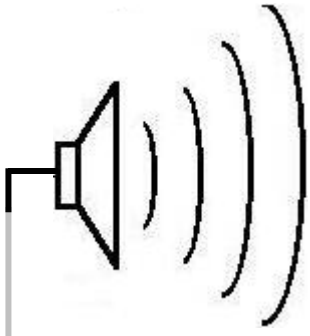


Introduction to MMSYSTEM.H

A written report on the basic functions of the multi-media-system header file for audio input/output programming

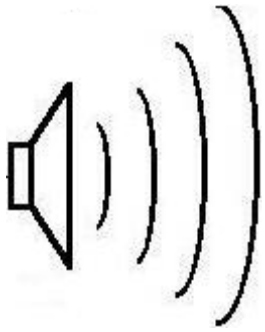
Prepared for:
Digital Signal Processing class
ELC 152 A

Prepared by:
Jeremiah C. Kwok
BS ECE '06

Instructor:
Mr. Luisito Agustin

Submitted on:
July 25, 2005

TABLE OF CONTENTS



I. Introduction: Why mmsystem?

From our basic background on C/C++ programming, we know that we can take in different kinds of input for various programs. We have been familiar of taking input from the computer keyboard, by typing values and allowing our programs to interpret and compute them in the way we created them to do so. Also from our Microprocessor class we studied how the input can be changed as we shift from one microprocessor to another, and how we can program these devices to take in inputs from switches and the like. But playing with digital signals and processing them is not limited to these types of input data. From past seminars in class we have found out how audio signals can be used, manipulated, converted, and even compressed (XMCS player, ADPCM compression, etc.) so that they can become our digital data. Thus, it can serve a distinct advantage for us if we can learn to create programs, not just in C/C++, that can utilize the sound coming in and out of our sound cards or other audio devices. Mmsystem.h is a header file that provides you with such capabilities.

For most of us, we are used to common header files such as stdio.h (library for standard input/output streams), math.h (descriptions for various math functions), string.h (string-handling capabilities), and stdlib.h (miscellaneous functions – malloc(), free(), etc.) Mmsystem.h is yet another library that we can add to our collection of header files from which we can access functions that will allow us to incorporate audio signals to our programs and use them for features such as voice recorder/players, mixers, volume controls, and the like. This header file uses the library winmm.lib for executing the functions. For many of us who have projects involving voice data, and are using open-source programming tools such as Dev C++, mmsystem.h can be a vital part in your program flow. This is also useful in Visual Basic programming, among others.

For my presentation, I have chosen this report because the group I am part of for the DSP project aims to develop an XMCS recorder, and as much as multimedia functions played a vital role in the XMCS player, so will it do so as well in our recorder. In the process, not being a selfish individual, I opted to share what I have learned thus far regarding mmsystem functions, and hopefully help not just my group, but also others as well. ☺

II. Scope and limitations

Because the functions covered by the mmsystem library is very extensive and for some, quite complicated, this report is limited to a couple of basic functions that can be very useful for basic programming involving audio data taken from a voice recorder or sound card. Mmsystem structures will not be discussed in great detail, as well as the exact procedures for interfacing your sound card with a specific working program, although it might be mentioned in passing as some of the functions are discussed. These are highly-recommended extra topics that can be covered, and the reader is encouraged to read up on these topics on their own leisure. Useful links and related information can be found in the Reference section at the end of the report.

Because of the limited time and absence of short, user-friendly functioning codes, the examples provided here for the functions are not meant to run on their own. They are rather taken from a much larger code, and is only illustrated here for the sake of clarification and example. Thus typing them as is, and compiling them alone to see the code actually may not always work. The idea is to understand how the functions can be used, and what parameters and values the functions will return. Links to some of the larger, running codes are also found in the latter sections of this report. Also these functions work only in the Windows environment. Such for Linux-based applications, other functions with different names but similar purposes probably exist.

