Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

О. Е. Елисеева, Р. Е. Сердюков

Речевой интерфейс. Лабораторный практикум

Под редакцией профессора В. В. Голенкова

В 2-х частях

Часть 2

Рекомендовано УМО вузов Республики Беларусь
по образованию в области информатики и радиоэлектроники
в качестве учебно-методического пособия
для студентов учреждений, обеспечивающих получение высшего образования
по специальности «Искусственный интеллект»

Минск БГУИР 2010

УДК 004.522(076.5)
ББК 32.973я73
Е51

Р е ц е н з е н т ы :

доцент кафедры прикладной лингвистики Белорусского государственного университета, кандидат филологических наук А. И. Головня;

заведующий кафедрой информационных систем и технологий учреждения образования «Барановичский государственный университет» О. И. Наранович

Елисеева, О. Е.

Е51 Речевой интерфейс. Лабораторный практикум : учеб.-метод. пособие. В 2 ч. Ч. 2 / О. Е. Елисеева, Р. Е. Сердюков ; под ред. проф. В. В. Голенкова. – Минск : БГУИР, 2010. – 48 с.

ISBN 978-985-488-486-8 (ч. 2).

Пособие включает лабораторные работы, связанные с программированием приложений, предназначенных для воспроизведения звука на компьютере. Излагается материал об основных функциях Windows API, обеспечивающих работу с оцифрованным звуком.

Содержание пособия соответствует типовой программе учебного курса «Речевой интерфейс» по специальности «Искусственный интеллект» (№ ТД-40-62/тип, 2003 г.).

Предназначено для студентов вузов, обучающихся по специальности «Искусственный интеллект» всех форм обучения, родственным специальностям, а также для специалистов в области разработки интеллектуальных интерфейсов и прикладных компьютерных систем различного назначения.

УДК 004.522(076.5)
ББК 32.973я73

Часть 1 издана в БГУИР в 2008 г.

ISBN 978-985-488-486-8 (ч. 2)
ISBN 978-985-488-247-5

Содержание

Предисловие

Данное учебно-методическое пособие является результатом обобщения многолетнего опыта проведения лабораторного практикума в рамках учебного курса «Речевой интерфейс» для студентов специальности «Искусственный интеллект» учреждения образования «Белорусский государственный университет информатики и радиоэлектроники» при участии и поддержке специалистов Лаборатории распознавания и синтеза речи Объединенного института проблем информатики (ОИПИ) НАН Беларуси.

Для более эффективного изучения вопросов, связанных с разработкой речевых интерфейсов прикладных интеллектуальных систем, рекомендуется ознакомиться с теоретическим материалом, представленном в пособии [2] (***Лобанов Б.М..2006кн-Речев_И_ИС***). Там же подробно изложены цели и задачи дисциплины «Речевой интерфейс», определено ее место в учебном процессе.

Авторы выражают искреннюю признательность всем сотрудникам Лаборатории распознавания и синтеза речи Объединенного института проблем информатики НАН Беларуси за помощь в постановке учебного курса и за предоставленные материалы, а также студентам специальности «Искусственный интеллект», которые принимали участие в сборе и подготовке материалов для данного пособия. Особую благодарность авторы выражают сотрудникам ОИПИ НАН Беларуси Б. М. Лобанову, Н. П. Дегтяреву, Г. В. Лосику, А. С. Рылову, Т. В. Левковской и Л. И. Цирульник за неоценимый вклад в развитие учебного курса «Речевой инетерфейс», а также рецензенту – доценту кафедры прикладной лингвистики БГУ, кандидату филологических наук А. И. Головня – за объективную критику и ценные замечания, позволившие существенно улучшить качество данного пособия.

Введение

Современный мир немыслим без использования компьютерной техники практически в любой сфере деятельности человека. Компьютеры используются везде, и каждый современный человек просто обязан знать, что такое компьютер и как с ним работать. Одной из самых естественных для человека форм взаимодействия является **речь**. Поэтому большую актуальность в настоящее время приобретает разработка речевых интерфейсов интеллектуальных систем. Это подтверждает и тот факт, что как современные компьютеры, так и соответствующие операционные системы снабжены практически всеми необходимыми средствами работы со звуком. Кроме того, операционной системе Microsoft Windows XP и более новых версиях Windows имеются также встроенные возможности для работы с речью. Несмотря на это существующие в настоящее время разработки пока не удовлетворяют всем требованиям пользователей, так как синтезируемая компьютером речь не является достаточно естественной, а средства распознавания нуждаются в настройке (обучении) на каждого пользователя, т.е. не обладают достаточной степенью универсальности.

Разработка речевого интерфейса интеллектуальной системы сопряжена с решением огромного количества разнородных проблем [2] (***Лобанов Б.М..2006кн-Речев_И_ИС***), которые условно можно разделить на два основных класса:

- исследование закономерностей и характеристик речевого сигнала;
- программная реализация компонентов речевого интерфейса.

В связи с этим лабораторные работы, основной целью которых является практическое изучение студентами программной реализации компонентов речевого интерфейса, подразделяются на два аналогичных класса.

Все рассмотренные в пособии лабораторные работы выполняются в операционной среде Windows на персональном компьютере, имеющем стандартный набор аудиоустройств: аудио-карту, динамики и микрофон (или телефонную гарнитуру).

Лабораторная работа №1
«Реализация программы, проигрывающей wav-файлы с помощью функций высокого уровня»

Цель работы – изучение функций высокого уровня *sndPlaySound*, *PlaySound* и *MessageBeep* для программирования приложения, проигрывающего wav-файлы, на языке C++ в операционной системе Windows.

Основные **задачи** работы следующие:
1. Изучить основы API-мультимедиа в Windows.
2. Изучить высокоуровневые функции sndPlaySound, PlaySound и MessageBeep.
3. Разработать приложение, использующее указанные функции для проигрывания wav-файлов.
4. Сравнить использованные функции между собой, выявить основные функциональные и технологические различия.
5. Закрепить навыки программирования на языке C++ в среде Microsoft Visual Studio.

Прежде чем приступить к выполнению лабораторной работы, просмотрите все предлагаемое ниже ее описание и обратите внимание на примечания, справочные материалы и другую дополнительную информацию.

Для реализации указанного в задании приложения следует разработать сначала его внешний интерфейс, затем изучить предлагаемые функции *sndPlaySound*, *PlaySound* и *MessageBeep* и подключить их к разработанному интерфейсу. Так как основной целью работы является изучение предложенных функций, то приложение проще всего реализовать в виде обычного диалогового окна, содержащего кнопки, при нажатии на которые осуществляется запуск соответствующей функции. Кроме указанных кнопок, целесообразно также включить в диалоговое окно:
- средства задания пути и имени wav-файла для проигрывания;
- текстовый компонент, в который будет выводиться сообщение о том, какая функция выполняется и успешно ли осуществлено ее выполнение;
- кнопка вызова помощи пользователю (в данном случае помощь может иметь вид окна с сообщением о том, каково назначение данного приложения с краткой инструкцией для пользователя).

Например, внешний интерфейс разрабатываемого приложения может иметь упрощенный вид, представленный на рис. 1.1.

Рис. 1.1. Общий вид пользовательского интерфейса

Придумайте свой вариант внешнего интерфейса приложения. Попытайтесь подойти к этому заданию творчески. При выполнении следующих лабораторных работ вы будете изучать другие функции работы со звуком. Поэтому в дальнейшем разрешается использовать разработанный вами интерфейс в качестве каркаса для других приложений. Допускается

также объединять в рамках одного приложения результаты нескольких лабораторных работ, посвященных разработке программ обработки звука на компьютере.

1.1. Теоретический материал

Прежде чем перейти к изучению вопросов программирования звука, рассмотрим кратко понятие wav-файла. На компьютере звук, как и любая другая информация, представляется в цифровой форме. Самый распространенный в настоящее время формат для хранения оцифрованного звука в Windows – это формат wav. Существуют и другие форматы, но они являются более специализированными и зачастую требуют установки дополнительного программного обеспечения для их воспроизведения. Появление такого количества форматов для хранения оцифрованного звука объясняется тем, что звук – это аналоговый сигнал, который в памяти компьютера кодируется последовательностью битов. Как правило, для хранения всей информации, заключенной в аналоговом сигнале, требуются довольно большие объемы памяти. Поэтому разработчики цифровых форматов для хранения звука стремятся сжать эту информацию. Очевидно, что для воспроизведения звука, который хранится в таком «сжатом» файле, требуются средства «распаковки» информации. Одним из наиболее распространенных сжатых форматов для хранения звука является формат MP3. В данном учебно-методическом пособии проблемы оцифровки и сжатия звука рассматриваться не будут. Изучение указанных вопросов рекомендуется осуществить самостоятельно.

Одни из простейших средств воспроизведения звука, хранящегося в wav-файле, – функции *sndPlaySound*, *PlaySound* и *MessageBeep*. Указанные функции являются мультимедиа-функциями системы Windows и находятся в динамических библиотеках (DLL – Dynamic Link Libraries). Для того чтобы программа получила доступ к мультимедиа-функциям, необходимо подключить к ней заголовочный файл *mmsystem.h* с помощью директивы *#include* и присоединить к проекту файл мультимедиа-библиотеки *winmm.lib*.

Система Windows-мультимедиа обеспечивает доступ к различным функциям, которые можно разделить на две категории – высокоуровневые и низкоуровневые. Рассматриваемые функции *sndPlaySound*, *PlaySound* и *MessageBeep* можно считать ключевыми, потому что они выполняют только одну высокоуровневую функцию – воспроизведение оцифрованного звука (wav-файлов). Кроме того, чтобы пользоваться этими функциями, необязательно знать детали устройства звуковых карт или структуру воспроизводимых файлов. Ниже указанные функции описаны более подробно.

Функция MessageBeep

```
BOOL MessageBeep (UINT uType);
```

Функция *MessageBeep()* является наиболее специализированной из всех рассматриваемых функций. При вызове ей передается только один параметр *uType* – флаг, сигнализирующий о степени «важности» сообщения. В табл. 1.1 перечислены возможные значения параметра *uType*.

Каждое из значений флага ассоциировано с каким-нибудь из звуков в файле *WIN.INI* в секции *[sounds]*. Например, файл *WIN.INI* содержит описание звуковой схемы Windows:

```
[Sounds]
SystemAsterisk=chord.wav, Asterisk
SystemHand=chord.wav, Critical Stop
SystemDefault=ding.wav, Default Beep
SystemExclamation=chord.wav, Exclamation
SystemQuestion=chord.wav, Question
```

Таблица 1.1

Значения флага *uType*

Значение флага	Звук
0xFFFFFFFF	Стандартный звуковой сигнал через встроенный динамик компьютера
MB_ICONASTERISK	SystemAsterisk
MB_ICONEXCLAMATION	SystemExclamation
MB_ICONHAND	SystemHand
MB_ICONQUESTION	SystemQuestion
MB_OK	SystemDefault

Заданные в данном описании wav-файлы можно найти на системном диске в папке ..\WINDOWS\Media. Все строчки в описании секции *[sounds]* файла *WIN.INI* ассоциируют определенные системные события со звуковыми файлами. Вызывая функцию *MessageBeep()* с одним из указанных в таблице флагов, вы заставляете Windows найти в *WIN.INI* ссылку на соответствующий звук и воспроизвести его. Возвращаемое значение функции имеет *MessageBeep()* булевский тип. Это означает, что при успехе выполнения *MessageBeep()* возвращает ненулевое значение *TRUE*, иначе – нулевое *FALSE*. Для получения расширенной информации о произошедшей ошибке рекомендуется использовать функцию *GetLastError* (подробнее см. MSDN Library).

Пример вызова функции:

```
MessageBeep (MB_ICONEXCLAMATION);
```

Для того чтобы определить, успешно ли выполнилась функция, необходимо проверить истинность возвращаемого ею значения, например, следующим образом:

```
if (MessageBeep (MB_ICONEXCLAMATION))
      // функция выполнилась успешно
      else
      // функция выполнилась с ошибкой, вызвать функцию GetLastError и
      // вывести сообщение об ошибке
```

Во время выполнения программы функция *MessageBeep()* осуществляет постановку в очередь на проигрывание звуковой файл и сразу возвращает управление в вызывающую ее функцию, так что звук проигрывается асинхронно, а запущенная программа продолжает выполняться, не ожидая окончания проигрывания звукового файла.

Если функции *MessageBeep()* не удается запустить указанный с помощью передаваемого ей параметра звук, то она пытается запустить системный звук, указанный по умолчанию. Если и это не удается, то функция *MessageBeep()* производит стандартный звук через компьютерный динамик.

Следует также заметить, что пользователь может отменить использование звуковых сигналов мультимедиа-системы Windows с помощью стандартного приложения Sound Control Panel.

Функция sndPlaySound

```
BOOL sndPlaySound (LPCTSTR lpszSoundName, UINT fuSound );
```

Для воспроизведения звуков, соответствующих стандартным сообщениям, можно использовать и функцию *sndPlaySound()*. В отличие от *MessageBeep()* эта функция

позволяет воспроизвести любой звуковой файл, а не только тот, который был заранее определен в *WIN.INI*. У этой функции два аргумента:

1. *lpszSoundName* – указатель на строку, в которой задано название системного события или файла, необходимого для воспроизведения. Если значение первого параметра равно NULL, то воспроизведения звука будет остановлено.
2. *fuSound* – целое число, содержащее одно или более значений флага, перечисленных в табл. 1.2.

Таблица 1.2

Список флагов для функции *sndPlaySound*

Значение флага	Описание
SND_SYNC	Звук воспроизводится синхронно, и функция не возвращает управление до окончания его проигрывания
SND_ASYNC	Звук воспроизводится асинхронно, функция возвращает управление сразу же после начала воспроизведения. Для прекращения воспроизведения необходимо вызвать *sndPlaySound()* с *NULL* в качестве аргумента-имени файла
SND_NODEFAULT	Если указанный файл отсутствует, функция ничего не делает
SND_MEMORY	Параметр *lpszSoundName* указывает на звук, образ которого уже находится в оперативной памяти
SND_LOOP	Воспроизведение звука повторяется до тех пор, пока не будет вызвана функция *sndPlaySound* с *NULL* в качестве параметра *lpszSoundName*. Для работы этого флага необходимо также установить флаг *SND_ASYNC*
SND_NOSTOP	Если звук уже воспроизводится, функция не прерывает его воспроизведение, а немедленно возвращает *FALSE*

Примечание. Многие функции API Windows требуют указать в качестве параметров названия флагов. Такие параметры также свойственны сообщениям Windows. Флаг – это переменная, используемая для указания одного из двух состояний устройства или процесса. Значение флага всегда равно одной из степеней двойки. Первый флаг имеет значение 1 (2 в степени 0), второй – 2 (2 в степени 1), третий – 4 (2 в степени 2) и т. д. Если вы знакомы с двоичной арифметикой, то уже догадались, что каждый флаг представляет собой один из битов двоичного регистра. Таким образом, в одном наборе может находиться до 16 флагов. Чтобы установить значения нескольких из них, мы можем сложить соответствующие константы или, что лучше, объединить их оператором “|” (OR, логическое ИЛИ).

