C 语言使用范围非常广，如何实现在 matalb 中执行 C 程序一直是大家关心的
比如我有一个用 C 语言写的函数，实现了一个功能，如一个简单的函数：

```
double add(double x, double y)
{
return x + y;
}
```

现在我想要在 Matlab 中使用它，比如输入：

```
>> a = add(1.1, 2.2)

    3.3000
```

要得出以上的结果，那应该怎样做呢？

解决方法之一是要通过使用 MEX 文件，MEX 文件使得调用 C 函数和调用 Matlab 的内置函数一样方便。MEX 文件是由原 C 代码加上 MEX 文件专用的接口函数后编译而成的。可以这样理解，MEX 文件实现了一种接口，它把在 Matlab 中调用函数时输入的自变量通过特定的接口调入了 C 函数，得出的结果再通过该接口调回 Matlab。该特定接口的操作，包含在 mexFunction 这个函数中，由使用者具体设定。

所以现在我们要写一个包含 add 和 mexFunction 的 C 文件，Matlab 调用函数，把函数中的自变量（如上例中的 1.1 和 2.2）传给 mexFunction 的一个参数，mexFunction 把该值传给 add，把得出的结果传回给 mexFunction 的另一个参数，Matlab 通过该参数来给出在 Matlab 语句中调用函数时的输出值（如上例中的 a）。

值得注意的是，mex 文件是与平台有关的，以我的理解，mex 文件就是另类的动态链接库。在 matlab6.5 中使用 mex -v　选项，你可以看到最后 mex 阶段有类似如下的信息：

```
--> "del _lib94902.obj"

--> "del "test.exp""

--> "del "test.lib""
```

也就是说，虽然在 matlab6.5 生成的是 dll 文件，但是中间确实有过 lib 文件生成。

比如该 C 文件已写好，名为 add.c。那么在 Matlab 中，输入：

```
>> mex add.c
```

就能把 add.c 编译为 MEX 文件（编译器的设置使用指令 mex -setup），在 Windows 中，MEX 文件类型为 mexw32，即现在我们得出 add.mexw32 文件。现在，我们就可以像调用 M 函数那样调用 MEX 文件，如上面说到的例子。所以，通过 MEX 文件，使用 C 函数就和使用 M 函数是一样的了。

我们现在来说 mexFunction 怎样写。

mexFunction 的定义为：

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])

{

/*.....*/

}
```

可以看到，mexFunction 是没返回值的，它不是通过返回值把结果传回 Matlab 的，而是通过对参数 plhs 的赋值。mexFunction 的四个参数皆是说明 Matlab 调用 MEX 文件时的具体信息，如这样调用函数时：

```
>> b = 1.1; c = 2.2;

>> a = add(b, c)
```

mexFunction 四个参数的意思为：

nlhs = 1，说明调用语句左手面（lhs－left hand side）有一个变量，即 a。

nrhs = 2，说明调用语句右手面（rhs－right hand side）有两个自变量，即 b 和 c。

plhs 是一个数组，其内容为指针，该指针指向数据类型 mxArray。因为现在左手面只有一个变量，即该数组只有一个指针，plhs[0]指向的结果会赋值给 a。

prhs 和 plhs 类似，因为右手面有两个自变量，即该数组有两个指针，prhs[0]指向了 b，prhs[1]指向了 c。要注意 prhs 是 const 的指针数组，即不能改变其指向内容。

因为 Matlab 最基本的单元为 array，无论是什么类型也好，如有 double array、cell array、struct array……所以 a,b,c 都是 array，b = 1.1 便是一个 1x1 的 double array。而在 C 语言中，Matlab 的 array 使用 mxArray 类型来表示。所以就不难明白为什么 plhs 和 prhs 都是指向 mxArray 类型的指针数组。

完整的 add.c 如下：

```
#include "mex.h" //  使用 MEX 文件必须包含的头文件

//  执行具体工作的 C 函数
```

```
double add(double x, double y)
{
    return x + y;
}
// MEX 文件接口函数
void mexFunction(int nlhs,mxArray *plhs[], int nrhs,const mxArray *prhs[])
{
    double *a;
    double b, c;
    plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
    a = mxGetPr(plhs[0]);
    b = *(mxGetPr(prhs[0]));
    c = *(mxGetPr(prhs[1]));
    *a = add(b, c);
}
```

mexFunction 的内容是什么意思呢？我们知道，如果这样调用函数时：

