А.И. ДАНИЛИН

ПРОГРАММИРОВАНИЕ В СРЕДЕ MICROSOFT VISUAL C++ С ИСПОЛЬЗОВАНИЕМ MFC

ДИАЛОГОВОЕ ПРИЛОЖЕНИЕ

2007

СГАУ

САМАРА

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА»

А.И. ДАНИЛИН

ПРОГРАММИРОВАНИЕ В СРЕДЕ MICROSOFT VISUAL C++ С ИСПОЛЬЗОВАНИЕМ MFC

ДИАЛОГОВОЕ ПРИЛОЖЕНИЕ

Утверждено Редакционно-издательским советом университета в качестве учебного пособия

САМАРА
Издательство СГАУ
2007

УДК 004.42(075)
ББК 32.973.26-018.2
Д 182

Инновационная образовательная программа "Развитие центра компетенции и подготовка специалистов мирового уровня в области аэрокосмических и геоинформационных технологий"

Рецензенты: д-р техн. наук, проф. Куренков В.И.
канд. техн. наук Харитонов С.И.

Данилин А.И.

Д 182 **Программирование в среде Microsoft Visual C++ с использованием MFC. Диалоговое приложение**: учеб. пособие / *А.И.Данилин* – Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2007. –116 с. : ил.

ISBN 978-5-7883-0533-2

Рассмотрены базовые идеи и методы создания диалоговых приложений в среде Visual C++ операционной системы Windows. На примерах показаны все этапы и приемы создания вполне профессиональных прикладных программ, использующих диалоговые окна, графическое и числовое представление результатов.

Учебное пособие предназначено для студентов факультета летательных аппаратов, обучающихся по специальностям «Автоматизированное управление жизненным циклом продукции» и «Компьютерные системы управления качеством для автоматизированных производств».

УДК 004.42(075)
ББК 32.973.26-018.2

ISBN 978-5-7883-0533-2

© Данилин А.И., 2007
© Самарский государственный аэрокосмический университет, 2007

Введение

Хотим мы того или нет, но операционная система Windows завоевала мир и практически все, кто имеет компьютер, работают в ней. Постепенно, но очень медленно, набирают силу программные продукты с открытым кодом (начало было положено операционными системами на базе системы Unix), однако, если программист хочет продать свой продукт, то этот продукт должен, среди прочего, работать и в среде Windows.

Какая фирма может создавать самые надежные, самые богатые возможностями, самые эффективные по конечному программному модулю инструментальные средства для программирования в какой-либо операционной среде?!? Ответ очевиден. Это фирма, разработавшая эту операционную систему, владеющая идеологией ее функционирования, а также исходным кодом всех управляющих модулей системы. Вы уже догадались, что это фирма Microsoft.

Что касается языка C++, то мы не будем приводить все преимущества этого языка в силу ограниченного объема данного пособия. Отметим лишь, что исполняемые программы, полученные в результате трансляции исходного кода (текста на C++) в машинные команды, по длине получаются соизмеримыми, а иногда даже короче, чем исходный код. Отсюда следует, что C++ - это реальный *язык программирования*, а не алгоритмический язык. И этим сказано очень многое.

И, наконец, последний аргумент. Microsoft Visual C++ имеет обширную библиотеку классов MFC (**M**icrosoft **F**oundation **C**lasses – базовые классы Microsoft), содержащую более 30.000 строк исходного кода и обладающую богатыми возможностями. Из-за этого он достаточно сложен в освоении, поскольку одного и того же результата можно добиться сотней разных способов. Однако на сегодняшний день самые высокооплачиваемые программисты – это программисты Visual C++.

1. Что такое Visual C++?

Visual C++ является «англоговорящим» продуктом и пока нет его русифицированной версии. В интернете можно найти русифицированную документацию по классам и функциям, но диалоги обще-

ния остаются на английском языке. Поэтому вспоминайте или учите заново «компьютерный» английский.

Visual C++ включает следующие основные компоненты.

1. Редактор (Editor), используемый для ввода, просмотра и анализа программистом исходного кода C++ (раньше текст программы, записанный на каком-либо языке программирования, назывался ***исходным текстом***; сейчас общеупотребительным является термин ***исходный код*** или просто ***код***).
2. Компилятор (Compiler), выполняющий трансляцию (перевод) исходного кода C++ в объектный код. ***Объектный код*** – это текст исходной программы, записанный в элементарных машинных командах, но без подключения библиотечных программных модулей, необходимых для ее выполнения.
3. Компоновщик (Linker), создающий выполняемые файлы за счет объединения объектного кода и библиотечных модулей. Этот процесс называют ***сборкой*** программы.
4. Библиотеки (Libraries), необходимые и применяемые для предварительно написанных программных модулей. Одна из наиболее важных библиотек включает Microsoft Foundation Classes (Базовые классы Microsoft, или MFC), используемые при написании программ, работающих под управлением Microsoft Windows. Помимо этого, стандартная библиотека C++ поддерживает операции ввода-вывода, оконный интерфейс (***Интерфейс*** – способ взаимодействия) и другие стандартные возможности языка C++.
5. Другие инструментальные средства, включая AppWizard (Мастер Приложений), ClassWizard (Мастер Классов), Resource Editor (Редактор ресурсов), Debugger (Отладчик) и др. Все необходимые инструментальные средства будут рассмотрены в последующих главах и Вы будете работать с ними.

Все описания и примеры данного пособия ориентированы на пакет **Microsoft Visual C++ версии 6.0**. На наш взгляд, эта версия наиболее надежна, наиболее типична и наиболее прозрачна. На июль 2007 г. доступна версия 8.0, однако главное отличие ее заключено только в «улучшении» сервиса и синхронизации работы нескольких программистов над единым проектом.

При запуске Visual C++ на короткое время появляется заставка. После ее исчезновения отображается рабочий экран Visual C++, выдержанный в серой цветовой гамме, как показано на рис 1. Намек на цвет появляется только в окне TipWizard (Мастер советов), всплывающем при каждом запуске Visual C++. Если нет необходимости в ежедневном расширении своих познаний в английском языке и в возможностях Visual C++, то TipWizard можно отключить, сбросив флажок Show tip at startup (Показывать совет при запуске).

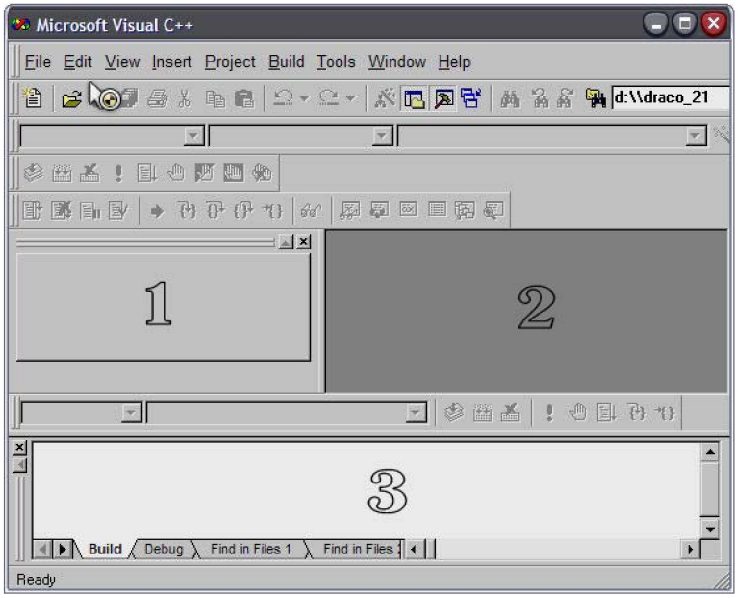


Рис. 1. Главное рабочее окно Visual C++

Рабочий экран Visual C++ включает три окна.

1. Окно Project Workspace (Рабочая область проекта) появляется с левого края экрана. Проектом в данном случае называется ***проект программы***, над которой работает программист. Это окно предназначено для показа всех функций и переменных, входящих в проект программы, разбивки на классы, для быстрого пе-

рехода к нужному коду программного модуля, ресурсу, файлу или к описанию переменной.

2. Окно Editor (Редактор) появляется справа от окна Project Workspace. Это окно используется для ручного ввода, чтения глазами и умственной проверки исходного кода.
3. Окно Output (Вывод) появляется в нижней части и на всю ширину экрана. В этом окне отображаются сообщения о ходе выполнения программы, о возникающих ошибках и последствиях, относящиеся к выполняемым командам. Например, сообщения об ошибках компиляции будут появляться в окне Output при попытке откомпилировать программу, содержащую ошибки.

На самом деле в Visual C++ ничего не происходит до тех пор, пока не создается проект, потому пришло время для выполнения этого шага. Вместо рассмотрения бесполезного примера, давайте создадим реальную программу на языке Visual C++, которая будет выполнять хоть что-нибудь полезное. Здесь не будет рассматриваться некая скучная программа типа "Здравствуй, мир!". Вместо этого мы создадим программу табулирования функции, которая для определенного интервала некоторой функции будет выводить на экран график этой функции и таблицу значений. Границы интервала, шаг табулирования вводит пользователь и он же управляет видом и содержанием выдаваемой информации. Кроме того, программа должна проверять данные на достоверность и быть выполнена в стиле диалогового приложения Windows с информацией об авторе и назначении программы.

2. Проект программы: предварительное обсуждение

При создании нового проекта в Visual C++ нужно начать с выбора команды **File -> New** (Файл -> Новый) из главного меню. В появившемся диалоговом окне выберите закладку **Projects** (Проекты). ***Закладка*** – это выделенный «язычок» с текстом обычно в верхней или нижней части диалогового окна. Закладки будут часто встречаться в дальнейшем, поэтому сразу откладывайте в памяти, что это такое.

К сожалению, в Visual C++ имеется обширный набор различных типов проектов. Ситуация напоминает проблему выбора незнакомой

мелодии из музыкального каталога. Подсказок нет и единственным способом является положиться на собственное везение. Но сейчас Вы читаете данное пособие и поэтому найдите пункт **MFC AppWizard (exe)** и щелкните по нему левой кнопкой мыши.

В дополнение к выбору типа проекта диалоговое окно **New** также предлагает озаглавить проект и определить место его размещения. Введите имя нашего проекта **TabFun** в текстовое поле **Project Name** (Имя проекта). После ввода имени проекта оно добавляется в конец текстового поля **Location** (Размещение), находящегося под полем Project Name.

Выбираемое имя проекта выполняет несколько задач. Во-первых, оно является именем для Вашей выполняемой программы, полученной после компиляции и сборки проекта (подключения библиотечных программ). Во-вторых, оно используется как имя подкаталога (папки), который называется рабочей областью (**workspace**). Это то место, где будут автоматически создаваться файлы проекта. В-третьих, при создании проекта Visual C++ это имя используется для образования имен классов приложения.

По умолчанию при создании нового проекта Visual C++ создает новую рабочую область. При необходимости можно щелкнуть на кнопке с тремя точками, находящейся около текстового поля Location, для выбора местоположения проекта.

ВНИМАНИЕ: При выполнении заданий по программированию в каждом компьютерном классе существуют определенные правила для размещения персональных данных пользователей.

Поэтому текст в поле Location
не может быть произвольным.

