

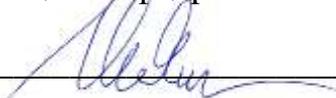
МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное  
образовательное учреждение высшего образования  
«Тульский государственный университет»

Институт прикладной математики и компьютерных наук  
Кафедра «Прикладная математика и информатика»

Утверждено на заседании кафедры  
«Прикладная математика и информатика»  
24 января 2022 г., протокол № 5

Заведующий кафедрой

  
\_\_\_\_\_ М.В. Грязев

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ**  
**по выполнению лабораторных работ**  
**по дисциплине (модулю)**  
**«Современные технологии программирования»**

**основной профессиональной образовательной программы**  
**высшего образования – программы магистратуры**

по направлению подготовки  
**01.04.02 Прикладная математика и информатика**  
с направленностью (профилем)  
**Искусственный интеллект в кибербезопасности**

Форма обучения: очная

Идентификационный номер образовательной программы: 010402-02-22

Тула 2022 год

## Разработчик методических указаний

Смирнов О.И., доцент каф. ПМий, к.ф.-м.н., доцент

---

*(ФИО, должность, ученая степень, ученое звание)*



---

*(подпись)*

Лабораторная работа № 1

## **РАБОТА В СРЕДЕ VISUAL C++**



### **ЦЕЛЬ РАБОТЫ**

Приобретение практических навыков работы в среде разработки Visual C++ 6.0.



### **ЗАДАНИЕ НА РАБОТУ**

Изучив представленный ниже материал создать самостоятельную программу рисования.

## Интерфейсы SDI и MDI

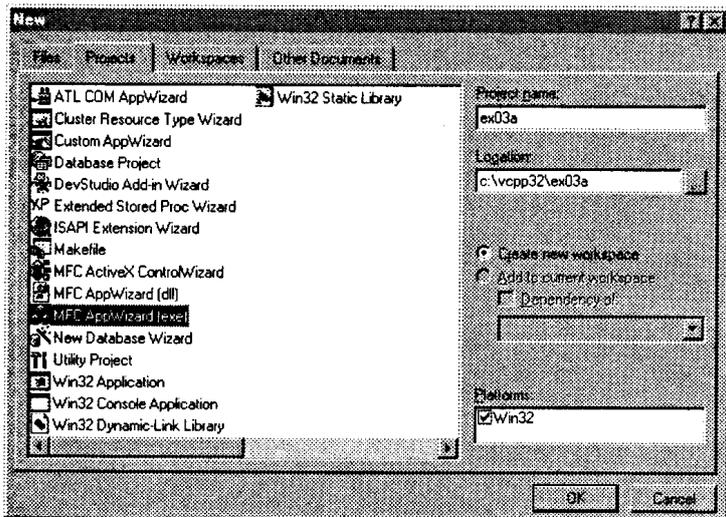
Библиотека MFC поддерживает приложения двух разных типов: *однодокументный интерфейс* (Single Document Interface, SDI) и *многодокументный интерфейс* (Multiple Document Interface, MDI). С точки зрения пользователя, SDI — это приложение, состоящее из единственного окна. Если такое приложение работает с «документами» в дисковых файлах, то в каждый момент времени можно загрузить только один документ. Примером приложения SDI служит Windows Notepad. В MDI-приложении имеется несколько *дочерних окон* (child windows), каждое из которых соответствует отдельному документу. Прекрасный пример такого приложения — Microsoft Word.

При запуске AppWizard для создания нового проекта параметр MDI выбран по умолчанию. В начальных примерах этой книги мы будем создавать SDI-приложения, так как в них меньше классов и требуется учитывать меньше особенностей. Убедитесь, что для этих примеров Вы выбрали тип приложения Single Document (в первом диалоговом окне AppWizard). Начиная с главы 18, Вы будете создавать MDI-приложения. Архитектура каркаса приложения MFC позволяет легко преобразовывать большинство SDI-примеров в MDI-приложения.

### Пример EX03A

AppWizard генерирует код работающего приложения MFC. Это приложение просто выводит на дисплей пустое окно с меню. Впоследствии Вы добавите к нему код, выполняющий отрисовку внутри окна. Далее приведены этапы создания приложения.

1. **Сгенерируйте приложение SDI с помощью AppWizard.** Из меню File в Visual C++ выберите New, затем щелкните в появившемся на экране диалоговом окне вкладку Projects, как показано ниже.



Убедитесь, что в списке выделен элемент MFC AppWizard (exe), и затем введите в поле Location путь `C:\vcpp32\`. В поле Project Name введите `ex03a` и щелкните ОК. Теперь Вам предстоит пройти через последовательность шагов AppWizard, первый из которых представлен на первом рисунке на стр. 24.

Убедитесь, что Вы выбрали Single Document. В следующих четырех диалоговых окнах оставьте все без изменений. Последнее диалоговое окно будет выглядеть как на втором рисунке на стр. 24.

Обратите внимание, что при генерации имен классов и исходных файлов за основу было принято название проекта EX03A. Сейчас Вы можете, при желании, изменить названия. Щелкните кнопку Finish. Непосредственно перед генерацией кода AppWizard отображает

После того, как Вы щелкнули кнопку ОК, AppWizard создает подкаталог приложения (ex03a в каталоге \vcpp32), а в нем ряд файлов. По окончании работы AppWizard посмотрите на содержимое каталога приложения. Интересны (на данный момент) следующие файлы.

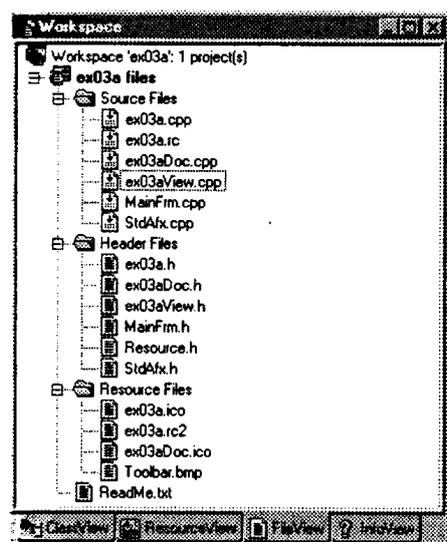
Файл	Описание
ex03a.dsp	Файл проекта, при помощи которого Visual C++ будет выполнять сборку приложения
ex03a.dsw	Файл рабочего пространства, содержащего единственный проект ex03a.dsp
ex03a.rc	Текстовый файл описаний ресурсов
ex03aView.cpp	Файл реализации класса «вид», в котором находятся функции-члены класса <i>CEx03aView</i>
ex03aView.h	Заголовочный файл класса «вид», содержащий объявление класса <i>CEx03aView</i>
ex03a.opt	Двоичный файл, в котором Visual C++ хранит информацию о том, какие файлы открыты для данного проекта и как расположены окна. (Файл не создается до тех пор, пока Вы не сохраните проект)
ReadMe.txt	Текстовый файл, в котором описано назначение сгенерированных файлов
Resource.h	Заголовочный файл, содержащий определения констант <i>#define</i>

Откройте файлы ex03aView.cpp и ex03aView.h и взгляните на исходный код. Вместе эти файлы определяют класс *CEx03aView* — центральный в приложении. Объект класса *CEx03aView* соответствует рабочему окну программы, где и происходят все «события».

**Выполните компиляцию и компоновку сгенерированного кода.** Помимо генерации кода, AppWizard создает для вашего приложения файлы проекта и рабочего пространства. Файл проекта ex03a.dsp описывает все зависимости файлов, а также параметры компилятора и компоновщика. Так как новый проект становится текущим проектом Visual C++, Вы можете собрать приложение, выбрав Build Ex03a.exe из меню Build или щелкнув показанную здесь кнопку Build на панели управления.



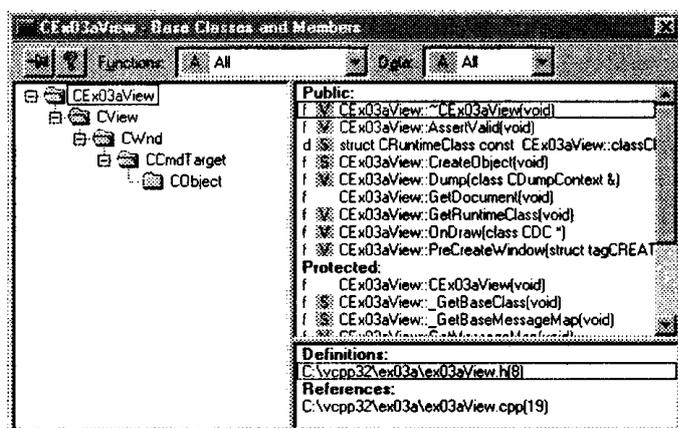
Если сборка прошла успешно, то в подкаталоге Debug каталога \vcpp32\ex03a будет создан исполняемый файл ex03a.exe. Файлы OBJ и другие промежуточные файлы также помещаются в каталог Debug. Сравните структуру каталогов на диске со структурой страницы FileView в окне Workspace.



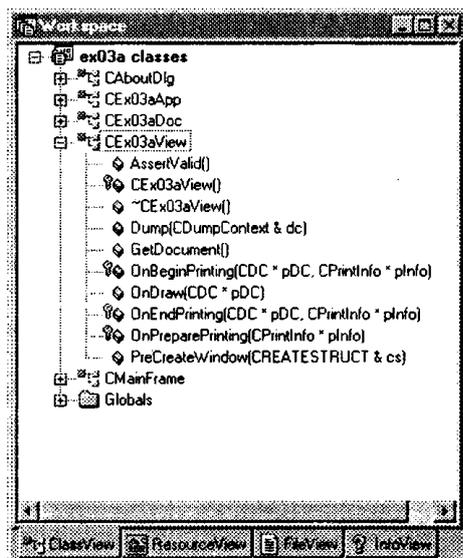
FileView содержит логическое представление проекта. Заголовочные файлы располагаются в разделе Header Files, хотя физически они хранятся в том же подкаталоге, что и файлы CPP. Файлы ресурсов хранятся в подкаталоге \res.

3. **Протестируйте полученное приложение.** Выберите из меню Build пункт Execute Ex03a.exe. Поэкспериментируйте с программой. Она мало на что способна, не так ли? (Но вряд ли можно ожидать большего, не написав ни строки кода.) На самом деле, как Вы, вероятно, догадываетесь, у программы много возможностей, просто они еще не активизированы. Закончив эксперименты, закройте окно программы.
4. **Просмотрите исходные тексты программы.** Выберите из меню Tools пункт Source Browser. Если параметры проекта не требуют создания базы данных средства просмотра, Visual C++ предложит изменить их соответствующим образом и перекомпилировать программу. (Чтобы изменить параметры самостоятельно, выберите Settings из меню Project. На вкладке C/C++ установите флажок Generate Browse Info, а на вкладке Browse Info установите флажок Build Browse Info File.)