```
>> output = add(1.1, 2.2);
```

在未涉及具体的计算时，output 的值是未知的，是未赋值的。所以在具体的程序中，我们建立一个 1x1 的实 double 矩阵（使用 mxCreateDoubleMatrix 函数，其返回指向刚建立的 mxArray 的指针），然后令 plhs[0]指向它。接着令指针 a 指向 plhs [0]所指向的 mxArray 的第一个元素（使用 mxGetPr 函数，返回指向 mxArray 的首元素的指针）。同样地，我们把 prhs[0]和 prhs [1]所指向的元素（即 1.1 和 2.2）取出来赋给 b 和 c。于是我们可以把 b 和 c 作自变量传给函数 add，得出给果赋给指针 a 所指向的 mxArray 中的元素。因为 a 是指向 plhs[0]所指向的 mxArray 的元素，所以最后作输出时，plhs[0]所指向的 mxArray 赋值给 output，则 output 便是已计算好的结果了。

上面说的一大堆指向这指向那，什么 mxArray，初学者肯定都会被弄到头晕眼花了。很抱歉，

要搞清楚这些乱糟糟的关系，只有多看多练。

实际上 mexFunction 是没有这么简单的，我们要对用户的输入自变量的个数和类型进行测试，以确保输入正确。如在 add 函数的例子中，用户输入 char array 便是一种错误了。

从上面的讲述中我们总结出，MEX 文件实现了一种接口，把 C 语言中的计算结果适当地返回给 Matlab 罢了。当我们已经有用 C 编写的大型程序时，大可不必在 Matlab 里重写，只写个接口，做成 MEX 文件就成了。另外，在 Matlab 程序中的部份计算瓶颈（如循环），可通过 MEX 文件用 C 语言实现，以提高计算速度。

以上是对 mex 文件的初步认识，下面详细介绍如何用 c 语言编写 mex 文件：

1 为什么要用 C 语言编写 MEX 文件

MATLAB 是矩阵语言，是为向量和矩阵操作设计的，一般来说，如果运算可以用向量或矩阵实现，其运算速度是非常快的。但若运算中涉及到大量的循环处理，MATLAB 的速度的令人难以忍受的。解决方法之一为，当必须使用 for 循环时，把它写为 MEX 文件，这样不必在每次运行循环中的语句时 MATLAB 都对它们进行解释。

2 编译器的安装与配置

要使用 MATLAB 编译器，用户计算机上应用事先安装与 MATLAB 适配的以下任何一种 ANSI C/C++编译器：

5.0、6.0 版的 MicroSoft Visual C++(MSVC)

5.0、5.2、5.3、5.4、5.5 版的 Borland C++

LCC(由 MATLAB 自带，只能用来产生 MEX 文件)

下面是安装与配置 MATLAB 编译器应用程序 MEX 的设置的步骤：

(1)在 MATLAB 命令窗口中运行 mex –setup，出现下列提示：

Please choose your compiler for building external interface (MEX) files:

Would you like mex to locate installed compilers [y]/n?

(2)选择 y，MATLAB 将自动搜索计算机上已安装的外部编译器的类型、版本及所在路径，并列出来让用户选择：

Select a compiler:

[1] Borland C++Builder version 6.0 in C:\Program Files\Borland

[2] Digital Visual Fortran version 6.0 in C:\Program Files\Microsoft Visual Studio

[3] Lcc C version 2.4 in D:\MATLAB6P5P1\sys\lcc

[4] Microsoft Visual C/C++ version 6.0 in C:\Program Files\Microsoft Visual Studio

[0] None

Compiler:

(3)选择其中一种(在这里选择了 3)，MATLAB 让用户进行确认：

Please verify your choices:

Compiler: Lcc C 2.4

Location: D:\MATLAB6P5P1\sys\lcc

Are these correct?([y]/n):

(4)选择 y，结束 MATLAB 编译器的配置。

3 一个简单的 MEX 文件例子

【例 1】用 m 文件建立一个 1000×1000 的 Hilbert 矩阵。