Для нашего проекта введите имя **TabFun** в текстовом поле Project Name и используйте значения по умолчанию для всех других опций. Экран должен принять вид, подобный показанному на рис. 2.

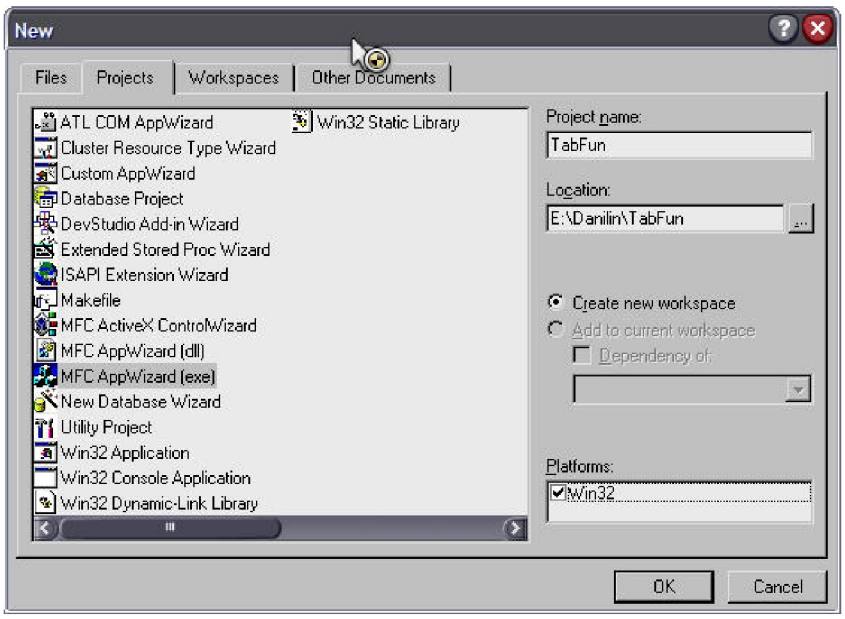


Рис. 2. Тип проекта

По завершении этой работы щелкните на кнопке OK для начала выполнения Visual C++ AppWizard (Мастера приложения).

3. Работаем с AppWizard

Шаг 1. Выбор типа приложения.

Как видно из рис. 3 AppWizard (Мастер приложения) сначала спрашивает о том, какой тип приложения необходимо создать.

Предлагается одна из следующих возможностей.

☞ Single document (Одиночный документ) – этот тип приложения реализует архитектуру «документ-представление» и позволяет открывать и работать только с одним документом в каждый момент времени. Представление документа автоматически помещается в основном окне Вашего приложения, не оставляя места для других документов.

☞ Multiple documents (Множественные документы) этот тип приложения также реализует архитектуру «документ-представление» и позволяет открывать и работать с несколькими документами в каждый момент времени, причем каждый документ может иметь несколько представлений – текст, диаграмма и др. Об архитектуре «документ-представление» речь пойдет в другом учебном пособии.

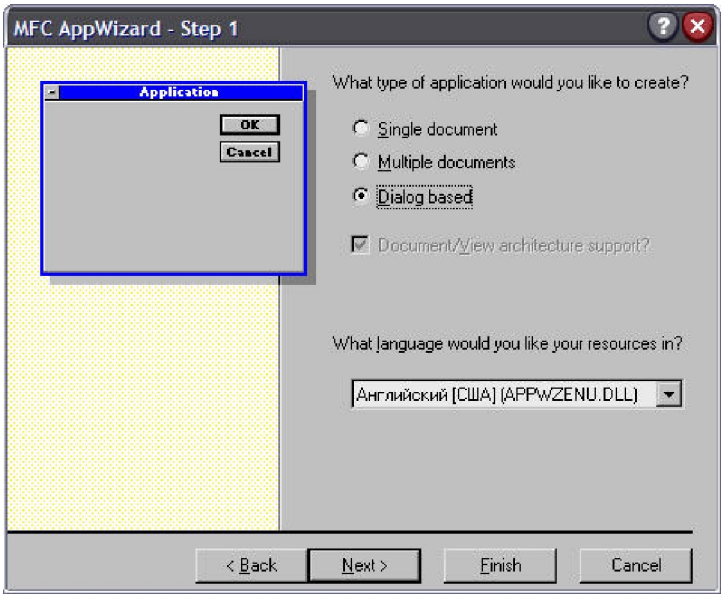


Рис. 3. Выбор типа приложения

☞ **Dialog based** (Основанный на диалоге) – этот тип приложения использует диалоговое окно в качестве главного. Visual C++ обеспечивает использование редактора диалоговых окон для разработки внешнего вида Вашего приложения. Приложения этого типа используют относительно небольшое количество классов, поэтому в них легко разобраться и именно этот тип приложения мы будем применять для нашего проекта.

Приведите окно в соответствие с видом, показанным на рис. 3, и щелкните по кнопке **Next** (Следующий), чтобы перейти к шагу 2.

Шаг 2. Внешний вид программы.

Во втором окне необходимо выбрать, будут ли добавлены несколько различных возможностей. Рассмотрим список свойств и возможностей, которым должна удовлетворять программа TabFun.

- Выберите флажок **About Box** (Окно о программе) для добавления экрана персонализации программы. Использование редактора ресурсов Visual C++ начинается с придания профессионального вида диалоговому окну About программы TabFun.
- Мы не будем использовать контекстно-зависимую справку – пункт **Context sensitive Help**. Она не выбирается по умолчанию, так что можно оставить флажок в том положении, в котором он находится.
- Если нужно использовать трехмерные элементы управления, оставьте флажок **3D controls** включенным.
- Если нет необходимости в использовании **OLE Automation** (механизма, позволяющего одним приложениям управлять и настраивать объекты других приложений), **ActiveX** (управляющих элементов, разработанных Microsoft) или **Windows sockets** (сетевых элементов Windows), оставьте соответствующие флажки неустановленными.
- В нижней части окна можно ввести заголовок для главного диалогового окна нашей программы (Please, enter a title for your dialog – Пожалуйста, введите заголовок для Вашего диалога). В этом окне введите «TabFun - табулирование функции».

  Приведите окно в соответствии с видом, показанным на рис. 4, и щелкните по кнопке **Next** (Следующий), чтобы перейти к шагу 3.

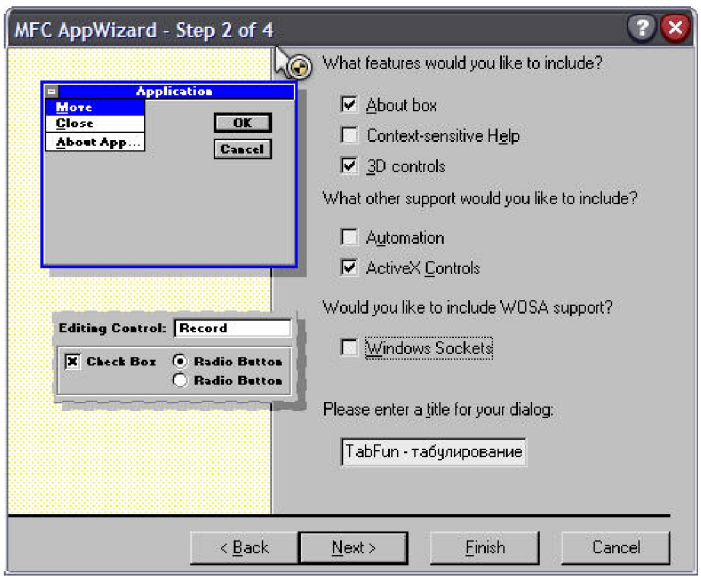


Рис. 4. Внешний вид программы

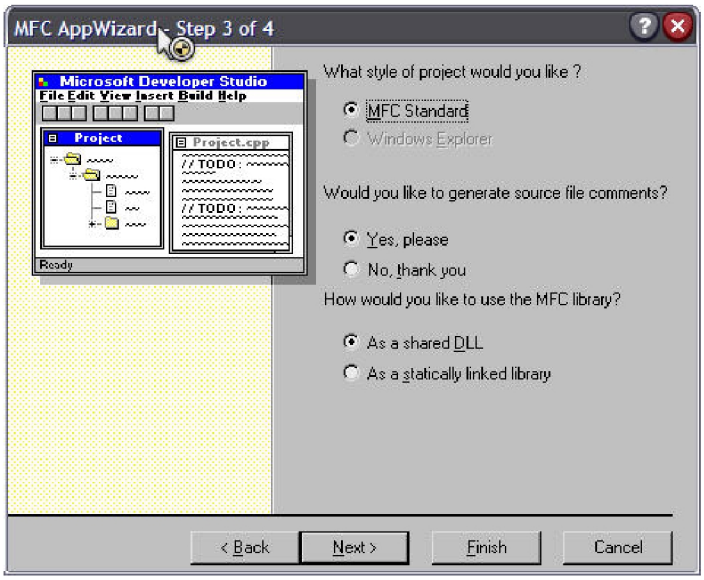


Рис. 5. Стили окон, комментарии и библиотеки

Шаг 3. Стили окон, комментарии и библиотеки.

На этом шаге оставьте предлагаемые положения флажков без изменения, в результате чего исходный код нашей программы TabFun будет представлен в стандарте MFC (флажок **MFC Standard**), в код будут добавлены комментарии **(Would you like to generate the comments ? -> Yes, please)** и программные модули стандартной библиотеки будут динамически подгружаться по мере надобности **(How would you like to use the MFC library ? -> As a shared DLL)**. Альтернативами являются отсутствие комментариев и полное присоединение библиотеки к исполняемому модулю. После того как окно примет вид, показанный на рис. 5, щелкните по кнопке **Next** (Следующий), чтобы перейти к последнему шагу 4.

Шаг 4. Информация о приложении.

Это окно дает Вам последнюю возможность что-либо изменить в проекте, поскольку дает обзор классов, используемых приложением, и отменить выборы, выполненные на предыдущих этапах. Вид окна показан на рис. 6. Обратите внимание на имена классов, сгенерированные AppWizard, и сравните их с именем проекта.

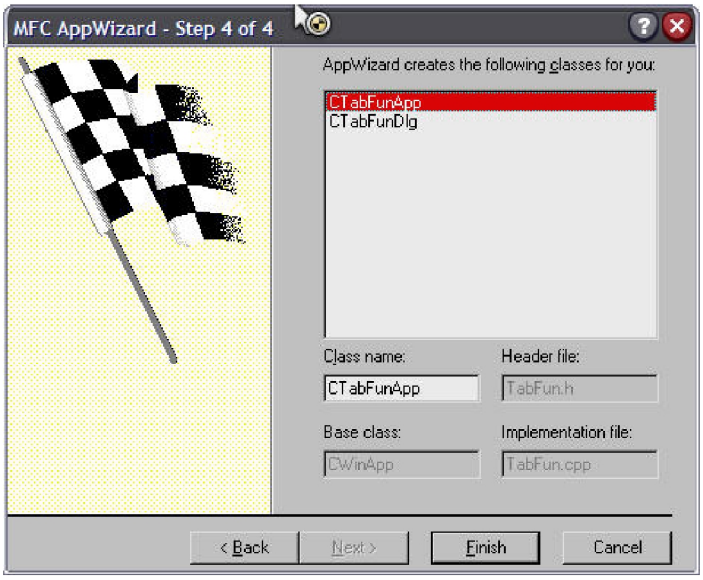


Рис. 6. Информация о приложении

После нажатия кнопки **Finish** (о том, что финиш достигнут, свидетельствует клетчатый флаг, и только здесь можно нажимать кнопку Finish !!!) выдается информационное окно с перечислением типа приложения, созданных классов и файлов, а также свойств (Features) приложения. Нажмите OK и Вы попадаете в каркас программы, сформированный Visual C++. Это вполне работоспособное приложение, в чем можно убедиться, нажав на кнопку с бордовым восклицательным знаком и запустив тем самым процесс компиляции, сборки и выполнения программы, но, увы, наша программа пока не выполняет никакой полезной работы. Закройте ее и придайте окну каркаса вид, показанный на рис. 7 путем нажатия на знаки + перед пиктограммами папок Dialog, Icon, String Table и Version закладки Resource View.