Функция *sndPlaySound()* возвращает числовое значение, представляющее собой логическую (BOOL) величину: TRUE – в случае успеха и FALSE – в случае неудачи.

Примеры вызова функции:

```
Int dummy = sndPlaySound (MB_ICONEXCLAMATION, SND_ASYNC);
sndPlaySound (“c:\\windows\\media\\chimes.wav”, SND_LOOP);
```

Если во время выполнения программы указанный в качестве первого аргумента функции *sndPlaySound()* звук не может быть найден, то она запускает системный звук SystemDefault, заданный в мультимедиа-системе Windows по умолчанию. Если это не удается, то функция не воспроизводит никакого звука и возвращает значение FALSE.

Следует также отметить, что указанный звук должен соответствовать размеру доступной физической памяти и быть проигрываемым установленным драйвером звукового устройства.

Функция PlaySound

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

Функция *PlaySound()* – это еще одна функция проигрывания звука в Windows. Данная функция позволяет воспроизводить wav-файл, звук системного события, определенного в *WIN.INI*, или звук из ресурса, сохраненного в приложении. У этой функции три аргумента:

1. *pszSound* – указатель на строку, в которой задано название файла, необходимого для воспроизведения. Если значение первого параметра равно *NULL*, то любой запущенный в текущий момент wav-файл будет остановлен. Для того чтобы остановить проигрывание звука в другом формате, необходимо в качестве третьего параметра задать флаг *SND_PURGE*.
2. *hmod* – дескриптор исполняемого файла, содержащего ресурс, который необходимо загрузить и воспроизвести через динамики. Его значение используется только в случае, если параметр *fdwSound* имеет значение *SND_RESOURCE*. В остальных случаях его следует устанавливать в *NULL*.
3. *fdwSound* – флаг, с помощью которого задается режим проигрывания звука. Список флагов описан в табл. 1.3.

Таблица 1.3

Список флагов для функции *PlaySound*

Значение флага	Описание
SND_APPLICATION	Звук воспроизводится с использованием установок приложения
SND_ALIAS	Параметр *pszSound* определяет псевдоним системного события в реестре Windows или в файле WIN.INI. Это значение нельзя использовать совместно со значениями SND_FILENAME и SND_RESOURCE
SND_ASYNC	Звук воспроизводится асинхронно, функция возвращает управление сразу же после начала воспроизведения. Для прекращения воспроизведения необходимо вызвать *PlaySound()* со значением *NULL* параметра *pszSound*
SND_FILENAME	Параметр *pszSound* является именем файла
SND_LOOP	Воспроизведение звука повторяется до тех пор, пока не будет вызвана функция *PlaySound* с *NULL* в качестве параметра *pszSound*. Для работы этого флага необходимо также установить флаг *SND_ASYNC* для асинхронного проигрывания звука
SND_MEMORY	Параметр *pszSound* указывает на звук, образ которого уже находится в оперативной памяти
SND_NODEFAULT	Если указанный файл или ресурс отсутствует, функция не должна поигрывать системный звук, заданный по умолчанию
SND_NOSTOP	Если какой-либо звук уже воспроизводится, функция не прерывает его воспроизведение, а немедленно возвращает *FALSE*. При этом запуска на проигрывание указанного в функции звука не происходит. Если этот флаг не указан, то функция *PlaySound*() попытается остановить проигрываемый в текущий момент времени звук, так чтобы устройство могло запустить на проигрывание указанный в функции новый звук
SND_NOWAIT	Если драйвер занят, функция сразу вернется без воспроизведения заданного звука
SND_RESOURCE	Параметр *pszSound* является идентификатором ресурса. При этом параметр *hmod* должен указывать на источник ресурса
SND_SYNC	Звук воспроизводится синхронно, и функция не возвращает управление до окончания его проигрывания

Функция *PlaySound()* возвращает TRUE в случае успеха и FALSE – в случае неудачи.

1.2. Порядок выполнения лабораторной работы

Задание 1.2.1. Прочитайте теоретический материал об изучаемых функциях (см. подразд. 1.1).

Задание 1.2.2. Запустите среду программирования Visual C++ и создайте новый проект, основанный на диалоговом интерфейсе (для справки см. подразд. 1.3).

Задание 1.2.3. Создайте свой интерфейс приложения с помощью средств визуального проектирования среды Visual C++.

Задание 1.2.4. Подключите требуемые функции к кнопкам на панели диалога вашего приложения.

Задание 1.2.5. Подготовьте несколько тестовых wav-файлов, осуществите тестирование и отладку разработанного вами приложения. При этом проверьте успешность выполнения функций для всех возможных значений флагов и их комбинаций, а в качестве исходных данных используйте по крайней мере три wav-файла, размещенных в разных папках. Отследите, какие значения возвращает каждая из трех тестируемых функций как в случае успешности их выполнения, так и в случае неудачи. Отметьте, какие возможны ошибки при выполнении указанных функций.

Задание 1.2.6. Оформите отчет о работе, в который необходимо включить следующее:

- краткое описание использованных функций и их сравнительный анализ;
- краткое описание результатов, полученных при выполнении функций с разными параметрами и комбинациями флагов;
- перечень ошибок, которые могут возникнуть в случае безуспешного выполнения использованных функций;
- (необязательно) обобщенную схему интерфейса разработанного приложения;
- (необязательно) ответы на вопросы для самопроверки (см. подразд. 1.4).

Задание 1.2.7. Покажите работу программы преподавателю. Ответьте на вопросы для самопроверки. Подробные ответы на них можно (но необязательно) включить в текст отчета.

1.3. Описание хода разработки приложения

Задание 1.3.1. Запустите среду программирования Visual C++ и создайте новый проект, основанный на диалоговом интерфейсе.

Задание 1.3.1.1. Запустите среду программирования *Visual C++*.

Задание 1.3.1.2. Создайте новый проект, основанный на диалоговом интерфейсе. Для этого войдите в меню *FILE -> NEW...*, выберите там *Project*. Затем в появившемся окне (рис. 1.2) выберите тип проекта *MFC Application*. В этом же окне в поле редактирования *Name* задайте имя программы, например *lab1*, и нажмите *OK*.

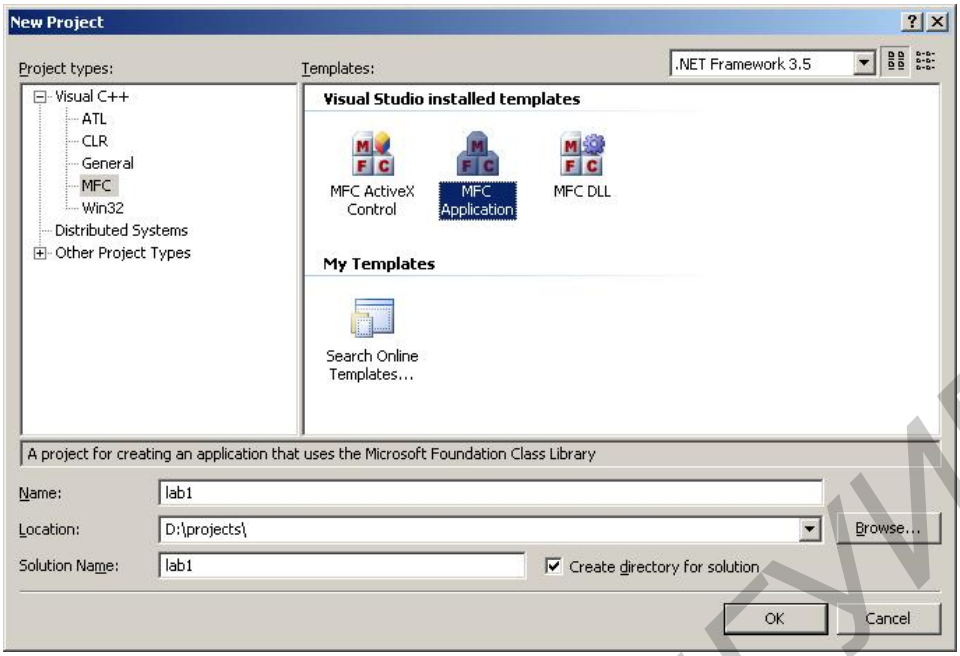

Рис. 1.2. Окно мастера создания нового проекта

Выбрав тип проекта *MFC Application*, мы задали режим создания приложения с помощью мастера создания приложений. После нажатия на кнопку *OK* осуществляется запуск мастера настройки приложения, в котором необходимо выбрать тип приложения *Dialog based* (рис. 1.3). Таким образом задается, что разрабатываемое приложение будет основано на диалоговых окнах. Нажмите кнопку *Next* для перехода к следующим шагам настройки приложения.

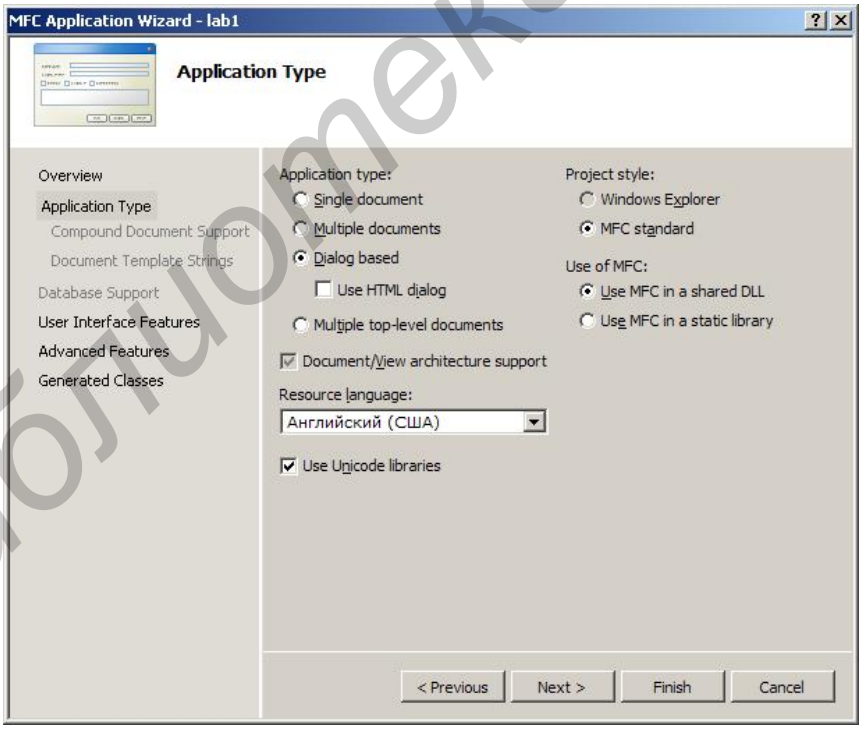

Рис. 1.3. Окно настройки параметров проекта

Далее необходимо указать, какие компоненты мы хотим использовать в своем проекте, а также задать строку, которая будет отображаться в заголовке создаваемого диалогового окна. После этого можно завершить настройку и нажать кнопку *Finish*.

Все каркасные файлы для создаваемой программы написал Visual C++ с помощью мастера настроек *MFC AppWizard*. MFC – это вещь очень полезная, с помощью нее можно быстро создавать программы, так как все дежурные файлы он пишет сам, что очень облегчает работу программиста.

Задание 1.3.2. Создайте свой интерфейс приложения с помощью средств визуального проектирования среды Visual C++.

В среде визуального проектирования интерфейса спроектируем диалоговую панель. Для этого в окне с перечнем компонентов проекта выберите вкладку *«Resourse View»* и раскройте в ней пункт *lab1 Resource*. Далее выберите ветвь *Dialog*, в ней будет ресурс *IDD_LAB1_DIALOG*, двойным кликом откройте этот ресурс. Он представляет собой макет создаваемого диалогового окна (рис. 1.4).

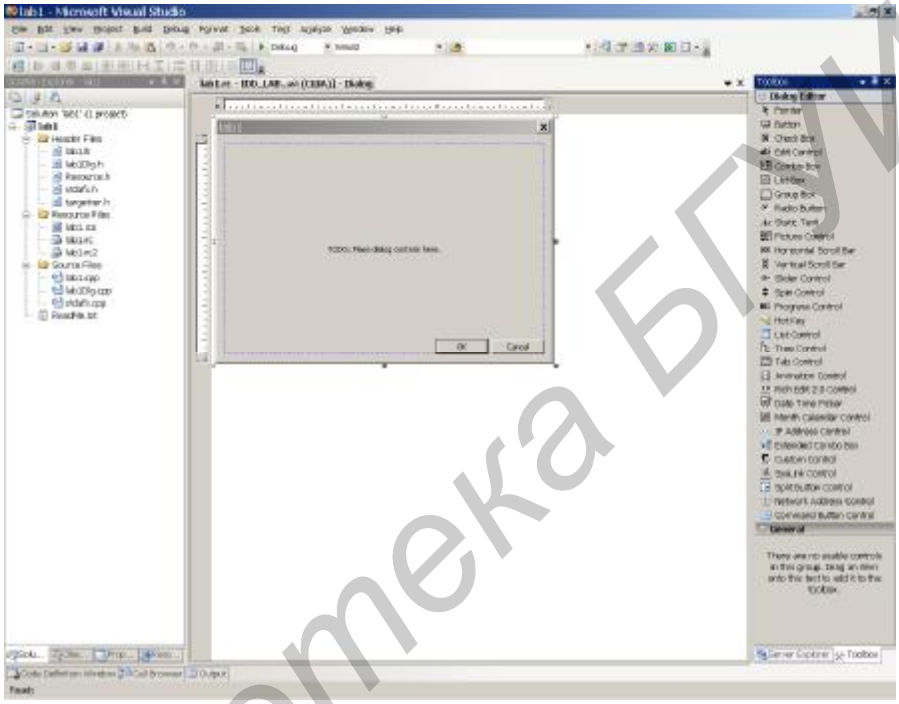

Рис. 1.4. Окно редактора ресурсов приложения

Справа от диалоговой панели находятся элементы управления – панель Toolbox. Панель можно редактировать в полноэкранном режиме, для этого зайдите в *VIEW->FULLSCREEN*.

По умолчанию на поле диалогового окна будут находиться три элемента: статический текст и две кнопки *OK* и *Cancel*. Статический текст и кнопку *Cancel* можно удалить, а кнопку *OK* переименовать в *Exit* (см. элемент 6 на рис. 1.5). Далее необходимо добавить несколько элементов в соответствии со структурой интерфейса (см. рис 1.5):

- **Элемент 1**. Поле *Sample edit box* служит для отображения пути к файлу. Выставить флаг *disable* и присвоить идентификатор *IDC_FILEPATH*.
- **Элемент 2**. Кнопка для вызова диалогового окна с выбором пути к файлу, идентификатор – *IDC_BROWSE*.
- **Элемент 3**. Кнопка для проверки функции *MessageBeep*, идентификатор – *IDC_MESSAGE_SOUND*.
- **Элемент 4**. Кнопка для проверки функции *sndPlaySound*, идентификатор – *IDC_PLAY_SOUND*.
- **Элемент 5**. Поле *Sample edit box* служит для отображения результата выполнения работы функции, идентификатор – *IDC_RESULT*.