```
tic
m=1000;
n=1000;
a=zeros(m,n);
for i=1:1000
        for j=1:1000
```

```
            a(i,j)=1/(i+j);
        end
end
toc
```

在 matlab 中新建一个 Matlab_1.cpp 文件并输入以下程序：

```
#include "mex.h"
//计算过程
void hilb(double *y,int n)
{
    int i,j;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            *(y+j+i*n)=1/((double)i+(double)j+1);
}
//接口过程
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
    double x,*y;
    int n;

    if (nrhs!=1)
```

```
        mexErrMsgTxt("One inputs required.");
    if (nlhs != 1)
        mexErrMsgTxt("One output required.");
    if (!mxIsDouble(prhs[0])||mxGetN(prhs[0])*mxGetM(prhs[0])!=1)
        mexErrMsgTxt("Input must be scalars.");
    x=mxGetScalar(prhs[0]);
    plhs[0]=mxCreateDoubleMatrix(x,x,mxREAL);
    n=mxGetM(plhs[0]);
    y=mxGetPr(plhs[0]);
    hilb(y,n);
}
```

该程序是一个 C 语言程序，它也实现了建立 Hilbert 矩阵的功能。在 MATLAB 命令窗口输入以下命令：mex Matlab_1.cpp，即可编译成功。进入该文件夹，会发现多了两个文件：Matlab_1.asv 和 Matlab_1.dll，其中 Matlab_1.dll 即是 MEX 文件。运行下面程序：

```
tic
a=Matlab_1(1000);
toc
  elapsed_time =
       0.0470
```

由上面看出，同样功能的 MEX 文件比 m 文件快得多。

4 MEX 文件的组成与参数

MEX 文件的源代码一般由两部分组成：

(1)计算过程。该过程包含了 MEX 文件实现计算功能的代码，是标准的 C 语言子程序。

(2)入口过程。该过程提供计算过程与 MATLAB 之间的接口，以入口函数 mxFunction 实现。在该过程中，通常所做的工作是检测输入、输出参数个数和类型的正确性，然后利用 mx-函数得到 MATLAB 传递过来的变量(比如矩阵的维数、向量的地址等)，传递给计算过程。

MEX 文件的计算过程和入口过程也可以合并在一起。但不管那种情况，都要包含#include "mex.h"，以保证入口点和接口过程的正确声明。注意，入口过程的名称必须是 mexFunction，并且包含四个参数，即：

void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])

其中，参数 nlhs 和 nrhs 表示 MATLAB 在调用该 MEX 文件时等式左端和右端变量的个数，例如在 MATLAB 命令窗口中输入以下命令：

[a,b,c]=Matlab_1(d,e,f,g)

则 nlhs 为 3，nrhs 为 4。

MATLAB 在调用 MEX 文件时，输入和输出参数保存在两个 mxArray*类型的指针数组中，分别为 prhs[]和 plhs[]。prhs[0]表示第一个输入参数，prhs[1]表示第二个输入参数，…，以此类推。如上例中，d→prhs[0]，e→prhs[1]，f→prhs[2]，f→prhs[3]。同时注意，这些参数的类型都是 mxArray *。

接口过程要把参数传递给计算过程，还需要从 prhs 中读出矩阵的信息，这就要用到下面的 mx-函数和 mex-函数。

5 常用的 mex-函数和 mx-函数

在 MATLAB6.5 版本中，提供的 mx-函数有 106 个，mex-函数有 38 个，下面我们仅介绍常用的函数。

5.1 入口函数 mexFunction

该函数是 C MEX 文件的入口函数，它的格式是固定的：

void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])

说明：MATLAB 函数的调用方式一般为：[a,b,c,…]=被调用函数名称(d,e,f,…)，nlhs 保存了等号左端输出参数的个数，指针数组 plhs 具体保存了等号左端各参数的地址，注意在 plhs 各元素针向的 mxArray 内存未分配，需在接口过程中分配内存；prhs 保存了等号右端输入参数的个数，指针数组 prhs 具体保存了等号右端各参数的地址，注意 MATLAB 在调用该 MEX 文件时，各输入参数已存在，所以在接口过程中不需要再为这些参数分配内存。

5.2 出错信息发布函数 mexErrMsgTxt，mexWarnMsgTxt

两函数的具体格式如下：