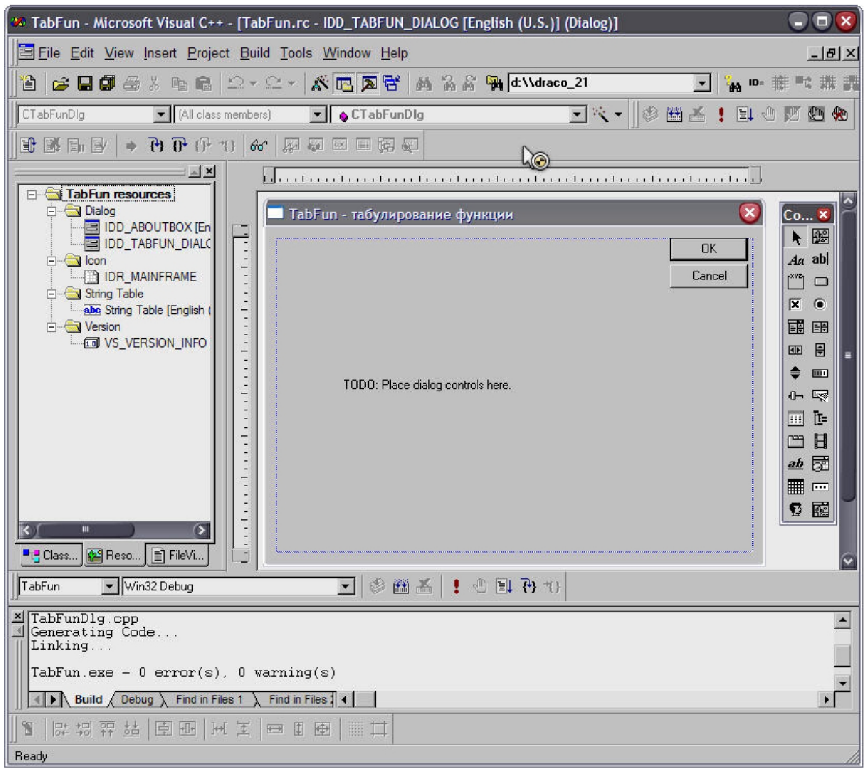


Рис. 7. Окно Visual C++

При рассмотрении рис. 7 отметим одно обстоятельство: в окне редактирования справа находится форма с кнопками **OK** (О'кей) и **Cancel** (Прекратить). Текст на форме гласит: **TODO: Place dialog controls here** (*ЧТО СДЕЛАТЬ: Поместить здесь элементы управления диалогом*). Рядом с формой находится панель с элементами управления, которые можно перемещать на нашу форму, зацепив левой кнопкой мыши (*Если у Вас открылся нестерпимый зуд и Вы уже перетащили на форму какой-либо элемент управления, то удалите его путем выделения и нажатия на клавиатуре клавиши Delete.* ***Кнопку «Cancel» не удалять никогда !!!***). Названия элементов управления можно посмотреть, поместив курсор мыши на элемент и немного подождав. Для создания полноценного осмысленного диалога Visual C++ имеет в своем составе **Dialog Editor** (Редактор диалога), которым мы и займемся.

4. Встреча с панелью управления

Панель, заполненная кнопками в Dialog Editor, называется ***Control Toolbar*** (Панель управления). Каждая кнопка из Control Toolbar – это элемент управления, который может размещаться в диалоговом окне нашей программы. Доступные элементы управления показаны на рис. 8.

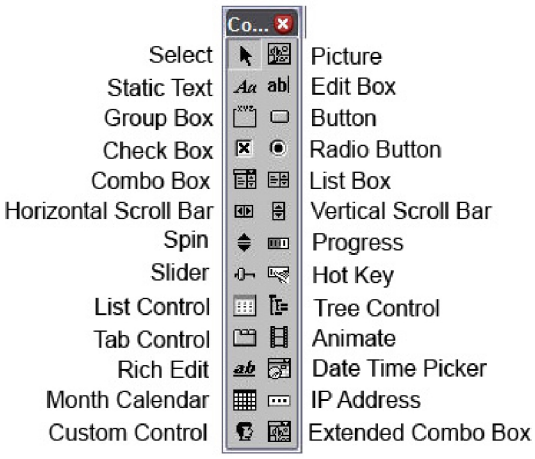
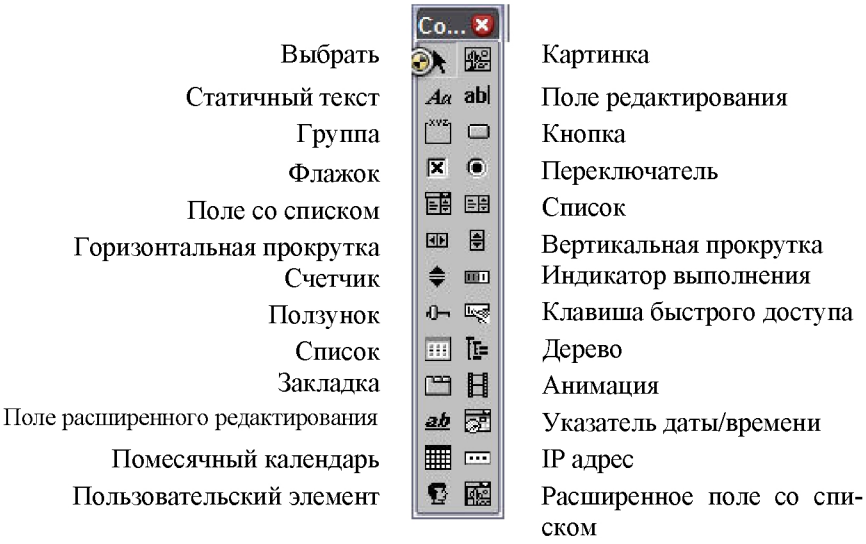


Рис. 8. Элементы управления диалогом

Для программы TabFun не нужны все элементы и мы будем давать подробный комментарий только к тем из них, которые будем использовать. Полное описание элементов управления можно найти в книге [2], здесь же дадим только перевод названий.



5. Персонализация программы

Хватит ходить вокруг да около. Как гласит французская пословица: «Чтобы стать кузнецом – надо ковать». Сейчас мы скуем персональную подпись под нашей программой, а именно: оформим диалоговое окно About (О программе).

Приведите Ваше окно Visual C++ в соответствие с окном на рис. 7. Внутри папки Dialog (Диалог) видны две небольшие пиктограммы с необычными именами **IDD_ABOUTBOX** и **IDD_TABFUN_DIALOG**.

Эти необычные имена – *идентификаторы ресурсов*. В них используются только заглавные буквы, соединенные символами нижнего подчеркивания. Первая часть идентификатора, в данном случае **IDD_**, является сокращением **ID**entifier for **D**ialog (идентификатор для диалогового окна). Если ресурс является пиктограммой, то идентификатор обычно начинается с букв **IDI_** (**I** – **I**con); если речь

идет о текстовом ресурсе или ресурсе управления, то **IDC_** (**C** – **C**haracter или **C**ontrol) и т.п.

Дважды щелкните левой кнопкой мыши на **IDD_ABOUTBOX** и начните редактировать диалоговое окно About TabFun в правой части окна оболочки Visual C++. Для начала расширьте это окно вниз. Для этого щелкните по границе окна About TabFun, и когда появится *прямоугольник выделения*, то зацепите левой кнопкой мыши нижнюю точку прямоугольника и после превращения курсора в стрелку острием книзу, потяните вниз. Прямоугольник увеличивается в размерах. При отпускании кнопки мыши процесс роста прекращается.

Щелкните по кнопке OK, и когда появится прямоугольник выделения, то подведите курсор так, чтобы появилось изображение курсора в виде стрелок в четырех направлениях Ї. Нажмите левую кнопку мыши и переместите кнопку OK к нижнему краю окна.

Щелкните по тексту TabFun Version 1.0. Расширьте вправо появившийся прямоугольник выделения. Щелкните по тексту TabFun Version 1.0 ***правой*** кнопкой мыши. Появится выпадающее меню, из которого надо выбрать пункт **Properties** (Свойства). В открывшемся окне свойств, рис. 9, (обратите внимание на закладку General) измените только текст в окне **Caption** (Заголовок).

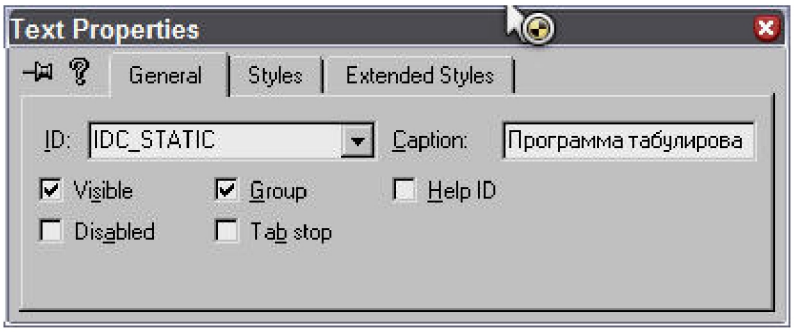


Рис. 9. Название программы

Переключитесь на русский язык и введите что-либо подобное «Программа табулирования функции. Версия 1.0». Старый текст, естественно, удалите. Закройте окно свойств и посмотрите на результат. Если текст пропечатан не полностью, то это означает, что размер выделенного прямоугольника маловат. Увеличьте его.

Аналогично обработайте поле Copyright © 2007. Сам текст оставьте - это международный признак держателя авторских прав, но справа укажите свою фамилию в качестве правообладателя.

Перетащите мышью с панели управления (см. рис. 8) пиктограмму Static Text (Статичный текст) на свободное место окна About TabFun. Расширьте получившийся прямоугольник вправо и вниз. В этом окне Вы в дальнейшем дадите короткую инструкцию по работе с программой. Для переноса текста на следующую строку служит комбинация **\n**, а для вставки шести пробелов комбинация **\t**. В закладках окна свойств Styles (Стили) и Extended Styles (Расширенные стили) можно задать принцип выравнивания текста и некоторые другие стили. Поэкспериментируйте с ними. Выберите на свой вкус. Имейте в виду, что длина текста в прямоугольнике не может превышать 256 символов. Если инструкция занимает больше, то Static Text используйте несколько раз.