Когда на экране появится окно Browse, выберите Base Classes And Members («Базовые классы и члены») и введите *CEx03aView*. Раскрыв ветви иерархии, Вы должны получить результат, аналогичный представленному ниже.



Сравните эти результаты с содержимым страницы ClassView окна Workspace.



ClassView не показывает иерархию классов, но зато не требует дополнительных накладных расходов, связанных со средством просмотра. Если Вам достаточно ClassView, не создавайте базу данных средства просмотра.

## Класс *CEx03aView*

Класс *CEx03aView* сгенерирован AppWizard и специфичен для приложения EX03A. (AppWizard строит имена классов на основании имени проекта, заданного Вами в его первом диалоговом окне.) *CEx03aView* находится внизу длинной цепочки наследования классов библиотеки MFC, как Вы могли видеть ранее в окне Browse. В классе собраны функции-члены и переменные-члены со всей цепочки. Информацию об этих классах можно получить из *Microsoft Foundation Class Reference* (в интерактивном режиме или в отпечатанной версии), но обязательно просматривайте описания всех базовых классов, так как описания наследуемых функций-членов обычно не повторяются для производных классов.

Наиболее важные базовые классы *CEx03aView* — *CWnd* и *CView*. *CWnd* придает *CEx03aView* свойства окна, а *CView* обеспечивает связь с остальными частями каркаса приложения, в частности с документом и рамочным окном, как Вы увидите в третьей части данной книги.

## Рисование внутри окна представления: Windows GDI

Теперь Вы готовы к тому, чтобы написать код, который будет рисовать в окне представления. Вы внесете несколько изменений непосредственно в исходный текст EX03A.

### Функция-член *OnDraw*

Конкретнее, Вам понадобится наполнить реализацию *OnDraw* в *ex03aView.cpp*. *OnDraw* — это виртуальная функция-член класса *CView*, которую каркас приложения вызывает всякий раз, когда необходимо перерисовать окно представления. Перерисовка требуется, когда пользователь изменил размеры окна или открыл ранее невидимые его части, либо если приложение изменило данные окна. В первых двух случаях *OnDraw* вызывается каркасом приложения автоматически; однако если данные окна изменены функцией внутри вашей программы, эта функция должна уведомить Windows об изменениях, вызвав унаследованную классом «вид» функцию-член *Invalidate* (или *InvalidateRect*). Код *Invalidate* впоследствии вызовет *OnDraw*.

Хотя внутри окна можно рисовать всегда, рекомендуется позволять ему накапливать изменения и обрабатывать их все вместе при вызове функции *OnDraw*. В этом случае ваша программа сможет реагировать как на события, сгенерированные ею самой, так и на события, сгенерированные Windows, например, изменения размеров окна.

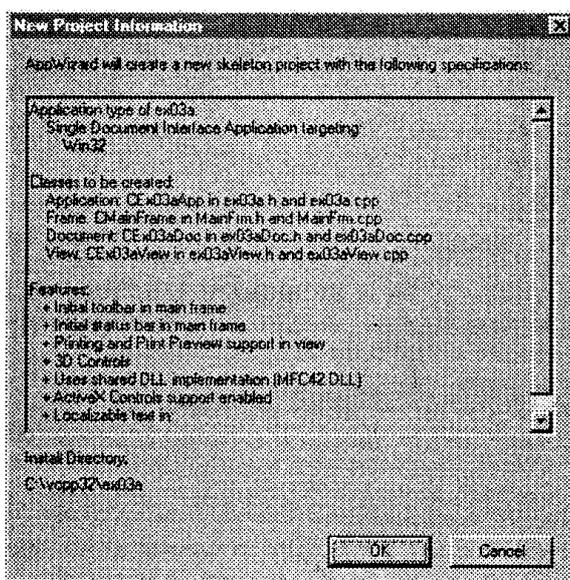
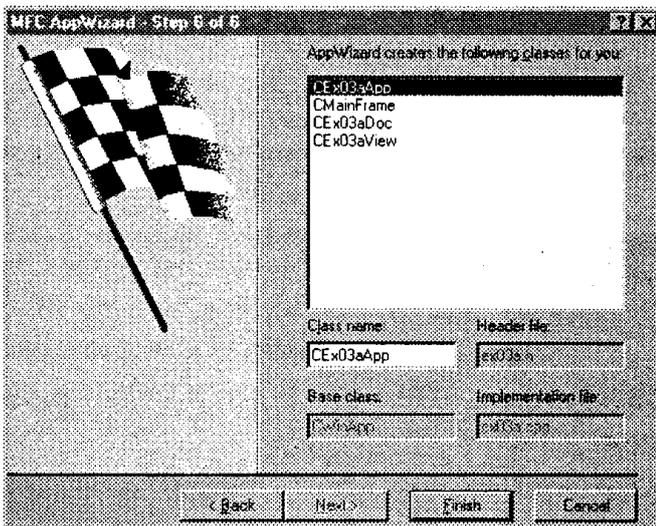
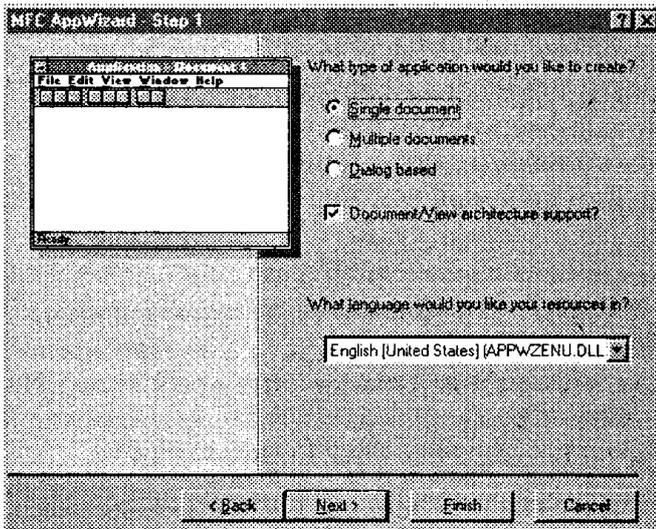
### Контекст устройства в Windows

Как Вы помните из главы 1, Windows не разрешает прямой доступ к аппаратуре дисплея, а взаимодействует с ней посредством абстракции под названием «контекст устройства» (*device context*), который связан с окном. В библиотеке MFC контекст устройства представлен в виде объекта класса C++ с именем *CDC*, который передается *OnDraw* как параметр (посредством указателя). Имея указатель на *CDC*, Вы можете воспользоваться множеством функций этого класса для рисования.

### Добавление кода рисования в программу EX03A

Теперь напишем код, рисующий внутри окна представления текст и круг. Убедитесь, что в Visual C++ открыт проект EX03A. Чтобы найти функцию, можно воспользоваться ClassView в окне Workspace (дважды щелкните *OnDraw*) либо открыть исходный файл *ex03aView.cpp* из FileView и отыскать функцию в тексте.

диалоговое окно New Project Information (см. третий рисунок), где представлена информация о новом проекте.



1. **Отредактируйте функцию *OnDraw* в *ex03aView.cpp*.** Найдите в файле *ex03aView.cpp* функцию *OnDraw*, сгенерированную AppWizard:

```
void CEx03aView::OnDraw(CDC* pDC)
{
    CEx03aDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // Доделать: добавьте код отображения своих данных.
}
```

Код, выделенный полужирным шрифтом, который Вы должны ввести, заменяет старый код:

```
void CEx03aView::OnDraw(CDC* pDC)
{
    pDC->TextOut(0, 0, "Hello, world!"); //Вывод шрифтом по умолчанию в левом верхнем углу
    pDC->SelectStockObject(GRAY_BRUSH); //Выбрать кисть для заполнения круга
    pDC->Ellipse(CRect(0, 20, 100, 120)); //Нарисовать серый круг диаметром 100 единиц
}
```

Вызов *GetDocument* можно спокойно удалить, поскольку пока мы не работаем с документами. Функции *TextOut*, *SelectStockObject* и *Ellipse* — члены класса контекста устройства *CDC* каркаса приложения. Функция *Ellipse* рисует круг, если длина ограничивающего прямоугольника равна его ширине.

Для работы с прямоугольниками Windows библиотека MFC предоставляет удобный класс *CRect*. Временный объект *CRect* — аргумент, ограничивающий прямоугольник для функции рисования эллипса. Класс *CRect* будет использоваться в примерах данной книги часто.

2. **Перекомпилируйте и протестируйте EX03A.** Выберите из меню Project пункт Build и, если нет ошибок компиляции, снова запустите программу. Теперь у Вас есть программа, работа которой сразу же видна!

#### Для тех, кто программирует в Win32

Стандартная функция *WinMain* и оконные процедуры скрыты внутри каркаса приложения. Вы увидите эти функции, когда мы будем рассматривать классы библиотеки MFC для окна-рамки и приложения. В данный момент Вы, возможно, удивляетесь, куда же делось сообщение *WM\_PAINT*? Вы ожидаете, что при обработке данного сообщения должно выполняться рисование в окне, а контекст устройства необходимо получать из структуры *PAINTSTRUCT*, возвращаемой функцией *Windows BeginPaint*.

Оказывается, каркас приложения сделал за Вас всю черновую работу и передал контекст устройства (в форме указателя на объект) виртуальной функции *OnDraw*. Как было отмечено в главе 2, истинно виртуальные функции в оконных классах MFC достаточно редки. Большинство сообщений Windows каркас приложения направляет на обработку функциям таблицы сообщений. Программисты, работавшие с MFC 1.0, всегда создавали для своих производных классов-окон функцию таблицы сообщений *OnPaint*. Однако, начиная с версии 2.5, функция *OnPaint* находится в таблице сообщений класса *CView* и выполняет полиморфный вызов функции *OnDraw*. Почему? Потому что *OnDraw* должна поддерживать не только дисплей, но и принтер. *OnDraw* вызывается и *OnPaint*, и *OnPrint*, что позволяет использовать один и тот же код рисования и для принтера, и для дисплея.

# СОЗДАНИЕ MDI ПРИЛОЖЕНИЙ С ГРАФИЧЕСКИМИ ЭЛЕМЕНТАМИ УПРАВЛЕНИЯ В VISUAL C++



## ЦЕЛЬ РАБОТЫ

Приобретение практических навыков создания MDI приложений с графическими элементами управления в Visual C++ 6.0.



## ЗАДАНИЕ НА РАБОТУ

Изучив представленный ниже материал создать самостоятельно аналогичную программу.

## Модальные диалоговые окна

Практически все Windows-программы взаимодействуют с пользователем при помощи диалоговых окон. Диалоговое окно может просто содержать какое-то сообщение и кнопку ОК, а может быть и очень сложной формой для ввода данных. Иногда этот мощный элемент называют «диалоговой панелью», что по меньшей мере несправедливо. Диалоговое окно — настоящее окно, которое можно перемещать и закрывать; оно принимает сообщения и даже отрабатывает команды отрисовки данных в своей клиентской области.

