## H文件

// TMatrix.h: interface for the CTMatrix class.

//

//////////////////////////////////////////////////////////////////////

#if !defined(AFX\_TMATRIX\_H\_\_D0F25BB1\_AE4C\_42ED\_BFD4\_89FECA42E86A\_\_INCLUDED\_)

#define AFX\_TMATRIX\_H\_\_D0F25BB1\_AE4C\_42ED\_BFD4\_89FECA42E86A\_\_INCLUDED\_

#if \_MSC\_VER > 1000

#pragma once

#endif // \_MSC\_VER > 1000

#include "TArray.h"

#include "ImagePoint.h"

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ]

// [ 模板类 CTMatrix< TYPE > 和 CTMatrixEx< TYPE > 的区别在于: ]

// [ 在构造矩阵内容( 矩阵复制 )函数( Construction )中进行矩阵复制 ]

// [ 时, CTMatrix 是整个矩阵内存的拷贝, CTMatrixEx 是逐个矩阵元素 ]

// [ 的拷贝. ]

// [ 功能说明: CTMatrix 中的 TYPE 是不存在动态分配内存的数据类型, 包括简 ]

// [ 单的数据类型( 如 int, double 等 )和 struct 类型 以及 ]

// [ class 类型( 如 CPoint 等 ). ]

// [ CTMatrixEx 中的 TYPE 最好是存在动态分配内存的数据类型, 包 ]

// [ 括 struct 类型和 class 类型( 如 CString 等 )。 ]

// [ CTMatrixEx 适用所有的数据类型, 但复制速度慢; CTMatrix 只 ]

// [ 适用不存在动态分配内存的数据类型, 但复制速度快. ]

// [ 可以说, CTMatrixEx 是 CTMatrix 的扩展, 而 CTMatrix 是 ]

// [ CTMatrixEx 的特例. ]

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ]

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ]

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* << 二维矩阵模板 >> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ]

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ]

template <class TYPE> class CTMatrix

{

// [ \*\*\*\*\*\*\*\* ] ......................................................

// [ 成员变量 ] ......................................................

// [ \*\*\*\*\*\*\*\* ] ......................................................

private:

TYPE\* m\_memory; // [ 矩阵数据内存指针 ]

long m\_rows; // [ 矩阵的行数 ]

long m\_columns; // [ 矩阵的列数 ]

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ............................................

// [ 构造函数和析构函数 ] ............................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ............................................

public:

CTMatrix() // [ 构造函数: 缺省构造函数 ]

{

Initialization(); // [ 初始化成员变量 ]

}

CTMatrix( long rows, long columns ) // [ 构造函数: 分配矩阵内存 ]

{

Initialization(); // [ 初始化成员变量 ]

Construction( rows, columns ); // [ 构造矩阵内容: 分配矩阵内存 ]

}

CTMatrix( const CTMatrix< TYPE >& matrix ) // [ 复制构造函数 ]

{

Initialization(); // [ 初始化成员变量 ]

Construction( matrix ); // [ 构造矩阵内容: 矩阵复制 ]

}

~CTMatrix() // [ 析构函数 ]

{

Destruction( ); // [ 析构矩阵内容 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 功能接口函数 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

public:

void Construction( long rows, long columns ) // [ 构造矩阵内容: 分配矩阵内存 ]

{

if( m\_rows == rows && m\_columns == columns && m\_memory != NULL ) // [ 如果所要分配的矩阵内存大小不变，则保留原有矩阵内存 ]

return ;

Destruction( ); // [ 析构矩阵内容 ]

if( rows > 0 && columns > 0 ) // [ 矩阵行数和列数都必须大于 0 ]

{

long elements = rows \* columns ; // [ 矩阵元素个数 ]

m\_memory = new TYPE[ elements ]; // [ 分配内存 ]

if( m\_memory != NULL ) // [ 判断分配内存是否成功 ]

{

m\_rows = rows; // [ 设置矩阵行数 ]

m\_columns = columns; // [ 设置矩阵列数 ]

}

else

{

Verification( "System Error << Can not allocate memory correctly >>" ); // [ 测试报告 ]

}

}

else

{

Verification( "Program Error << Matrix rows and columns must be bigger than 0 >>" ); // [ 测试报告 ]

}

}

void Construction( const CTMatrix< TYPE >& matrix ) // [ 构造矩阵内容: 矩阵复制 ]

{

if( matrix.m\_memory != NULL ) // [ 判断矩阵内容是否为空 ]

{

Construction( matrix.m\_rows, matrix.m\_columns ); // [ 构造相同尺寸的矩阵: 分配矩阵内存 ]

long sizeInBytes = m\_rows \* m\_columns \* sizeof( TYPE ); // [ 矩阵内容所占字节数 ]

memcpy( m\_memory, matrix.m\_memory, sizeInBytes ); // [ 复制矩阵内容 ]

}

else

{

Destruction( ); // [ 析构矩阵内容 ]

}

}

void Destruction() // [ 析构矩阵内容 ]

{

if( m\_memory != NULL ) // [ 判断是否分配了内存 ]

{

delete [] m\_memory; // [ 释放内存 ]

Initialization( ); // [ 初始化成员变量 ]

}

}

void ImportFrom( const void\* source ) // [ 导入矩阵内容 ]

{

if( m\_memory != NULL ) // [ 判断矩阵是否为空 ]