Теперь о пиктограмме. Вы, вероятно, изначально заметили пиктограмму MFC в диалоговом окне About TabFun. Все проекты MFC используют эту стандартную пиктограмму и как пиктограмму программы, и как пиктограмму, появляющуюся в заданном по умолчанию диалоговом окне About, но пользователь практически всегда заменяет эту пиктограмму своей собственной.

Если снова обратить внимание на окно ProjectView, можно отметить, что пиктограмма именуется при помощи идентификатора **IDR_MAINFRAME** (Main – главный, Frame - рамка). Учитывая предыдущее обсуждение идентификаторов, можно было бы ожидать, что пиктограмма получит имя IDI_MAINFRAME. Однако рассматриваемый случай является исключением для стандарта обозначений имен: **IDR** означает **ID**entifier for a **R**esource, поскольку один и тот же идентификатор используется для пиктограмм, курсоров, таблиц акселераторов и ряда других ресурсов. Как вы думаете, почему MFC имеет несколько ресурсов с тем же именем? Если рассматривать оставшуюся часть имени, то это могло бы иметь смысл. Любой ресурс, который использует идентификатор IDR_MAINFRAME – это заданный по умолчанию ресурс для главного окна Вашего приложения. Пиктограммы, курсоры и другие ресурсы могут совместно использовать это имя.

Обратим наши взоры на пиктограмму, представляющую нашу программу. Эта пиктограмма отображается в нескольких местах:

1. Возле имени программы в Проводнике.
2. Возле имени программы в Windows Taskbar при выполнении программы.
3. В названии окна приложения.
4. На рабочем столе, если был создан ярлык для приложения.
5. В диалоговом окне About.

Пиктограммы бывают разных размеров. Обычный размер – 32 пиксела в ширину и 32 пиксела в высоту. Такого рода пиктограмма появляется на рабочем столе, либо же когда пользователь выбирает пункт Large Icons (Большие пиктограммы) из меню настройки. Пиктограммы с меньшей стороной (половиной ранее указанной величины, т.е. 16 пикселов) появляются в названии окна приложения и в Windows Taskbar, а также, когда пользователь выбирает пункт Small Icons или List в меню настройки внешнего вида.

Файл изображения пиктограммы может содержать оба типа изображений, хотя в окне ResourceView будет виден только один идентификатор. Для переключения между размерами изображения пиктограммы потребуется выбрать пункт меню из раскрывающегося списка, который появляется над изображением при редактировании пиктограммы. При выполнении такого переключения AppWizard генерирует пиктограммы размером 16x16 и 32x32 пиксела.

Поскольку нет необходимости создавать пользовательскую пиктограмму для обоих размеров, то нужно *удалить* маленькую пиктограмму из файла. После удаления, когда Windows выполняет программу TabFun и не находит маленькую пиктограмму, она просто соответствующим образом изменяет размеры большой пиктограммы. Сначала убедитесь в том, что загружена пиктограмма IDR_MAINFRAME. Для этого щелкните два раза по Icon -> IDR_MAINFRAME. Затем можно удалить маленькую пиктограмму, ***выбрав*** пункт Small (16 x 16) из раскрывающегося списка Device, а ***затем*** выбрав команду Delete Device Image (Удалить изображение устройства) из меню Image (во второй сверху строке главного окна). Убедитесь, что **не** выбран пункт Standard (32x32), иначе придется создавать пиктограмму заново !?!

Отредактируйте пиктограмму программы TabFun, используя встроенный графический редактор. Нарисуйте свое изображение, демонстрирующее Вашу индивидуальность и художественный вкус.

После всех операций окно About TabFun должно принять вид, подобный изображенному на рис. 10.

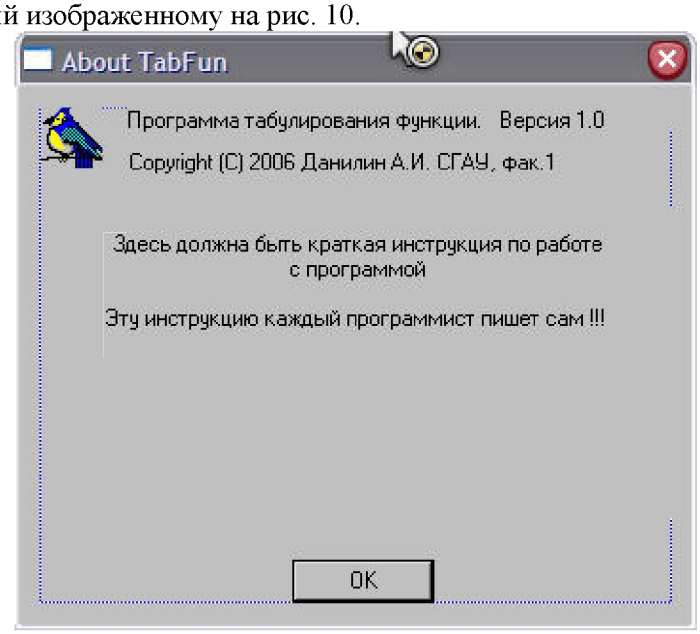


Рис. 10. Окно «О программе»

И, наконец, последний штрих. В закладке Resource View (Просмотр ресурсов) раскройте папку Dialog, дважды щелкните по строке IDD_ABOUTBOX и в раскрывшемся окне щелкните правой кнопкой мыши по выделенному прямоугольнику главного окна About TabFun. В раскрывшемся окне свойств замените текст в поле Caption на текст «О программе TabFun».

Теперь можно запускать программу на выполнение и смотреть, что получилось.

6. Трансляция, сборка, выполнение программы

Трансляция исходного кода программы в машинные команды и подключение библиотечных модулей - сборка программы - называется ***построением приложения***. Выполнить построение приложе-

ния можно тремя способами: 1) выбрать из главного меню **Build** (Построить) команду **Build TabFun.exe**; или 2) щелкнуть по пиктограмме на панели управления; или 3) нажать клавишу **F7**. Но не торопитесь нажимать указанные клавиши - почитайте дальше.

Как только запускается процесс построения приложения, Visual C++ предоставляет информацию пользователю о ходе этого процесса, выводя сообщения в окне Output (Вывод), которое обычно появляется в нижней части экрана. Если окно Output было закрыто, его вновь можно открыть, выбрав команду View -> Output из главного меню. Если окно Output отображается, убедитесь, что выбрана закладка Build, иначе будет невозможно наблюдать за процессом трансляции приложения.

При первом построении программы TabFun понадобится терпение. Этот процесс может потребовать времени даже на быстром компьютере. Все дальнейшие попытки построения одного и того же проекта занимают существенно меньшее время, поскольку Visual C++ транслирует только те файлы, которые подвергались изменению.

Если компилятор Visual C++ находит какие-либо ошибки, сообщения об ошибках помещаются в окно Output. Однако в Visual C++ сообщения об ошибках специфичны.

Если дважды щелкнуть на сообщении об ошибке в окне Output, происходит автоматический переход к месту нахождения ошибки, так что можно провести анализ и исправить исходный код.

После успешного построения приложения можно запускать его на выполнение. Это может быть исполнено также тремя способами: 1) выбрать из главного меню **Build** (Построить) команду **Execute TabFun.exe**; или 2) щелкнуть по пиктограмме с бордовым восклицательным знаком, расположенной на панели управления; или 3) нажать комбинацию клавиш **Ctrl+F5**.

Вовсе необязательно выполнять каждый раз оба эти этапа: построение приложения и запуск его на выполнение. Visual C++ внимательно следит за Вашими действиями, и если Вы вносите какие-либо исправления в файлы исходного кода или ресурсов (например, редактируете диалоговые окна), то он фиксирует это, и если Вы дае-

те указание запустить приложение на выполнение (минуя этап построения), то Visual C++ сообщит Вам, какие файлы нуждаются в перестроении и спросит, нужно ли их построить. На этот вопрос нужно ответить «Да» и тогда Visual C++ построит приложение и запустит его на выполнение. Поэтому после внесения изменений в свою программу для проверки эффекта этих изменений сразу нажимайте бордовый восклицательный знак и полагайтесь на неусыпного сторожа Visual C++.

Итак, щелкаем на бордовый восклицательный знак, нажимаем кнопку «**Да**» на запрос «Would You like to build them ?» («Хотите построить их ?») и получаем диалоговое окно Вашего приложения, показанное на рис. 11.

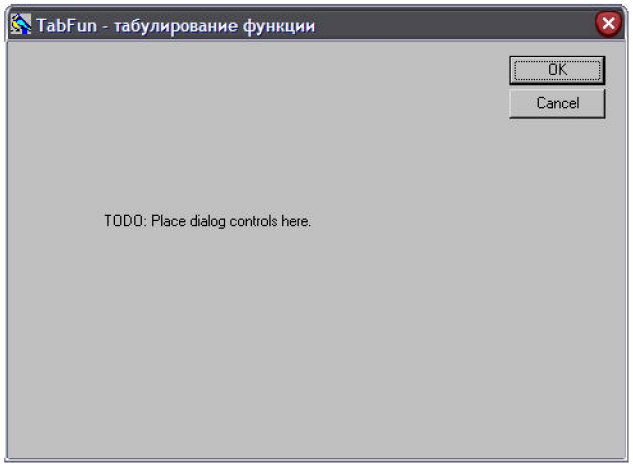


Рис. 11. Работа каркаса программы табулирования функции

Visual C++ сложил с себя все полномочия и теперь работает **только** Ваша программа TabFun. А где же окно с информацией о программе, над которым мы трудились ? Чтобы вызвать его, щелкните правой кнопкой мыши по верхней строке с текстом «TabFun - табулирование функции» и в выпавшем меню выберите пункт «About TabFun». Посмотрите, что получилось… После нажатия кнопки «OK» Вы возвращаетесь в диалоговое окно Вашего приложения - в программу TabFun.

Чтобы выйти из программы TabFun и вернуться в Visual C++, нужно нажать одну из кнопок: «ОК» или «Cancel». Если Вы войдете в Visual C++ **не выходя** из программы TabFun (например, щелкнув по окну Visual C++) и попытаетесь вновь построить и выполнить программу TabFun, то попытка окончится крахом на этапе сборки модуля.

Нельзя строить программу, которая в данный момент работает !?!
Поэтому не забывайте выходить из своей программы перед возвратом в Visual C++.

Мы славно поработали, теперь можно передохнуть и сделать перерыв. Закройте главное окно Visual C++, как это принято в среде Windows. Не беспокойтесь о сохранении результатов Вашей работы - Visual C++ сделает это автоматически при выходе.

Если Вы хотите для надежности переписать созданный проект на флэш-память или дискету, то перед копированием **удалите папку Debug из папки TabFun**, поскольку папка Debug содержит промежуточные файлы Visual C++, занимает много места и формируется каждый раз, когда происходит построение приложения.

А что же с главной задачей программы: табулированием функции и построением ее графика ? Мы этим займемся, когда Вы наберетесь новых сил…

7. Проектирование программы и создание диалога

Чтобы писать программу, необходимо сначала спроектировать ее. Проектирование программы включает в себя выполнение следующих действий.