Существует два типа диалоговых окон, *модальные* (modal) и *немодальные* (modeless). В этой главе мы рассмотрим более распространенный модальный тип. В первом из двух примеров данной главы Вы будете работать с давно знакомыми элементами управления, унаследованными от Win 16, например, полем ввода и списком. Во втором — с новыми элементами управления, впервые появившимися в Windows 95.

### Модальные и немодальные диалоговые окна

Базовый класс *CDialog* поддерживает как модальные, так и немодальные диалоговые окна. Пока открыто модальное диалоговое окно (например, Open File — «Открытие файла»), пользователь не может работать ни с каким другим окном программы (точнее, окном того же потока пользовательского интерфейса). Если же открыто немодальное диалоговое окно, работать с другим окном программы можно. Пример — диалоговое окно Find and Replace редактора Microsoft Word, которое совершенно не мешает редактировать текст.

Выбор конкретного типа диалогового окна зависит от характера создаваемого приложения. Программировать модальные диалоговые окна намного проще, и это может повлиять на ваше решение.

**ПРИМЕЧАНИЕ** 16-разрядные версии Windows поддерживают особый вид модального диалогового окна — *системное модальное* (system modal), который не дает пользователю переключаться в другое приложение. Win32 тоже поддерживает системные модальные диалоговые окна, правда, весьма своеобразно: переключаться в другую программу можно, но системное модальное диалоговое окно всегда останется сверху других окон. Так что в Win32-приложениях такие диалоговые окна, по-видимому, не стоит применять.

### Ресурсы и элементы управления

Итак, диалоговое окно — это настоящее окно. Но чем же оно все-таки отличается от окон класса *CView*, с которыми Вы успели поработать? Хотя бы тем, что диалоговое окно почти всегда связано с каким-нибудь ресурсом Windows, идентифицирующим элементы и определяющим структуру окна. Поскольку диалоговый ресурс можно создавать и модифицировать в редакторе диалоговых окон (одном из редакторов ресурсов), диалоговые окна формируются быстро, эффективно и наглядно.

Диалоговое окно содержит набор *элементов управления* (controls): *поля ввода* (edit controls; их еще называют *текстовыми окнами* — text boxes), *кнопки* (buttons), *списки* (list boxes), *комбинированные списки* (combo boxes), *статический текст* (static text) или *метки* (labels), *списки с древовидным отображением* (tree views), *индикаторы продвижения* (progress indicators), *ползунки* (sliders) и т. д. Windows управляет этими элементами, используя специальную, значительно упрощающую труд программиста логику группирования и обхода. На элементы управления можно ссылаться либо по указателю на *CWnd* (поскольку элементы сами являются окнами), либо по индексу (с сопоставленной константой *\*define*), назначенному в ресурсе. Реагируя на действия пользователя, скажем, ввод текста или щелчок кнопки, элементы управления передают сообщения родительскому диалоговому окну.

MFC-библиотека и ClassWizard, тесно взаимодействуя, облегчают программирование диалоговых окон Windows. ClassWizard генерирует класс, производный от *CDialog*, а затем позволяет соотнести переменные-члены класса «диалоговое окно» с элементами управления. Вы можете указывать такие параметры, как максимальная длина текста или границы диапазона для вводимых чисел. После этого ClassWizard генерирует вызовы MFC-функций, отвечающих за обмен данными и проверку их достоверности. Эти функции перемещают информацию между экранными элемен-

тами и переменными-членами соответствующего класса.

## Программирование модального диалогового окна

Модальные диалоговые окна используют чаще всего. Пользователь совершает какое-то действие (например, выбирает команду в меню), на экране появляется диалоговое окно, пользователь вводит данные, а затем закрывает его. Чтобы дополнить существующий проект модальным диалоговым окном, сделайте следующее.

1. Используя редактор диалоговых окон, создайте диалоговый ресурс с элементами управления. Редактор обновит файл описания ресурсов (RC-файл) вашего проекта, включив в него новый ресурс, и файл resource.h, дополнив его соответствующими константами *^define*.

2. С помощью ClassWizard создайте класс «диалоговое окно», производный от *CDialog*, и закрепите его за ресурсом, созданным в п. 1. ClassWizard добавит в проект требуемый код и заголовочный файл.

**ПРИМЕЧАНИЕ** При генерации производного класса диалогового окна ClassWizard формирует конструктор, который запускает конструктор *CDialog*, принимающий в качестве параметра идентификатор ресурса. Заголовочный файл диалогового окна содержит константу класса *IDD*, которой присвоен идентификатор диалогового ресурса. А реализация конструктора в CPP-файле выглядит так:

```

CMyDialog::CMyDialog(CWnd* pParent /*=NULL*/): CDialog(CMyDialog::IDD, pParent)
{
    // здесь должен быть инициализирующий код
}

```

Применение *enum IDD* избавляет CPP-файл от идентификаторов ресурсов, определяемых в файле resource.h данного проекта<sup>1</sup>.

3. Используя ClassWizard, добавьте в класс диалогового окна переменные-члены и функции, предназначенные для обмена и проверки данных.

4. С помощью ClassWizard добавьте обработчики сообщений для кнопок и других (генерирующих события) элементов управления диалогового окна.

5. Напишите код для инициализации (в *OnInitDialog*) элементов управления и для обработчиков сообщений. Убедитесь, что при закрытии диалогового окна (если только пользователь не нажмет кнопку «Отмена») вызывается виртуальная функция-член *OnOK* класса *CDialog* (она вызывается по умолчанию).

6. В вашем классе «вид» напишите код, активизирующий диалоговое окно. Он сводится к вызову конструктора вашего класса диалогового окна и последующему вызову функции-члена *DoModal* класса диалогового окна. *DoModal* возвращает управление только после закрытия диалогового окна.

А теперь рассмотрим все эти операции на примере.

### Пример EX06A

Мы не станем возиться с каким-нибудь простеньким примером, а встроим в наше диалоговое окно почти все возможные элементы управления. Работа эта несложная, потому что нам поможет редактор диалоговых окон Visual C++. Законченный вид диалогового окна представлен на рисунке.

Как видите, это диалоговое окно предназначено для учета кадров. Это пример довольно скучного делового приложения — неплохо бы сделать с ним что-нибудь такое, что было невозможно во времена перфокарт. Программу слегка оживляют полосы прокрутки «loyalty» («лояльность») и «reliability» («надежность») — классический пример прямого ввода и наглядного отображения данных.

### Построение диалогового ресурса

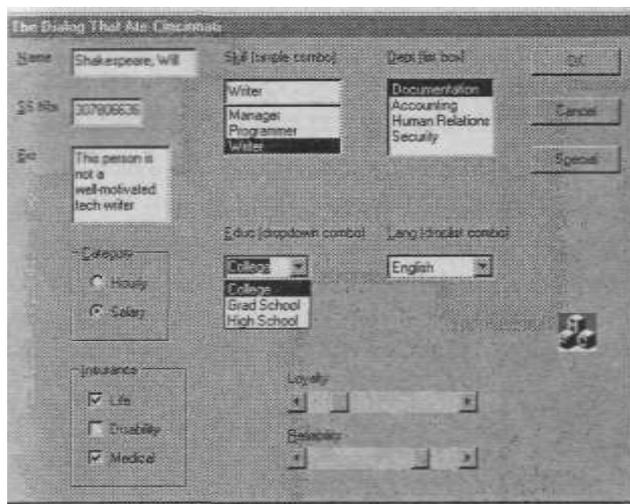
Для создания диалогового ресурса сделайте следующее.

1. Запустите AppWizard, чтобы сгенерировать проект EX06A. Выберите из меню File команду New, а затем щелкните вкладку Projects и выберите MFC AppWizard (exe). Примите параметры по умолчанию, но установите Single Document Interface и откажитесь от Printing And Print Preview. Параметры и имена классов по умолчанию показаны на рисунке.

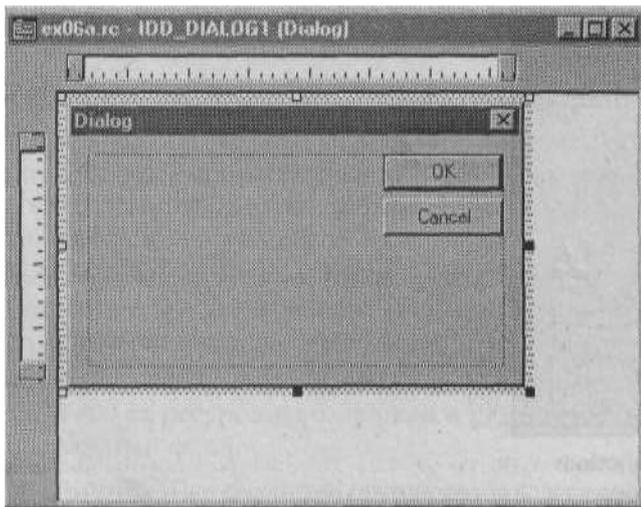
Как обычно, AppWizard будет считать новый проект текущим.

2. Создайте новый диалоговый ресурс с идентификатором *IDD\_DIALOG1*. Выберите из меню Insert среды разработки команду Resource и в появившемся диалоговом окне Insert Resource щелкните строку Dialog, а затем — кнопку New. Visual C++ создаст новый диалоговый ресурс, как показано на рисунке.

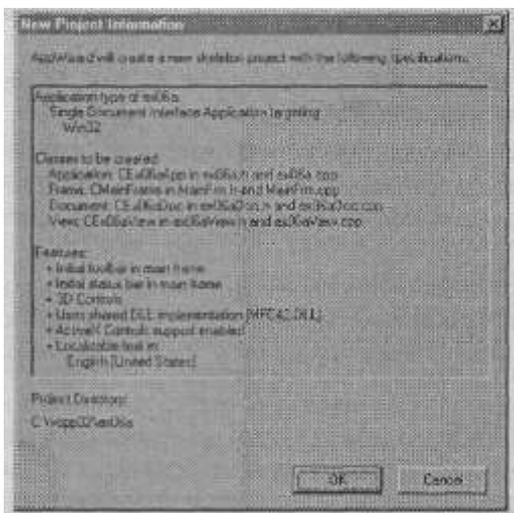
Редактор диалоговых окон присвоит новому диалоговому окну идентификатор ресурса *IDD\_DIALOG1*. Заметьте: он вставляет в новое диалоговое окно кнопки «OK» и «Cancel».



