依赖条件：1 #pragma comment ( lib, "winmm.lib" )

音频的输入大体分三步

1 打开设备 -----waveInOpen（打开一个音频输入设备）、

2 开始录音------waveInStart开始录音

3关闭设备-------waveInClose关闭录音。之前调用一下waveInReset，这样可以清掉尚在等待录音的缓冲区

常用的相关API为：

waveInOpen（打开一个音频输入设备）

waveInPrepareHeader（为一个即将在waveInAddBuffer中调用的输入缓冲区准备头部）

waveInAddBuffer（添加一个输入用的数据缓冲区）

waveInStart（开始录音）

waveInClose（关闭音频输入设备）等几个，以及需要在waveInOpen中指定的一个回调函数或者线程，其作用是在一个数据缓冲区被录满后被调用，以对这些数据进行处理，和其他一些相关的操作。注意这里的一个数据缓冲区。

下面详细说明他们相对应的关系。

1---------------waveInOpen

MMRESULT waveInOpen( LPHWAVEIN phwi,  //phwi是返回的句柄存放地址  
UINT uDeviceID,   // uDeviceID是要打开的音频设备ID号，一般都指定为WAVE\_MAPPER  
LPWAVEFORMATEX pwfx,  
DWORD dwCallback,  //dwCallback则为指定的回调函数或线程,窗口等的地址  
DWORD dwCallbackInstance,   // dwCallbackInstance为需要向回调函数或线程送入的用户参数  
DWORD fdwOpen  // fdwOpen指定回调方式：CALLBACK\_FUNCTION, CALLBACK\_THREAD和CALLBACK\_WINDOW  
);  
  至于pwfx，则比较关键，它指定了要以什么音频格式打开音频输入设备， 它是一个结构WAVEFORMATEX：  
typedef struct { WORD wFormatTag;  
WORD nChannels;  
DWORD nSamplesPerSec;  
DWORD nAvgBytesPerSec;  
WORD nBlockAlign;  
WORD wBitsPerSample;  
WORD cbSize;  
} WAVEFORMATEX;  
    可以在wFormatTag中指定一些压缩的音频格式，如G723.1，TURE DSP，等之类。不过一般都是选用WAVEFORMAT\_PCM格式，即未压缩的音频格式，至于压缩，可以在录完后调用下面将要谈到的ACM单独进行。  
　　nChannels为声道数，1或者2。

        nSamplesPerSec为每秒采样数，8000、11025、22050、44100为几个标准值。

        nAvgBytesPerSec为每秒平均的字节数，在PCM方式中就等于nChannels\*nSamplesPerSec\*wBitsPerSample/8，但对于其它的压缩的音频格式，由于很多压缩方式是按时间片进行的，如G723.1，就是以30ms为一个压缩单位，这样，nAvgBytesPerSec只是一个大概的数字，并不准确，程序中的计算是不应该以这个量为准的。这一点在下面的压缩音频输出和ACM音频压缩中非常重要。

       nBlockAlign是一个比较特殊的值，表示对音频处理时的最小处理单位，对于PCM非压缩，它就是wBitsPerSample\*nChannels/8，而对于非压缩格式，则表示压缩/解压处理的最小单位了，如G723.1，就是30ms的数据大小（20bytes或者24bytes）。

       wBitsPerSample就是每采样值的位数，8或者16。

       cbSize则是表示该WAVEFORMATEX的结构在标准的头部之后还有多少字节数，对于很多非PCM的音频格式，有一些自己的定义格式参数，这些就紧跟在标准的WAVEFORMATEX后面，其大小就由cbSize指定。对于PCM格式而言，为0，或者忽略不检查。  
　　这样，指定了这些参数后，你应该就能够打开音频输入设备了。下面要做的事情就是准备几个用做录音的缓冲区。常准备多个缓冲区，并在回调中循环使用。对于缓冲区，得使用waveInPerpareHeader准备一下头部，这个API比较简单，如果你是循环使用缓冲区，对每个缓冲区也只需要调用一次waveInPrepareHeader。为什么要使用一次就可以。参看waveInPerpareHeader说明就明白。此函数功能就是定位缓冲区的数据区地址，和数据大小。以便为系统所用。  
  A）

首先得确定一下需要用什么回调方式，即在某个时间片的音频数据被录完后，Windows将通过这个回调来激活对这些数据的处理过程，一般用到的无非是FUNCTION、THREAD和EVENT这几类，而比较方便简单的就是FUNCTION和THREAD了。FUNCTION方式是指Windows会调用你这个函数，而THREAD则是由 Windows来激活你所指定的线程。这些都在waveInOpen中指定。  
   b)  
　　一切准备好之后，就可以调用waveInAddBuffer和waveInStart开始录音了，只要你一调用这个waveInStart，录音就开始了，即使这个缓冲区录满之后你没有加入新的缓冲区进去，录音也不会停，只是这中间的语音数据全都丢了。当通过 waveInAddBuffer送入的缓冲区被录满后，Windows就会通过你在waveInOpen中指定的方式进行回调，在回调中把录好的语音数据取出来，并且，如果还想继续录音的话，得将下一个缓冲区添加进去。考虑到这个处理是有时间延迟的，而且音频对时间很敏感，