1. Продумать и выбрать интерфейс с пользователем программы.
2. Выбрать алгоритм, который будет положен в основу программы. (Час, потраченный на выбор алгоритма, стоит четырех часов написания кода. То есть, если Вы не потратили времени на

выбор алгоритма, а сразу принялись писать код, то четыре часа Вашей жизни – на ветер).

3. Продумать побочные эффекты и защиту от неквалифицированного пользователя.

В нашем случае алгоритм табулирования функции очевиден. Продвигаясь в цикле от начала интервала с заданным шагом, вычисляем значение функции и строим ее график, соединяя прямыми линиями текущую точку с предыдущей. Процесс прекращаем при достижении конца интервала. Алгебраическое выражение функции жестко запрограммировано в исходном коде, но начало и конец интервала, а также шаг табулирования необходимо вводить с клавиатуры.

Запустите Visual C++. В главном меню выполните команду File -> Open Workspace (Файл -> Открыть рабочую область) и в выпавшем диалоговом окне откройте ***свой*** проект **TabFun.dsw**. Откройте закладку Resource View в окне рабочей области проекта и разверните все пункты в TabFun Resources. Дважды щелкните левой кнопкой мыши на IDD_TABFUN_DIALOG. В окне справа щелкните по тексту «TODO: Place dialog controls here» и нажмите клавишу Delete. Щелкните по границе окна «TabFun – табулирование функции» и когда появится прямоугольник выделения, увеличьте его в размерах. На этом поле мы будем располагать элементы управления, рисовать график и выдавать таблицу значений. Удалите кнопку «OK». Щелкните по ней и после появления прямоугольника выделения нажмите клавишу Delete. Сместите кнопку «Cancel» вниз-влево до упора, зацепив левой кнопкой мыши. Щелкните по кнопке «Cancel» ***правой*** кнопкой мыши и войдите в пункт Properties (Свойства) выпавшего меню. В окне диалога свойств на закладке General (Общие) измените Caption (Заголовок) на «В ы х о д».

Текст «IDCANCEL» в поле ID не изменять ни в коем случае. Попробуйте сделать это – и все предыдущие труды пойдут насмарку. Ваш проект будет испорчен безвозвратно. Придется начинать все с создания нового проекта.

Постройте и выполните программу. При нажатии на кнопку «Выход» Вы возвращаетесь в Visual C++. Из панели управления

возьмите левой кнопкой мыши элемент «Edit Box» (Поле редактирования) и перетащите на поле Вашей программы. Убедитесь, что элемент окружен прямоугольником выделения. Щелкните по нему ***правой*** кнопкой мыши и войдите в пункт Properties выпавшего меню. В окне диалога свойств на закладке General измените идентификатор ID на IDC_BEGIN, как это показано на рис. 12. Полю редактирования теперь присвоено имя и с этим полем надо связать некоторую переменную в программе, чтобы число, которое наберет пользователь в этом поле, было присвоено этой переменной.

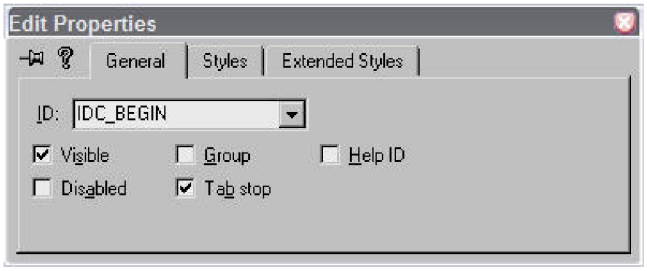


Рис. 12. Поле данных для начала интервала табулирования

Щелкните по элементу Edit Box ***правой*** кнопкой мыши и выберите пункт **Class Wizard** (Мастер классов) выпавшего меню. В появившемся окне мастера перейдите на закладку **Member Variables** (Переменные-члены), выделите IDC_BEGIN и щелкните кнопку «**Add Variable**» (Добавить переменную). Убедитесь, что вы добавляете переменную в проект **TabFun** и в класс с именем **CTabFunDlg**. Visual C++ предложит Вам задать имя переменной, категорию и тип. Задайте имя **m_Begin**, категорию **Value** (значение) и тип **double** (с двойной точностью). Итог показан на рис. 13.

Щелкните по кнопке OK и задайте для переменной m_Begin минимальное и максимальное значение, как это показано на рис. 14. Нажмите кнопку OK.

Аналогичным образом организуйте Edit Box с идентификаторами **IDC_END** и **IDC_STEP** и соответствующими им переменными **m_End** и **m_Step**.

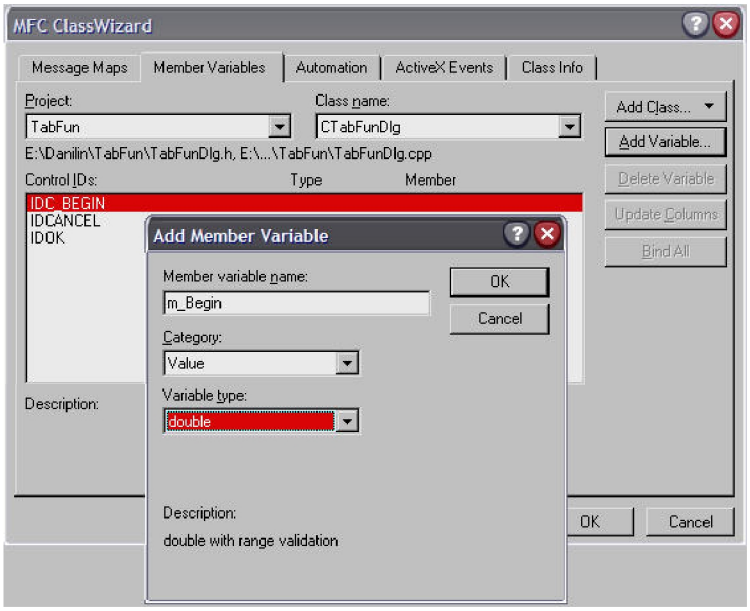


Рис. 13. Переменная для поля IDC_BEGIN

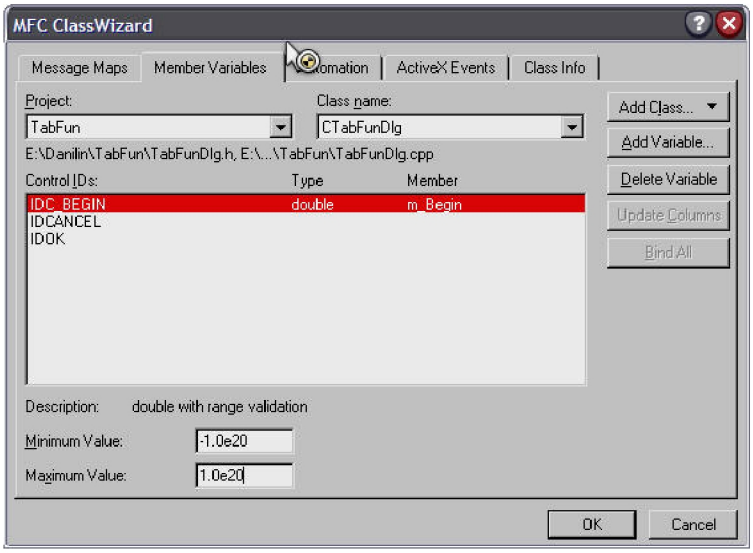


Рис. 14. Тип и диапазон значений

Visual C++ использует соглашение об именах переменных. Имя класса начинается с заглавной буквы **C** (Class); имя переменной-члена класса начинается с **m_** (member – член); указатель – с маленькой буквы **p** (pointer – указатель) и т.п. Признаком хорошего стиля программирования считается придерживаться этих соглашений.

После всех этих действий закладка Member Variables должна принять вид, показанный на рис.15.

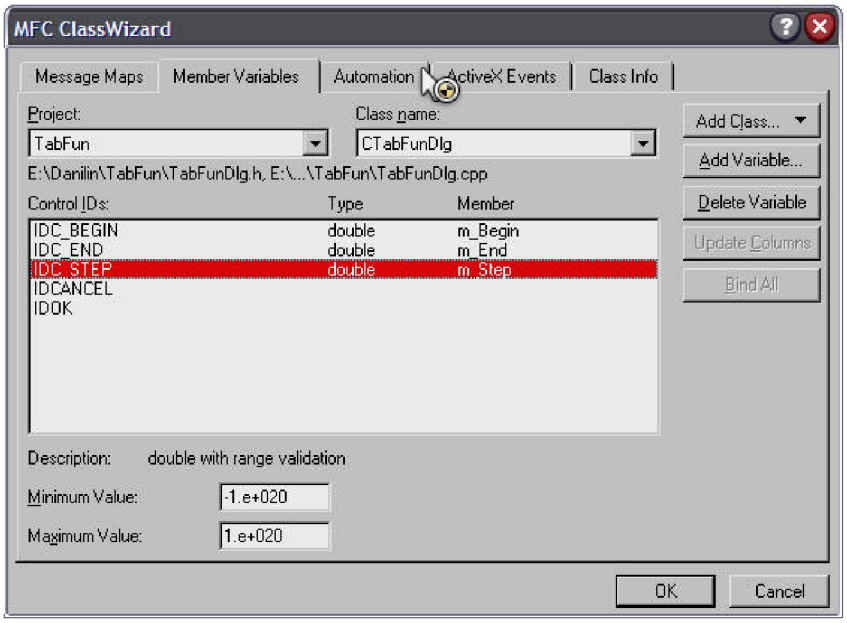


Рис. 15. Переменные для интервала и шага табулирования

Постройте приложение и запустите его на выполнение. На диалоговом поле появились три окошка, в которых можно набирать текст. Окошки безликие и кроме Вас никто не знает, в каком из них нужно вводить начало интервала, конец интервала и шаг. Поэтому диалог надо доработать.

Раздвиньте поля редактирования по горизонтали. Не беспокойтесь, чтобы они были на одном уровне по вертикали. Перетащите с панели элементов управления элемент Group Box и разместите его так, чтобы он **охватывал** поле редактирования, соответствующее, например, m_Begin (IDC_BEGIN). У меня это левое поле. При необходимости растяните или сожмите Group Box. Щелкните **правой** кнопкой мыши по выделенному прямоугольнику Group Box и войдите в окно Properties. Измените заголовок (Caption) на «Начало интервала». Постройте приложение и запустите его на выполнение. Не правда ли, ситуация с вводом начинает проясняться? Аналогично постройте Group Box вокруг полей m_End (IDC_END) и m_Step (IDC_STEP). У меня получился диалог, фрагмент которого показан на рис. 16. Не очень эстетично, не так ли?

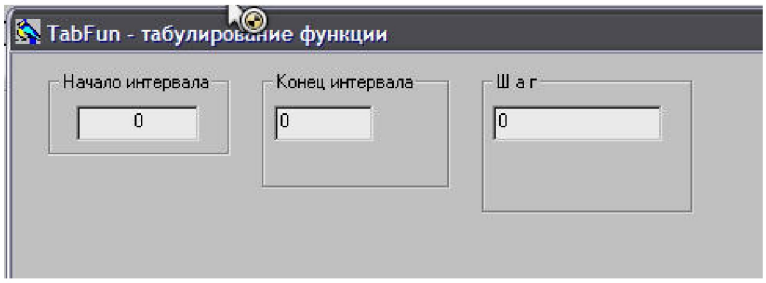


Рис. 16. Диалоговый ввод значений

В окне «Начало интервала» 0 расположен в центре, а в остальных прижат к левой границе. Все нули в центр!! Для этого надо войти **в окно свойств** IDC_END и IDC_STEP и в закладке **Styles** выбрать **Centered** в окне **Align text**.