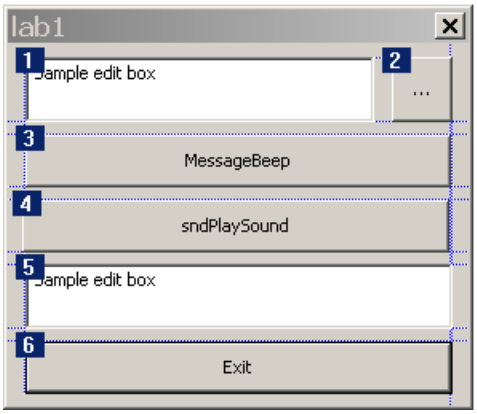

Рис. 1.5. Вид разрабатываемого диалогового окна

Далее создадим строковые переменные, которые будут соответствовать значениям полей элементов *IDC_FILEPATH* и *IDC_RESULT*. Для этого вызовем настройщик классов *ClassWizard*, пункт меню *VIEW->ClassWizard*. В появившемся окне выберем вторую вкладку, где будет доступен список элементов, которым можно поставить в соответствие переменные (рис. 1.6).

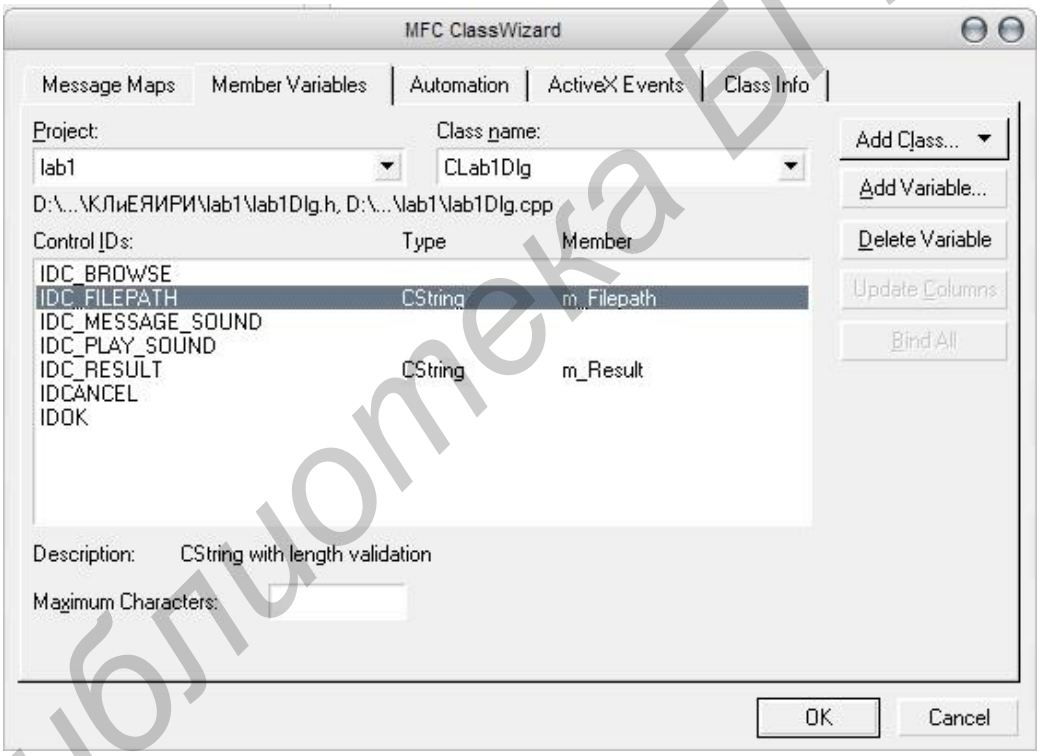

Рис. 1.6. Окно диалога мастера классов

Для задания переменных необходимо выполнить следующее: выделить строку, в которой указан идентификатор *IDC_FILEPATH*, и нажать кнопку *AddVariable…*, в результате чего появится еще один диалог, где необходимо указать имя переменной и ее тип (рис. 1.7).

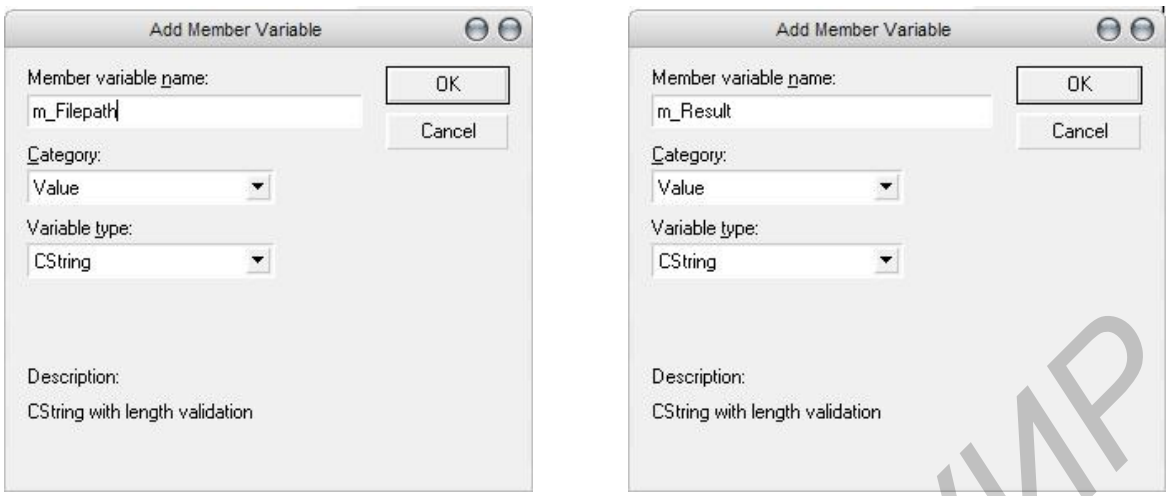

Рис. 1.7. Окна диалогов добавления переменных в классы

Аналогичным образом задайте переменную для идентификатора *IDC_RESULT*.

Задание 1.3.3. Подключите требуемые функции к кнопкам на панели диалога приложения.

Создадим функции, которые будут обрабатываться при нажатии на кнопки создаваемого диалогового окна. Для этого необходимо вернуться к редактору ресурсов (рис. 1.8).

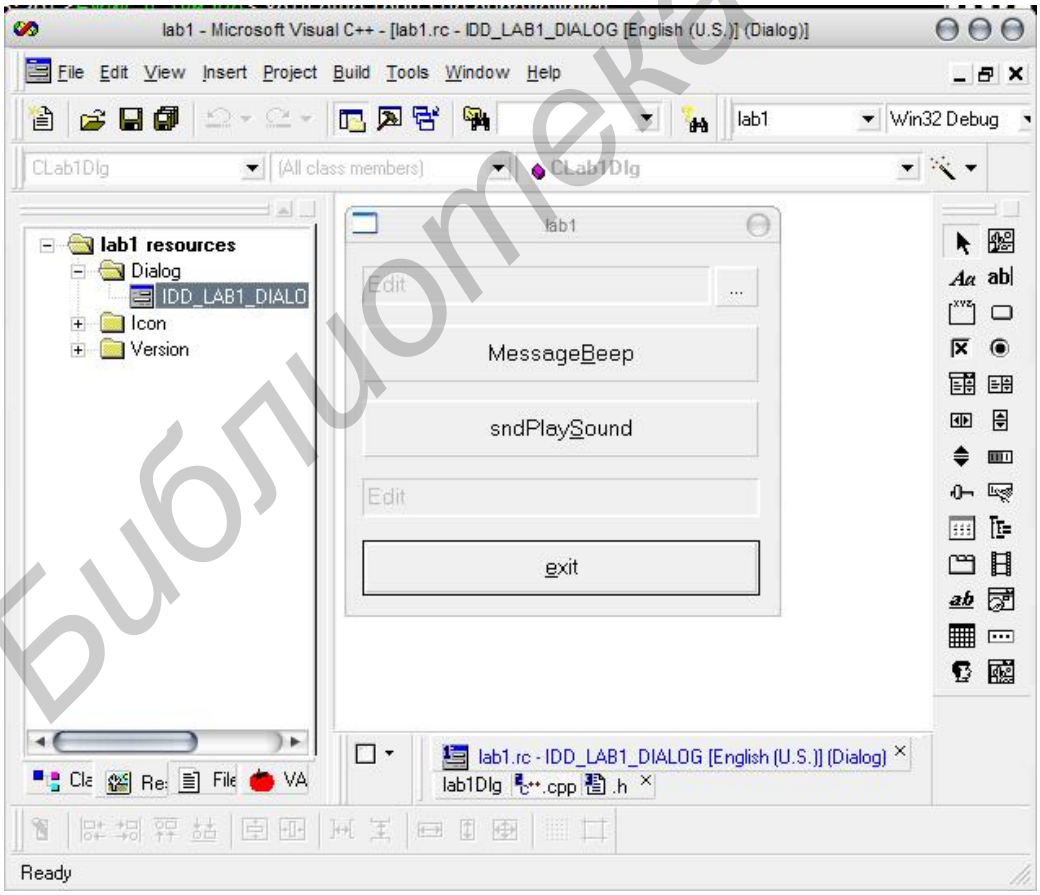

Рис. 1.8. Окно редактора ресурсов приложения

Двойным щелчком мыши по элементу *IDC_MESSAGE_SOUND* вызовите диалог (рис. 1.9).

Рис. 1.9. Окно диалога добавления переменной в класс

В диалоге будет указано имя функции, которая будет вызываться при нажатии этой кнопки – *OnMessageBeep*. После нажатия на кнопку *OK* откроется файл *lab1Dlg.cpp* в том месте, куда была добавлена функция *OnMessageBeep().*

В этом месте в тело функции необходимо добавить вызов функции *MessageBeep()* и обработку результата ее выполнения:

```
void CLab1Dlg::OnMessageSound ()
{
	if (MessageBeep(MB_ICONEXCLAMATION))
		m_Result = "MessageBeep() - ok";
	else
	{
		m_Result = "MessageBeep() - error";
		MessageBox("[error] MessageBeep() !");
	}
	UpdateData(FALSE);
}
```

Аналогично двойным щелчком по элементам *IDC_PLAY_SOUND* и *IDC_BROWSE* получаем соответственно функции *OnMessageSound* и *OnBrowse*, тела которых необходимо заполнить следующим кодом:

```
void CLab1Dlg::OnPlaySound () {
	if (m_Filepath == "")
		m_Result = "sndPlaySound() - select file";
	else {
		if (sndPlaySound(m_Filepath,SND_ASYNC))
			m_Result = "sndPlaySound() - ok";
		else {
			m_Result = "sndPlaySound() - error";
			MessageBox ("[error] sndPlaySound() !");
		}
	}
	UpdateData(FALSE);
}

void CLab1Dlg::OnBrowse () {
	// choose wav file
	CFileDialog Dlg(TRUE, NULL, NULL,
				OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
				TEXT("sound files (*.wav)|*.wav||"));
	if (IDOK != Dlg.DoModal())
		return;
	m_Filepath = Dlg.GetPathName();
	UpdateData(FALSE);
}
```

После этого необходимо подключить к проекту заголовочный файл *mmsystem.h* и библиотеку *winmm.lib*, в которых определены функции для работы с мультимедиа. Для этого откройте файл *lab1Dlg.h* и добавьте в его начало строку:

```
#include "mmsystem.h"
```

Для того чтобы указать VC++ на необходимость использования мультимедиа-библиотеки при создании кода программы, нажмите комбинацию клавиш Alt+F7 либо выберите пункт меню *PROJECT-> SETTINGS*. В появившемся диалоговом окне выберите вкладку *Link* и впишите в поле «*Object/library modules*» *winmm.lib* (рис. 1.10).

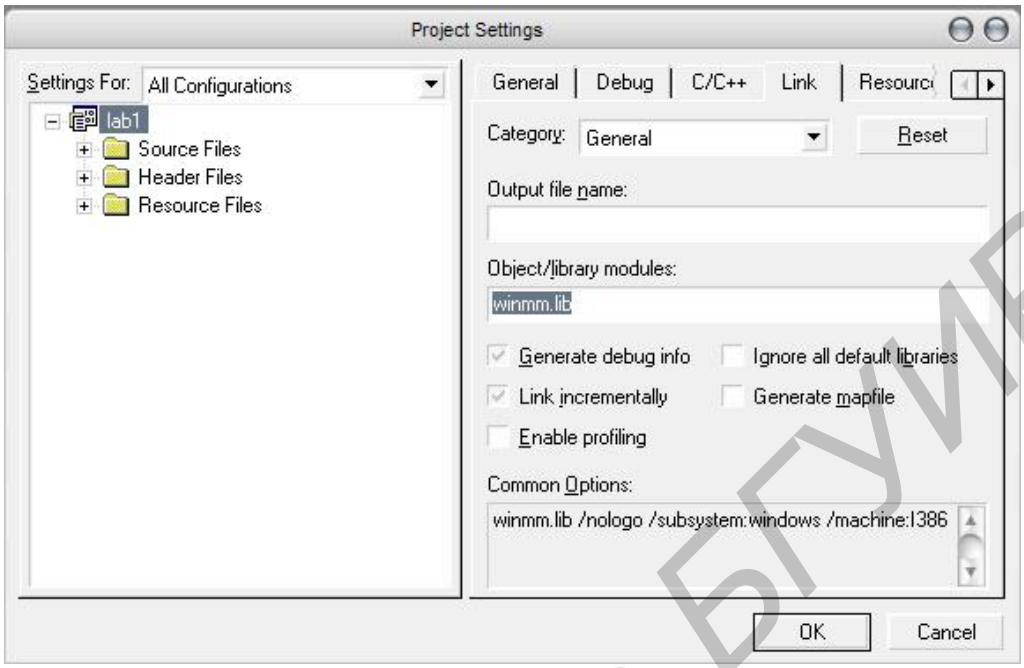

Рис. 1.10. Окно настройки свойств проекта

Теперь можно компилировать проект. Для этого либо нажмите на кнопку с восклицательным знаком на панели управления среды *Visual C++*, либо Ctrl+F5, либо выберите пункт меню *BUILD -> EXECUTE lab1.exe*.