**Отмасштабируйте диалоговое окно и присвойте ему заголовок.** Увеличьте размеры окна редактирования, чтобы было удобно работать. Щелкнув новое диалоговое окно правой кнопкой мыши и выбрав из всплывающего меню команду Properties, Вы увидите диалоговое окно Dialog Properties. Введите заголовок создаваемого диалогового окна — так, как показано на следующем рисунке. Состояние кнопки в левом верхнем углу определяет, будет ли диалоговое окно Dialog Properties располагаться поверх остальных окон. Когда эта кнопка нажата, окно всегда находится поверх других. Щелкнув кнопку Toggle Grid на панели инструментов Dialog, включите линии сетки для создаваемого



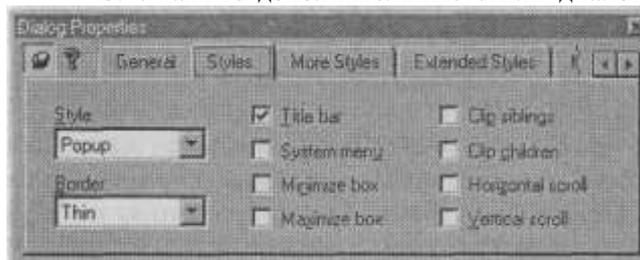
окна — это поможет выровнять элементы управления.



Щелкните вкладку More Styles и задайте параметры.

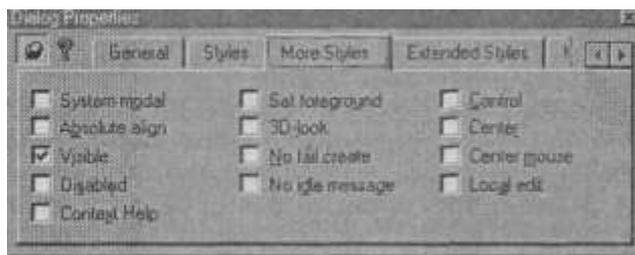
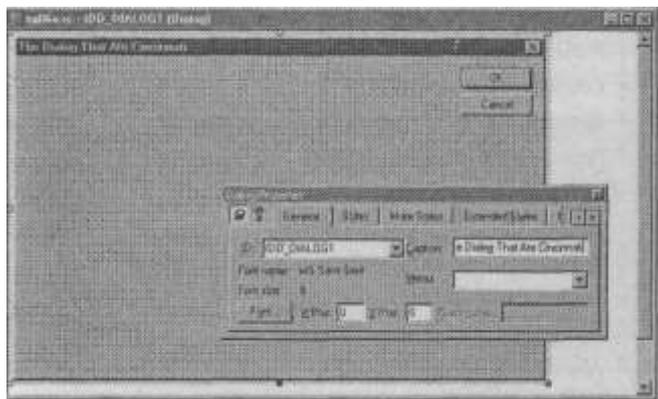
**4. Выберите стиль диалогового окна.** Щелкните вкладку Styles в верхней части окна Dialog Properties и установите параметры стиля, как показано на рисунке.

**5. Укажите дополнительные стили диалогового окна.**



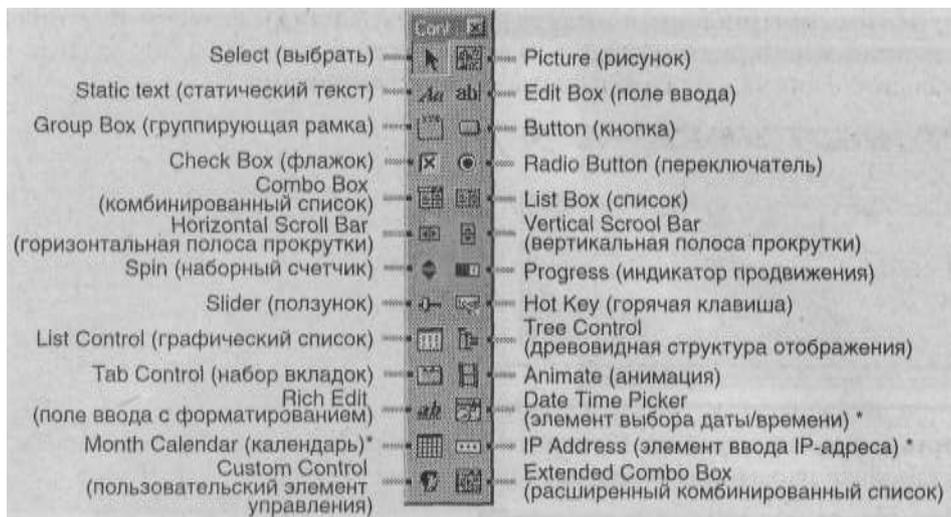
**6. Разместите элементы управления диалогового окна.** Это делается с помощью палитры элементов управления. (Если палитра

не видна, щелкните правой кнопкой мыши любую панель инструментов и выберите из списка пункт Controls.) Перетащите мышью нужные элементы с палитры в создаваемое диалоговое окно, а затем сдвиньте и отмасштабируйте их



так, чтобы они разместились, как показано на рисунке.

**ПРИМЕЧАНИЕ** Редактор диалоговых окон сообщает о положении и размере каждого элемента управления в своей строке состояния. При этом координаты выражаются не в аппаратных, а в специальных «диалоговых единицах» (DLUs). Горизонтальная DLU — это средняя ширина шрифта, используемого в диалоговом окне, деленная на 4; вертикальная DLU — средняя высота этого шрифта, деленная на 8. Ну, а шрифт — это обычно MS Sans Serif размером 8 пунктов.



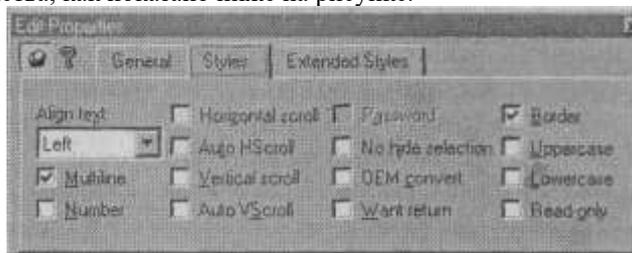
Теперь коротко рассмотрим элементы управления нашего диалогового окна.

- **Статический текст для поля Name (имя).** Этот элемент просто выводит на экран указанные символы и собственно к диалогу с пользователем отношения не имеет. Расположив ограничивающий прямоугольник, введите текст (размеры прямоугольника можно при необходимости изменить). Это единственный элемент управления «статический текст», который здесь упоминается, но на рисунке 6-1 есть и другие подобные элементы; создавайте их по аналогии. Все они имеют один и тот же идентификатор, но это совершенно не важно, поскольку доступ к ним программе не нужен<sup>1</sup>.

- **Поле ввода Name.** Поле ввода — основное средство ввода текста в диалоговых окнах. Смените его идентификатор с *IDCJEDIT1* на *IDC\_NAME*. Остальные свойства оставьте такими, какими они предлагаются по умолчанию. Заметьте: по умолчанию устанавливается *Auto HScroll*, т. е. текст по мере заполнения текстового поля прокручивается справа налево.

- **Поле ввода SS Nbr (social security number — номер карточки социального страхования).** Этот элемент управления идентичен предыдущему. Замените его идентификатор на *IDC\_SSN*. Впоследствии при помощи *Class Wizard* Вы превратите это поле в числовое.

- **Поле ввода Bio (biography — биография).** Это многострочное поле ввода. Присвойте ему идентификатор *IDC\_BIO* и установите его свойства, как показано ниже на рисунке.



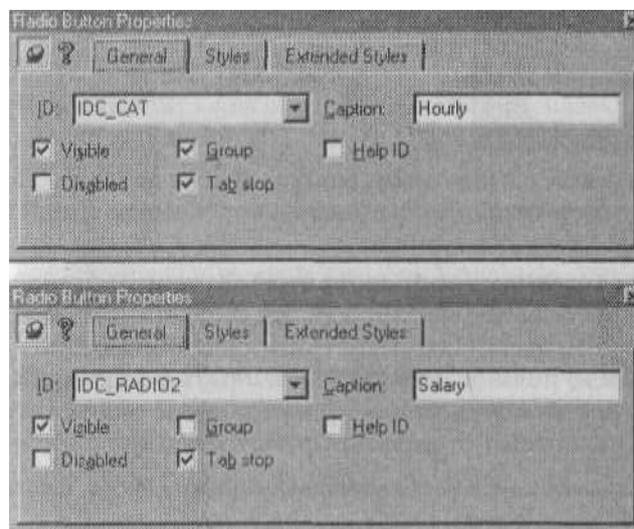
- **Группирующая рамка Category (категория).** Этот элемент управления нужен только для того, чтобы сгруппировать два переключателя. Введите его название: *Category*. Идентификатор, присвоенный по умолчанию, нас устроит.

- **Переключатели Hourly (почасовая оплата) и Salary (оклад).** Разместите эти переключатели в группирующей рамке *Category*. Идентификатор кнопки *Hourly* задайте как *IDC\_CAT*, а остальные свойства укажите в соответствии с рисунками.

Убедитесь, что у обеих кнопок на вкладке *Styles* установлено свойство *Auto* (по умолчанию) и что только у кнопки *Hourly* установлено свойство *Group*. При правильном задании этих свойств *Windows* гарантирует, что пользователь сможет одновременно выбрать только одну из двух кнопок. Группирующая рамка *Category* никак не сказывается на их функционировании.

- **Группирующая рамка Insurance (страховка).** В этом элементе управления размещаются три флажка. Введите название рамки — *Insurance*.

**ПРИМЕЧАНИЕ** Позже, установив порядок обхода элементов в диалоговом окне, Вы добьетесь того, чтобы группирующая рамка *Insurance* следовала за по-



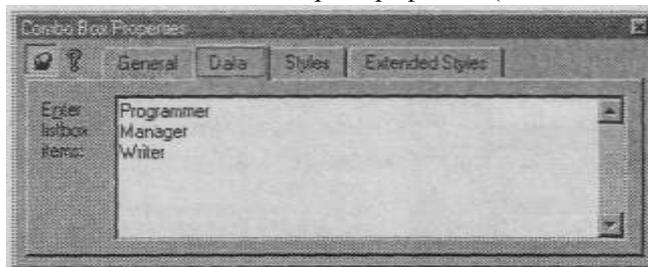
следней кнопкой рамки Category. Только не забудьте пометить у элемента управления Insurance свойство Group, чтобы «завершить» предыдущую группу. Если этого не сделать, особой беды не будет, но, запустив программу под отладчиком, Вы получите предупреждающее сообщение.

- **Флажки Life (жизни), Disability (по инвалидности) и Medical (медицинская).**

Разместите эти элементы управления в группирующей рамке Insurance. Примите свойства, предлагаемые по умолчанию, но замените идентификаторы на *IDC\_LIFE*, *IDC\_DIS* и *IDC\_MED*. В отличие от переключателей, флажки ведут себя независимо — пользователь сможет устанавливать любую их комбинацию.