Размеры полей редактирования различны? Сделать их одинаковыми!! Щелкните левой кнопкой мыши по правому полю редактирования. Появится прямоугольник выделения. Нажмите на клавиатуре клавишу Ctrl и, ***не отпуская*** ее, щелкните по среднему и левому полям редактирования. Левый прямоугольник выделения остался цветным, а остальные побледнели. Мы выделили группу элементов, причем ***цветной элемент является доминирующим***. Обратите внимание на линейку пиктограмм

1 2 3 4 5 6 7 8 9 10 11 12 13 14

Идем слева направо.

1. Выключить/включить линейку.
2. Выравнивать выделенные элементы по ***левой*** границе доминирующего элемента.
3. Выравнивать выделенные элементы по ***правой*** границе доминирующего элемента.
4. Выравнивать выделенные элементы по ***верхней*** границе доминирующего элемента.
5. Выравнивать выделенные элементы по ***нижней*** границе доминирующего элемента.
6. Расположить выделенные элементы ***посредине высоты*** окна.
7. Расположить выделенные элементы ***посредине ширины*** окна.
8. Расположить выделенные элементы на равном расстоянии ***по горизонтали***.
9. Расположить выделенные элементы на равном расстоянии ***по вертикали***.
10. Сделать ширину выделенных элементов ***равной ширине доминирующего***.
11. Сделать высоту выделенных элементов ***равной высоте доминирующего***.
12. Сделать ширину и высоту выделенных элементов ***равными ширине и высоте доминирующего***.
13. Показать сетку привязки.
14. Включить/выключить направляющие.

Щелкните пиктограмму №12, затем №5 и затем №8. Обработайте таким же образом элементы Group Box. Здесь будьте внимательны при использовании пиктограммы №8. Предварительно нужно разместить крайние элементы и только потом работать с группой. Окончательный вид фрагмента окна программы TabFun показан на рис. 17.

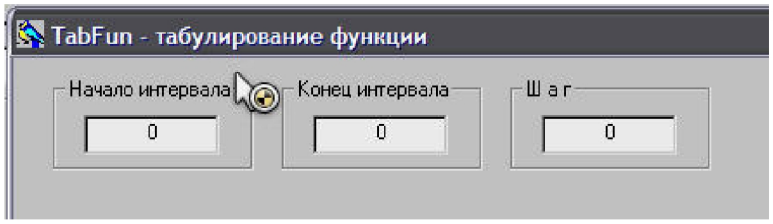


Рис. 17. Окончательный вид окна ввода значений

Постройте приложение и запустите его на выполнение. Нас ждет неприятный сюрприз. Оказывается, можно набирать ЛЮБОЙ текст в окнах редактирования и программа не реагирует на нечисловые данные !?! Чтобы этот текст преобразовывался в число с соответствующей диагностикой и затем передавался в память, отведенную для переменных m_Begin, m_End и m_Step, нужно вызвать программу **UpdateData(TRUE)**.

Как это сделать ?

8. Знакомство с каркасом приложения

Пока мы лишь эксплуатировали Visual C++, а сами не написали ни одной строчки исходного кода на языке C++. Настала пора посмотреть, какую программу создал наш помощник, и определить, где модернизировать ее, чтобы она выполняла наши прихоти.

Перейдите на закладку ClassView в рабочей области проекта и разверните все классы, созданные AppWizard. Определения классов помечены пиктограммами, изображающими прямоугольники, соединенные линиями. Малиновые кубики слева от имен означают, что имя соответствует функции, а голубые – переменным. Среди переменных узнаем наши переменные m_Begin, m_End, m_Step. Ключик слева от кубика означает, что переменная имеет тип **protected** (защищенная), а отсутствие какой-либо пиктограммы, - что переменная общедоступна (тип **public**). Переменная типа **private** (личная) обозначена пиктограммой в виде висячего замка. Щелкните два раза по CTabFunApp и в окне справа появится текст определения этого класса. Нам придется много работать с текстами программ, поэтому запомните следующее.

1. Двойной щелчок по ***имени класса*** вызывает появление в окне справа определения этого класса. В определение класса входит объявление всех функций класса и всех переменных класса.
2. Двойной щелчок по ***имени функции*** вызывает появление в окне справа исполняемого кода этой функции.
3. Двойной щелчок по ***имени переменной*** вызывает появление в окне справа объявления этой переменной.
4. В тексте исходного кода комментарии начинаются с // и выделены ***зеленым цветом***. Комментарии в Visual C++ специфичны и эту специфику мы рассмотрим дальше.
5. ***Синим цветом*** выделены ключевые слова языка C++.
6. ***Черный цвет*** соответствует именам переменных, функций, знакам операций, выражениям и т.п.

Даже если Вы – опытный программист на языке C/C++, Вы, вероятно, будете удивлены, когда впервые откроете файл, сгенерированный при помощи AppWizard (Мастера приложений). Может сложиться впечатление, что код чрезвычайно странный и, кроме того, что означают эти очень уж непонятные комментарии:

```
//{{AFX_MSG(CAboutDlg)
//}}AFX_MSG
```

А что вы будете делать с такого рода строкой?

```
#define
AFX_TABFUN_H__8536FBE7_8757_4917_8BFA_40195173AFEC__IN
CLUDED_
```

Что бы вы ни думали об этом, код AppWizard имеет вполне логичное объяснение. Он создан для улучшения работы инструментальных средств автоматизации программирования типа ClassWizard, а не для того, чтобы его расшифровывал пользователь.

Поскольку код, предназначенный для ClassWizard (Мастера классов), столь необычен, то маловероятно спутать его с написанным Вами кодом, а также поддаться соблазну изменять его. Также сомнительно, чтобы один из Ваших идентификаторов случайно вошел в противоречие с идентификатором, выбранным AppWizard (даже если Ваш вкус в выборе имен идентификаторов исключитель-

но оригинален). Чтение программы, созданной AppWizard, сходно с чтением текста, написанного на двух различных языках. Если Вы не понимаете один из языков, то просто проходите далее и отыскивайте части, имеющие для Вас смысл.

Проект TabFun в качестве своего класса приложения использует CTabFunApp. Проекты AppWizard следуют практике, общепринятой при работе с C++ и состоящей в сохранении *определения класса в файле заголовков*, который использует расширение **.h**. Также принято сохранять *выполнение или фактические функциональные модули* для класса в отдельном файле с расширением **.cpp**.

Файлы заголовков могут содержать встроенные функции, но они не создают переменных и не распределяют память.

Приложение TabFun следует этому соглашению, записывая файл заголовков для класса CTabFunApp под именем **TabFun.h**.

Правило «Не влезай – убьет»:

Не изменяйте код, сгенерированный не Вами.

Замена кода обязательно приведет к непредсказуемым последствиям, в большинстве случаев не поддающихся исправлению.

Поэтому никогда не редактируйте непосредственно файлы с расширениями **.h** и **.cpp**. Работа через Visual C++ - это залог беспроблемной работы Ваших программ. Но вернемся к тексту нашего приложения.

Определение для класса CTabFunApp занимает большую часть TabFun.h. Вместо того чтобы повторять каждый фрагмент кода, рассмотрим лишь некоторые моменты.

☞ Обратите внимание на то, что из самого заголовка следует, что класс публично унаследован от CWinApp. На это указывает строка **class CTabFunApp : public CWinApp**. Знак «:» разделяет родителя (справа) и потомка (слева). Это означает возможность вызова любого из методов, унаследованных от CWinApp или его суперклассов.

☞ Для упрощения поиска файлы заголовков, которые создает AppWizard, делятся на разделы, выделяемые строками из символов «/». Заголовок начинается с раздела конструктора. ***Конструктор класса создает экземпляр класса.*** Затем следует раздел, отмеченный комментарием

`// Overrides` (Подмены)
Этот раздел содержит виртуальные функции, которые вы можете расширять. Вообще, термин «*Подмены*», на мой взгляд, более понятный, чем термин «*Перекрытия*», принятый в литературе по объектно-ориентированному программированию. Фактически виртуальная функция, которая описана в данном разделе, **заменяется** (подменяется, перекрывается) реальной функцией класса.

Раздел, отмеченный комментарием
`// Implementation` (Выполнение)
содержит код AppWizard, который может изменяться от версии к версии. Не следует полагаться на код в разделе выполнения – вообще не изменяйте его.

☞ Наконец, обратите внимание на *AFX*-комментарии. В этом месте ClassWizard добавляет или удаляет функции и тому подобное. Как указывает один из комментариев, не следует редактировать что-либо, находящееся между маркерами комментария, которые выглядят следующим образом.

```
//{{
//}}
```

☞ Последняя строка файла заголовка содержит загадочную строку

```
DECLARE_MESSAGE_MAP ()
```

Эта *макрокоманда* сообщает Visual C++, что класс будет отвечать на сообщения Windows. Эта запись и соответствующие записи отображаемых сообщений в файле выполнения являются частью сложной макросистемы для формирования таб-

лиц ответных сообщений (message response tables). Таблица ответных сообщений гарантирует, что сообщения Windows направляются соответствующему обработчику и окну. Нет необходимости в дополнительных знаниях для практического использования системы отображения сообщений.

Как работает операционная система Windows и какую роль играют сообщения, - об этом особый и достаточно длинный разговор, поэтому обратитесь к Приложению 2 или к другой книге, например [6].

Код для ***выполнения*** класса CTabFunApp находится в файле **TabFun.cpp**. Вместо того чтобы читать код программы целиком, обратим внимание на его существенные фрагменты.

Первые три строки

```
#include "stdafx.h"
#include "TabFun.h"
#include "TabFunDlg.h"
```

подключают стандартный файл заголовков каркаса приложения и файлы заголовков, составляющих наш проект.

Далее идут строки, определяющие отладочные режимы.

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

После этого идет раздел отображения сообщений класса, который мы опустим.

Конструктор класса **CTabFunApp::CTabFunApp()**, который идет следом и должен создавать экземпляр нашего приложения, вообще вызывает недоумение: тело процедуры между {...} не содержит ничего, кроме комментария !?! Пустота порождает работающую программу !!!

Ответ находится в последней строке раздела, а именно

```
// The one and only CTabFunApp object
// Один и единственный объект CTabFunApp

CTabFunApp theApp;
```

В классическом объектно-ориентированном программировании распределение памяти для объекта, все начальные присвоения переменным (инициализация) и задание всей управляющей информации происходит в конструкторе класса. Что же здесь ?