1.4. Вопросы для самопроверки

1. Какие функции высокого уровня можно использовать для воспроизведения wav-файлов в Windows?
2. Чем различаются указанные функции?
3. Почему указанные функции называются функциями высокого уровня?
4. Что такое мультимедиа-система Windows?
5. Что такое wav-файл?
6. Какие основные режимы воспроизведения звука в Windows?
7. Чем отличается асинхронный режим воспроизведения звука от синхронного?
8. Какая из трех функций *sndPlaySound*, *PlaySound* и *MessageBeep* является наиболее универсальной и почему?
9. Какая из трех функций *sndPlaySound*, *PlaySound* и *MessageBeep* является наиболее специализированной и почему?
10. Чем отличаются друг от друга функции *sndPlaySound* и *PlaySound* ?
11. Какие библиотеки содержат описание функций высокого уровня для проигрывания wav-файлов в Windows?

Лабораторная работа №2
«Реализация программы, проигрывающей wav-файлы с помощью функций интерфейса MCI»

Цель работы – изучение функций интерфейса MCI *mciSendString*, *mciSendCommand*, *mciGetErrorString* для программирования приложения, проигрывающего wav-файлы, на языке C++ в операционной системе Windows.

Основные **задачи** работы следующие:
1. Закрепить основы API-мультимедиа в Windows.
2. Изучить высокоуровневые функции интерфейса MCI *mciSendString*, *mciSendCommand*, *mciGetErrorString*.
3. Разработать приложение, использующее указанные функции для проигрывания wav-файлов.
4. Сравнить использованные функции между собой, выявить основные функциональные и технологические различия.
5. Закрепить навыки программирования на языке C++ в среде Microsoft Visual Studio.

Прежде чем приступить к выполнению лабораторной работы, просмотрите все предлагаемое ниже ее описание и обратите внимание на примечания, справочные материалы и другую дополнительную информацию.

Для реализации указанного в задании приложения (аналогично тому, как было сделано при выполнении лабораторной работы №1) следует разработать сначала его внешний интерфейс, затем изучить предлагаемые функции и подключить их к разработанному интерфейсу. Так как основной целью работы является изучение предложенных функций, то приложение проще всего реализовать в виде обычного диалогового окна, содержащего кнопки, при нажатии на которые осуществляется запуск соответствующей функции. Кроме указанных кнопок необходимо также включить в диалоговое окно:
- средства задания пути и имени wav-файла для проигрывания;
- текстовый компонент для ввода текста выполняемой команды (для тестирования функций с различными параметрами);
- текстовый компонент, в который будет выводиться сообщение о том, какая функция выполняется и успешно ли осуществлено ее выполнение;
- кнопка вызова помощи пользователю (в данном случае помощь может иметь вид окна с сообщением о том, каково назначение данного приложения с краткой инструкцией для пользователя).

Придумайте свой вариант внешнего интерфейса приложения. Попытайтесь подойти к этому вопросу творчески. При этом разрешается использовать уже разработанный вами интерфейс в лабораторной работе №1.

2.1. Теоретический материал

Интерфейс управления устройствами *MCI* (*Media Control Interface*) позволяет программам для Windows взаимодействовать со множеством устройств мультимедиа – компакт-диск-плеерами, цифровыми аудио- и midi-синтезаторами на звуковых платах, проигрывателями видеодисков, видеомагнитофонами и т. п.

При помощи функций MCI мы можем управлять любым из перечисленных выше устройств, посылая ему специальные команды: начать воспроизведение, остановиться, переместиться в начало или конец файла и т. д. Во многом такое управление похоже на манипуляции с кнопками на панели управления магнитофоном. Конкретный набор команд для каждого устройства определяется его возможностями. Например, устройство цифрового ввода-вывода на звуковой карте может записывать звук в файл, в то время как компакт-плеер может только воспроизводить. Любую команду мы передаем MCI-устройству в качестве

аргумента одной из функций интерфейса MCI. Кроме командных функций, интерфейс MCI включает еще и две функции поддержки.

Все функции MCI начинаются с префикса *mci* и делятся на три группы:
- интерфейс команд-сообщений:
- *mciSendCommand();*
- *mciGetDeviceID();*
- интерфейс команд-строк:
- *mciSendString();*
- оба типа интерфейса:
- *mciGetErrorString();*
- *mciSetYieldProc().*

Два командных интерфейса высокого уровня Command-Message и Command-String выполняют примерно одни и те же функции. Они позволяют начать или остановить воспроизведение wav- или midi-файла, перейти к седьмой дорожке на компакт-диске и т. п. – примерно то же самое, что вы можете сделать при помощи пульта дистанционного управления своей стереосистемой. Различие между этими двумя интерфейсами заключается только в способе подачи команд и сводится к различию между числами и словами – конечный результат всегда один и тот же.

Указанные функции являются мультимедиа-функциями системы Windows и находятся в динамических библиотеках (DLL – Dynamic Link Libraries). Для того чтобы программа получила доступ к мультимедиа-функциям, необходимо подключить к ней заголовочный файл *mmsystem.h* с помощью директивы *#include* и присоединить к проекту файл мультимедиа-библиотеки *winmm.lib*.

Ниже указанные функции описаны более подробно.

Функция **mciSendString**

```
MCIERROR mciSendString (
       LPCTSTR lpszCommand,
       LPTSTR lpszReturnString,
       UINT cchReturn,
       HANDLE hwndCallback
);
```

Функция *mciSendString()* является функцией высокого уровня и предназначена для воспроизведения wav-файлов. Можно сказать, что эта функция представляет самый высокий уровень в иерархии мультимедиа-интерфейса, так как она умеет транслировать команды обычного английского языка, превращая их в команды управления мультимедиа-устройствами. У этой функции четыре аргумента:
1. *lpszCommand* – указатель на завершающуюся нулем строку с командой в следующей форме:
   [команда] [устройство] [параметры] ;
2. *lpszReturnString* – указывает на буфер для получения информации о результате. Если такая информация не нужна, то этот параметр можно установить равным *NULL*;
3. *cchReturn* – указывает размер (в символах) определенного предыдущим параметром буфера. Если буфер не нужен, этот параметр должен быть равен нулю;
4. *hwndCallback* – указывает на окно «отклика», если в командной строке указан параметр «*notify*» (подтвердить). Если «*notify*» не указан, то этот параметр может быть *NULL*.

Эта функция возвращают величину типа *MCIERROR*, которая на самом деле является не чем иным, как типом *DWORD*. Если функция возвращает ноль, то все в порядке – команда выполнена успешно. В случае какой-либо ошибки функция возвращает ненулевой код ошибки. Собственно, сам код находится в младшем байте слова. Для генерации текстового описания ошибки на основании этого кода есть функция *mciGetErrorString*.
Примеры вызова функции *mciSendString*:

```
mciSendString (“play c:\\windows\\tada.wav”, NULL, 0, NULL);
// простейший способ вызова функции для проигрывания wav-файла
#define BUF_LEN, 256
char reply [BUF_LEN];
mciSendString (m_commandStringText , reply, BUF_LEN, NULL);
// здесь задан буфер памяти reply, в который будет помещена
// дополнительная информация от MCI, и его длина BUF_LEN
// в строке m_commandStringText передается описание действия,
// которое следует совершить
```

Для того чтобы определить, успешно ли выполнилась функция, необходимо проверить «истинность» возвращаемого ею значения, например, следующим образом:

```
if (mciSendString(szCmdLine, NULL, 0, NULL)) {
      // функция выполнилась с ошибкой, вызвать функцию mciGetErrorString
      //  и вывести сообщение об ошибке
} else {
      // функция выполнилась успешно
}
```

Рассмотрим отдельно последний параметр функции *mciSendString()*. Он играет роль, известную как «возврат» (callback) – она свойственна многим вызовам API. «Возврат» – это средство, при помощи которого функция API может уведомить программу о завершении какого-либо действия. «Возвраты» работают одним из двух способов: они либо вызывают функцию, специально написанную программистом для их обработки, либо, как *mciSendString()*, посылают сообщение в окно, которое должно содержать соответствующий обработчик. В последнем случае «возврат» аналогичен любому другому сообщению Windows с тем отличием, что вызывается не определенными действиями пользователя или системными событиями, а фактом завершения указанной API операции. Окно-получатель сообщения указывается в качестве четвертого параметра функции *mciSendString()*, а само сообщение называется MM_MCINOTIFY. VC++ полностью поддерживает обработку этого сообщения. Важно помнить, что *mciSendString()* будет посылать сообщение MM_MCINOTIFY только в том случае, если в командной строке указан параметр «notify», как в следующем примере:

```
“play c:\\windows\\tada.wav notify” .
```

Функция mciSendCommand

Кроме *mciSendString()* для воспроизведения wav-файлов можно использовать и функцию *mciSendCommand()*:

```
MCIERROR mciSendCommand (
       MCIDEVICEID  IDDevice ,
       UINT  uMsg ,
       DWORD  fdwCommand ,
       DWORD dwParam
);
```

У этой функции тоже четыре аргумента:
1. *IDDevice* – идентификатор MCI-устройства, которому адресуется команда. Этот параметр не используется для команды *MCI_OPEN*;
2. *uMsg* – командное сообщение;
3. *fdwCommand* – флаги для командного сообщения;
4. *dwParam* – указатель на структуру с параметрами для командного сообщения.

Эта функция тоже возвращают величину типа *MCIERROR*. Если она возвращает ноль, то все в порядке – команда выполнена успешно. В случае какой-либо ошибки функция возвращает ненулевой код ошибки. При этом сам код находится в младшем байте слова. Для генерации текстового описания ошибки на основании этого кода есть функция *mciGetErrorString*. Если

ошибка специфична для определенного устройства, то функция *mciSendCommand()* помещает в старший байт возвращаемого слова идентификатор устройства, в противном случае старший байт остается равным нулю.

Обратите внимание на четвертый параметр – указатель на структуру с данными, необходимыми для выполнения команды. Мы ничего не сказали о типе этой структуры. Просто различные команды пользуются разными наборами исходных данных. Вызывая *mciSendCommand()*, мы создаем структуру с данными, необходимыми конкретной команде, и передаем функции указатель на нее. При помощи такой техники специалисты из Microsoft сумели свести множество операций MCI к одному интерфейсу. В любом другом случае вам понадобились бы отдельные функции для каждого действия, каждая со своим собственным набором параметров. Мы же вместо всего этого имеем возможность пользоваться одной единственной функцией, сообщая ей, где найти необходимую информацию.

Структуры данных, необходимые для воспроизведения wav-файлов при помощи интерфейса команд-сообщений:

```
// Структура должна использоваться для открытия wav-устройства
// перед началом любого воспроизведения звука
typedef struct {
      // окно, используемое для «возврата», если необходимость
      // последнего указана флагом MCI_NOTIFY
      DWORD cwCallback;
      // возвращаемый пользователю идентификатор устройства
      MCIDEVICEID wDeviceID;
      // указатель на буфер, определяющий тип устройства
      LPCTSTR lpstrDeviceType;
      // указатель на имя элемента (обычно путь)
      LPCTSTR lpstrElementName;
      // указатель на буфер, содержащий алиас устройства
      LPCTSTR lpstrAlias;
      // определяет длину буфера в секундах
      DWORD dwBufferSeconds;
} MCI_WAVE_OPEN_PARMS;
// Структуру можно использовать, если нет желания указывать
// длину буфера
typedef struct {
// окно, используемое для «возврата», если необходимость
// последнего указана флагом MCI_NOTIFY
DWORD dwCallback;
// начало проигрываемого участка (позиция в файле)
DWORD dwFrom;
// окончание проигрываемого участка (позиция в файле)
DWORD dwTo;
} MCI_PLAY_PARMS;
```

Но не торопитесь набирать эти определения – они уже сделаны за вас в файле *mmsystem.h*, который вы должны включить в свой проект.

Сообщение «Play» в интерфейсе команд-сообщений не выполняет операции открытия и закрытия устройства. Поэтому для воспроизведения файла необходимо послать три команды-сообщения:

- открыть файл;
- воспроизвести файл;
- закрыть файл.

Две из этих команд используют блоки параметров.

Заметим также, что структура *MCI_WAVE_OPEN_PARMS* является несколько расширенным вариантом более общей структуры *MCI_OPEN_PARMS*, в которой отсутствует поле *dwBufferSeconds*. Если вы не хотите указывать длину буфера, то можете воспользоваться структурой *MCI_PLAY_PARMS* – в ней есть все параметры, необходимые для воспроизведения wav-файла.

Функция mciGetErrorString

```
BOOL mciGetErrorString (
DWORD   fdwError,
LPTSTR  lpszErrorText,
UINT    cchErrorText
);
```

У этой функции три аргумента:
1) *fdwError* – код ошибки, выданный одной из функций *mciSendCommand()* или *mciSendString()*;
2) *lpszErrorText* – указатель на буфер, предназначенный для помещения в него текста, описывающего ошибку. Сам текст завершается символом с кодом ноль;
3) *cchErrorText* – указывает размер буфера *lpszErrorText* в символах.

Другими словами, в качестве первого параметра мы передаем этой функции код интересующей нас ошибки, а второй и третий параметры описывают буфер, в который должна быть помещена текстовая информация об ошибке. Функция *mciGetErrorString()* возвращает опять-таки код ошибки: TRUE – в случае удачи и FALSE – при неудаче. Здесь в качестве неудачи может быть попытка поинтересоваться описанием несуществующей ошибки.

2.2. Порядок выполнения лабораторной работы

Задание 2.2.1. Прочитайте теоретические сведения об изучаемых функциях (см. подразд. 2.1).

Задание 2.2.2. Запустите среду программирования Visual C++ и создайте новый проект, основанный на диалоговом интерфейсе (для справки см. подразд. 2.3.), либо откройте проект, созданный при выполнении лабораторной работы №1.

Задание 2.2.3. Создайте (или доработайте) интерфейс приложения с помощью средств визуального проектирования среды Visual C++.

Задание 2.2.4. Подключите требуемые функции к кнопкам на панели диалога приложения.

Задание 2.2.5. Подготовьте несколько тестовых wav-файлов, осуществите тестирование и отладку разработанного приложения. При этом проверьте успешность выполнения функций для всех возможных типов вводимых команд, а в качестве исходных данных используйте по крайней мере три wav-файла, размещенных в разных папках. Отследите, какие значения возвращает каждая из тестируемых функций как в случае успешности их выполнения, так и в случае неудачи. Отметьте, какие возможны ошибки при выполнении указанных функций.

Задание 2.2.6. Оформите отчет о работе, в который необходимо включить следующее:
- краткое описание использованных функций и их сравнительный анализ (в том числе сравнение с функциями, которые изучались при выполнении лабораторной работы №1);
- краткое описание результатов, полученных при выполнении функций с разными входными параметрами и с использованием разных структур;
- перечень ошибок, которые могут происходить в случае безуспешного выполнения использованных функций;
- (необязательно) обобщенную схему интерфейса разработанного приложения.

Задание 2.2.7. Покажите работу программы преподавателю.

2.3. Описание хода разработки приложения

Данное описание является одним из вариантов реализации приложения. Вы можете разработать свое собственное приложение, которое будет отличаться от предлагаемого ниже как пользовательским интерфейсом, так и набором инструментов. Например, вместо (или наряду с) текстового поля для ввода команды вы может продумать все возможные кнопки для управления процессом воспроизведения wav-файлов.