- **Комбинированный список Skill (профессиональные навыки).** Это первый из трех типов комбинированных списков. Измените идентификатор на *IDC\_SKILL*, а затем щелкните вкладку Styles и установите параметр Type в Simple. Теперь щелкните вкладку Data и введите в поле Enter Listbox Items названия трех профессий (заканчивая каждую строку нажатием сочетания клавиш Ctrl+Enter).

Этот тип комбинированного списка называется *простым* (Simple). В поле ввода, располагающемся сверху, пользователь может вводить любые данные и при помощи мыши или клавиш-стрелок Up/Down выделять элемент в окне списка.



- **Комбинированный список Educ (education — образование).** Замените идентификатор на *IDC\_EDUC*, а прочие параметры оставьте такими, какими они предлагаются по умолчанию. Введите в поле Enter Listbox Items вкладки Data три уровня образования (см. рис. 6-1). В этом комбинированном списке пользователь сможет набрать в поле ввода все, что ему заблагорассудится или, щелкнув стрелку, раскрыть прикрепленное окно списка и выбрать любой элемент (мышью или клавишами-стрелками Up/Down).

**ПРИМЕЧАНИЕ** Чтобы задать размер раскрываемой части комбинированного списка, щелкните стрелку в правой части поля и сместите ее мышью вниз.

- **Список Dept (department — отдел).** Замените идентификатор на *IDC\_DEPT*, а прочие параметры оставьте такими, какими они предлагаются по умолчанию. В этом списке пользователь сможет выбрать лишь один элемент мышью, клавишами-стрелками Up/Down или, введя первый символ нужной строки. Обратите внимание: ввести строку, предлагаемую пользователю по умолчанию, в графическом редакторе нельзя; как это сделать, Вы узнаете чуть позже.

- **Комбинированный список Lang (language — язык).** Замените идентификатор на *IDC\_JLANG*, выберите вкладку Styles и присвойте параметру Type значение Drop List. Введите в поле Enter Listbox Items вкладки Data названия трех языков: English, French и Spanish. В этом комбинированном списке пользователь сможет выбирать элементы только из раскрываемого списка. Для этого он должен будет, щелкнув стрелку, указать нужную строку или ввести ее первую букву и при необходимости уточнить выбор при помощи стрелок.

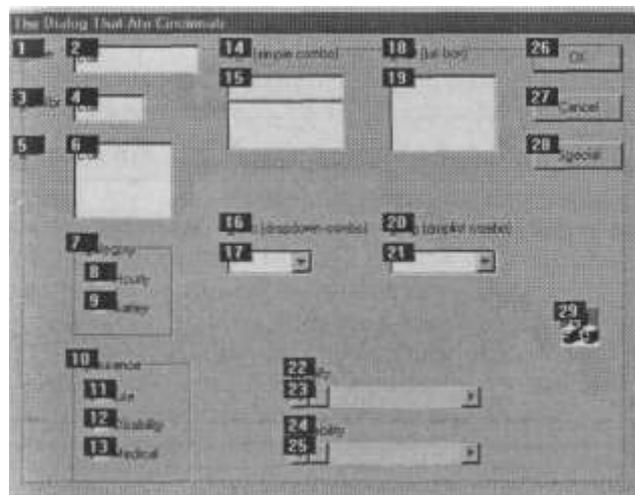
- **Полосы прокрутки Loyalty (лояльность) и Reliability (надежность).** Не путайте элемент управления «полоса прокрутки» с полосами прокрутки, встроенными в окно. Элемент «полоса прокрутки» ведет себя так же, как и остальные элементы управления, — в частности, в период проектирования его можно масштабировать. Разместите горизонтальные полосы прокрутки, отмасштабируйте их (см. рис. 6-1) и присвойте им идентификаторы *IDC\_LOYAL* и *IDC\_RELY*.

- **Кнопки OK, Cancel и Special.** Введите названия кнопок *OK*, *Cancel* и *Special*, затем присвойте кнопке Special идентификатор *IDC\_SPECIAL*

- **Произвольный значок (для примера показывается значок MFC).** В диалоговом окне при помощи элемента управления Picture можно отобразить любой значок или растровое изображение, если они определены в описании ресурсов. Воспользуемся MFC-значком программы, обозначенным как *IDR\_MAINFRAME*. Установите параметр Type как Icon, а параметр Image — в *IDR\_MAINFRAME*. Идентификатор оставьте *IDC\_STATIC*.

**7. Проверьте порядок перехода между элементами управления.** Выберите из меню Layout в редакторе диалоговых окон команду Tab Order. Укажите мышью порядок переключения между элементами управления при нажатии клавиши Tab. Для этого щелкните мышью каждый из элементов управления в таком порядке, как показано на рисунке, а затем нажмите Enter.

**ПРИМЕЧАНИЕ** В текст заголовка элемента управления «статический текст» (например, Name и Skill) можно вставить знак «&». В период выполнения символ, за которым стоит этот знак, будет подчеркнут (см. рис. 6-1). Это позволяет переходить к нужным элементам управления нажатием Alt+«подчеркнутый символ». Так, Alt+N сместит фокус на поле ввода Name, а Alt+K — на комбинированный список Skill. Ясно, что такие символы должны быть уникальны в пределах диалогового окна. Буква K используется для элемента управления Skill потому, что буква S уже задействована в поле ввода SS Nbr.



- **Сохраните файл ресурсов на диске.** Осторожности ради сохраните файл exОба.ге выберите из меню File

команду Save или щелкните одноименную кнопку на панели инструментов. Не закрывайте пока редактор диалоговых окон и оставьте на экране только что созданное диалоговое окно.

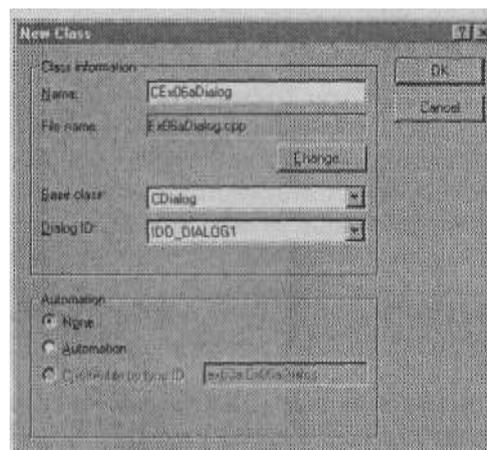
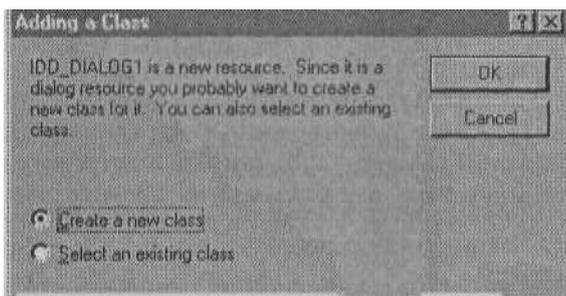
### ClassWizard и класс «диалоговое окно»

Теперь Вы создали диалоговый ресурс, но без соответствующего класса диалогового окна воспользоваться им нельзя. Действия ClassWizard и редактора диалогов при создании класса диалогового окна.

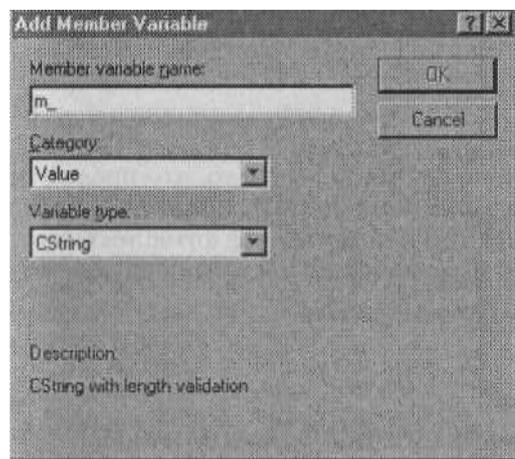
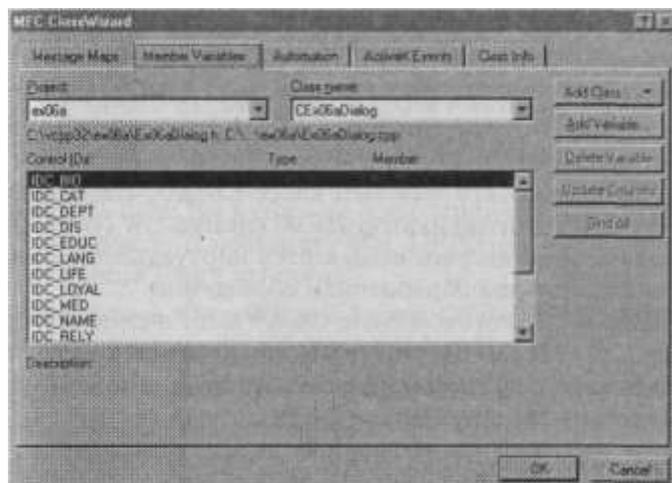
**1. Выберите ClassWizard из меню View в Visual C++ (или нажмите Ctrl+W).** Убедитесь, что в редакторе диалоговых окон по-прежнему выделено только что созданное диалоговое окно `IDD_DIALOG1` и что EX06A — текущий проект Visual C++.

**2. Добавьте класс `CEx06aDialog`.** ClassWizard обнаружит, что Вы создали диалоговый ресурс и что ему еще не сопоставлен C++-класс, после чего вежливо осведомится, не хотите ли Вы создать класс.

Щелкните кнопку ОК, соглашаясь с предложенным по умолчанию выбором Create A Class, и заполните верхнее поле в диалоговом окне Create New Class, как показано ниже.



**3- Добавьте переменные класса `CEx06aDialog`.** После того как ClassWizard сформирует класс `CEx06aDialog`, на экране появится диалоговое окно MFC ClassWizard. Щелкните вкладку Member Variables, и Вы увидите одноименную страничку. Вам надо сопоставить переменные-члены каждому элементу управления диалогового окна. Для этого щелкните идентификатор элемента управления,



а затем — кнопку Add Variable. В результате появится диалоговое окно Add Member Variable. Введите имя переменной-члена и выберите ее тип в соответствии со следующей таблицей. Убедитесь, что Вы набрали имя переменной-члена именно так, как показано в таблице, — здесь важен регистр каждой буквы. Далее щелкните кнопку ОК, чтобы вернуться в диалоговое окно MFC ClassWizard, и повторите только что описанную процедуру применительно к остальным элементам управления.