```
#include "mex.h"

void mexErrMsgTxt(const char *error_msg);

void mexWarnMsgTxt(const char *warning_msg);
```

其中 error_msg 包含了要显示错误信息，warning_msg 包含要显示的警告信息。两函数的区别在于 mexErrMsgTxt 显示出错信息后即返回到 MATLAB，而 mexWarnMsgTxt 显示警告信息后继续执行。

5.3 mexCallMATLAB 和 mexEvalString

两函数具体格式如下：

```
#include "mex.h"

int mexCallMATLAB(int nlhs, mxArray *plhs[],

int nrhs, mxArray *prhs[], const char *command_name);

int mexEvalString(const char *command);
```

mexCallMATLAB 前四个参数的含义与 mexFunction 的参数相同，command_name 可以 MATLAB 内建函数名、用户自定义函数、M 文件或 MEX 文件名构成的字符串，也可以 MATLAB 合法的运算符。

mexEvalString 用来操作 MATLAB 空间已存在的变量，它不返回任何参数。

mexCallMATLAB 与 mexEvalString 差异较大，请看下面的例子。

【例 2】试用 MEX 文件求 5 阶完全图邻接矩阵 的特征值及对应的特征向量。

5 阶完全图的邻接矩阵为：（这里找不到图片了，抱歉。不过不会影响您对本文的理解。）

下面是求该矩阵的 MEX 文件。

[Matlab_2.cpp]

```
#include "mex.h"
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
    double x;
    mxArray *y,*z,*w;
    int n;
    if (nrhs!=1)
        mexErrMsgTxt("One inputs required.");
    if (nlhs != 3)
        mexErrMsgTxt("Three output required.");
    if (!mxIsDouble(prhs[0])||mxGetN(prhs[0])*mxGetM(prhs[0])!=1)
        mexErrMsgTxt("Input must be a scalar.");
    x=mxGetScalar(prhs[0]);
    plhs[0]=mxCreateDoubleMatrix(x,x,mxREAL);
    plhs[1]=mxCreateDoubleMatrix(x,x,mxREAL);
    plhs[2]=mxCreateDoubleMatrix(x,x,mxREAL);
    n=mxGetM(plhs[0]);
    y=plhs[0];
    z=plhs[1];
    w=plhs[2];
    //利用 mexCallMATLAB 计算特征值
    mexCallMATLAB(1,&plhs[1],1,prhs,"ones");
```

```
    mexCallMATLAB(1,&plhs[2],1,prhs,"eye");
    mexCallMATLAB(1,&plhs[0],2,&plhs[1],"-");
    mexCallMATLAB(2,&plhs[1],1,&plhs[0],"eig");
    //演示 mexEvalString 的功能
    mexEvalString("y=y*2");
    mexEvalString("a=a*2");
}
```

在 MATLAB 命令窗口输入以下命令：