Задание 2.3.1. Запустите среду программирования Visual C++ и создайте новый проект, основанный на диалоговом интерфейсе.

Задание 2.3.1.1. Запустите среду программирования *Visual C++*.

Задание 2.3.1.2. Создайте новый проект, так же как и в подразд. 1.3.

Задание 2.3.2. Создайте интерфейс приложения, как показано на рис. 2.1, с помощью средств визуального проектирования среды Visual C++, руководствуйтесь описанием из подразд. 1.3.

Рис. 2.1. Вид разрабатываемого диалогового окна

Задание 2.3.3. Подключите требуемые функции к кнопкам на панели диалога приложения. Создайте функции, которые будут обрабатываться при нажатии на кнопки создаваемого диалогового окна.

Для элемента *IDC_SEND_COMMAND* (*mciSendCommand*) добавьте соответствующую функцию (см. подразд. 1.3) и следующий код в тело функции:

```
void CLab2Dlg::OnSendCommand() {
     if (m_Filepath == "") {
          m_Result = "mciSendCommand() - select file";
          MessageBox ("mciSendCommand() - select file");
          UpdateData(FALSE);
          return;
     }
     // open
     MCI_OPEN_PARMS mciOpenParms;
     mciOpenParms.lpstrDeviceType = "waveaudio";
     char szFileName [1024];
     strcpy(szFileName, m_Filepath);
     mciOpenParms.lpstrElementName = szFileName;
     if (mciSendCommand(0, MCI_OPEN,
          MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
          (DWORD)(LPVOID) &mciOpenParms))
```

```
	{
		m_Result = "mciSendCommand() - error open";
		MessageBox ("mciSendCommand() - error open");
		UpdateData(FALSE);
		return;
	}
	// play
	UINT wDeviceID = mciOpenParms.wDeviceID;
	MCI_PLAY_PARMS mciPlayParms;
	mciPlayParms.dwCallback = (DWORD) NULL;
	if (mciSendCommand(wDeviceID, MCI_PLAY, MCI_WAIT,
		(DWORD)(LPVOID) &mciPlayParms))
	{
		mciSendCommand(wDeviceID, MCI_CLOSE, 0, NULL);
		m_Result = "mciSendCommand() - error play";
		MessageBox ("mciSendCommand() - error play");
		UpdateData(FALSE);
		return;
	}
	// close
	mciSendCommand(wDeviceID, MCI_CLOSE, MCI_WAIT, NULL);
	m_Result = "mciSendCommand() - ok";
	UpdateData(FALSE);
}
```

Двойным кликом по элементам *IDC_SEND_STRING* (*sndPlaySound*) и *IDC_BROWSE* (…) соответственно получаем функции *OnSendString* и *OnBrowse*, которые необходимо заполнить:

```
void CLab2Dlg::OnSendString() {
	if (m_Filepath == "")
	{
		m_Result = "mciSendString() - select file";
		MessageBox ("mciSendString() - select file");
		UpdateData(FALSE);
		return;
	}

	// init
	char  szCmdLine  [1024];

	// open
	wsprintf(szCmdLine, "open \"%s\" type waveaudio alias mysound",
		m_Filepath);
	if (mciSendString(szCmdLine, NULL, 0, NULL)) {
		m_Result = "mciSendString() - error open command";
		MessageBox ("mciSendString() - error open command");
		UpdateData(FALSE);
		return;
	}

	// play
	wsprintf(szCmdLine, "play mysound wait");
	if (mciSendString(szCmdLine, NULL, 0, NULL)) {
		m_Result = "mciSendString() - error play command";
		MessageBox ("mciSendString() - error play command");
		UpdateData(FALSE);
		return;
	}
```

```
	// close
	wsprintf(szCmdLine, "close mysound wait");
	if (mciSendString(szCmdLine, NULL, 0, NULL)) {
		m_Result = "mciSendString() - error close command";
		MessageBox ("mciSendString() - error close command");
		UpdateData(FALSE);
		return;
	}

	m_Result = "mciSendString() - ok";
	UpdateData(FALSE);
}

void Clab2Dlg::OnBrowse ()
{
	// choose wav file
	CFileDialog Dlg(TRUE, NULL, NULL,
			OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
			TEXT("sound files (*.wav)|*.wav||"));
	if (IDOK != Dlg.DoModal())
		return;
	m_Filepath = Dlg.GetPathName();
	UpdateData(FALSE);
}
```

После этого необходимо подключить к проекту заголовочный файл *mmsystem.h* и библиотеку *winmm.lib*, в которых определены функции и структуры для работы с мультимедиа. Для этого откройте файл *lab2Dlg.h* и добавьте в его начало строки:

```
#include "mmsystem.h"
```

Для того чтобы указать VC++ на необходимость использования мультимедиа-библиотеки при создании кода программы, нажмите комбинацию клавиш Alt+F7 либо выберите пункт меню *PROJECT-> SETTINGS*. В появившемся диалоговом окне выберите вкладку *Link* и впишите во второе поле *winmm.lib* (см. рис. 1.10).

Теперь можно компилировать проект. Для этого либо нажмите на кнопку с восклицательным знаком на панели управления среды *Visual C++*, либо Ctrl+F5, либо выберите пункт меню *BUILD -> EXECUTE lab2.exe*.

2.4. Вопросы для самопроверки

1. Для чего предназначен интерфейс MCI?
2. На какие группы подразделяются функции интерфейса MCI?
3. Чем различаются указанные функции?
4. Чем указанные функции отличаются от функций высокого уровня, изученных при выполнении лабораторной работы №1?

Лабораторная работа №3
«Реализация программы, проигрывающей wav-файлы с помощью низкоуровневых аудиофункций»

Цель работы – изучение низкоуровневых аудиофункций для программирования приложения, проигрывающего wav-файлы, на языке C++ в операционной системе Windows.

Основные **задачи** работы следующие:
1. Изучить формат файлов RIFF.
2. Изучить функции ввода-вывода мультимедиа файлов (mmioOpen, mmioRead, mmioAscend и др.).
3. Изучить функции управления аудиоустройством (waveOutOpen, waveInOpen и др.).
4. Разработать приложение, которое осуществляет разбор формата wav-файла и его проигрывание.
5. Сравнить использованные функции между собой, выявить основные функциональные и технологические различия.
6. Сравнить использованные низкоуровневые функции с высокоуровневыми функциями, изученными при выполнении лабораторных работ №1 и №2.
7. Закрепить навыки программирования звука на языке C++ в среде Microsoft Visual Studio.

Лабораторная работа выполняется в операционной системе Windows на персональном компьютере IBM PC типа Pentium, имеющем стандартный набор аудиоустройств: аудиокарту и динамики.

Прежде чем приступить к выполнению лабораторной работы, просмотрите все предлагаемое ниже ее описание и обратите внимание на примечания, справочные материалы и другую дополнительную информацию.

Примечание. Данная лабораторная работа рассчитана на два занятия. Поэтому на 1-м занятии позволяется показать только предварительные результаты. Например, как минимум, интерфейс и попытки открыть файл. При этом обязательно по листингу программы объяснить, что и где написано.

Для реализации указанного в задании приложения (аналогично тому, как было сделано при выполнении лабораторных работ №1 и №2) следует разработать сначала его внешний интерфейс, затем изучить предлагаемые функции и подключить их к разработанному интерфейсу. Так как основной целью работы является изучение предложенных функций, то приложение проще всего реализовать в виде обычного диалогового окна, содержащего кнопки, при нажатии на которые осуществляется запуск соответствующей функции. Кроме указанных кнопок необходимо также включить в диалоговое окно:
- средства задания пути и имени wav-файла для проигрывания;
- текстовый компонент, в который будет выводиться структура воспроизводимого wav-файла (этот компонент можно оформить также в виде дерева);
- текстовый компонент, в который будет выводиться сообщение о том, какая функция выполняется и успешно ли осуществлено ее выполнение;
- кнопка вызова помощи пользователю (в данном случае помощь может иметь вид окна с сообщением о том, каково назначение данного приложения с краткой инструкцией для пользователя).

Придумайте свой вариант внешнего интерфейса приложения. Попытайтесь подойти к этому вопросу творчески. При этом разрешается использовать уже разработанный вами интерфейс в лабораторных работах №1 и №2.

3.1. Теоретический материал

Чтобы воспроизвести wav-файл, система мультимедиа должна открыть его, прочитать и интерпретировать заголовок, загрузить аудиоданные в память, открыть устройство «waveaudio», воспроизвести звук и затем закрыть устройство и исполнявшийся файл. Для того чтобы справиться со всем этим, высокоуровневые функции MCI «пользуются услугами» нескольких низкоуровневых функций. Для воспроизведения wav-файла требуется более десятка таких функций. Все они могут быть разделены на две группы:

- функции, которые читают файлы формата RIFF (Resource In-terchange File Format, формат файлов для обмена ресурсами);
- функции, управляющие устройством воспроизведения звука.

Перед тем как открыть устройство «waveaudio», вы должны знать, какую информацию собираетесь туда посылать. Ведь данные в wav-файлах записываются в нескольких форматах, различающихся частотой дискретизации, количеством каналов и разрешением (количеством битов на один отсчет). Вся интересующая нас информация находится в одном блоке («куске» chunk – в терминологии RIFF, в дальнейшем вместо «chunk» мы будем пользоваться словом «блок») в самом начале файла.

Файлы RIFF

Файлы RIFF имеют иерархическую структуру (см. рис. 3.1). Это значит, что блоки могут включать в себя другие блоки, которые в свою очередь содержат в себе третьи. На самом высшем уровне иерархии блоков находится собственно блок RIFF, т. е. весь файл. Все блоки помечены метками, которые называются идентификаторами блоков (chunk IDs). Если вы посмотрите начало wav-файла при помощи какого-нибудь редактора, то увидите, что первые четыре символа файла содержат в себе слово RIFF.

Для чтения и записи RIFF-файлов используется стандартная структура данных, называемая MMCKINFO (multimedia chunk information). Определение этой структуры на VC++ выглядит следующим образом:

```
// структура информации о блоке RIFF
typedef  struct _MMCKINFO {
      FOURCC ckid;
      DWORD cksize;
      FOURCC fccType;
      DWORD dwDataOffset;
      DWORDdwFlags;
} MMCKINFO;
```

Первое и третье поля – это массивы из четырех символов, основанные на типе *DWORD*. Каждый из четырех байтов *DWORD* содержит один из этих символов. *FOURCC*, таким образом, является просто мнемоническим обозначением типа данных.

Предостережем вас от весьма распространенной ошибки — *MMCKINFO* не определяет поле с собственно данными! Эта структура используется только для обмена информацией с функциями ввода-вывода мультимедиа-файлов. Блоки RIFF содержат строго определенные фрагменты информации, однако могут иметь очень разную длину, и поэтому мы не можем загрузить их в буфер с заранее определенной длиной.

Все блоки начинаются с двух полей – идентификатора и размера блока. Они соответствуют первым двум полям структуры *MMCKINFO*. Когда мы пользуемся для чтения блока низкоуровневой мультимедиа-функцией *mmioDescend()*, мы передаем ей адрес структуры типа *MMCKINFO*, и функция заполняет эту структуру необходимой информацией. Важно, что мы не читаем данные из файла прямо в структуру *MMCKINFO*. RIFF-файлы обычно не содержат в себе блоков одинаковой длины, поэтому наиболее эффективный способ чтения таких файлов заключается в том, чтобы пройтись по ним при помощи функций *mmio* (multimedia i/o, мультимедиа ввод-вывод).

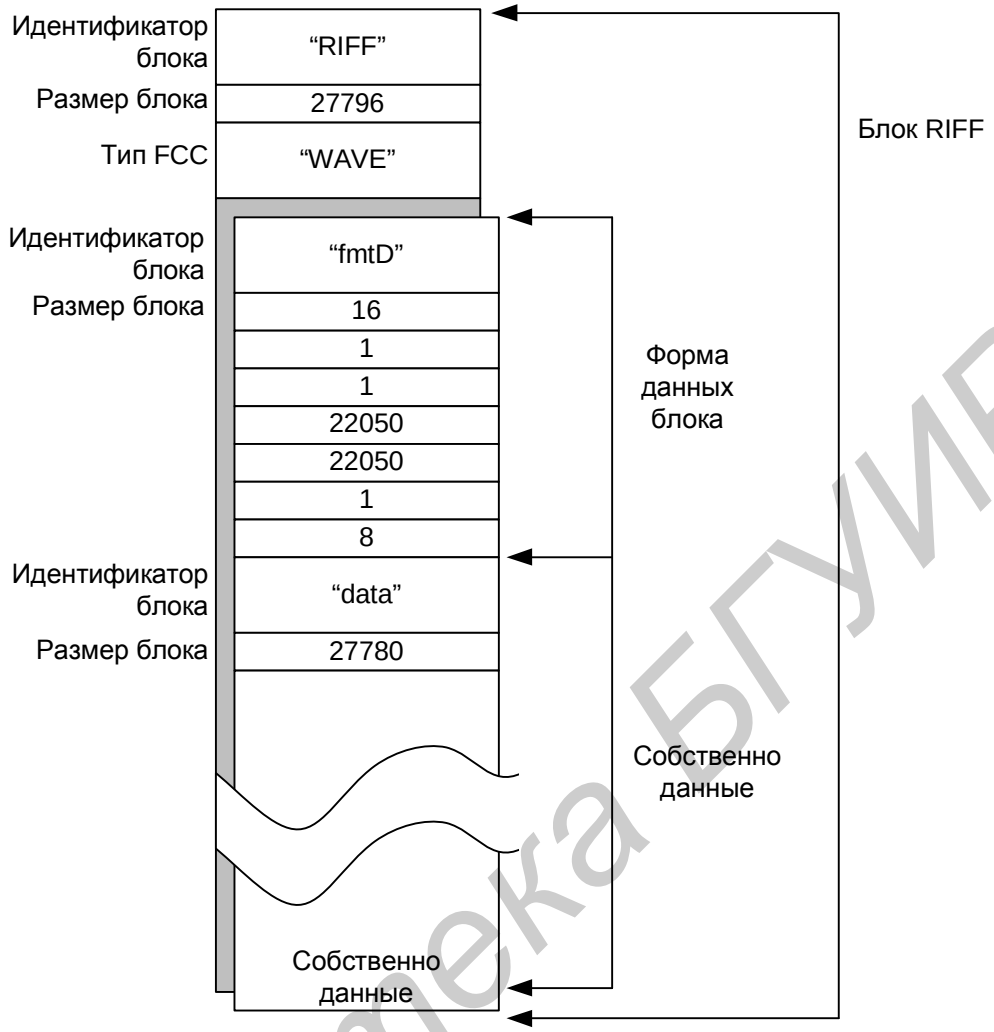

Рис. 3.1. Структура RIFF-файла

Для перемещения по RIFF-файлу используются функции *mmioDescend()* и *mmioAscend()*, которые изменяют текущее положение внутри блока. В зависимости от типа блока вы указываете либо его идентификатор (поле *ckid*), либо тип (поле *fccType*), устанавливаете флаг поиска и вызываете *mmioDescend()*.Если вам просто требуется следующий блок, то вы можете не указывать *ckid* или *fccType*, и функция *mmioDescend()* заполнит эти поля данными из найденного ею блока.