Идентификатор элемента управления	Переменная-член	Тип
<code>IDD_BIO</code>	<code>m_strBio</code>	<code>CString</code>
<code>IDD_CAT</code>	<code>m_nCat</code>	<code>int</code>
<code>IDD_DEPT</code>	<code>m_strDept</code>	<code>CString</code>
<code>IDD_DIS</code>	<code>m_b!nsDis</code>	<code>BOOL</code>
<code>IDD_EDUC</code>	<code>m_strEduc</code>	<code>CString</code>

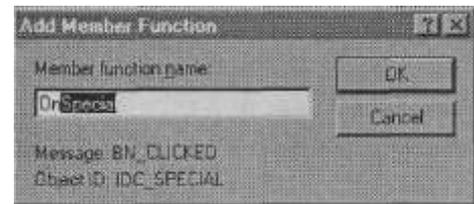
<i>IDC_LANG</i>	<i>m_strLang</i>	<i>CString</i>
<i>IDC_LIFE</i>	<i>m_bInsLife</i>	<i>BOOL</i>
<i>IDC_LOYAL</i>	<i>m_nLoyal</i>	<i>int</i>
<i>JDC_MED</i>	<i>m_bInsMed</i>	<i>BOOL</i>
<i>IDC_NAME</i>	<i>m_strName</i>	<i>CString</i>
<i>IDC_RELY</i>	<i>m_nRely</i>	<i>int</i>
<i>IDC_SKILL</i>	<i>m_strSkill</i>	<i>CString</i>
<i>IDC_SSN</i>	<i>m_nSsn</i>	<i>int</i>

Когда Вы выбираете элементы управления в диалоговом окне MFC ClassWizard, в нижней его части появляются соответствующие поля ввода. Указав переменную *CString*, можно установить ее максимальную длину, а выбрав числовую переменную — задать ее верхний и нижний пределы. Введите для *IDC\_SSN* минимальное значение 0 и максимальное — 999999999. Большинство взаимосвязей между типами элементов управления и типами переменных совершенно прозрачны. Однако уже не столь очевидно, как кнопки-переключатели соотносятся с переменными. Класс *CDialog* сопоставляет целочисленную переменную каждой *группе* переключателей; при этом первой кнопке соответствует значение 0, второй — 1 и т. д.

**4. Добавьте функцию-обработчик сообщений для кнопки Special.** Классу *Ex06aDialog* не требуется много функций-обработчиков сообщений, поскольку большую-часть работы по управлению диалоговым окном выполняет его базовый класс *CDialog* совместно с Windows. Если Вы, например, присвоили идентификатор *ШОК* кнопке ОК (установка ClassWizard по умолчанию), при щелчке этой кнопки вызывается виртуальная *CDialog*-функция *OnOK*. Однако для других кнопок нужны обработчики сообщений. Поэтому щелкните вкладку Message Map. В диалоговом окне ClassWizard в списке Object IDs должна присутствовать запись для *IDC\_SPECIAL*. Щелкните эту запись, а затем дважды - сообщение BN\_CLICKED в списке Messages. ClassWizard присваивает функции-члену имя *OnSpecial* и открывает диалоговое окно Add Member Function.

Здесь можно было бы ввести свое имя функции, но сейчас примите то, что предлагается по умолчанию, и щелкните кнопку ОК. Далее в диалоговом окне MFC ClassWizard щелкните кнопку Edit Code — откроется файл *ex06aDialog.cpp* и появится код функции *OnSpecial*. Замените существующий код оператором *TRACE*, выделенным в листинге полужирным:

```
void CEx06aDialog::OnSpecial()
{
    TRACE("CEx06aDialog::OnSpecial\n");
}
```



**5. Используя ClassWizard, добавьте функцию-обработчик OnInitDialog.** Как Вы скоро увидите, ClassWizard генерирует код, инициализирующий элементы управления диалогового окна. Однако этот DDX-код (Dialog Data Exchange) не будет инициализировать варианты, предлагаемые в списках, поэтому нужно переопределить функцию *CDialog::OnInitDialog*. Хотя *OnInitDialog* — это виртуальная функция-член, ClassWizard сформирует для нее прототип и заготовку, если Вы решите обработать сообщение WM\_INITDIALOG в производном классе «диалоговое окно». Для этого выберите в списке Object IDs класс *CEx06aDialog*, а в списке Messages дважды щелкните сообщение WM\_INITDIALOG. Далее щелкните кнопку Edit Code в диалоговом окне MFC ClassWizard, чтобы отредактировать функцию *OnInitDialog*. Замените существующий код текстом, который выделен полужирным:

```
BOOL CEx06aDialog::OnInitDialog()
{
    // CDialog::OnInitDialog можно вызвать в этой функции только один раз -
    // будьте внимательны
    CListBox* pLB = (CListBox*) GetDlgItem(IDC_DEPT);
    pLB->InsertString(-1, "Канцелярия");
    pLB->InsertString(-1, "Бухгалтерия");
    pLB->InsertString(-1, "Отдел кадров");
    pLB->InsertString(-1, "Безопасность");
    // вызываем после инициализации
    return CDialog::OnInitDialog();
}
```

Для комбинированных списков вместо инициализации в ресурсе при желании можно воспользоваться такой же процедурой.

## Подключение диалогового окна к классу «вид»

Теперь у Вас есть ресурс и код для диалогового окна, но окно не подключено к классу «вид». В большинстве приложений диалоговое окно открывается при выборе какой-либо команды из меню, но пока Вы меню «не проходили». Поэтому для открытия диалогового окна прибегнем к знакомому сообщению WM\_LBUTTONDOW, которое генерируется при щелчке левой кнопки мыши. Ваши действия.

**1. Выберите в ClassWizard класс CEx06aView.** Убедитесь, что EX06A — текущий проект Visual C++.

2. С помощью ClassWizard добавьте функцию-член *OnLButtonDown*. Просто выделите имя класса *CEx06aView*, щелкните идентификатор объекта *CEx06aView*, а затем дважды — сообщение WM\_LBUTTONDOWN.

3. Добавьте код функции *OnLButtonDown* в файле *ex06aView.cpp*. Добавьте в заготовку тела функции код, выделенный полужирным. Большая часть кода состоит из операторов *TRACE*, выводящих значения переменных диалогового окна после того, как пользователь закрыл диалоговое окно. Но главное здесь — вызовы конструктора класса *CEx06aDialog* и функции *DoModal*.

```
void CEx06aView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CEx06aDialog dlg;
    dlg.m_strName = "Шекспир, Уилл";
    dlg.m_nSsn     = 307806636;
    dlg.m_nCat     =1; // 0 = почасовая, 1 = оклад
    dlg.m_strBio   = "Не слишком увлеченный своей работой технический писатель";
    dlg.m_bInsLife = TRUE;
    dlg.m_bInsDis  = FALSE;
    dlg.m_bInsMed  = TRUE;
    dlg.m_strDept  = "Канцелярия";
    dlg.m_strSkill = "Писатель";
    dlg.m_nLang    = 0;
    dlg.m_strEduc  = "Колледж";
    dlg.m_nLoyal   = dlg.m_nRelay = 50;
    int ret = dlg.DoModal();
    TRACE("Возврат из DoModal = %d\n", ret);
    TRACE("name = %s,  ssn = %d,  cat = %d\n",
          dlg.m_strName,  dlg.m_nSsn,  dlg.m_nCat);
    TRACE("dept = %s,  skill = %s,  lang = %d,  educ = %s\n",
          dlg.m_strDept,  dlg.m_strSkill,  dlg.m_nLang,  dlg.m_strEduc);
    TRACE("life = %d,  dis = %d,  med = %d,  bio = %s\n",
          dlg.m_bInsLife,  dlg.m_bInsDis,  dlg.m_bInsMed,  dlg.m_strBio);
    TRACE("loyalty = %d,  reliability = %d\n",
          dlg.m_nLoyal,  dlg.m_nRelay);
}
```

4. Добавьте код в виртуальную функцию *OnDraw* в файле *ex06aView.cpp*. Чтобы предложить пользователю нажать левую кнопку мыши, закодируйте *CEx06aView*-функцию *OnDraw* (ее заготовка сгенерирована мастером AppWizard). Замените существующий код текстом, выделенным полужирным:

```
void CEx06aView::OnDraw(CDC* pDC)
{
    pDC->TextOut(0, 0, "Щелкните здесь левой кнопкой мыши.");
}
```

5. Добавьте в файл *ex06aView.cpp* оператор включения класса «диалоговое окно».

Показанная выше функция *OnLButtonDown* зависит от объявления класса *CEx06aDialog*. Вы должны включить следующий оператор *#include*.

```
#include "ex06aDialog.h"
```

в начало файла с исходным кодом для класса *CEx06aView* (*ex06aView.cpp*) после оператора:

```
#include "ex06aView.h"
```

6. Соберите и протестируйте приложение. Если все сделано правильно, Вы сможете собрать и запустить приложение EX06A из Visual C++. Попробуйте, вводя данные в каждый элемент управления, щелкать кнопку ОК и наблюдать за выводом операторов *TRACE* в окне Debug. Полосы прокрутки пока ничего особенного не делают — о них речь пойдет позже. Обратите внимание также на то, что происходит при нажатии клавиши Enter в момент ввода текстовых данных в каком-либо элементе управления — диалоговое окно немедленно закрывается.

## Усовершенствование программы EX06A

Программа EX06A, обладая массой функциональных возможностей, не требовала от программиста особых усилий. Теперь создадим новую версию с расширенной функциональностью. Избавим EX06A от скверной привычки завершать свою работу при нажатии клавиши Enter и введем поддержку элементов управления «полосы прокрутки».

### Перехват контроля при выходе по *OnOK*

В исходной программе EX06A виртуальная функция *CDialog::OnOK* обрабатывала щелчок кнопки ОК, что запускало процесс обмена данными и процедуру завершения диалогового 1 окна. Как оказалось, клавиша Enter давала тот же результат — может быть, это как раз то, чего Вы хотели, а может быть, и нет. Если пользователь нажмет клавишу Enter, скажем, в поле ввода Хате, его немедленно «выбросит» из диалогового окна. Вряд ли это ему понравится.

Что же происходит? В момент нажатия этой клавиши Windows ищет кнопку, на которой установлен *фокус ввода* (input focus) — на экране он выглядит как пунктирная рамка. Если фокус не установлен ни на одну кнопку, Windows ищет указанную программой или в ресурсе *кнопку по умолчанию* (default pushbutton) — у такой кнопки более толстые границы. Если такой кнопки нет, вызывается виртуальная функция *OnOK* — даже если кнопка ОК отсутству-

ет.

Клавишу Enter можно отключить, просто написав ничего не делающую функцию *CEx06aDialog::OnOK* и добавив код завершения в новую функцию, реагирующую на щелчок кнопки ОК. Последовательность действий такова.

**1. С помощью ClassWizard сопоставьте кнопку IDOK виртуальной функции OnOK.**

В ClassWizard выберите *IDOK* в списке Object IDs для *CEx06aDialog*, а затем дважды щелкните *BN\_CLICKED*. В результате будут сгенерированы прототип и заготовка для *OnOK*.

