矩阵实现

> 优先备注： 我在实现矩阵类的时候，提前实现了向量类。值得注意的是，我把行向量作为主序，同样的，我把矩阵看作由多个行向量组成的向量组
> 另外，本代码实现的原理源于我对《线性代数》（同济大学第六版）部分知识的贮备

向量类（Vector）

属性

属性名称	类型	作用	本身属性
data	Element*	记录向量数据，初始化时根据长度全部赋值为0.0	public
length	int	记录向量长度	public

方法

方法名称	类型	作用	本身属性
Vector()	--	默认构造函数，你会获取一个长度为1，唯一值为0.0的向量	public
Vector(Element* _data, int _length = MAXLEN)	--	可传参的构造函数，根据你传入的数据，初始化向量，如果你不指定长度，会默认为最大指定长度，并在值没传满的情况全部补全为0.0	public
~Vector()	--	析构函数	public
printVec(int len=MAXLEN, bool ifself= true, int round=ROUND)	void	以向量形式打印，并修改终端输出样式，这三个参	public

方法名称	类型	作用	本身属性
		数分别指定了打印向量前len个数据、是否是自己在打印（被矩阵打印方法调用时需要指定false，一般不需要在意此方法）、打印当前向量数据精确到小数点后几位	
transpose()	Vector*	获取矩阵的转置矩阵，从1xn变成nx1形状，但是返回的是一个n个1x1向量组成向量组	public
Vector(const Vector& other)	--	拷贝函数	public
isEqual(Vector & other)	const bool	判断两个向量一样的，即长度一样，对应索引位置数据一样	public
multi(Element k)	Vector	获取k倍向量	public
add(Vector & other)	Vector	获取两个向量相加后的向量	public
splice(Vector & other, bool pos= true)	Vector	获取两个向量拼接后的向量，如果pos为true，把other拼到右侧，否则左侧	public
expand(int index, Element k)	Vector	获取索引index扩大k倍后的向量	public
unit()	Vector	获取单位化后的向量，但引用向量本身不能是零向量	public
isZero()	const bool	判断一个向量是不是零向量	public
transpose2(Vector *vec, int row)	Vector	把一个nx1的向量组转回到1xn向量，这是一个友元函数	public

方法名称	类型	作用	本身属性
inner(Vector & other)	const Element	两个行向量做内积，但是other（即被乘向量，在右侧那个）在矩阵（或者可以说向量组）我视之为列向量的作用	public
getReverseOrderNumber(Vector & vec)	int	把一个向量视为逆序数组，求取逆序数，这是一个友元函数	public

矩阵类（Matrix）

属性

属性名称	类型	作用	本身属性
_vecs	Vector[]	我把矩阵看成了行向量组	private
_row	int	记录矩阵行数	private
_column	int	记录矩阵的列数	private
spc	Special	标识特定矩阵，具体请见下面Special结构体	private
shape	Shape	矩阵的形状	public

方法

方法名称	类型	作用	本身属性
Matrix(Vector * vectors, int _row, int _column)	--	矩阵类唯一提供的构造方法，需要传入向量组、矩阵的行数或者列数	public
Matrix(const Matrix & other)	--	拷贝函数	public

方法名称	类型	作用	本身属性
~Matrix()	--	析构函数	public
printMatrix(int round=ROUND)	void	直观的打印出矩阵，round参数可以指定矩阵数值精确到小数点后几位	public
isSquare()	const bool	判断矩阵是不是方阵	public
E()	Matrix	在方阵的前提下获取同形单位矩阵	public
transpose()	Matrix	获取矩阵的转置矩阵	public
isDig(bool if_main=true)	bool	在方阵的前提下，如果传入true，则判断矩阵是不是主对角矩阵，否则判断是不是副对角矩阵	public
isSymmetric()	bool	在方阵的前提下，判断矩阵是不是对称矩阵（即M(i, j) == M(j, i)的矩阵）	public
isEqual(Matrix & other)、==	bool	判断两个矩阵是不是相同矩阵，即形状一样的前提下，对应位置数据一样视为相同矩阵，同时在相同前提下，把拥有一方spc属性中true的成员传入到另一方spc属性中对应的成员	public
splice(Matrix & other, Splice _splice={true, true})	Matrix	按模式拼接两个矩阵，模式传入请参考Splice结构体	public

方法名称	类型	作用	本身属性
add(Matrix & other)、+	Matrix	矩阵形状一样的前提下做矩阵加法运算并返回一个新矩阵	public
multi(Element k)、*	Matrix	矩阵的数乘，返回扩大k倍的矩阵	public
innerMulti(Matrix & other)、^	Matrix	满足内积条件的前提下做矩阵内积	public
getRemainder(int row, int col)	Matrix	去掉row行和col列，获取剩下数据组成的余子矩阵，请注意，需要区别于余子式；传入的矩阵至少是二阶的	public
det()	Matrix	是方阵的前提下，采用代数余子式的方法递归求取矩阵的值，一般情况的时间复杂度是O（n！）	public
ifTriMatrix(bool if_up=true)	bool	传入true时，判断方阵是不是一个上三角矩阵，反之	public
isSingularMat()	bool	判断方阵是不是奇异矩阵	public
trace()	Element	获取方阵的迹	public
getAccompany()	Matrix	使用AA* = det(A)E原理求取方阵A的伴随矩阵，逆矩阵同样使用此方法	public
getAccompanyT()	Matrix	同样使用上面原理，采用另一种方式获取方阵A的伴随矩阵	public