Структура wav-файла

Чтобы лучше разобраться в деталях доступа к wav-файлам, рассмотрим небольшой пример. В табл. 3.1 приведена структура файла TADA.WAV.

В самом простом случае wav-файл содержит в себе три блока. Самый большой из них – это блок RIFF, т. е. в сущности весь файл. Поле *cksize*, находящееся сразу же за полем *ckid*, содержит в себе длину файла за вычетом восьми байтов, которые используются для хранения этих двух полей. В поле *ckid* находятся символы «RIFF». Второй и третий блоки (их еще называют субблоками) содержатся внутри блока RIFF. Первый из них, блок «fmt», содержит информацию, необходимую для заполнения структуры *PCMWAVEFORMAT*. Второй субблок, «data», следует за блоком «fmt» и содержит собственно цифровые данные о звуке, являясь самой большой частью файла. Конец субблока «data» совпадает с концом блока RIFF. Содержимое поля *cksize* блока RIFF равно суммарной длине субблоков «fmt» и «data».

Таблица 3.1

Образец структуры wav-файла

Смещение		Размер в байтах	Содержимое	Комментарий
шестн.	дес.			
0000	0	4	"RIFF"	ckid, по символу в каждом байте
0004	4	4	27796	cksize, размер файла – 8 байтов
0008	8	4	"WAVE"	fccType
000C	12	4	"fmt"	следующий ckid. Обратите внимание на дополнение пробелом до четырех символов
0010	16	4	16	cksize, размер блока формата – 16 байтов
0014	20	2	1	wFormatTag, 1 – формат PCM Wave
0016	22	2	1	nChannels, количество каналов
0018	24	4	22050	nSamplesPerSec, частота дискретизации
001C	28	4	22050	nAvgBytesPerSec
0020	32	2	1	nBlockAlign, байт на отсчет
0022	34	2	8	wBitsPerSample, разрядность
0024	36	4	"data"	ckid, блок с данными
0028	40	4	27760	cksize, размер данных
002C	44	В зависимости от данных	Цифровые аудиоданные	

RIFF-фаплы могут содержать еще один тип блоков – блоки «LIST». Они используются для хранения дополнительной информации, например, сведений об авторских правах или описаний файлов. Мы не будем касаться этого типа блоков, тем более, что нам они не нужны и в используемых нами файлах не встречаются. Мы упоминаем о них только потому, что их наличие еще раз доказывает полезность и необходимость функций мультимедиа ввода-вывода, которые были специально разработаны фирмой Microsoft для чтения и записи riff-файлов.

Функции мультимедиа ввода-вывода

Прочитать RIFF-файл и воспользоваться имеющейся в нем информацией можно при помощи обыкновенных функций ввода-вывода. Но для этого понадобится вычислять смещения и читать блоки разной длины. Гораздо проще воспользоваться специально разработанными для этой цели функциями. Все функции мультимедиа ввода-вывода начинаются с префикса mmio. Вот их полный список:

- mmioOpen();
- mmioClose();
- mmioSeek();
- mmioRead();
- mmioWrite();
- mmioDescend();
- mmioAscend();
- mmioCreateChunk();
- mmioFOURCC();
- mmioStringToFOURCC();
- mmioAdvance();
- mmioGetInfo();
- mmioSetInfo();
- mmioInstallIOProc();

- mmioSendMessage();
- mmioFlush();
- mmioRename();
- mmioSetBuffer().

Для чтения и воспроизведения wav-файла нам потребуются только пять из этих восемнадцати функций: *mmioOpen()*, *mmioClose()*, *mmioRead()*, *mmioDescend()* и *mmioAscend()*.

Все эти функции могут возвращать сообщения об ошибках, поэтому необходимо проверять результат после выполнения каждой из них. Для чтения wav-файла потребуется несколько вызовов функций, поэтому процедура обработки ошибок может оказаться достаточно длинной.

3.2. Порядок выполнения лабораторной работы

Задание 3.2.1. Прочитайте теоретические сведения о структуре wav-файла и об изучаемых функциях.

Задание 3.2.2. Запустите среду программирования Visual C++ и создайте новый проект, основанный на диалоговом интерфейсе (для справки см. подразд. 3.3), либо откройте проект, созданный при выполнении лабораторных работ №1 и №2.

Задание 3.2.3. Создайте (или доработайте) интерфейс приложения с помощью средств визуального проектирования среды Visual C++.

Задание 3.2.4. Подключите требуемые функции к кнопкам на панели диалога вашего приложения.

Задание 3.2.5. Продумайте и реализуйте средства вывода информации о структуре воспроизводимого wav-файла.

Задание 3.2.6. Подготовьте несколько тестовых wav-файлов, осуществите тестирование и отладку разработанного вами приложения. При этом сравните данные о структурах воспроизводимых файлов и отметьте для себя последовательность выполняемых операций при воспроизведении wav-файла с использованием низкоуровневых аудиофункций.

Задание 3.2.7. Оформите отчет о работе, в который необходимо включить следующее:
- краткое описание структуры wav-файла;
- краткое описание технологии воспроизведения wav-файлов с помощью низкоуровневых аудиофункций;
- краткое описание использованных функций и их сравнительный анализ с функциями, которые изучались при выполнении лабораторных работ №1 и №2;
- (необязательно) обобщенную схему интерфейса разработанного приложения.

Задание 3.2.8. Покажите работу программы преподавателю и объясните выполненные вами действия.

3.3. Описание хода разработки приложения

Данное описание является одним из вариантов реализации приложения. Вы можете разработать свое собственное приложение, которое будет отличаться от предлагаемого ниже как внешним интерфейсом, так и набором инструментов. Например, вместо (или наряду с) текстового поля для ввода команды вы может продумать все возможные кнопки для управления процессом воспроизведения wav-файлов.

Задание 3.3.1. Запустите среду программирования Visual C++ и создайте новый проект, основанный на диалоговом интерфейсе.

Задание 3.3.1.1. Запустите среду программирования *Visual C++*.

Задание 3.3.1.2. Создайте новый проект, который продемонстрирует возможности использования низкоуровневых аудиофункций. Назовите его, например **KLLab3**. Для этого выполните следующие шаги (см. также аналогичное иллюстрированное описание в лабораторных работах №1 и №2):

- в меню *File* выберите *New…* (можно также воспользоваться комбинацией клавиш *Ctrl+N*);
- в появившемся диалоговом окне выберите *MFC AppWizard (exe)*, в качестве имени проекта (Project name) введите *KLLab3*. Нажмите кнопку ОК;
- далее при помощи *AppWizard* настройте будущий проект. Для этого выберите *Dialog based* (приложение будет основано на диалоговых окнах) и нажмите кнопку *Next*. В появившемся окне оставьте выбранным только *3D controls*, во всех остальных местах галочки снимите и нажмите кнопку *Finish*.

Задание 3.3.2. Создайте (или доработайте) свой интерфейс приложения с помощью средств визуального проектирования среды Visual C++.

Для этого сначала выберите вкладку *«ResourseView»* и раскройте пункт *KLLab3 resources*. Далее выберите ветвь *Dialog*, в которой будет ресурс *IDD_KLLAB3_DIALOG*. Двойным щелчком мыши откройте этот ресурс. Он представляет собой макет будущего диалогового окна. Здесь необходимо создать кнопку, при нажатии на которую в приложении должен открываться диалог выбора файла. Для этого на панели инструментов *Controls* выберите элемент *Button* и поместите его на диалоговое окно. На созданной кнопке нажмите на правую кнопку мыши и в появившемся контекстном меню выберите *Properties* и задайте *ID* – IDC_WAVEOUTWRITE, *Caption* – Wave Out Write. Создайте остальные необходимые элементы управления и разместите их, например, так, как показано на рис. 3.2:

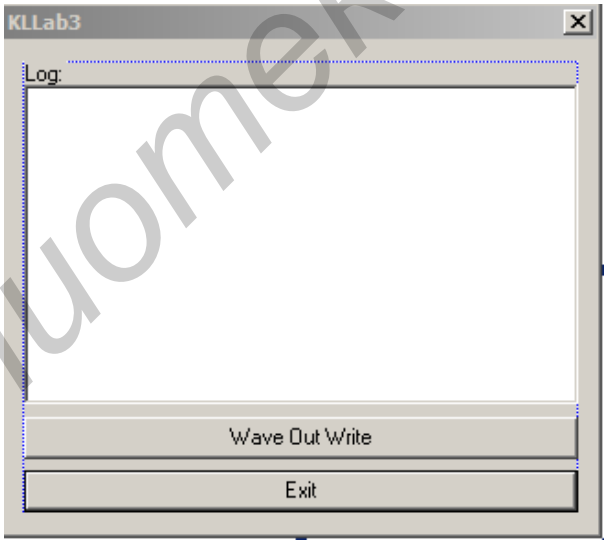

Рис. 3.2. Вид разрабатываемого диалога

Требуется создать следующие элементы управления:

- Static Text: Caption – “Log:”, ID - IDC_STATIC_LOG. Помещаем его в верхний левый угол. Ниже помещаем:
- List Box: ID - IDC_LIST_LOG.
  Еще ниже:
- наша созданная ранее кнопка;
- кнопка ОК, переименованная в Exit;
- кнопку Cancel – удаляем.

Теперь необходимо создать две функции, которые будут выполнять открытие и воспроизведение wav-файла. Пусть это будут функции *OpenWaveFile()* и *WavePlay()*. Начнем с размещения описания этих функций в файле *KLLab3Dlg.h*.

```
private:
      // открывает файл
      HANDLE OpenWaveFile(CString FileName);
      // посылает данные в wav-устройство
      BOOL WavePlay();
```

В этот же файл добавьте объявление двух необходимых переменных.

```
private:
      PCMWAVEFORMAT PCMWaveFmtRecord;
      WAVEHDR WaveHeader;
```

Две структуры, *PCMWAVEFORMAT* и *WAVEHDR*, необходимы для работы некоторых из используемых функций. Они определены в файле *mmsystem.h*, который необходимо включить в начало файла *KLLab3Dlg.h* :

```
#include "mmsystem.h"
```

Так как ListBox служит для ведения протокола работы программы, то необходимо создать для него соответствующую переменную. Для этого вызовите настройщик классов *ClassWizard…*, выберите меню *View* и затем *ClassWizard*. В появившемся диалоге откройте вкладку *Member Variables* – здесь выводится список элементов, для которых можно задать переменные (рис. 3.3).

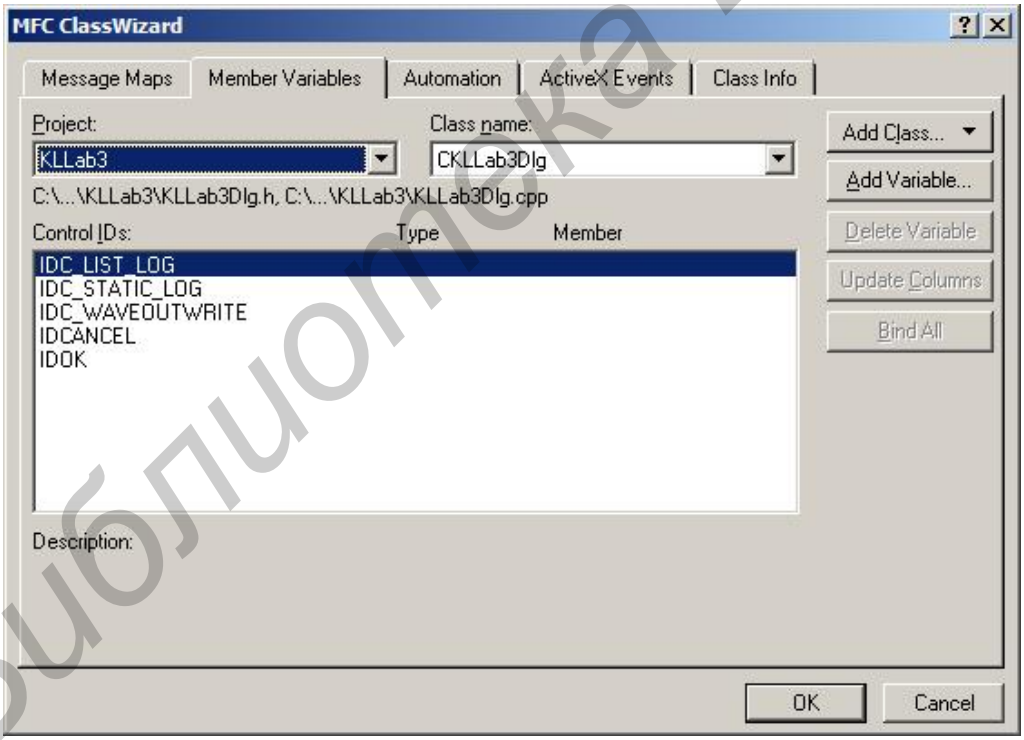

Рис. 3.3. Диалоговое окно мастера классов

Выберите *IDC_LIST_LOG* и нажмите кнопку *Add Variable… .* В появившемся диалоговом окне (рис. 3.4) в поле *«Member variable name:»* укажите имя переменной, например log. В списке *«Category:»* выберите *Control*, а в *«Variable type»* – *CListBox*. Закройте диалоговые окна, нажав кнопку OK.

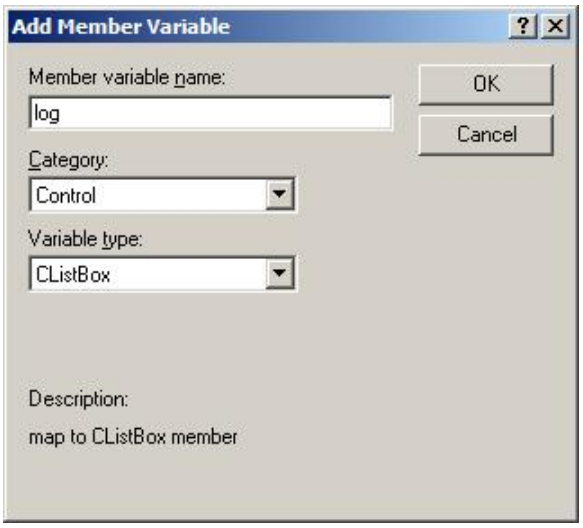

Рис. 3.4. Диалоговое окно добавления переменной в класс

Задание 3.3.3. Подключите требуемые функции к кнопкам на панели диалога вашего приложения.