**2. Используя редактор диалоговых окон, измените идентификатор кнопки ОК.** Выберите кнопку ОК, измените ее идентификатор с *ШОК* на *ЮС\_ОК* и отмените у нее свойство Default Button. Оставьте только функцию *OnOK*.

**3. С помощью ClassWizard создайте функцию-член OnClickedOk.** Эта функция-член класса *CEx06aDialog* играет ключевую роль в обработке сообщения *BN\_CLICKED* от только что переименованной кнопки *IDC\_OK*.

**4. Отредактируйте тело функции OnClickedOk в ex06aDialog.cpp.** Она вызывает функцию *OnOK* базового класса, как это делала исходная функция *CEx06aDialog::OnOK*. Вот ее код:

```
void CExOGaDialog::OnClickedOk()
{
    TRACE("CEx06aDialog::OnClickedOk\n");
    CDialog::OnOK();
}
```

**5. Отредактируйте исходную функцию OnOK в ex06aDialog.cpp.** Эта функция — «пустой» обработчик кнопки с прежним идентификатором *IDOK*. Модифицируйте код следующим образом:

```
void CEx06aDialog::OnOK()
{
    // заглушка для функции OnOK - НЕ вызывайте CDialog::OnOK()
    TRACE("CEx06aDialog::OnOK\n");
}
```

**6. Соберите и протестируйте приложение.** Попробуйте теперь нажать клавишу Enter. Ничего произойти не должно, но в окне Debug появится вывод оператора *TRACE*. Однако щелчок кнопки ОК должен, как и раньше, завершать диалоговое окно.

## Обработка OnCancel

Так же, как нажатие клавиши Enter приводило к вызову *OnOK*, нажатие клавиши Esc инициирует вызов *OnCancel*, в результате чего диалоговое окно завершается с кодом возврата *IDCANCEL* от функции *DoModal*. Программа *EX06A* не предусматривает особой обработки *IDCANCEL*, - поэтому нажатие клавиши Esc (или щелчок кнопки Close) закрывает диалоговое окно. Вы можете обойти этот процесс, применив функцию-заглушку *OnCancel* по аналогии с тем, что уже делалось для кнопки ОК.

## Подключение элементов управления «полосы прокрутки»

Графический редактор позволяет размещать в диалоговом окне элементы управления «полосы прокрутки», а Class Wizard — создавать для них целочисленные переменные-члены. Чтобы полосы прокрутки *Loyalty* и *Reliability* заработали, надо дополнить программу соответствующим кодом.

Эти элементы управления позволяют считывать и записывать текущую позицию движка и границы заданного диапазона. Если Вы установили диапазон, скажем, от 0 до 100, то соответствующая переменная-член со значением 50 поместит движок в центр полосы прокрутки, (Функция *CScrollbar::SetScrollPos* тоже задает позицию движка на полосе прокрутки.) Когда пользователь перемещает движок или щелкает стрелки, полоса прокрутки отправляет в диалоговое окно сообщения *WM\_HSCROLL* и *WM\_VSCROLL*. Обработчики сообщений в диалоговом окне расшифровывают эти сообщения и соответственно изменяют позицию движка на полосе прокрутки.

Особенность элементов управления «полоса прокрутки» в том, что все горизонтальные -: полосы посылают одно сообщение — *WM\_HSCROLL*, а все вертикальные — *WM\_VSCROLL*. А в нашем «навороченном» диалоговом окне две горизонтальные полосы прокрутки, значит, один-единственный обработчик сообщения *WM\_HSCROLL* должен как-то различать, какая из них прислала сообщение.

Добавим в программу *EX06A* логику управления полосами прокрутки.

**1. Добавьте операторы enum, чтобы задать предельные значения диапазона прокрутки.** Включите в файл *ex06aDialog.h*, в самое начало объявления класса, следующие строки:

```
enum { nMin = 0 };
enum { nMax = 100 };
```

**2. Отредактируйте функцию OnInitDialog, чтобы инициализировать границы диапазона прокрутки.** Эта функция должна устанавливать минимальное и максимальное значения диапазона прокрутки так, чтобы переменные-члены *CEx06aDialog* отражали величины, выраженные в процентах. Значение, равное 100, означает «установить движок в крайнюю правую позицию», а нулевое значение — «установить движок в крайнюю левую позицию». Добавьте в файле *ex06aDialog.cpp* в функцию-член *OnInitDialog* класса *CEx06aDialog* такой код:

```
CScrollbar* pSB = (CScrollbar*) GetDlgItem(IDC_LOYAL);
pSB->SetScrollRange(nMin, nMax);
pSB = (CScrollbar*) GetDlgItem(IDC_REL);
pSB->SetScrollRange(nMin, nMax);
```

**3. Используя ClassWizard, добавьте в *CEx06aDialog* обработчик сообщений от полос прокрутки.** Выберите сообщение `WM_HSCROLL`, а затем добавьте функцию-член *OnHScroll*. Введите код, выделенный полужирным: `void CEx06aDialog::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)`

```
{
    int nTemp1, nTemp2;
    nTemp1 = pScrollBar->GetScrollPos();
    switch (nSBCode)
    {
        case SB_THUMBPOSITION:
            pScrollBar->SetScrollPos(nPos);
            break;
        case SB_LINELEFT: // кнопка со стрелкой влево
            nTemp2 = (nMax - nMin) / 10;
            if ((nTemp1 - nTemp2) > nMin)
                nTemp1 -= nTemp2;
            else
                nTemp1 = nMin;
            pScrollBar->SetScrollPos(nTemp1);
            break;
        case SB_LINERIGHT: // кнопка со стрелкой вправо
            nTemp2 = (nMax - nMin) / 10;
            if ((nTemp1 + nTemp2) < nMax)
                nTemp1 += nTemp2;
            else
                nTemp1 = nMax;
            pScrollBar->SetScrollPos(nTemp1);
            break;
    }
}
```

**ПРИМЕЧАНИЕ** Функции, связанные с полосами прокрутки, пользуются 16-разрядными целыми числами и для позиции движка, и для границ диапазона прокрутки.

**4. Соберите и протестируйте приложение.** Вновь соберите и запустите программу ЕХОБА. Ну как, полосы прокрутки на этот раз работают? Движки должны перемещаться, когда Вы щелкаете стрелки на полосах прокрутки, а также подчиняться перетаскиванию мышью. (Обратите внимание: пока в программу не включена логика, позволяющая реагировать на щелчок пользователем самой полосы прокрутки.)

## Доступ к элементам управления: *CWnd*-указатели и идентификаторы

Размечая диалоговый ресурс в графическом редакторе, Вы определяете элементы управления при помощи идентификаторов, таких, как *IDC\_SSN*. Однако в программном коде бывает необходим доступ к стоящему за элементом управления оконному объекту. Поэтому в MFC-библиотеке предусмотрена функция *CWnd::GetDlgItem*, преобразующая идентификатор в *CWnd*-указатель. Вы уже видели такое преобразование в функциях-членах *OnInitDialog* и *OnClickedOk* класса *CEx06aDialog*. Каркас приложений «создавал» этот *CWnd*-указатель, потому что там не было вызова конструктора для объектов управления. Этот указатель временный, и сохранять его нельзя.

**СОВЕТ** Чтобы преобразовать *CWnd*-указатель в идентификатор элемента управления, воспользуйтесь функцией-членом *GetDlgItemID* класса *CWnd*.

Выбор фона диалогового окна и цвета элементов управления

Чтобы изменить фон отдельных диалоговых окон или конкретных элементов управления в каком-либо диалоговом окне, придется потрудиться. Каждый элемент управления, прежде чем появиться на экране, посылает родительскому диалоговому окну сообщение `WM_CTLCOLOR`. Такое же сообщение отправляется и самому диалоговому окну. Если написать в производном

классе диалогового окна обработчик этого сообщения, можно установить цвет текста и его фон, а также выбрать кисть для заполнения нетекстовой области элемента управления или диалогового окна.

Вот пример функции *OnCtlColor*, задающей желтый фон для всех полей ввода и красный фон для диалогового окна. Переменные *m\_hYellowBrush* и *m\_hRedBrush* — это переменные-члены типа *HBRUSH*, которые инициализируются в *OnInitDialog*. Параметр *nCtrlColor* определяет тип элемента управления, *apWnd* — конкретный элемент управления. Чтобы установить цвет отдельного поля ввода, нужно преобразовать *pWnd* в идентификатор дочернего окна и проверить его.

```
HBRUSH CMyDialog::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtrlColor)
{
    if (nCtrlColor == CTLCOLOR_EDIT)
    {
        pDC->SetBkColor( RGB(255,255,0) ); //желтый
    }
}
```

```

    return m_hYellowBrush;
}
if (nCtlColor == CTLCOLOR_DLG)
{
    pDC->SetBkColor(RGB(255,0,0)); // красный
    return m_hRedBrush;
}
return CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
}

```

## Вывод графики в диалоговом окне

Графику можно выводить прямо в клиентскую область диалогового окна, но наклейки на диалоговые элементы не произойдет, только если рисовать внутри окна какого-либо элемента управления. Если Вы хотите вывести лишь текст, создайте с помощью графического редактора пустой статический элемент управления (пустую метку) с уникальным идентификатором, а затем, чтобы поместить текст в элемент управления, вызовите функцию *CWnd::SetDlgItemText* в функции-члене диалогового окна, например, в *OnInitDialog*.

Вывод собственно графики сложнее. Вы должны с помощью ClassWizard добавить к диалоговому окну функцию-член *OnPaint*, которая преобразует идентификатор статического элемента управления в *CWnd*-указатель, и получить его контекст устройства. Весь «фокус» заключается в том, чтобы рисовать в поле элемента управления и в то же время не дать Windows впоследствии перезаписать вашу работу. Это достигается за счет последовательных вызовов *Invalidate/UpdateWindow*. Взгляните на программный код функции *OnPaint*, которая рисует в статическом элементе управления маленький черный квадрат:

```

void CMyDialog::OnPaint()
{
    CWnd* pWnd=GetDlgItem(IDC_STATIC1); // IDC_STATIC1 - идентификатор статического элемента управления
    CDC* pControlDC = pWnd->GetDC();
    pWnd->Invalidate();
    pWnd->UpdateWindow();
    pControlDC->SelectStockObject(BLACK_BRUSH);
    pControlDC->Rectangle(0,0,10,10); // маркер в виде черного квадратика
    pWnd->ReleaseDC(pControlDC);
}

```

Как и в случае с любым окном, функция *OnPaint* диалогового окна вызывается, только если часть окна объявлена недействительной. *OnPaint* можно вызвать принудительно из другой функции-члена объекта диалогового окна, применив оператор: *Invalidate()*;

# ПОСТРОЕНИЕ ИНТЕРФЕЙСА К БАЗЕ ДАННЫХ

---



## ЦЕЛЬ РАБОТЫ

Приобретение практических навыков построения интерфейса к базе данных MS Access на Visual C++ с использованием технологий ODBC или DAO.