Объект **theApp** создается как глобальный (текст его создания находится **вне** фигурных скобок какой-либо процедуры). В результате конструктор нашего приложения вызывается при загрузке программы и ***перед*** вызовом главной управляющей программы WinMain, которая производит регистрацию нашего приложения в среде Windows и настраивает все подсистемы MFC. Все настройки происходят **после** конструирования основного объекта приложения и **на код**, помещенный в теле конструктора, реагировать некому. Поэтому, если нужно провести инициализацию каких-либо переменных, то конструктор класса CTabFunApp – не помощник. Вместо этого класс CTabFunApp подменяет виртуальную функцию InitInstance() своей собственной, в код которой надо войти. Приведем ее код полностью, заменив язык комментариев и добавив новые.

```
BOOL CTabFunApp::InitInstance() //Знак «::» означает
                                //принадлежность классу
{
      AfxEnableControlContainer();

// Стандартная инициализация
// Если вы не используете эти свойства и
// хотите уменьшить размер или это окончательный вариант,
// то вы должны удалить следующие те
// специфические программы инициализации,
// которые не нужны.

    #ifdef _AFXDLL
       Enable3dControls(); // Вызов при динамическом
                           // использовании библиотек
    #else
```

```
        Enable3dControlsStatic(); // Вызов при статическом
                                  // подключении библиотек
    #endif

        CTabFunDlg dlg;     //Объявление диалогового окна
        m_pMainWnd = &dlg;  //Наложение главного окна на
                            // объявленное
        int nResponse = dlg.DoModal();   //Выполнение
                                         //диалогового окна
        if (nResponse == IDOK) //Если выход по кнопке OK
        {
              // Поместить здесь код, который надо
              // выполнить при выходе по кнопке OK
        }
        else if (nResponse == IDCANCEL)  // Если выход по
                                         // кнопке Cancel
        {
              // Поместить здесь код, который надо
              // выполнить при выходе по кнопке Cancel

        }

// Поскольку диалог закрыт, возвратить FALSE, чтобы мы
// вышли из приложения раньше, чем начнется подкачка
// сообщений приложения

        return FALSE;
    }
```

Обратите внимание на две выделенные строчки. Первая создает экземпляр класса **CTabFunDlg**, а вторая запускает его на выполнение и мы видим на экране работу нашей программы. Таким образом,

Всю полезную работу по табулированию функции мы будем выполнять в классе CTabFunDlg.

Разверните его и войдите в конструктор класса, щелкнув два раза по функции CTabFunDlg(…). Она содержит немного кода: инициализацию наших переменных **m_Begin**, **m_End** и **m_Step** и загрузку пиктограммы программы. Если при последующих отладках нулевые начальные значения для этих переменных неудобны, то здесь можно ввести нужные значения. Однако с учетом правила «Не влезай - убьет» делать это надо осторожно. Просто добавьте перед

закрывающейся фигурной скобкой конструктора нужные строки. Конструктор будет выглядеть примерно так.

```
/////////////////////////////////////////////////////////
// CTabFunDlg dialog

CTabFunDlg::CTabFunDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CTabFunDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CTabFunDlg)
	m_Begin = 0.0;
	m_End = 0.0;
	m_Step = 0.0;
	//}}AFX_DATA_INIT
// Note that LoadIcon does not require a subsequent
// DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);

	m_Begin = 1.0;
	m_End =  10.0;
	m_Step =  0.01;
}
```

Выполните программу.

Что появилось в окнах редактирования ?

9. Начинаем работать с графикой

Куда нам поместить код, который будет принимать данные из полей редактирования и рисовать график ? Давайте посмотрим на функции, которые предлагает нам Visual C++ в классе CTabFunDlg и обсудим их назначение.

CTabFunDlg(CWnd* pParent /*=NULL*/) – конструктор класса, который вызывается только при создании экземпляра класса. Если поместить код рисования графика здесь, то график будет нарисован один раз на интервале и с шагом, которые заданы в программе. Изменения в полях редактирования интервала и шага на графике отражаться не будут.

DoDataExchange(CDataExchange* pDX) – отвечает за поля редактирования и обмен данными между «переменными» и отображениями их значений на экране. Код рисования графика здесь совершенно неуместен.

OnInitDialog() – инициализирует диалог и отображает окно диалога на экране. Вызывается один раз при загрузке программы. Код рисования, помещенный здесь, также будет выполняться один раз и наша программа не будет реагировать на изменения интервала и шага табулирования.

OnPaint() – кандидат номер один на то, чтобы здесь поместить код рисования графика. Эта функция обрабатывает сообщение **WM_PAINT**, которое указывает, что окно должно быть перерисовано. Это сообщение посылается операционной системой Windows по ее усмотрению. Достоверно можно сказать, что диалоговое окно нашей программы будет перерисовано, когда оно сворачивается в системный трэй (минимизируется), а затем снова становится видимым и когда после перекрытия другим окном снова становится активным. А вообще, Windows сама решает, когда перерисовывать. Если поместить код рисования здесь, то программа работать будет, но мы получим кучу побочных эффектов и на нашем поле рисования будут оставаться «хвосты» от других окон.

OnQueryDragIcon() - обрабатывает сообщение **WM_QUERYDRAGICON**, которое посылает Windows, когда пользователь перетаскивает минимизированное окно приложения.

OnSysCommand(UINT nID, LPARAM lParam) - обрабатывает системное сообщение **WM_SYSCOMMAND**, которое посылается каждый раз, когда пользователь выбирает какой-либо пункт в системном меню. В дополнение к системным сообщениям функция проверяет, не выбран ли пункт About (О программе).

И что же в итоге ?... Вклиниться просто негде !! Не будем впадать в отчаяние. Когда выбирать не из чего, надо организовать новую возможность.

Давайте создадим кнопку на диалоговом поле программы с надписью «Чертить график»; при нажатии на нее будет вызвана функция, которая и нарисует график.

Войдите в закладку **Resource View**, разверните папку Dialog и дважды щелкните левой кнопкой мыши по **IDD_TABFUN_DIALOG**. Перетащите с панели управления элемент **Button** (Кнопка). Расположите его справа от окон ввода значений на

одном уровне с полями редактирования и несколько растяните по горизонтали. Войдите в окно свойств кнопки и замените IDC_BUTTON1 на **IDC_DRAW**, а заголовок в окне Caption на «Чертить график». Результат должен выглядеть, как на рис. 18.

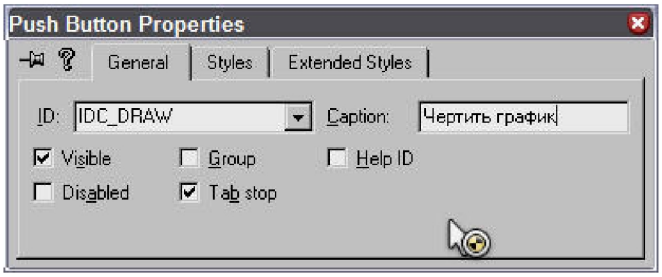


Рис. 18. Свойства кнопки

Щелкните правой кнопкой мыши по кнопке «Чертить график» и вызовите **ClassWizard** (Мастер классов). Войдите в закладку **Message Maps** (Карты сообщений) и выделите IDC_DRAW, BN_CLICKED, после чего нажмите кнопку **AddFunction** (Добавить функцию). Итоговый вид окна показан на рис. 19. После щелчка по AddFunction будет предложено имя функции **OnDraw**, с которым нужно согласиться, нажав «ОК».

Мы создали функцию, которая вызывается, если пользователь щелкнет по кнопке «Чертить график». Проверим это.

В окне на рис. 19 нажмите кнопку **Edit Code** (Редактировать код) и Вы попадете внутрь функции OnDraw. Пока единственной строкой этой программы является комментарий **//ЧТО СДЕЛАТЬ: Добавить код обработчика элемента управления**. Замените эту строку своим комментарием:

```
//Программа черчения графика функции на заданном интервале.
```

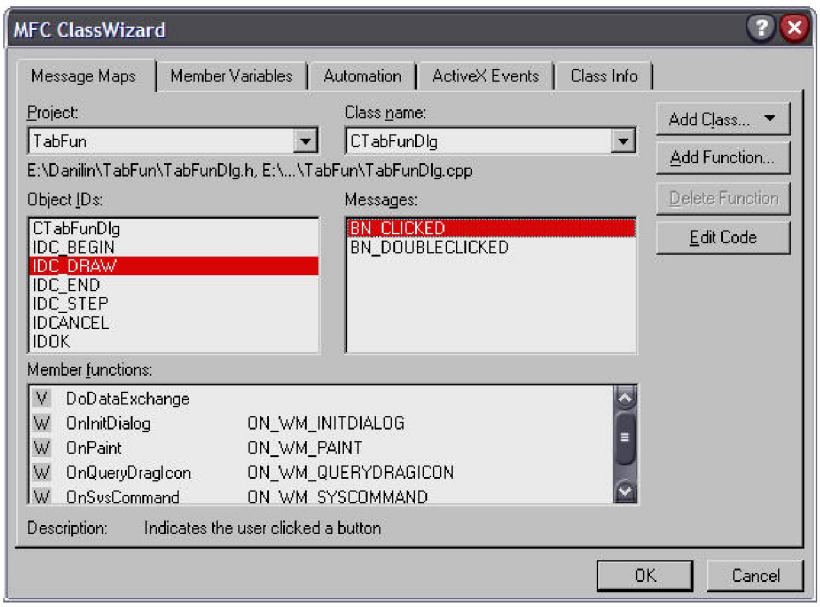


Рис. 19. Задание обработчика нажатия кнопки

После комментария наберите строку

```
MessageBox("Мы находимся внутри OnDraw","ОТЛАДКА",MB_OK);
```

и запустите программу на выполнение. Щелкните по кнопке «Чертить график». Работает, не так ли ? Не удаляйте строку **MessageBox(…)**, просто сделайте ее комментарием, добавив // в начало строки.

Теперь можно заняться подготовкой к вычерчиванию графика функции. Для примера мы возьмем функцию $Y = 5.0 \times \sin(x) + 0.1 \times x^2$ и рисовать график будем в области нашего диалогового окна, внутри прямоугольника, который будем считать нашим холстом.

Для этого выполните следующие действия. Откройте диалоговое окно **IDD_TABFUN_DIALOG.** Перетащите с панели управления элемент **Picture** (Картинка) в диалоговое окно. В окне его свойств установите идентификатор **IDC_HOLST**, тип **Frame** (Рам-

ка) и цвет **Gray** (Серый) (см. рис. 20). В закладке **Styles** (Стили) ***отметьте*** только флажок **Sunken** (Вдавленный), а в закладке **Extended Styles** (Расширенные стили) только флажок **Client Edge** (Клиентская кромка). Увеличьте его размеры, чтобы будущий график не надо было рассматривать в микроскоп.

В закладке **ClassView** щелкните правой кнопкой мыши по ***классу*** **CTabFunDlg**. В открывшемся окне выберите **Add Member Variable…** (Добавить переменную-член). Когда откроется диалоговое окно, введите в качестве типа переменной **CRect** и в качестве имени переменной **m_Holst**. Для доступа выберите тип **private** (личный). Завершенный вид окна Add Member Variable показан на рис. 20. Переменная m_Holst, равно как переменные m_Begin, m_End и m_Step, принадлежат классу CTabFunDlg и доступны во всех функциях этого класса.