This written report was based from a 40-minute presentation prepared for the DSP series of seminars in class. Due to the lack of interaction for this seminar, the readers are more than encouraged to write down and compile their questions or comments as you read through the report. The presenter is more than willing to entertain these questions through the DSP groups, or through future classroom interactions. ☺

III. Some basic multimedia functions

A. auxSetVolume
This function sets the volume of the specified auxiliary output device.

Format:
MMRESULT auxSetVolume(**UINT** *uDeviceID*, **DWORD** *dwVolume*);

Parameters:
uDeviceID – Identifier of the auxiliary output device to be queried. Device identifiers are determined from the total number of devices present in the system. Values range from zero to one-less-than-the-number of devices present. Use the auxGetNumDevs function to determine the number of auxiliary devices in the system.

dwVolume – Specifies the new volume setting. The low-order word specifies the left-channel volume setting, and the high-order word specifies the right-channel setting. A value of 0xFFFF represents full volume, and 0x0000 is silence. Should the device not support both left and right volume control, the low-order word of dwVolume specifies the volume level, and the high-order word is ignored.

Return Values:
MMSYSERR_NOERROR → If successful and no error is detected.
MMSYSERR_BADDEVICEID → The specified device identifier is beyond range or non-existent.

Note: Volume settings are logarithmically interpreted. This means that the actual volume increase is the same when increasing the volume level from 0x5000 to 0x6000 as it is from 0x4000 to 0x5000.

Example:
retval = auxSetVolume(0, &H80008000)

**retval is declared as a long integer
**This sets the volume of both left and right channels to a 50% increase (H8000 halfway to HFFFF).
**The statement is for a 2-channel volume setting. If the device does not support two channels, it should be written as (&H8000).

B. auxGetVolume
This function retrieves the current volume setting of the specified auxiliary output device.

Format:
MMRESULT auxGetVolume(**UINT** *uDeviceID*, **LPDWORD** *lpdwVolume*);

Parameters:
uDeviceID – Identifier of the auxiliary output device to be queried.

lpdwVolume – Pointer to a variable to be filled with the current volume setting. The low-order word contains the left channel volume setting, and the high-order word has the volume setting of the right channel. A value of 0xFFFF represents full volume, and a value of 0x0000 is silence. Should the device not support both left and right volume controls, the low-order word contains the volume level. The full 16-bit settings with the auxSetVolume function are returned, irregardless of whether the device supports the full 16 bits of volume-level control.

Return Values:
MMSYSERR_NOERROR → If successful or no error detected.
MMSYSERR_BADDEVICEID → Error: specified device identifier is out of range.

Example: This short program done in Visual Basic determines the volume setting of a particular auxiliary device, confirms how many is detected, and displays the volume. (Phrases in // are comments.)

```
//Declaraction of variables to be used
Dim auxinfo As AUXCAPS                  //receives information about the device
Dim numvols As Long                     //identifies number of volumes on the device
Dim lrvol As Long                       //volumes of both channels
Dim lvol As Integer, rvol As Integer    //volumes of left and right channels
Dim retval As Long                      //return value

//Figure out whether the device has one or two volume settings.
retval = auxGetDevCaps(0, auxinfo, Len(auxinfo))
If retval <> 0 Then                     //error checking
  Debug.Print "Could not access auxiliary audio device 0 -- aborting."
End If
If (auxinfo.dwSupport And AUXCAPS_LRVOLUME) = AUXCAPS_LRVOLUME Then
  numvols = 2                           //separate left and right volumes
Else
  numvols = 1                           //only one overall volume
End If
```

```
retval = auxGetVolume(0, lrvol)          //Determine the current volume & displays it.
If numvols = 2 Then                      //Separate the left and right channel volumes.
  lvol = Val("&H" & Hex(lrvol And (Not &HFFFF0000)))
  rvol = (lrvol And &HFFFF0000) / &H10000
  Debug.Print "Left Channel volume: "; Hex(lvol)      //Displays results in hexadecimal.
  Debug.Print "Right Channel volume: "; Hex(rvol)
Else
  lvol = Val("&H" & Hex(lrvol And (Not &HFFFF0000)))
  Debug.Print "Volume: "; hex(lvol)                  //here, lvol is the overall volume
End If
```

C. auxGetNumDevs

This function retrieves the number of auxiliary output devices present in the system.

Format

UINT auxGetNumDevs(VOID);

Return Values

Returns the number of devices detected on the system. A return value of zero means that no devices are present or that an error occurred.

D. playSound

This function plays a sound specified by the given filename, resource, or system event. (A system event may be associated with a sound in the registry or in the WIN.INI file.)