方法名称	类型	作用	本身属性
inv()	Matrix	获取非奇异矩阵的逆矩阵	public
getElementaryTransposeReSize(int lr, Element k, bool if_line= true)	void	对矩阵本身进行初等变换——把某行（列）扩大k倍，但是除了初等变换，我还希望k可以为0，if_line为true则指定行变换，反之；if_line作用下同	public
getElementaryTransposeExchange(int lr1, int lr2, bool if_line=true);	void	对矩阵本身进行初等变换——把两行（列）对换位置	public
getElementaryTransposAdd(int lr1, int lr2, double k, bool if_line);	void	对矩阵本身进行初等变换——把lr2行（列）的k倍加到lr1行（列），但lr2本身不变	public
Zero()	Matrix	获取矩阵同形状的零矩阵，但是不会继承原来的是spc属性被检测过为true的成员	public
isOrthogon()	bool	判断方阵是不是正交矩阵	public
norm(bool if_line=false)	Matrix	组成矩阵的向量组单位化，if_line=true决定的是行方向，反之	

特殊的独立方法（Special）

方法名称	类型	作用	文件源
printVecC(Vector* vecs, int length, int round=ROUND)	void	列向量专用的独立打印方式，length参数实质是形状nx1中的n，在这里，你仍然可以指定当前输出的精确小数的位置	rematrix
error_models(int model)	void	指定报错模式	auxiliary
swap_array(Element *arr1, Element *arr2, int len)	void	交换两个长度为len的数组交换数据	auxiliary
power(int x, int y)	Element	只考虑整数x的正整数次幂，但是没有进行数据检测处理	auxiliary
sums(Element * data, int len)	Element	求取长度为len的数据集的和	auxiliary
copy_mat(Matrix & other)	Matrix	复制一个Matrix，静态方法	rematrix

特殊结构（Structure）

Splice结构

> Splice结构决定了拼接矩阵的方式

成员名称	类型	作用
if_lr	bool	表示是否左右拼接，如果为true，左右拼接，否则上下拼接
other_lr	bool	如果选择了左右拼接，传入true，则把other拼接到右侧，反之；如果选择的是上下拼接，传入true，则把other拼接到下面，反之

Shape结构

> Shape结构用于标识矩阵的形状

成员名称	类型	作用
row	int	矩阵的行数

成员名称	类型	作用
column	int	矩阵的列数

Special结构

- Special是一个用于标识某些矩阵在运行特定方法后获取的新属性

> - 注意的是，必须是某些特殊矩阵在具有的属性（比如说，确定了这是一个对角矩阵、三角矩阵等等等）
> - 一旦一个矩阵被标识了这类属性，由此类矩阵衍生出的新矩阵可能会继承某些属性
> - 或者，在进行大量高时间复杂度计算时，特殊矩阵可以明显的降低时间复杂度，比如说使用递归求方阵的行列式值，需要O（n！）时间复杂度，而上三角矩阵只需要O（1）时间复杂度

成员名称	类型	作用
triUp	bool	是否上三角矩阵
triDown	bool	是否下三角矩阵
digUp	bool	是否主对角线矩阵
digDown	bool	是否副对角线矩阵
unit	bool	是否是的单位矩阵
symmetry	bool	是否是对称矩阵

- 下列成员为true时，支持解决的方法

triUP

> - Matrix(const Matrix &other)
> - det

triDown

> - Matrix(const Matrix &other)
> - det

digUp

- Matrix(const Matrix &other)
- det
- getAccompany
- getAccompanyT
- inv

digDown

- Matrix(const Matrix &other)
- det
- inv

unit

- Matrix(const Matrix &other)
- tanspose
- isDig
- isSymmetric
- ifTriMatrix
- det
- getAccompany
- getAccompanyT
- innerMulti
- inv

symmetry

- Matrix(const Matrix &other)
- tanspose
- isDig

特别备注
虽然有以下例子： 如果一个矩阵是和单位矩阵（已经记录它是）相加的话，我可以更快，只使用时间复杂度为O（n）的计算方法代码来更快的运算，但是我不打算这样做。
如果诸如此类情况都考虑的话，确实可以让运算性能大幅度提升，但是有一些方法中我不打算这么做

预编译（Macro）

- 预编译定义

宏名称	作用
_GLIBCXX_REMATRIX	独立库标识
Element	double类型的别名
MAXLEN	允许向量数据传入的最大的长度，在矩阵中，代表着矩阵允许最大列数
MAXCOUNT	在矩阵中，代表着矩阵允许最大行数
NEWLINE	帮助快速换行
T_ERROR	用于测试代码运行是否不符合预期
CHECKSUC	确实是我想的那样
CHECKNOT	确实不是我想的那样
elif	else if的代替
END_SUCCESSFULLY	成功退出，exit(EXIT_SUCCESS)
ROUND	全局，输出向量/矩阵小数精确度