Рассмотрим первую функцию *OpenWaveFile()*, так как чтобы воспроизвести файл, его необходимо сначала открыть. Чтобы открыть wav-файл, необходимо знать его имя и путь к нему. Для этого служит параметр типа *CString*. Рассматриваемая функция будет возвращать величину типа *HANDLE*, указывающую на блок памяти, в которой помещаются прочитанные данные. Это необходимо для того, чтобы по окончании работы можно было освободить занятую область памяти.

Для открытия файла используется функция *mmioOpen()*. Она определена в файле *mmsystem.h* следующим образом:

```
HMMIO mmioOpen(LPSTR lpszFileName, LPMMIOINFO lpmmioinfo, DWORD fdwOpen);
```

Данная функция имеет три параметра:
1. Первый параметр *lpszFileName* содержит имя открываемого файла.
2. Второй параметр *lpmmioinfo* является указателем на структуру *MMIOINFO*.
3. Третий параметр *fdwOpen* используется для передачи флагов, указывающих на определенные опции *mmioOpen()*.

Функция *mmioOpen()* возвращает указатель на открытый файл или NULL, если файл не может быть открыт.

Строки протокола добавляются функцией *AddString()* класса *CListBox*. В качестве параметра ей передается строка, содержащая описание действия, которое в данный момент выполняет программа.

```
// Открыть wav-файл
HMMIO hmmio = mmioOpen(FileName.GetBuffer(80), NULL, MMIO_READ);
sprintf(info, "Открываем файл: %s",FileName.GetBuffer(80));
log.AddString(_T(info));
// Успешно?
if (!hmmio) {
      MessageBox("Не могу открыть файл");
      return NULL;
}
```

После открытия файла необходимо найти его главный блок, т. е. блок RIFF. Для поиска блока используется функция *mmioDescend()*. Она объявляется следующим образом:

```
MMRESULT mmioDescend( HMMIO hmmio, LPMMCKINFO lpmmcki,
                      LPMMCKINFO lpmmckiParent, UINT fuSearch );
```

В случае успешного выполнения операции эта функция возвращает 0, иначе – код ошибки.

У данной функции четыре параметра:

- первый параметр – это указатель на открытый wav-файл;
- второй параметр – адрес структуры *MMCKINFO*, в которую эта функция поместит определенную информацию;
- третий параметр – это адрес «старшей по иерархии» структуры *MMCKINFO*;
- четвертый параметр – это один из флагов, определяющих режим поиска.

Для поиска определенного типа riff-блока необходимо явно указать этот тип. Он хранится в третьем поле заголовка блока RIFF и является третьим полем структуры *MMCKINFO*. Для поиска riff-блока с данными WAV необходимо установить поле *fccType* структуры *MMCKINFO* в значение WAV, воспользовавшись при этом макросом *mmioFOURCC()*, который преобразует символьный массив в нужный тип. Объявление структур выглядит следующим образом:

```
MMCKINFO MMCkInfoParent;
MMCKINFO MMCkInfoChild;
```

Код поиска основного блока:

```
// Найти основной блок RIFF
MMCkInfoParent.fccType = mmioFOURCC('W','A','V','E');
int errorCode = mmioDescend(hmmio, &MMCkInfoParent, NULL, MMIO_FINDRIFF);
if (errorCode) {
      MessageBox("Ошибка поиска блока WAVE в файле");
      mmioClose(hmmio, 0);
      return (NULL);
}
```

Далее необходимо найти блок формата. Так как блок RIFF с типом WAV является «родителем» всех блоков в wav-файле, то для поиска любого другого блока необходимо указать идентификатор *ckid* и передать еще одну структуру *MMCKINFO* в качестве третьего параметра. Код для поиска блока формата следующий:

```
// Найти блок формата
MMCkInfoChild.ckid = mmioFOURCC('f','m','t',' ');
errorCode = mmioDescend(hmmio, &MMCkInfoChild,
                        &MMCkInfoParent, MMIO_FINDCHUNK);
if (errorCode) {
      MessageBox("Ошибка поиска блока формата в файле");
      mmioClose(hmmio, 0);
      return (NULL);
}
```

Теперь можно получить данные для структуры *PCMWAVEFORMAT*:

```
// Прочитать запись о формате данных PCM
DWORD bytesRead = mmioRead(hmmio, (HPSTR)&PCMWaveFmtRecord,
                           MMCkInfoChild.cksize);
if (bytesRead <= 0) {
      MessageBox("Ошибка чтения данных о формате PCM");
      mmioClose(hmmio, 0);
      return (NULL);
}
```

Функция *mmioRead()* читает информацию из указанного файла и заполняет структуру *PCMWAVEFORMAT* сведениями о формате открытого файла.

На следующем шаге необходимо проверить совместимость прочитанных данных со звуковым устройством. Если данные не совместимы, то все остальные действия будут бесполезны. Для этой проверки следует использовать функцию *waveOutOpen()* с флагом *WAVE_FORMAT_QUERY*. Если файл совместим с устройством, то эта функция вернет ноль, иначе – код ошибки.

```
// Проверить совместимость данных с устройством
HWAVEOUT hWaveOut;
errorCode = waveOutOpen(&hWaveOut, WAVE_MAPPER,
                        (LPWAVEFORMATEX) &PCMWaveFmtRecord,
                        NULL, NULL, WAVE_FORMAT_QUERY);
if (errorCode) {
      MessageBox("Несовместимый формат файла");
      mmioClose(hmmio, 0);
      return (NULL);
}
```

После выяснения формата данных требуется найти блок с этими данными. Для этого необходимо подняться в иерархии RIFF-блоков на предыдущий уровень и затем прочитать блок данных:

```
// Подняться на один уровень в иерархии RIFF
errorCode = mmioAscend(hmmio, &MMCkInfoChild, 0);
if (errorCode) {
      MessageBox("Не могу вернуться");
      mmioClose(hmmio, 0);
      return (NULL);
}

// прочитать блок данных
MMCkInfoChild.ckid = mmioFOURCC('d','a','t','a');
errorCode = mmioDescend (hmmio, &MMCkInfoChild, &MMCkInfoParent,
                         MMIO_FINDCHUNK);
if (errorCode) {
      MessageBox("Не могу прочитать блок данных");
      mmioClose(hmmio, 0);
      return(NULL);
}
```

Далее следует прочитать сам звук. Для этого необходим достаточно большой буфер для хранения данных в памяти. Размер блока данных можно узнать в поле *cksize* структуры *MMCkInfoChild*. Для выделения блока памяти в предлагаемом примере программы используется функция *GlobalAlloc()*, которой передаются два параметра: первый – это флаг, определяющий режим выделения памяти, а второй указывает необходимое количество байтов памяти.

Если необходимый блок памяти был выделен, то можно в него записать данные из обрабатываемого звукового файла. Ниже приведен соответствующий исходный код:

```
// Выделить память для wav-данных
long lDataSize = MMCkInfoChild.cksize;
HANDLE waveDataBlock = ::GlobalAlloc(GMEM_MOVEABLE, lDataSize);
if (waveDataBlock == NULL) {
      MessageBox("Ошибка выделения памяти");
      mmioClose(hmmio, 0);
      return (NULL);
}

// читаем данные
char* pWave = (char*)::GlobalLock(waveDataBlock);
if (mmioRead(hmmio, (LPSTR)pWave, lDataSize) != lDataSize) {
      MessageBox("Ошибка чтения данных");
```

```
	mmioClose(hmmio, 0);
	::GlobalFree(waveDataBlock);
	return (NULL);
}
```

Теперь есть вся необходимая информация для заполнения структуры *WAVEHDR*, описывающей отдельный wav-буфер. Закончив чтение файла и подготовку заголовков, можно закрыть файл. Ниже приведен полный листинг функции *OpenWaveFile()*.

```
// Открывает и читает указанный wav-файл. Возвращает ссылку
// на область памяти, где хранятся данные WAV или NULL при неудаче
HANDLE CKLLab3Dlg::OpenWaveFile(CString FileName) {
	MMCKINFO    MMCkInfoParent;
	MMCKINFO    MMCkInfoChild;
	char        info[30];

	// Открыть wav-файл
	HMMIO hmmio = mmioOpen(FileName.GetBuffer(80), NULL, MMIO_READ);

	sprintf(info, "Открываем файл: %s", FileName.GetBuffer(80));
	log.AddString(_T(info));

	// Успешно?
	if (!hmmio) {
		MessageBox("Не могу открыть файл");
		return (NULL);
	}

	log.AddString(_T("Ищем основной блок RIFF..."));

	// Найти основной блок RIFF
	MMCkInfoParent.fccType = mmioFOURCC('W','A','V','E');
	int errorCode = mmioDescend(hmmio, &MMCkInfoParent, NULL,
				MMIO_FINDRIFF);
	if (errorCode) {
		MessageBox("Ошибка поиска блока WAVE в файле");
		mmioClose(hmmio, 0);
		return (NULL);
	}
	sprintf(info, "--- Размер блока RIFF: %i", MMCkInfoParent.cksize);
	log.AddString(_T(info));

	log.AddString(_T("Ищем блок формата..."));

	// Найти блок формата
	MMCkInfoChild.ckid = mmioFOURCC('f','m','t',' ');
	errorCode = mmioDescend(hmmio, &MMCkInfoChild,
				&MMCkInfoParent, MMIO_FINDCHUNK);
	if (errorCode) {
		MessageBox("Ошибка поиска блока формата в файле");
		mmioClose(hmmio, 0);
		return (NULL);
	}
	sprintf(info, "--- Размер блока формата: %i",
			MMCkInfoChild.cksize);
	log.AddString(_T(info));

	log.AddString(_T("Читаем запись о формате данных PCM"));

	// Прочитать запись о формате данных PCM
	DWORD bytesRead = mmioRead(hmmio, (HPSTR)&PCMWaveFmtRecord,
```

```
                                    MMCkInfoChild.cksize);
if (bytesRead <= 0) {
      MessageBox("Ошибка чтения данных о формате PCM");
      mmioClose(hmmio, 0);
      return (NULL);
}

log.AddString(_T("Проверяем совместимость данных с устройством"));

// Проверить совместимость данных с устройством
HWAVEOUT hWaveOut;
errorCode = waveOutOpen(&hWaveOut, WAVE_MAPPER,
                        (LPWAVEFORMATEX) &PCMWaveFmtRecord,
                        NULL, NULL, WAVE_FORMAT_QUERY);
if (errorCode) {
      MessageBox("Несовместимый формат файла");
      mmioClose(hmmio, 0);
      return (NULL);
}

log.AddString(_T("Поднимаемся на один уровень в иерархии RIFF"));

// Подняться на один уровень в иерархии RIFF
errorCode = mmioAscend(hmmio, &MMCkInfoChild, 0);
if (errorCode) {
      MessageBox("Не могу вернуться");
      mmioClose(hmmio, 0);
      return (NULL);
}

log.AddString(_T("Читаем блок данных"));

// Прочитать блок данных
MMCkInfoChild.ckid = mmioFOURCC('d','a','t','a');
errorCode = mmioDescend (hmmio, &MMCkInfoChild,
                         &MMCkInfoParent, MMIO_FINDCHUNK);
if (errorCode) {
      MessageBox("Не могу прочитать блок данных");
      mmioClose(hmmio, 0);
      return(NULL);
}
sprintf(info, "--- Размер блока данных: %i", MMCkInfoChild.cksize);
log.AddString(_T(info));

log.AddString(_T("Выделяем память для wav-данных"));

// Выделить память для wav-данных
long lDataSize = MMCkInfoChild.cksize;
HANDLE waveDataBlock = ::GlobalAlloc(GMEM_MOVEABLE, lDataSize);
if (waveDataBlock == NULL) {
      MessageBox("Ошибка выделения памяти");
      mmioClose(hmmio, 0);
      return (NULL);
}

log.AddString(_T("Читаем данные"));

// Читаем данные
char* pWave = (char*)::GlobalLock(waveDataBlock);
if (mmioRead(hmmio, (LPSTR)pWave, lDataSize) != lDataSize) {
      MessageBox("Ошибка чтения данных");
```

```
            mmioClose(hmmio, 0);
            ::GlobalFree(waveDataBlock);
            return (NULL);
      }

      log.AddString(_T("Заполняем заголовок"));

      // Заполняем заголовок
      WaveHeader.lpData = pWave;
      WaveHeader.dwBufferLength = lDataSize;
      WaveHeader.dwFlags = 0L;
      WaveHeader.dwLoops = 0L;

      log.AddString(_T("закрываем wav-файл"));

      // Закрываем wav-файл
      mmioClose(hmmio,0);

      // Возвращаем ссылку на блок памяти
      return waveDataBlock;
}
```

Итак, файл открыт и из него прочитаны данные. На следующем этапе необходимо реализовать воспроизведение wav-файла через устройство *Waveaudio*. Для этого следует выполнить пять шагов:

1. Открыть устройство Waveaudio.
2. Подготовить заголовок сегмента.
3. Вывести данные в устройство.
4. Вернуть заголовок сегмента в исходное состояние.
5. Закрыть устройство.

Начнем с открытия устройства Waveaudio. Для этого используется функция *waveOutOpen()*. Для вызова *waveOutOpen()* используется шесть параметров:

1. Первый – это адрес свободной ссылки на устройство, он используется для присваивания этой ссылке определенного значения, впоследствии эта ссылка используется во всех операциях с устройством.
2. Второй параметр указывает, какое из аудиоустройств мы хотим открыть.
3. Третий параметр указывает на адрес структуры с информацией о формате данных.
4. Четвертый и пятый параметры определяют функцию «возврата» и необходимую для нее информацию.
5. С помощью последнего параметра передается флаг для управления методом «возврата».