Format:

BOOL PlaySound(**LPCSTR** *pszSound*, **HMODULE** *hmod*, **DWORD** *fdwSound*);

Parameters:

pszSound – A string that specifies the sound to play. The maximum length, including the null terminator, is 256 characters. If this parameter is NULL, any currently playing waveform sound is stopped. To stop a non-waveform sound, specify SND_PURGE in the fdwSound parameter.

hmod – Serves as the handle to the executable file that contains the resource to be loaded. This parameter must be NULL unless SND_RESOURCE is specified in fdwSound.

fdwSound – Flags for playing the sound. The following values are defined.

SND_APPLICATION	→ The sound is played using an application-specific association.
SND_ALIAS	→ The pszSound parameter is a system-event alias in the registry or the WIN.INI file. Do not use with either SND_FILENAME or SND_RESOURCE.
SND_ALIAS_ID	→ The pszSound parameter is a predefined sound identifier.
SND_ASYNC	→ The sound is played asynchronously and PlaySound returns immediately after beginning the sound. To terminate an asynchronously played waveform sound, call PlaySound with pszSound set to NULL.
SND_FILENAME	→ The pszSound parameter is a filename.
SND_LOOP	→ The sound plays repeatedly until PlaySound is called again with the pszSound parameter set to NULL. You must also specify the SND_ASYNC flag to indicate an asynchronous sound event.
SND_MEMORY	→ A sound event's file is loaded in RAM. The parameter specified by pszSound must point to an image of a sound in memory.
SND_NODEFAULT	→ No default sound event is used. If the sound cannot be found, PlaySound returns silently without playing the default sound.
SND_NOSTOP	→ The specified sound event will yield to another sound event that is already playing. If a sound cannot be played because the resource needed to generate that sound is busy playing another sound, the function immediately returns FALSE without playing the requested sound. → If this flag is not specified, PlaySound attempts to stop the currently playing sound so that the device can be used to play the new sound.
SND_NOWAIT	→ If the driver is busy, return immediately without playing the sound.
SND_PURGE	→ Sounds are to be stopped for the calling task. If pszSound is not NULL, all instances of the specified sound are stopped. If pszSound is NULL, all sounds that are playing on behalf of the calling task are stopped.
SND_RESOURCE	→ The pszSound parameter is a resource identifier; hmod must identify the instance that contains the resource.
SND_SYNC	→ Synchronous playback of a sound event. PlaySound returns after the sound event completes.

Return Values
Returns TRUE if sound is successfully played, or FALSE otherwise.

Note: The sound specified by pszSound must fit into available physical memory and playable through an installed audio device driver. PlaySound searches for sound files in the following directories: the current directory, the Windows directory, the Windows system directory, directories listed in the PATH environment variable, and the list of directories mapped in a network.

Note: If it cannot find the specified sound, PlaySound uses the default system event sound entry instead. If the function can find neither the system default entry nor the default sound, it makes no sound and returns FALSE.

Example: This VB program plays the Windows SystemStart event sound synchronously. Then it plays the file C:\Sounds\scream.wav for 5 seconds before stopping. (Phrases in // are comments.)

```
Dim retval As Long
retval = PlaySound("SystemStart", 0, SND_ALIAS Or SND_SYNC)
//Synchronously play the SystemStart sound.
//This function returns when the sound is done.
retval = PlaySound("C:\Sounds\scream.wav", 0, SND_FILENAME Or SND_ASYNC Or
SND_NODEFAULT Or SND_LOOP)          // plays the scream.wav file
Sleep 5000                          //wait for 5 seconds while sound loops
retval = PlaySound("", 0, SND_PURGE Or SND_NODEFAULT)  //stop playback
```

E. waveInOpen
This function opens the waveform-audio input device and readies it for recording. Also for reading incoming digitized audio data to the particular device's ADC, passing the device ID of the desired device.

Format:

```
MMRESULT waveInOpen(
  LPHWAVEIN          phwi,
  UINT_PTR           uDeviceID,
  LPWAVEFORMATEX     pwfx,
  DWORD_PTR          dwCallback,
  DWORD_PTR          dwCallbackInstance,
  DWORD              fdwOpen );
```

Parameters:

phwi – Pointer to a buffer that receives a handle that identifies the open waveform-audio input device. This handle is used to identify the device when calling other waveform-audio input functions. This parameter can be declared also as NULL if WAVE_FORMAT_QUERY is specified for fdwOpen.

uDeviceID – Identifier of the waveform-audio input device to open. It can either be a device identifier or a handler of an open waveform-audio input device. You can use the following flag instead of a device identifier.

