<Matrix.h>

C 语言基本矩阵运算宏。需要 typeof 特性。

以二维数组表示矩阵，形状同数组初始化器。

对于编译时已知形状的矩阵运算，静态推导、检查其类型、形状。

若不使用由 VLA 表示的矩阵，建议定义宏 NDEBUG 以关闭对矩阵形状的动态检查。由 VLA 表示的矩阵忽略对其形状的静态检查。

未能解决的缺陷：对于矩阵运算中具有 VM 类型的 C 表达式，将额外求值 1 次以获得类型。

用户可在头文件处修改宏的名字，但仅限于下文介绍的宏。

矩阵求值

声明于<MatrixBase.h>

```
#define _Mvl    /* Matrix Evaluate With Depth 1 */
#define _Mvl3   /* Matrix Evaluate With Depth 3 */
#define _Mvl9   /* Matrix Evaluate With Depth 9 */
#define _Mvl27  /* Matrix Evaluate With Depth 27 */
#define _Mvl81  /* Matrix Evaluate With Depth 81 */
#define _Mvl243 /* Matrix Evaluate With Depth 243 */
#define _Mvl729 /* Matrix Evaluate With Depth 729 */
```

描述

矩阵

二维数组类型的 C 表达式或复合矩阵表达式。

应注意，作为函数形参的数组是指针。建议在参数声明中额外用指针包装这些数组，再解引用，以保留原数组类型。

_Mvl*NN*(矩阵)

将矩阵表达式编译为 C，展开深度为 NN。

示例

```
void nope(int (*A)[3][3])
{ _Mvl(*A); }
```

简单地返回并丢弃 A 的值。

```
size_t N = rand();
typedef float matNxN[N][N];
_Mvl(*(matNxN*)malloc(sizeof(matNxN)));
```

由于该 C 表达式具有 VM 类型，实际发生 2 次分配。

矩阵加、乘法

声明于<MatrixTrivial.h>

```
#define _Mst /* Matrix Set */
#define _Mgt /* Matrix Get */
```

声明于<MatrixTrivialCall.h>

```
#define _Mcl /* Matrix Call */
#define _Mrt /* Matrix Return */
```

声明于<MatrixXT.h>

```
#define $T$  /* ^T */
#define $    /* dot */
#define $T   /* dot ^T */
#define T$   /* ^T dot */
#define T$T  /* ^T dot ^T */
```

描述

矩阵二次式

矩阵加法表达式中，具有以下格式之一的参数形式。A、B 均为矩阵。

`A` 表示 A

`A $T$` 表示 A^T

`A $ B` 表示 AB

`A $T B` 表示 AB^T

`A T$ B` 表示 A^TB

`A T$T B` 表示 A^TB^T

连接符

可连接两个对象以形成表达式的 C 语句片段。

`_Mst(矩阵，连接符，矩阵二次式，连接符，... ，矩阵二次式)`

执行对应的矩阵操作。返回参数中的第一个矩阵。

`_Mgt(矩阵二次式，连接符，... ，矩阵二次式)`

执行对应的矩阵运算。返回临时矩阵以表示结果。

_Mcl(可调用对象，矩阵，矩阵二次式，... ，矩阵二次式)

逐元素调用对象，执行对应的矩阵操作。返回参数中的第一个矩阵。

_Mrt(可调用对象，矩阵二次式，... ，矩阵二次式)

逐元素调用对象，执行对应的矩阵运算。返回临时矩阵以表示结果。

示例

```
_Mvl9(_Mst(A, = ,B, + ,C $T$, + ,D $ E));
```

将 A 置为 $B + C^T + DE$。

```
_Mvl9(_Mst(A, += 2*,B, - 3*,C));
```

将 A 置为 $2B - 3C$。

```
_Mvl9(_Mst(A, = ,_Mgt(F T$ A)));
```

将 A 置为 F^TA。使用临时矩阵以防止就地乘法产生错误结果。

```
_Mvl9(_Mst(A, = ,F T$ _Mgt(P $ F)));
```

将 A 置为 F^TPF。超过 2 次的矩阵多项式须用临时矩阵计算。

```
#define set0(x) x = 0
_Mvl3(_Mcl(set0, A));
```

将 A 置为 0。

矩阵除法

声明于<MatrixDivide.h>

```
#define _Mdv /* Matrix Divide */
#define _Mnv /* Matrix Inverse */
#define I$   /* ^-1 dot */
#define IT$  /* ^-T dot */
#define $I   /* dot ^-1 */
#define $IT  /* dot ^-T */
```

描述

A、B 均为矩阵。

```
_Mdv(A I$ B)
```

将 B 置为 $A^{-1}B$，A 置为不确定值。返回 B。

```
_Mdv(A IT$ B)
```

将 B 置为 $A^{-T}B$，A 置为不确定值。返回 B。

```
_Mdv(A $I B)
```

将 A 置为 AB^{-1}，B 置为不确定值。返回 A。

```
_Mdv(A $IT B)
```

将 A 置为 AB^{-T}，B 置为不确定值。返回 A。

```
_Mnv(A)
```

返回临时矩阵 A^{-1}，并将 A 置为不确定值。

示例

```
_Mvl9(_Mst(A, = ,_Mdv(_Mgt(C) I$ _Mgt(D))));
```

将 A 置为 $C^{-1}D$。使用临时矩阵保护 C、D 不被修改。

矩阵单位加法

声明于<MatrixGrow.h>

```
#define _Mgr /* Matrix Grow */
```

描述

`_Mgr(矩阵, 数值)`

将矩阵增加数值倍的单位矩阵并返回。

矩阵分块

声明于<MatrixPartition.h>

```
#define _Mpt /* Matrix Partition */
```

描述

_Mpt(矩阵，自然数,自然数，自然数,自然数)

返回矩阵中指定左上角、形状的子矩阵。

示例

```
_Mvl3(_Mst(_Mpt(A, 0,0, 3,3), -= ,_Mpt(A, 3,3, 3,3)));
```

将 A 中左上角 3x3 的分块减去其右下方的 3x3 分块。