    一般都要先预加入若干个缓冲区，有人提出：比如，一共定义了8个缓冲区，而为了保险起见，最好保证任一时刻至少有3个缓冲区可被录音使用，那么在开始录音时，则先加入4个缓冲区，然后在回调中，如果当前录好的缓冲区第n个，则对第(n+4)%8调用waveInAddBuffer，这时，还有第(n+1)%8,(n+2)%8, (n+3)%8这三个缓冲区可用，即基本上就可以保证所录得音频中不会有断开的间隔。比如0，1，2，3这些个先加入，当0好的时候对4，5 ，6 ，7调用waveInAddBuffer。

   如此这样何不：开始的时候把8个全部放入缓冲区，当一个缓冲区满后调用回调，处理后立即把这个缓冲区重用，继续添加到缓冲区队列中。不更简单明了。如下

mmReturn = ::waveInPrepareHeader ( m\_hRecord, pHdr, sizeof(WAVEHDR) ); //准备

mmReturn = ::waveInAddBuffer ( m\_hRecord, pHdr, sizeof(WAVEHDR) );//添加

注意这两步都是在回调，或者线程中处理的。

c)

想结束录音时，最好在waveInClose之前调用一下waveInReset，这样可以清掉尚在等待录音的缓冲区，这里常见的问题是等待的缓冲区清理了，可是正在用的缓冲区怎么办？如果这个时候就用waveInClose，那么系统会出错。解决方法一：在回调函数中注意，一个缓冲区满后，不要再用waveInAddBuffer增加缓存，当缓冲区用到1的时候调用waveInReset清掉尚在等待录音的缓冲区继续waveInClose。

总结上面的注意3点：回调的选取，注意缓冲区的原理，注意结束的处理

windows waveform方式实现录音要通过这么几步：（注意顺序！！）

一、打开录音设备

waveinopen（）函数

注意，调用之前要填写好wav头信息（包含采样率、采样位数等）；还要定义好回调函数等，回调函数的解释后面讲。

二、准备好录音缓存空间

waveinprepareheader（）函数

这一步为了准备好将要送入录音设备的缓存，以便之后可以供之使用。

一般至少需要准备两块缓存。因为录音不能间断，当一块填满时没有时间等待你去送入下一块缓存，所以必须提前就准备好。

三、将缓存送入录音设备

waveinaddbuffer（）函数

将缓存送入录音设备，供之存入已录下的音频。开始录音时，应至少送入两块不同的缓存，即调用两次这个函数。之后为了不致录音产生间断，应保证至少有一块缓存在录音时为空，以备衔接。

四、开始录音

waveinstart（）函数

当以上的工作都准备好时，便可调用此函数开始录音了。一旦调用，录音设备便立即录音并存入已经送来的缓存块内，当被送来的有多个缓存块时，按照FIFO的原则向缓存块内存入录音数据。此函数执行之后可以执行一个while（）循环，来等待录音设备录音。为了减少cpu使用率，可以在循环中加人sleepex（x，TRUE），x单位ms，TRUE必须要有。

每个缓冲块存满时，会产生一个回调信号，并调用回调函数（或回调窗口等，具体定义在waveinopen函数内，这里只讲回调函数的情况）；回调信号自动被回调函数接收，回调函数根据回调信号来执行各种相应的操作。回调函数执行完后，程序跳回到原来执行位置继续执行。回调函数的具体如下：

回调信号一般有三个，对应着三种回调函数被调用的情况：

1、  WIM\_OPEN

当执行waveinopen（）函数时，会调用回调函数，并产生这个回调信号。代表录音设备已经打开。在这次回调函数的调用中，可以自己设定一些操作，也可以没有操作。

2、  WIM\_DATA

当每块缓存块填满时，产生这个回调信号，并调用回调函数。在这次调用中，回调函数应当完成这样的工作，以便录音连续进行：

        将存满的缓存块处理，例如存入文件，或送入其他设备；

        向录音设备送入新的缓存块；录音设备任何时刻应当拥有不少于2个的缓存块，以保证录音不间断性。

3、  WIM\_CLOSE

当调用waveinclose函数时，会产生这个回调信号，代表录音设备关闭成功。这次回调函数调用中，可以执行相应的一些关闭文件保存信息等等的操作，自定义。

五、停止录音，关闭设备

waveinstop（）停止录音

waveinreset（）复位

waveinclose（）关闭设备

    依次调用这些函数，来结束录音。

最后，注意在代码开头要包含windows.h和mmsystem.h两个头文件，还要加人库winmm.lib，用#pragma comment(lib,”winmm.lib”)即可。

主要顺序就是这些，注意每一步要完成的工作，一旦没有按顺序执行或者没有把每步应当完成的工作做完，录音是不能够启动的