WAVE_MAPPER → The function selects a waveform-audio input device capable of recording in the specified format.

pwfx – Pointer to a WAVEFORMATEX structure that identifies the desired format for recording an audio data. You can free this structure as soon as waveInOpen returns.

dwCallback – Points to a fixed callback function, an event handle, or the identifier of a thread to be called during waveform-audio recording to process messages related to the progress of recording. If no callback function is required, this value can be zero.

dwCallbackInstance – User-instance data passed to the callback mechanism.

fdwOpen – Flags for opening the device. It is defined by any of the following values:

CALLBACK_EVENT → The dwCallback parameter is an event handle.
CALLBACK_FUNCTION → The parameter is a callback procedure address.
CALLBACK_NULL → No callback mechanism. This is the default setting.
CALLBACK_THREAD → The parameter is a thread identifier.
CALLBACK_WINDOW → The parameter is a window handle.
WAVE_FORMAT_DIRECT → If this flag is specified, the ACM driver does not perform audio data conversion.
WAVE_FORMAT_QUERY → The function queries the device to determine whether it supports the given format or not. It does not directly open the device though.
WAVE_MAPPED → The uDeviceID parameter specifies a waveform-audio device to be mapped to by the wave mapper.

Return Values

MMSYSERR_NOERROR → If operation is successful
MMSYSERR_ALLOCATED → Error: Specified resource is already allocated.
MMSYSERR_BADDEVICEID → Error: Specified device identifier is out of range.
MMSYSERR_NODRIVER → Error: No device driver is present.
MMSYSERR_NOMEM → Error: Unable to allocate or lock memory.
WAVERR_BADFORMAT → Error: Attempt to open with an unsupported waveform-audio format.

Note: If you choose to have a window or thread receive callback information, the following messages are sent to the window procedure or thread to indicate the progress of waveform-audio input: MM_WIM_OPEN, MM_WIM_CLOSE, and MM_WIM_DATA. If you choose to have a function receive callback information, the following messages are sent to the function to indicate the progress of waveform-audio input: WIM_OPEN, WIM_CLOSE, and WIM_DATA.

Example: Part of a code for our project, XMC recorder using Dev C++.

```
//Checks if device can be used for recording.
if(waveInOpen(&hWaveIn, WAVE_MAPPER, &wfx, 0, 0, CALLBACK_NULL) !=
MMSYSERR_NOERROR)
  {fprintf(stderr, "unable to open WAVE_MAPPER device\n");
    ExitProcess(1);}
  printf("The Wave Mapper device was opened successfully!\n");
```

F. waveInClose
This function closes the given waveform-audio input device.

Format: **MMRESULT** waveInClose(**HWAVEIN** *hwi*);

Parameters:
hwi – Handles the waveform-audio input device. If the function succeeds, the handle is no longer valid after this call.

Return Values:

MMSYSERR_NOERROR	→ If operation is successful
MMSYSERR_INVALHANDLE	→ Error: Specified device handle is invalid.
MMSYSERR_NODRIVER	→ Error: No device driver is present.
MMSYSERR_NOMEM	→ Error: Unable to allocate or lock memory.
WAVERR_STILLPLAYING	→ Error: There are still buffers in the queue that are being used.

Note: If there are input buffers that have been sent with the waveInAddBuffer function and that haven't been returned to the application, the close operation will fail. Call the **waveInReset** function to mark all pending buffers as done.

Example: Part of a code for our project, XMC recorder using Dev C++.

```
waveInStop(hWaveIn);              //Stop recording from hWaveIn pointing to a device
waveInUnprepareHeader(hWaveIn, &WaveHeader, sizeof(WAVEHDR));
  if (!checkkey())
```

```
{waveInClose(hWaveIn);          //Closes device if key is pressed to stop recording
 return 0; }
```

G. waveInAddBuffer

This function sends an input buffer to the given waveform-audio input device. When the buffer is filled, the application is notified.

Format:

MMRESULT waveInAddBuffer(**HWAVEIN** *hwi*, **LPWAVEHDR** *pwh*, **UINT** *cbwh*);

Parameters:

hwi – Handle for the waveform-audio input device.

pwh – Points to a WAVEHDR structure that identifies the buffer.

cbwh – Size (bytes) of the WAVEHDR structure.