Далее необходимо подготовить заголовок для воспроизведения, воспользовавшись функцией *waveOutPrepareHeader()*. После этого можно смело посылать его в устройство *Waveaudio* при помощи функции *waveOutWrite()*. У этой функции есть три аргумента: ссылка на открытое устройство, адрес заголовка и размер заголовка. Закончив вывод данных, необходимо вернуть заголовок в исходное состояние и закрыть устройство. Но этого нельзя делать до тех пор, пока не закончится реальное воспроизведение посланных данных, для чего организуется цикл, ожидающий, пока функция *waveOutWrite()* не установит флаг *WHDR_DONE* в структуре *WaveHeader*. Вернуть заголовок в исходное состояние можно функцией *waveOutUnprepareHeader()* и затем обязательно закрыть устройство. Приведем полный листинг функции *WavePlay()*:

```
// Воспроизводит звуковой файл, определенный заголовком
// WaveHeader и структурой PCMWaveFormatRecord
BOOL CKLLab3Dlg::WavePlay() {
      // ссылка на wav-устройство
      HWAVEOUT hWaveOut;
```

```
        log.AddString(_T("Открываем wav-устройство"));

        // Открыть wav-устройство
        MMRESULT ReturnCode = waveOutOpen (&hWaveOut, WAVE_MAPPER,
                              (LPWAVEFORMATEX) &PCMWaveFmtRecord,
                              NULL, NULL, CALLBACK_NULL);
        if (ReturnCode) {
              MessageBox("Ошибка при открытии wav-устройства");
              return (FALSE);
        }

        log.AddString(_T("Подготавливаем заголовок"));

        // Подготавливаем заголовок
        ReturnCode = waveOutPrepareHeader(hWaveOut, &WaveHeader,
                                          sizeof(WaveHeader));
        if (ReturnCode) {
              MessageBox("Ошибка при подготовке заголовка");
              waveOutClose(hWaveOut);
              return (FALSE);
        }

        log.AddString(_T("Выводим данные в wav-устройство"));

        // Выводим данные в WAVE-устройство
        ReturnCode = waveOutWrite(hWaveOut, &WaveHeader,
                                  sizeof(WaveHeader));
        if (ReturnCode) {
              MessageBox("Ошибка записи в wav-устройство");
              waveOutClose(hWaveOut);
              return (FALSE);
        }

        // Цикл до окончания воспроизведения
        do {}
        while (!(WaveHeader.dwFlags & WHDR_DONE));

        log.AddString(_T("Возвращаем заголовок в исходное состояние"));

        // Вернуть заголовок в исходное состояние
        ReturnCode = waveOutUnprepareHeader(hWaveOut, &WaveHeader,
                                            sizeof(WaveHeader));
        if (ReturnCode) {
              MessageBox("Ошибка при восстановлении заголовка");
              waveOutClose(hWaveOut);
              return (FALSE);
        }
        WaveHeader.dwFlags = 0L;

        log.AddString(_T("Закрываем wav-устройство"));

        // Закрываем wav-устройство
        ReturnCode = waveOutClose(hWaveOut);
        if (ReturnCode) {
              AfxMessageBox("Ошибка при закрытии wav-устройства");
              waveOutClose(hWaveOut);
              return (FALSE);
        }
        return (TRUE);
}
```

Завершим написание данной программы, добавив функциональности кнопке. Это можно сделать двумя путями:

- через *ClassWizard* на вкладке *Message Maps* выбрать в списке элементов *IDC_WAVEOUTWRITE*, затем в списке *Messages* выбрать сообщение *BN_CLICKED* и нажать на кнопку *AddFunction*. В появившемся диалоговом окне нажать *OK*;
- при визуальном проектировании диалога кликнуть два раза левой кнопкой мыши по нужной кнопке, при этом появиться диалоговое окно, в котором необходимо нажать *OK*, если имя функции верное. В любом случае *Visual C++* автоматически создаст необходимый код.

Теперь, когда создан обработчик нажатия кнопки, добавим следующий код:

```
void CKLLab3Dlg::OnWaveoutwrite()
{
	// Выбираем wav-файл
	CFileDialog Dlg(TRUE, NULL, NULL,
			OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
			TEXT("sound files (*.wav)|*.wav||"));

	if (IDOK != Dlg.DoModal())
		return;

	log.AddString(_T("Начинаем работу..."));

	log.AddString(_T("Открываем wav-файл"));
	log.AddString(_T("------------------------"));

	// Открываем wav-файл
	HANDLE handle = OpenWaveFile(Dlg.GetPathName());
	if (!handle) {
		MessageBox("Ошибка при открытии файла");
		return;
	}

	log.AddString(_T(""));
	log.AddString(_T("Воспроизводим wav-файл"));
	log.AddString(_T("------------------------"));

	// Воспроизводим wav-файла
	if (!WavePlay()) {
		MessageBox("Ошибка при воспроизведении wav-файла");
	}

	log.AddString(_T(""));
	log.AddString(_T("Освобождаем память"));

	// Освобождаем память
	if (::GlobalFree(handle)) {
		MessageBox("Ошибка при освобождении памяти");
	}

	log.AddString(_T("Завершение работы."));
}
```

В данном фрагменте кода сначала создается диалог открытия файла *CFileDialog*. Если файл был выбран, то работа продолжается; иначе – выходим из функции, так как нечего будет воспроизводить. Получить путь и имя файла можно с помощью функции *GetPathName()* класса *CFileDialog*. Полученный путь и имя файла передаем функции *OpenWaveFile()*. При удачном завершении функции *OpenWaveFile()* вызываем функцию *WavePlay()*. И затем обязательно освобождаем выделенную память.

Теперь, чтобы приложение успешно скомпилировалось, необходимо подключить к проекту библиотеку *winmm.lib*. Для этого выберите меню *Project -> Settings…* или нажмите комбинацию клавиш *Alt+F7*. В диалоговом окне необходимо выбрать вкладку *Link* и в поле *«Object/library modules:»* написать *winmm.lib* и нажать ОК (см. рис. 1.10).

Проект готов к компиляции. Для этого либо нажмите *CTRL+F5*, либо выберите пункт меню *Build -> Execute KLLab3.exe*.

3.4. Вопросы для самопроверки

1. Опишите кратко формат файлов RIFF. Для чего он предназначен?
2. Какой алгоритм воспроизведения звука из wav-файла с использованием низкоуровневых мультимедиа функций ввода-вывода?
3. Чем указанные функции отличаются от функций высокого уровня, изученных при выполнении лабораторной работы №1, и функций интерфейса MCI, которые были изучены при выполнении лабораторной работы №2?
4. Каким образом обрабатываются ошибки при использовании низкоуровневых мультимедиа-функций ввода-вывода?

Лабораторная работа №4
«Программная реализация системы работы со звуком»

Цель работы – разработать приложение, работающее со звуком в операционной системе Windows.

Основные **задачи** работы следующие:
1. Изучить технологию разработки программных приложений, работающих со звуком.
2. Научиться проектировать приложения по работе со звуком.
3. Научиться самостоятельно осваивать инструментальные средства, библиотеки программ и т.п. по работе со звуком.
4. Разработать программное приложение, которое осуществляет обработку звукового файла или речи.
5. Закрепить навыки программирования звука в среде Microsoft Visual C.

Прежде чем приступить к выполнению лабораторной работы, просмотрите все предлагаемое ниже ее описание и обратите внимание на примечания, справочные материалы и другую дополнительную информацию.

Примечание. Данная лабораторная работа рассчитана на четыре занятия. Поэтому на 1-м занятии необходимо определиться с постановкой задачи и согласовать с преподавателем график работы на все четыре занятия. Предположительно график может включать следующие этапы:

1-е занятие:
- выбор варианта реализации приложения;
- уточнение постановки задачи, выбор инструментальных средств (с учетом возможностей их установки в учебной лаборатории – по согласованию с преподавателем и лаборантом);
- формулировка задач, решаемых в рамках планируемого приложения;
- разработка структуры и схем алгоритмов работы приложения;

2-е занятие:
- разработка интерфейса приложения;
- реализация основных компонентов приложения;

3-е занятие:
- завершение реализации;
- тестирование и отладка приложения;

4-е занятие:
- завершение тестирования и отладки;
- разработка help-а для пользователя;
- составление отчета о работе;
- защита лабораторной работы.

Варианты «звуковых приложений» под Windows могут быть самыми разными: от аудиоплееров и программ синтеза звука до речевых приложений с использованием существующих библиотек программ и технологий (например Speech API). Инструментальные средства тоже могут быть практически любыми, но с учетом возможностей их установки в лаборатории.

4.1. Варианты заданий

Вариант 1. «Программная реализация синтезатора речи с использованием компиляционного метода»

Метод компиляции (фонемный, или аллофонный, или по слогам и т. п.) выбирается самостоятельно. Программа должна обеспечивать следующие минимальные возможности:
- создание/редактирование базы фонем/слогов;
- ввод текста (орфографического, или предварительно размеченного, или с возможностью ручной разметки);
- воспроизведение сгенерированного речевого сигнала для введенного текста.

Вариант 2. «Программная реализация синтезатора речи с использованием параметрического метода»

Параметрический метод (например формантный) выбирается самостоятельно. Программа должна обеспечивать следующие минимальные возможности:
- ввод текста (орфографического, или предварительно размеченного, или с возможностью ручной разметки);
- воспроизведение сгенерированного речевого сигнала для введенного текста;
- регулировка параметров воспроизведения: громкость, темп, тембр и т. п.;
- создание/редактирование речевой базы – по возможности.

Вариант 3. «Разработка речевого приложения с использованием Speech API»

Программа должна обеспечивать следующие минимальные возможности:
- ввод текста, или копирование через буфер, или поддержка указателя мыши в любом другом приложении (например на html-страничке);
- воспроизведение сгенерированного речевого сигнала для введенного текста;
- настройки (выбор) голоса;
- выбор темпа, громкости и др. параметров для чтения.

Вариант 4. «Разработка персонального говорящего агента в рамках любого другого приложения»

Реализовать такого агента можно с использованием технологии Microsoft Agents. Программа должна обеспечивать следующие минимальные возможности:
- ввод текста, или копирование через буфер, или поддержка указателя мыши в любом другом приложении (например на html-страничке);
- воспроизведение сгенерированного речевого сигнала для введенного текста;
- речевое воспроизведение сообщений приложения, в рамках которого используется агент;
- настройки (выбор) персонажа и голоса.

В качестве приложения, для которого будет разработан такой «речевой» персональный агент, может быть использована любая реализованная вами программа. Например, это может быть какая-либо обучающая программа, или система компьютерного тестирования знаний, или текстовый (графический) редактор, или просто ваша домашняя страничка (в Интернете есть такие примеры) и т. д.

Вариант 5. «Разработка аудиоплеера»

Программа должна обеспечивать следующие минимальные возможности:
- открытие и воспроизведение звукового файла (поддержка какого-либо заранее выбранного формата или несколько самых распространенных форматов);
- регулировка громкости воспроизведения;
- поиск по файлу («пролистывание» в начало, конец, внутри файла);
- циклическое воспроизведение одного и того же файла;
- создание и сохранение плэй-листа;

- циклическое или случайное воспроизведение файлов из плэй-листа – по возможности;
- регулировка тембра, баланса и т.п. – по возможности;
- вывод осциллограммы воспроизводимого файла – по возможности.

Вариант 6. «Разработка программы обработки аудиофайлов»

Это программа аналогична Sound Forge. Программа должна обеспечивать следующие минимальные возможности:
- открытие и воспроизведение звукового файла (поддержка какого-либо заранее выбранного формата или несколько самых распространенных форматов);
- регулировка громкости воспроизведения;
- поиск по файлу («пролистывание» в начало, конец, внутри файла);
- вывод осциллограммы воспроизводимого файла;
- редактирование файла: элементарные функции вырезания, добавления фрагментов осциллограммы;
- дополнительная обработка звука: добавление реверберации, подчистка частот, изменение темпа и т. п.

Вариант 7. «Разработка программы синтеза звука»

Это программа способна синтезировать звук по заданным (регулируемым пользователем) характеристикам. Для ее разработки необходимо иметь сведения о том, как вообще синтезируется звук на компьютере, а также какие параметры его (звук) описывают. Это может быть программа, синтезирующая любые звуки, аналогичная музыкальным синтезаторам, либо программа, синтезирующая речевой сигнал на основе любого параметрического метода (например формантного). Программа должна обеспечивать следующие минимальные возможности:
- ввод параметров синтеза звука (если это речь, то можно обеспечить выбор синтезируемых фонем или слов с параметрами голоса, темпа, тембра и т. п.);
- регулировка параметров синтеза звука (синхронная или асинхронная воспроизведению);
- отображение значений параметров в виде чисел, а также различных графиков и диаграмм;
- сохранение наборов параметров и их динамического изменения;
- синтез звука на основе сохраненных наборов динамически изменяющихся параметров;
- удобный для пользователя интерфейс (подобный реальным аудиосистемам и музыкальным синтезаторам).

Вариант 8. «Разработка программного аудиомикшера»

Это программа, аналогичная микшеру, встроенному в системе Microsoft Windows, но разработанная самостоятельно и перехватывающая настройки Windows. Такими приложениями, как правило, сопровождается драйвер любой более-менее «продвинутой» звуковой карты. Программа должна обеспечивать следующие минимальные возможности:
- ввод пользователем основных параметров управления воспроизведением звука: громкость, баланс, понижение/повышение частоты и т. п., а также (по возможности) настроек для записи звука с микрофона или линейного входа;
- перехват воспроизводимого в аудиосистеме Windows звукового файла и применение к нему настроек пользователя;
- поддержка возможности воспроизведения нескольких звуковых файлов одновременно (их синхронизация во времени);
- удобный для пользователя интерфейс (содержащий всяческие «крутящиеся ручки» и «передвигающиеся рычажки»).

Варианты 9, 10, … Любые другие ваши пожелания (по согласованию с преподавателем)

4.2. Требования к отчету

В качестве **отчета** по работе необходимо представить достаточно подробное описание разработки по аналогии с тем, как вы это делаете при написании курсовых проектов, но в несколько меньшем объеме. Основные составляющие отчета:

1. Назначение разработанного приложения.
2. Перечень возможностей приложения.
3. Структура приложения и схема внешнего интерфейса.
4. Схемы алгоритмов (основных, в которых происходит работа со звуком или речью).
5. Схематичное описание технологии разработки приложений выбранного класса.
6. Основные выводы по работе.

Детали содержания отчета можно уточнить у преподавателя в индивидуальном порядке в зависимости от выбранного варианта.

Литература

1. ***Елисеева О. Е.2007мет-Речев_И***
   Елисеева, О. Е. Речевой интерфейс. Лабораторный практикум : учеб.-метод. пособие. В 2 ч. Ч. 1 / О. Е. Елисеева ; под ред. проф. В. В. Голенкова. – Минск : БГУИР, 2008. – 44 с.
2. ***Лобанов Б.М..2006кн-Речев_И_ИС***
   Лобанов, Б. М. Речевой интерфейс интеллектуальных систем : учеб. пособие / Б. М. Лобанов, О. Е. Елисеева ; под науч. ред. В. В. Голенкова . – Минск : БГУИР, 2006. – 152 с.
3. ***Мешков А..2004кн-Visual_C***
   Мешков, А. Visual C++ и MFC. Руководство для профессионалов / А. Мешков, Ю. Тихомиров ; пер. с англ. – 2-е изд. – СПб. : БХВ – Санкт-Петербург, 2004. – 1040 с.
4. ***Страуструп Б.2008кн-Язык_П_С***
   Страуструп, Б. Язык программирования C++. Специальное издание / Б. Страуструп. – М. : Бином, 2008. – 1104 с.

Учебное издание

Елисеева Ольга Евгеньевна
Сердюков Роман Евгеньевич

РЕЧЕВОЙ ИНТЕРФЕЙС. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Учебно-методическое пособие

В 2-х частях

Часть 2

Редактор *Л. А. Шичко*

Корректор *Е. Н. Батурчик*

Дизайн обложки *А. А. Макаров*

Подписано в печать .04.2009. Формат 60x84 1/8. Бумага офсетная. Гарнитура «Arial».
Печать ризографическая. Усл.печ.л. . Уч.-изд.л. 4,0. Тираж 100 экз. Заказ 329 .

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Ли № 02330/0494371 от 16.03.2009.
Ли № 02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6.