非零元素的次对角线，……，最外层上下各有一条包含 n-b 个非零元素的次对角线。则总共的非零元素个数有

$$n + 2(n-1) + 2(n-2) + \cdots + 2(n-b) = n + 2((n-1) + (n-2) + \cdots + (n-b))$$

$$= n + 2*\frac{((n-1)+(n-b))b}{2} = n + b(2n-b-1) = (2b+1)n - b - b^2$$

(2) 在用一个一维数组 B 按行顺序存放各行的非零元素时，若设 b≤n/2，则可按各行非零元素个数变化情况，分 3 种情况讨论。

① 当 1≤i≤b+1 时，矩阵第 1 行有 b+1 个元素，第 2 行有 b+2 个元素，第 3 行有 b+3 个元素，……，第 i 行存有 b+i 个元素，因此，数组元素 A[i][j]在 B[ ]中位置分析如下：

第 i 行(i≥1)前面有 i−1 行，元素个数为 (b+1)+(b+2)+…+(b+i−1) = (i−1)*b+i*(i−1)/2，在第 i 行第 j 列(j≥1)前面有 j−1 个元素，则数组元素 A[i][j]在 B[ ]中位置为

$$(i-1)*b+\frac{i*(i-1)}{2}+j-1$$

② 当 b+1<i≤n−b+1 时，各行都有 2b+1 个元素。因为数组 A[ ][ ]前 b 行共有 b*b+(b+1)*b/2 = b*(3*b+1)/2 个元素，所以数组元素 A[i][j]在 B[ ]中位置为

$$\frac{b*(3*b+1)}{2}+(i-b-1)*(2*b+1)+j-i+b$$

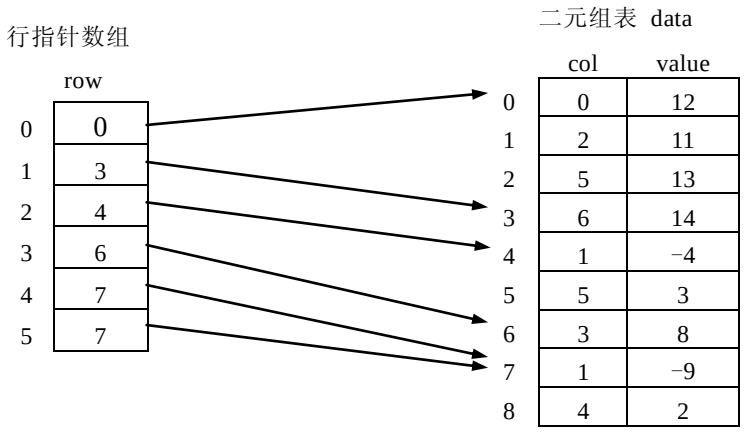
③ 当 n−b+1<i≤n 时，各行元素个数逐步减少。当 i=n−b+1 时有 2b 个非零元素，当 i=n−b+2 时有 2b−1 个非零元素，当 i=n−b+3 时有 2b−2 个非零元素，…，当 i=n 时有 b+1 个非零元素。因为前面 n−b 行总共有 b*(3*b+1)/2+(n−2*b)*(2*b+1)个非零元素，所以在最后各行数组元素 A[i][j]在 B[ ]中位置为

$$\frac{b*(3*b+1)}{2}+(n-2*b)*(2*b+1)+\frac{(i-n+b-1)*(4*b-i+n-b+2)}{2}+j-i+b$$

2−13 稀疏矩阵的三元组表可以用带行指针数组的二元组表代替。稀疏矩阵有多少行，在行指针数组中就有多少个元素：第 i 个元素的数组下标 i 代表矩阵的第 i 行，元素的内容即为稀疏矩阵第 i 行的第一个非零元素在二元组表中的存放位置。二元组表中每个二元组只记录非零元素的列号和元素值，且各二元组按行号递增的顺序排列。试对右图所示的稀疏矩阵，分别建立它的三元组表和带行指针数组的二元组表。

$$\begin{pmatrix} 12 & 0 & 11 & 0 & 0 & 13 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 14 \\ 0 & -4 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -9 & 0 & 0 & 2 & 0 & 0 \end{pmatrix}$$

【解答】



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”，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
        if ( curLen+ id + v.curLen <= maxLen )
            l = v.curLen;                                    //确定替换串 v 传送字符数 l
        else l = maxLen− curLen− id;
        for ( j = 0; j < l; j++ ) ch[k++] = v.ch[j];         //连接替换串 v 到结果串 ch 后面
        curLen += id + l;                                    //修改结果串连接后的长度
        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】

```
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	b
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";
}
```

2-16 设串 s 为“aaab”，串 t 为“abcabaa”，串 r 为“abc□aabbabcabaacbacba”，试分别计算它们的失效函数 f (j)的值。

【解答】

j	0	1	2	3	j	0	1	2	3	4	5	6
s	a	a	a	b	t	a	b	c	a	b	a	a
f (j)	−1	0	1	−1	f (j)	−1	−1	−1	0	1	0	0

2-17 设定整数数组 B[m+1][n+1]的数据在行、列方向上都按从小到大的顺序排序，且整型变量 x 中的数据在 B 中存在。试设计一个算法，找出一对满足 B[i][j] == x 的 i, j 值。要求比较次数不超过 m+n。

【解答】

算法的思想是逐次二维数组右上角的元素进行比较。每次比较有三种可能的结果：若相等，则比较结束；若右上角的元素小于 x，则可断定二维数组的最上面一行肯定没有与 x 相等的数据，下次比较时搜索范围可减少一行；若右上角的元素大于 x，则可断定二维数组的最右面一列肯定不包含与 x 相等的数据，下次比较时可把最右一列剔除出搜索范围。这样，每次比较可使搜索范围减少一行或一列，最多经过 m+n 次比较就可找到要求的与 x 相等的数据。

```
void find ( int B[ ][ ], int m, int n, int x, int& i, int& j ) {
//在二维数组 B[m][n]中寻找与 x 相等的元素，找到后，由 i 与 j 返回该数组元素的位置
    i = 0;   j = n;
    while ( B[i][j] != x )
        if ( B[i][j] < x ) i++;
        else j--;
}
```