Return Values:

MMSYSERR_NOERROR	→ If operation is successful
MMSYSERR_INVALHANDLE	→ Error: Specified device handle is invalid.
MMSYSERR_NODRIVER	→ Error: No device driver is present.
MMSYSERR_NOMEM	→ Error: Unable to allocate or lock memory.
WAVERR_UNPREPARED	→ The buffer pointed to by the pwh parameter hasn't been prepared or is not ready to be used.

Note: The buffer stores the recorded digital audio. After recording and the buffer is full, the driver signals and is processed by another program when it is full. The driver then stores data into a second, similarly-sized buffer. Both have to be declared. This is simultaneously stored and written to disk. The buffer to be used must first be prepared with the **waveInPrepareHeader** function before it is passed to this function.

H. / I. waveInStart/Stop

This starts/stops input on the given waveform-audio input device.

Format:

MMRESULT waveInStart(**HWAVEIN** *hwi*); → waveInStart
MMRESULT waveInStop(**HWAVEIN** *hwi*); → WaveInStop

Parameters:

hwi – Handle to the waveform-audio input device.

Return Values:

MMSYSERR_NOERROR	→ If record operation is successful.
MMSYSERR_INVALHANDLE	→ Error: Specified device handle is invalid.
MMSYSERR_NODRIVER	→ Error: No device driver is present.
MMSYSERR_NOMEM	→ Error: Unable to allocate or lock memory.

Note: Calling this function when input is already started has no effect, and the function returns zero.

J. waveOutOpen

This function opens the given waveform-audio output device to enable sound playback after recording.

Format:

MMRESULT waveOutOpen(

LPHWAVEOUT	*phwo*,
UINT_PTR	*uDeviceID*,
LPWAVEFORMATEX	*pwfx*,
DWORD_PTR	*dwCallback*,
DWORD_PTR	*dwCallbackInstance*,
DWORD	*fdwOpen*

);

Parameters:

Parameters have same description as in the waveInOpen function. (See page 9)

Return Values:

MMSYSERR_NOERROR	→ If operation is successful
MMSYSERR_ALLOCATED	→ Error: Specified resource is already allocated.
MMSYSERR_BADDEVICEID	→ Error: Specified device identifier is out of range.
MMSYSERR_NODRIVER	→ Error: No device driver is present.
MMSYSERR_NOMEM	→ Error: Unable to allocate or lock memory.
WAVERR_BADFORMAT	→ Error: Attempt to open with an unsupported waveform-audio format.
WAVERR_SYNC	→ The device is synchronous but waveOutOpen was called w/out the use of WAVE_ALLOWSYNC flag.

Note: Use the waveOutGetNumDevs function to determine the number of waveform-audio output devices present in the system. The structure pointed to by pwfx can be extended to include type-specific information for certain data formats. For example, for PCM data, an extra UINT is added to specify the number of bits per sample. Use the

PCMWAVEFORMAT structure in this case. For all other waveform-audio formats, use the WAVEFORMATEX structure to specify the length of the additional data.

K. waveOutClose
This function closes the given waveform-audio output device (useful after recording).

Format:
MMRESULT waveOutClose(**HWAVEOUT** *hwo*);

Parameters:
hwo – Provides handle to the audio output device. If the function is successful, the handle is no longer valid afterwards.

Return Values:

MMSYSERR_NOERROR	→ If operation is successful
MMSYSERR_INVALHANDLE	→ Error: Specified device handle is invalid.
MMSYSERR_NODRIVER	→ Error: No device driver is present.
MMSYSERR_NOMEM	→ Error: Unable to allocate or lock memory.
WAVERR_STILLPLAYING	→ Error: There are still buffers in the queue that are being used.

Note: If the device is still playing a waveform-audio file, the close operation fail will fail. Use the **waveOutReset** function to terminate playback before calling waveOutClose.

L. waveOutSetPlaybackRate
The function sets the playback rate for the specified waveform-audio output device.

Format:
MMRESULT waveOutSetPlaybackRate(**HWAVEOUT** *hwo*, **DWORD** *dwRate*);

Parameters:
hwo – Provides a handle to the waveform-audio output device.

dwRate – Sets new playback rate setting. This setting is a multiplier to the current change in playback rate from the original authored setting. The value must be a positive number.