{

long sizeInBytes = m\_rows \* m\_columns \* sizeof( TYPE ); // [ 矩阵内容所占字节数 ]

memcpy( m\_memory, source, sizeInBytes ); // [ 导入矩阵内容 ]

}

}

void ExportTo( void\* destination ) const // [ 导出矩阵内容 ]

{

if( m\_memory != NULL ) // [ 判断矩阵是否为空 ]

{

long sizeInBytes = m\_rows \* m\_columns \* sizeof( TYPE ); // [ 矩阵内容所占字节数 ]

memcpy( destination, m\_memory, sizeInBytes ); // [ 导出矩阵内容 ]

}

}

CTMatrix< TYPE >& operator=( const CTMatrix< TYPE >& matrix ) // [ 赋值操作符 ]

{

Construction( matrix ); // [ 构造矩阵内容: 矩阵复制 ]

return \* this; // [ 返回当前对象 ]

}

#ifdef \_DEBUG

CTArray< TYPE >& operator[]( long index ) const // [ 下标操作符 ]

{

static CTArray< TYPE > error; // [ 出错返回的静态变量 ]

if( !( index >= 0 && index < m\_rows ) ) // [ 矩阵行定位有效性测试 ]

{

Verification( "Program Error << Matrix row index beyond bound >>" ); // [ 测试报告 ]

return error; // [ 出错返回 ]

}

static CTArray< TYPE > rowArray; // [ 矩阵行数组 ]

rowArray.Construction( m\_memory + index \* m\_columns, m\_columns ); // [ 构造数组内容: 指派数组内存 ]

return rowArray; // [ 返回矩阵行数组 ]

}

#else // \_DEBUG

TYPE\* operator[]( long index ) const // [ 下标操作符 ]

{

return m\_memory + index \* m\_columns; // [ 返回矩阵行数组地址 ]

}

#endif // \_DEBUG

TYPE& operator[]( const CImagePoint& point ) const // [ 下标操作符 ]

{

#ifdef \_DEBUG

return ( \*this )[ point.m\_row ][ point.m\_column ] ;

#else // \_DEBUG

return \*( m\_memory + point.m\_row \* m\_columns + point.m\_column );

#endif // \_DEBUG

}

bool Is\_point\_valid( const CImagePoint& point ) const // [ 点坐标是否正确 ]

{

return point.m\_row >= 0 && point.m\_row < m\_rows // [ 行坐标 ]

&& point.m\_column >= 0 && point.m\_column < m\_columns; // [ 列坐标 ]

}

bool Is\_row\_valid( long row ) const // [ 行索引是否正确 ]

{

return row >= 0 && row < m\_rows; // [ 行索引 ]

}

bool Is\_column\_valid( long column ) const // [ 列索引是否正确 ]

{

return column >= 0 && column < m\_columns; // [ 列索引 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 状态参数函数 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

public:

bool IsNull() const // [ 是否为空 ]

{

return ( m\_memory == NULL );

}

bool IsNotNull() const // [ 是否不空 ]

{

return ( m\_memory != NULL );

}

long GetRows() const // [ 矩阵行数 ]

{

return m\_rows;

}

long GetColumns() const // [ 矩阵列数 ]

{

return m\_columns;

}

long Get\_height() const // [ 矩阵高度( 矩阵行数 ) ]

{

return m\_rows;

}

long Get\_width() const // [ 矩阵宽度( 矩阵列数 ) ]

{

return m\_columns;

}

long GetRowUpperBound() const // [ 矩阵行上界 ]

{

return ( m\_rows - 1 );

}

long GetColumnUpperBound() const // [ 矩阵列上界 ]

{

return ( m\_columns - 1 );

}

long GetElements() const // [ 矩阵元素个数 ]

{

return ( m\_rows \* m\_columns );

}

long GetSizeInBytes() const // [ 矩阵内容所占字节数 ]

{

return ( m\_rows \* m\_columns \* sizeof( TYPE ) );

}

CImagePoint Get\_size() const // [ 矩阵尺寸( 矩阵行数和列数 ) ]

{

return CImagePoint( m\_rows, m\_columns );

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 内部实现函数 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

private:

void Initialization() // [ 初始化成员变量 ]

{

m\_memory = NULL; // [ 矩阵内存指针清空 ]

m\_rows = 0; // [ 矩阵行数归零 ]

m\_columns = 0; // [ 矩阵列数归零 ]

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ]

// [ 初始化成员变量( Initialization )后, 成员变量遵守如下规则: ]

// [ 1. 如果矩阵内存指针不空( m\_memory != NULL ), 则矩阵行数和列 ]

// [ 数都不为零( m\_rows > 0 && m\_columns > 0 ). ]

// [ 2. 如果矩阵内存指针为空( m\_memory == NULL ), 则矩阵行数和列 ]

// [ 数都为零( m\_rows == 0 && m\_columns == 0 ). ]

// [ 3. 矩阵行数和列数要么都为零，要么都不为零. ]

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ]

}

void Verification( CString message ) const // [ 测试报告 ]

{

#ifdef \_DEBUG

AfxMessageBox( message ); // [ 测试报告 ]

#endif // \_DEBUG

}

};

#endif // !defined(AFX\_TMATRIX\_H\_\_D0F25BB1\_AE4C\_42ED\_BFD4\_89FECA42E86A\_\_INCLUDED\_)

## Cpp文件