## ЗАДАНИЕ НА РАБОТУ

Разработать и отладить программу, предоставляющий следующий интерфейс к базе данных MS Access: просмотр имеющихся записей; добавление записей; удаление записей; отбор записей по критерию.



## ВАРИАНТЫ ЗАДАНИЯ

1. Структура таблицы: автор; название книги; год издания; количество экземпляров. Вывести количество книг заданного года издания.
2. Структура таблицы: номер избирательного участка; адрес; количество избирателей, принявших участие в голосовании. Вывести участки, на которых процент проголосовавших не менее заданной величины.
3. Структура таблицы: название компании; дата; объем продаж акций. Определить компанию, имеющую максимальный объем продаж указанный месяц.
4. Структура таблицы: название товара; дата; объем продаж. Определить товар, наиболее продавшийся в указанном месяце.
5. Структура таблицы: кафедра; количество сотрудников; объем хоздоговорных работ. Вывести кафедры, у которых удельный объема работ не менее заданной величины.
6. Структура таблицы: район; выполнение плана; техническая оснащенность. Определить районы, в которых выполнение плана не менее заданной величины.
7. Структура таблицы: шифр детали; вес детали; стоимость детали; количество. Указать общее количество деталей стоимостью до заданной суммы.
8. Структура таблицы: фамилия преподавателя; возраст; должность; количество публикаций. Указать преподавателей, которые имеют не менее заданного числа публикаций.
9. Структура таблицы: фамилия; должность; зарплата. Указать фамилии инженеров с окладом более заданного числа.
10. Структура таблицы: название клуба; количество членов; средний возраст. Определить клубы с численностью более заданного числа человек.
11. Структура таблицы: тема работы; исполнитель; затраченное время. Вывести темы, исполнители которых затратили суммарное время не более указанного.
12. Структура таблицы: название компании; дата; стоимость акций. Определить компанию, имеющую минимальную среднюю стоимость акции за указанный месяц.

13. Структура таблицы: шифр изделия; шифр комплектующего; цена единицы комплектующего; количество данных комплектующих на одно изделие. Указать изделия с общей стоимостью, находящейся в заданных пределах.
14. Информация о магнитных дисках представлена в виде таблицы следующей структуры: имя файла; пользователь; последняя дата обновления файла. Определить пользователя с наиболее «старой» информацией.
15. Структура таблицы: ФИО продавца; дата; объем продаж. Определить продавца, продавшего за указанный месяц на наименьшую сумму.
16. Задания на ЭВМ содержат информацию в виде таблицы следующей структуры: пользователь; объем памяти; время выполнения. Определить программы, которые требуют более заданной продолжительности процессорного времени и более заданного объема памяти.
17. Структура таблицы: фамилия вкладчика; номер счета; сумма вклада. Определить вкладчиков с суммой вклада больше указанной.
18. Сведения о студентах хранятся в виде записей следующей структуры: фамилия; группа; номер зачетной книжки; средняя оценка за сессию. Определить количество отличников в заданной группе.
19. Сведения о работе цехов представлены записями, содержащими: название цеха; плановое задание; фактическое выполнение плана. Определить цеха, выполнившие план не менее чем на заданную величину.
20. Определить наиболее экономичную ЭВМ по критерию стоимость/быстродействие, если каждая модель характеризуется записью следующей структуры: название ЭВМ; объем оперативной памяти; быстродействие; стоимость.
21. Меню в столовой представлено записями вида: название блюда; калорийность; цена. Вывести сведения о блюдах в заданном ценовом диапазоне.
22. Структура таблицы: название компании; дата; стоимость акций. Определить среднюю стоимость акций на указанную дату.
23. Структура таблицы: номер школы; количество классов; общее число учеников. Определить школы в которых среднее количество учеников в одном классе не менее заданной величины.
24. Структура таблицы: шифр изделия; количество; цена. Определить шифр изделия с общей стоимостью большей указанной величины.
25. Структура таблицы: номер телефона; дата разговора; длительность разговора. Определить общую стоимость переговоров в заданном месяце по указанному номеру.
26. Каждая модель ЭВМ характеризуется записью следующей структуры: название; объем оперативной памяти; быстродействие; стоимость. Выбрать ЭВМ, у которой быстродействие выше заданного числа и объем памяти больше заданного числа.
27. Структура таблицы: название компании; дата; стоимость акций. Определить компанию, имеющую максимальную среднюю стоимость акции за указанный месяц.
28. Структура таблицы: ФИО продавца; дата; объем продаж. Определить продавца, продавшего за указанный месяц на наибольшую сумму.

# ЭКСПОРТ ИНФОРМАЦИИ ИЗ БАЗЫ ДАННЫХ В MS EXCEL НА VISUAL C++

---



## ЦЕЛЬ РАБОТЫ

Приобретение практических навыков построения интерфейса из базе данных MS Access к MS Excel на Visual C++ с использованием технологий COM.



## ЗАДАНИЕ НА РАБОТУ

Разработать и отладить программу, экспортирующую информацию из базы данных MS Access в MS Excel с использованием технологий COM. Структура базы приведена в индивидуальном задании.



## ВАРИАНТЫ ЗАДАНИЯ

1. Структура таблицы: автор; название книги; год издания; количество экземпляров. Вывести количество книг заданного года издания.
2. Структура таблицы: номер избирательного участка; адрес; количество избирателей, принявших участие в голосовании. Вывести участки, на которых процент проголосовавших не менее заданной величины.
3. Структура таблицы: название компании; дата; объем продаж акций. Определить компанию, имеющую максимальный объем продаж указанный месяц.
4. Структура таблицы: название товара; дата; объем продаж. Определить товар, наиболее продавшийся в указанном месяце.
5. Структура таблицы: кафедра; количество сотрудников; объем хоздоговорных работ. Вывести кафедры, у которых удельный объема работ не менее заданной величины.
6. Структура таблицы: район; выполнение плана; техническая оснащенность. Определить районы, в которых выполнение плана не менее заданной величины.
7. Структура таблицы: шифр детали; вес детали; стоимость детали; количество. Указать общее количество деталей стоимостью до заданной суммы.
8. Структура таблицы: фамилия преподавателя; возраст; должность; количество публикаций. Указать преподавателей, которые имеют не менее заданного числа публикаций.
9. Структура таблицы: фамилия; должность; зарплата. Указать фамилии инженеров с окладом более заданного числа.
10. Структура таблицы: название клуба; количество членов; средний возраст. Определить клубы с численностью более заданного числа человек.
11. Структура таблицы: тема работы; исполнитель; затраченное время. Вывести темы, исполнители которых затратили суммарное время не более указанного.

12. Структура таблицы: название компании; дата; стоимость акций. Определить компанию, имеющую минимальную среднюю стоимость акции за указанный месяц.
13. Структура таблицы: шифр изделия; шифр комплектующего; цена единицы комплектующего; количество данных комплектующих на одно изделие. Указать изделия с общей стоимостью, находящейся в заданных пределах.
14. Информация о магнитных дисках представлена в виде таблицы следующей структуры: имя файла; пользователь; последняя дата обновления файла. Определить пользователя с наиболее «старой» информацией.
15. Структура таблицы: ФИО продавца; дата; объем продаж. Определить продавца, продавшего за указанный месяц на наименьшую сумму.
16. Задания на ЭВМ содержат информацию в виде таблицы следующей структуры: пользователь; объем памяти; время выполнения. Определить программы, которые требуют более заданной продолжительности процессорного времени и более заданного объема памяти.
17. Структура таблицы: фамилия вкладчика; номер счета; сумма вклада. Определить вкладчиков с суммой вклада больше указанной.
18. Сведения о студентах хранятся в виде записей следующей структуры: фамилия; группа; номер зачетной книжки; средняя оценка за сессию. Определить количество отличников в заданной группе.
19. Сведения о работе цехов представлены записями, содержащими: название цеха; плановое задание; фактическое выполнение плана. Определить цеха, выполнившие план не менее чем на заданную величину.
20. Определить наиболее экономичную ЭВМ по критерию стоимость/быстродействие, если каждая модель характеризуется записью следующей структуры: название ЭВМ; объем оперативной памяти; быстродействие; стоимость.
21. Меню в столовой представлено записями вида: название блюда; калорийность; цена. Вывести сведения о блюдах в заданном ценовом диапазоне.
22. Структура таблицы: название компании; дата; стоимость акций. Определить среднюю стоимость акций на указанную дату.
23. Структура таблицы: номер школы; количество классов; общее число учеников. Определить школы, в которых среднее количество учеников в одном классе не менее заданной величины.
24. Структура таблицы: шифр изделия; количество; цена. Определить шифр изделия с общей стоимостью большей указанной величины.
25. Структура таблицы: номер телефона; дата разговора; длительность разговора. Определить общую стоимость переговоров в заданном месяце по указанному номеру.
26. Каждая модель ЭВМ характеризуется записью следующей структуры: название; объем оперативной памяти; быстродействие; стоимость. Выбрать ЭВМ у которой быстродействие выше заданного числа и объем памяти больше заданного числа.
27. Структура таблицы: название компании; дата; стоимость акций. Определить компанию, имеющую максимальную среднюю стоимость акции за указанный месяц.

28. Структура таблицы: ФИО продавца; дата; объем продаж. Определить продавца, продавшего за указанный месяц на наибольшую сумму.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

### Основная литература

1. Хорев, П.Б. Технологии объектно-ориентированного программирования: Учеб. пособие для вузов / П.Б. Хорев.— М. : Академия, 2004 .— 448с.

### Дополнительная литература

1. Буч. Язык UML : Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон; пер.с англ. Мухин Н. — 2-е изд. — М. : ДМК Пресс: Академия Айти, 2007 .— 496с.
2. Круглински. Программирование на Microsoft Visual C++6.0 : пер.с англ. / Д.Д. Круглински, С. Уингоу, Д. Шеферд .— СПб.[и др.] : Русская ред., 2002 .— 864с. : ил. + 1 CD.
3. Microsoft SQL Server 2005 Analysis Services.OLAP и многомерный анализ данных : наиболее полное руководство / А.Б. Бергер [и др.]; под общ. ред. А.Б. Бергера, И.В. Горбач.— СПб. : БХВ-Петербург, 2007 .— 928с.
4. Рофэйл, Э. COM и COM+ : Полное руководство; Пер.с англ. / Э.Рофэйл, Я.Шохауд .— Киев : ВЕК+: НТИ; М.: Энтроп, 2000 .— 560с.
5. Понамарев, В.А. Программирование на C++/C# в Visual Studio.NET 2003.— СПб.: БХВ-Петербург, 2004 .— 352с.