```
>> mex Matlab_2.cpp
>> clear
>> a=magic(5)
a =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
>> [y,z,w]=Matlab_2(5)
??? Undefined function or variable 'y'.
a =
    34    48     2    16    30
```

46	10	14	28	32
8	12	26	40	44
20	24	38	42	6
22	36	50	4	18

y =

0	1	1	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

z =

0.8333	-0.1667	-0.1667	0.2236	0.4472
-0.1667	0.8333	-0.1667	0.2236	0.4472
-0.1667	-0.1667	0.8333	0.2236	0.4472
-0.5000	-0.5000	-0.5000	0.2236	0.4472
0	0	0	-0.8944	0.4472

w =

-1	0	0	0	0
0	-1	0	0	0
0	0	-1	0	0
0	0	0	-1	0
0	0	0	0	4

由上面可以看出，K5 的特征值为–1 和 4，其中–1 是四重根。MATLAB 提供了 mexGetVariable、mexPutVariable 函数，以实现 MEX 空间与其它空间交换数据的任务，具体可以参看 MATLAB 帮助文档。

5.4 建立二维双精度矩阵函数 mxCreateDoubleMatrix

其格式具体如下：

#include "matrix.h"

mxArray *mxCreateDoubleMatrix(int m, int n, mxComplexity ComplexFlag);

其中 m 代表行数，n 代表列数，ComplexFlag 可取值 mxREAL 或 mxCOMPLEX。如果创建的矩阵需要虚部，选择 mxCOMPLEX，否则选用 mxREAL。

类似的函数有： mxCreateCellArray
创建 n 维元胞 mxArray

mxCreateCellMatrix
创建二维元胞 mxArray

mxCreateCharArray
创建 n 维字符串 mxArray

mxCreateCharMatrixFromStrings
创建二维字符串 mxArray

mxCreateDoubleMatrix
创建二维双精度浮点 mxArray

mxCreateDoubleScalar
创建指定值的二维精度浮点 mxArray

mxCreateLogicalArray
创建 n 维逻辑 mxArray，初值为 false

mxCreateLogicalMatrix
创建二维逻辑 mxArray，初值为 false

mxCreateLogicalScalar
创建指定值的二维逻辑 mxArray

mxCreateNumericArray

创建 n 维数值 mxArray

mxCreateNumericMatrix
创建二维数值 mxArray，初值为 0

mxCreateScalarDouble
创建指定值的双精度 mxArray

MxCreateSparse
创建二维稀疏 mxArray

mxCreateSparseLogicalMatrix
创建二维稀疏逻辑 mxArray

MxCreateString
创建指定字符串的 1 n 的串 mxArray

mxCreateStructArray
创建 n 维架构 mxArray

mxCreateStructMatrix
创建二维架构 mxArray

5.5 获取行维和列维函数 mxGetM、mxGetN

其格式如下：

#include "matrix.h"

int mxGetM(const mxArray *array_ptr);

int mxGetN(const mxArray *array_ptr);

与之相关的还有：

mxSetM：设置矩阵的行维

mxSetN：设置矩阵的列维

5.6 获取矩阵实部和虚部函数 mxGetPr、mxGetPi

其格式如下：

```
#include "matrix.h"

double *mxGetPr(const mxArray *array_ptr);

double *mxGetPi(const mxArray *array_ptr);
```

与之相关的函数还有：

mxSetPr：设置矩阵的实部

mxSetPi：设置矩阵的虚部

【例 3】实现字符串的倒序输出。

```
#include "mex.h"

void revord(char *input_buf,int buflen,char *output_buf)

{

    int i;

    //实现字符串倒序

    for(i=0;i<buflen-1;i++)

        *(output_buf+i)=*(input_buf+buflen-i-2);

}

void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])

{

    //定义输入和输出参量的指针

    char *input_buf,*output_buf;

    int buflen,status;
```

```
//检查输入参数个数
if(nrhs!=1)
    mexErrMsgTxt("One input required.");
else if(nlhs>1)
    mexErrMsgTxt("Too many output arguments.");
//检查输入参数是否是一个字符串
if(mxIsChar(prhs[0])!=1)
    mexErrMsgTxt("Input must be a string.");
//检查输入参数是否是一个行变量
if(mxGetM(prhs[0])!=1)
    mexErrMsgTxt("Input must a row vector.");
//得到输入字符串的长度
buflen=(mxGetM(prhs[0])*mxGetN(prhs[0]))+1;
//为输入和输出字符串分配内存
input_buf=mxCalloc(buflen,sizeof(char));
output_buf=mxCalloc(buflen,sizeof(char));
//将输入参量的 mxArray 结构中的数值拷贝到 C 类型字符串指针
status=mxGetString(prhs[0],input_buf,buflen);
if(status!=0)
    mexWarnMsgTxt("Not enough space. String is truncated.");
//调用 C 程序
revord(input_buf,buflen,output_buf);
```

```
    plhs[0]=mxCreateString(output_buf);

}
```

这个程序中需要注意的地方是 mxCalloc 函数，它代替了标准 C 程序中的 calloc 函数用于动态分配内存，而 mxCalloc 函数采用的是 MATLAB 的内存管理机制，并将所有申请的内存初始化为 0，因此凡是 C 代码需要使用 calloc 函数的地方，对应的 Mex 文件应该使用 mxCalloc 函数。同样，凡是 C 代码需要使用 realloc 函数的地方，对应的 Mex 文件应该使用 mxRealloc 函数。

在 MATLAB 命令窗口中对 revord.cpp 程序代码编译链接：

```
>> mex revord.cpp
```

在 MATLAB 命令窗口中对 C-MEX 文件 revord.dll 进行测试：

```
>> x='I am student.';

>> revord(x)

ans =

.tneduts ma I
```