The playback rate is specified as a fixed-point value. The high-order word contains the signed integer part of the number, and the low-order word contains the fractional part. A value of 0x8000 in the low-order word represents one-half, and 0x4000 represents

one-quarter. For example, the value 0x00010000 specifies a multiplier of 1.0 (no playback rate change), and a value of 0x000F8000 specifies a multiplier of 15.5.

Return Values:

MMSYSERR_NOERROR	→ If change to playback rate is successful.
MMSYSERR_INVALHANDLE	→ Error: Specified device handle is invalid.
MMSYSERR_NODRIVER	→ Error: No device driver is present.
MMSYSERR_NOMEM	→ Error: Unable to allocate or lock memory.
MMSYSERR_NOTSUPPORTED	→ Error: Declared function isn't supported.

Note: Changing the playback rate does not change the sample rate but does change the playback time. Not all devices support playback rate changes. Check other available functions in your library if playback rate change is supported.

M. waveOutWrite

This function sends a data block to the given waveform-audio output device.

Format:

MMRESULT waveOutWrite(**HWAVEOUT** *hwo*, **LPWAVEHDR** *pwh*, **UINT** *cbwh*);

Parameters:

hwo – Provides handle to the waveform-audio output device.

pwh – Points to a WAVEHDR structure containing information about the data block.

cbwh – Size (bytes) of the WAVEHDR structure.

Return Values:

MMSYSERR_NOERROR	→ If data block is successfully written.
MMSYSERR_INVALHANDLE	→ Error: Specified device handle is invalid.
MMSYSERR_NODRIVER	→ Error: No device driver is present.
MMSYSERR_NOMEM	→ Error: Unable to allocate or lock memory.
WAVERR_UNPREPARED	→ Error: The data block pointed to by the pwh parameter has not been prepared.

Note: When the buffer is finished, the WHDR_DONE bit is set in the dwFlags member of the WAVEHDR structure. The buffer must be prepared with the waveOutPrepareHeader function before it is passed to waveOutWrite. Unless the device is paused by calling the waveOutPause function, playback begins when the first data block is sent to the device.

IV. Summary of functions

auxSetVolume	Sets the volume of the specified output device
auxGetVolume	Retrieves current volume setting of the specified output device
auxGetNumDevs	Retrieves the number of output devices present in the system
playSound	Plays a sound specified by a given filename, resource, or system event
waveInOpen	Opens the waveform-audio input device for recording
waveInClose	Closes the waveform-audio device after recording
waveInAddBuffer	Sends an input buffer to the given waveform-audio input device
waveInStart	Starts input for the given waveform-audio input device
waveInStop	Stops input for the given waveform-audio input device
waveOutOpen	Opens the waveform-audio output device for playback
waveOutClose	Closes the waveform-audio output device for playback
waveOutSetPlaybackRate	Sets playback rate for the specified waveform-audio output device
waveOutWrite	Prepares data block to be used by the waveform-audio output device

These functions are but a few of the many available functions in the mmsystem header file. There are other useful functions that can be accessed, like switching formats, adding ACM drivers for recording and playback, manipulating various formats such as avi, midi, or wav, setting time for record and break, creating editable data streams, among others. Everything can be found in the MSDN website, or checking your program libraries. Mmsystem structures are also another vital part of the study that can be undertaken in a separate report. It is the goal of the author that these simple examples would challenge the reader to play around with the mmsystem functions and discover how such can add to the development of the various projects in class.

V. References

- Primary source: http://msdn.microsoft.com → MSDN Library
- http://www.pellesoft.se → Sample codes and brief discussions
- http://www.cisco.com → Useful boards and information on mmsystem
- http://www.piclist.com/techref/io/serial/midi/winapi.html → Programming and configuring the sound card
- http://www.insomniavisions.com/documents/tutorials/wave.php → Useful site for playing with digital audio data, recording and playback.
- http://www.borg.com/~jglatt/tech/lowaud.htm → Supplement data on audio input/output devices
- Kelly, Al; Pohl, Ira: A Book on C: Programming in C; 4th Edition 1997; Addison-Wesley Professional Inc.
- Schildt, Herbert: Turbo C/C++: The Complete Reference; 2nd Edition 1992; Borland-Osborne/McGraw-Hill.