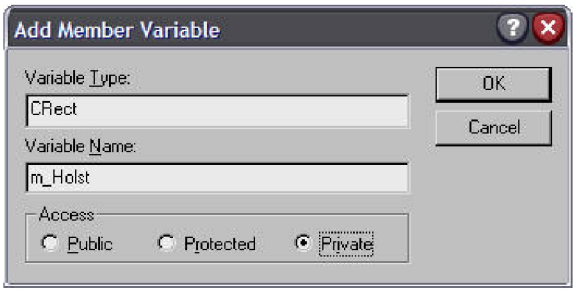


Рис. 20. Задание холста для рисования графика

CRect – это структура, в которой хранятся координаты левого верхнего и правого нижнего угла прямоугольника с именем m_Holst. Внутри работающей программы нам необходимо «наложить» переменную m_Holst на рамку ресурса IDC_HOLST и загнать в переменную m_Holst фактическое положение рамки на экране, чтобы мы могли расположить график внутри рамки IDC_HOLST, то есть на нашем холсте. Это делается так.

В закладке ClassView войдите в функцию OnInitDialog класса CTabFunDlg и после комментария введите следующие строки.

```
// TODO: Add extra initialization here

CWnd* pHolst = GetDlgItem(IDC_HOLST);//Получить
```

```
                          // указатель на рамку холста
pHolst->GetWindowRect(&m_Holst); //Получить
                          // его положение на экране
ScreenToClient(&m_Holst); // Получить координаты
                          // холста внутри главного окна
m_Holst.DeflateRect(2, 2, 1, 1); //Исключить из
                                 // холста толщину рамки
```

Назначение программ ясно из комментариев.

Каждое окно имеет оси координат с началом в левом верхнем углу. *Ось «X» направлена* ***вправо,*** *ось «Y»* ***вниз.*** Главное окно имеет так называемую «клиентскую» область, из которой исключена верхняя строка главного окна нашего приложения. Координаты клиентской области могут быть вычислены как в экранных координатах (относительно левого верхнего угла экрана), так и в оконных координатах (относительно левого верхнего угла главного окна). При перемещении окна по экрану ***Windows*** отслеживает положение главного окна приложения и положение клиентской области. Нам удобнее работать в клиентских координатах, возлагая черновую работу по отслеживанию положения окон на ***Windows***.

Первое, что мы сделаем в программе **OnDraw()**, так это заполним наш холст цветным фоном, который заодно сотрет график от предыдущего вызова функции. Для этого функция **OnDraw()** должна принять следующий вид.

```
void CTabFunDlg::OnDraw()
{
// Программа черчения графика функции
// на заданном интервале
//MessageBox("Мы находимся внутри OnDraw",
// "ОТЛАДКА",MB_OK);

CClientDC dc(this); //Объявить и создать контекст
                    // устройства

CBrush m_Brush;     //Объявить кисть
m_Brush.CreateSolidBrush(RGB(255,255,255));//Создать
                                 // кисть белого цвета
dc.FillRect(m_Holst,&m_Brush);   //Замазать холст
}
```

Запустите программу на выполнение. Поэкспериментируйте с цветом. Цифры в параметрах программы **RGB** означают «количест-

во» красного **R** (Red), зеленого **G** (Green) и синего **B** (Blue). Диапазон изменения «количества» каждого цвета от 0 до 255. Черный цвет соответствует RGB(0, 0, 0), бледно-желтый – RGB(255, 255, 200), красный – RGB(255, 0, 0), зеленый – RGB(0, 255, 0) и т.д.

Теперь обсудим, что такое **контекст** устройства, зачем он нужен и как он работает.

Под Windows любой вывод – графический, поскольку используется одна и та же библиотека **GDI** (**G**raphical **D**evice **I**nterface – интерфейс графического устройства) независимо от того, направляется ли вывод на экран, принтер, графопостроитель или на какое-то другое экзотическое устройство вывода. Каждая буква, каждая строка, каждый знак, отображаемые программой, пропускаются через утилиты графической библиотеки GDI, обеспечивающие корректное отображение вывода. Не имеет значения, используется при этом старинный дисплей VGA с разрешением 640 x 480 пикселей или суперсовременное фотонаборное устройство с высокой разрешающей способностью.

Чтобы такая гибкость оказалась возможной, GDI использует три механизма.

- *Контекст устройства* (**D**evice **C**ontext или **DC**) — это логический холст, который используется для вывода в системе ***Windows***. Вместо того чтобы выводить данные непосредственно на экран или принтер, Вы неизменно пользуетесь контекстом устройства, направляя туда потоки информации для вывода.
- *Драйвер устройства.* Контекст устройства (**DC**), то есть логическое выходное устройство, соединен с физическим устройством вывода через драйвер устройства ***Windows***. Драйвер устройства транслирует команды **GDI** в физические команды, которые устройство выполняет с целью отображения выходных данных. Этот процесс не требует при написании программы знания конечного выходного устройства. С другой стороны, Вы иногда можете потребовать от устройства вывода выполнить то, чего оно сделать не может, например, *нарисовать цветной текст на черно-белом принтере*. Заботу об удовлетворении запросов пользователя, даже завышенных и нелепых, ***Windows*** возлагает на драйвер устройства. Мы намеренно употребили словосочета-

ние «*нарисовать ... текст*», чтобы лишний раз подчеркнуть, что текстового режима в Windows нет.

☞ *Режим отображения.* Физические устройства вывода выпускаются с различными разрешающими способностями, от стандартной электронно-лучевой трубки с разрешением 72 dpi (dots per inch – точек на дюйм) до фотонаборного устройства с разрешением 14400 dpi. Следовательно, независимая от устройств графическая библиотека должна выполнять преобразование координат и масштабирование изображения. Воспроизведенный на экране квадрат из 100x100 пикселей занимает место, чуть большее, чем 35x35 мм, но тот же квадрат, отображенный фотонаборным устройством, выглядит бесконечно малым. Библиотека GDI решает эту проблему посредством реализации различных режимов логического отображения. Выполняя графический вывод, программист может указать масштаб (пикселы/дюймы/миллиметры), а также начало координат и направление масштабирования.

В дополнение к этим трем свойствам GDI включает в себя все "навороты", какие только можно ожидать от полномасштабной графической библиотеки – от примитивных графических функций типа **SetPixel(...)** (Задать пиксель) до сложных функций управления растром и цветом.

Все это остается за кадром, в недрах ***Windows*** и ***Visual C++***. Мы работаем с контекстом устройства **DC**, который, в свою очередь, выполняет три различных роли.

1. Он обеспечивает логическую связь с физическим устройством (через драйвер).
2. Он предоставляет пользователю набор изобразительных инструментов.
3. Он выполняет роль сторожа, следя за тем, чтобы Ваша программа не «наехала» своим окном на другие окна, когда она неактивна или когда Вы пытаетесь нарисовать что-либо за пределами своего окна.

Контекст устройства в MFC может быть четырех видов: CWindowDC, CMetaFileDC, CPaintDC и CClientDC. Большинство рисунков создаются с использованием класса **CPaintDC** или класса **CCli-**

entDC. Каждый из них позволяет рисовать в любом месте в пределах окна. ***В классе CClientDC позиция 0,0 соответствует верхнему левому углу клиентской области окна, но не фактическому положению на экране***. Именно поэтому мы используем этот вид контекста устройства в нашей программе OnDraw. Если Вы хотите рисовать на всем окне, включая полосу заголовка и обрамление окна, то используйте класс CWindowDC. Для него позиция 0,0 соответствует верхнему левому углу окна, а не клиентской области.

Посмотрите на введенный код программы **OnDraw()**. В объявлении контекста устройства аргумент (**this** – *это*) означает, что контекст принадлежит данной программе и обеспечивает вывод на дисплей. Затем мы объявляем объект «кисть» и создаем сплошную кисть с помощью функции **CreateSolidBrush(…)**. Последняя введенная нами строка использует функцию контекста устройства для раскраски нашего холста (то, что холст наш, свидетельствует первый аргумент метода **FillRect(…)**). Теперь порисуем на холсте, чтобы почувствовать, как это делается.

Хотя художники рисуют всем, что оставляет след, в нашем случае перед рисованием нужно создать перо. Мы создадим несколько перьев, чтобы рисовать линии разного цвета и разной толщины. Добавьте в конец программы **OnDraw()** следующие строки.

```
CPen m_Pen1, m_Pen2, m_Pen3;//Объявить перья
m_Pen1.CreatePen(PS_SOLID, 3, RGB(255, 0, 0)); //Красное
                          // перо толщиной 3 пикселя
m_Pen2.CreatePen(PS_SOLID, 1, RGB(0, 0, 255)); //Синее
                          // перо толщиной 1 пиксель
m_Pen3.CreatePen(PS_SOLID, 2, RGB(0, 0, 0)); //Черное
                          // перо толщиной 2 пикселя

dc.SelectObject(&m_Pen1);        //Выбрать перо 1
dc.MoveTo(m_Holst.left+20,m_Holst.top+20); //Подвести
                  // перо в точку с координатами x,y
dc.LineTo(m_Holst.left+120,m_Holst.top+20);//Опустить
                  // перо на холст и провести линию

dc.SelectObject(&m_Pen2);        //Выбрать перо 2
dc.MoveTo(m_Holst.left+20,m_Holst.top+20);//Подвести
                  // перо в точку с координатами x,y
dc.LineTo(m_Holst.left+120,m_Holst.top+120);  //Опустить
```

```
                    // перо на холст и провести линию

dc.SelectObject(&m_Pen3);          //Выбрать перо 3
dc.MoveTo(m_Holst.left+20,m_Holst.top+20);//Подвести
              // перо в точку с координатами x,y
dc.LineTo(m_Holst.left+20,m_Holst.top+120);//Опустить
                    // перо на холст и провести линию
```

Выполните программу. Нажмите кнопку «Чертить график». На холсте должны получиться три линии из одной точки, расположенной на 20 пикселей правее и ниже левого верхнего угла холста. Аргументами для функций **MoveTo(…)** и **LineTo(…)** являются координаты **x, y** точки на экране, разделенные запятой. Наш холст после инициализации в программе **OnInitDialog()** получил реальное местоположение на экране, поэтому отсчет идет от него.

m_Holst.left – значение координаты «**x**» *левого верхнего* угла.
m_Holst.top – значение координаты «**y**» *левого верхнего* угла.
m_Holst.right – значение координаты «**x**» *правого нижнего* угла.
m_Holst.bottom – значение координаты «**y**» *правого нижнего* угла.

Теперь «удалите» из программы строку **dc.MoveTo(m_Holst.left+20,m_Holst.top+20);** для ***второго*** пера, сделав ее комментарием (введите // в начало строки). Выполните программу. Что получилось? Почему?

Теперь «удалите» вдобавок еще и строку **dc.MoveTo(m_Holst.left+20,m_Holst.top+20);** для ***третьего*** пера. Прямоугольник повернулся… Почему?

Функция **LineTo(…)** рисует линию из *текущей точки в заданную*, поэтому при рисовании ломаной не нужно подводить перо к началу каждого отрезка - достаточно сделать это один раз: подвести перо к началу *первого* отрезка. Эта особенность нам пригодится при рисовании графика.

Первый аргумент функции **CreatePen(…)** задает стиль пера. На выбор MFC предоставляет семь стилей. Наиболее распространены следующие четыре стиля.

PS_SOLID – сплошная линия.
PS_DASH – пунктирная линия.
PS_DOT – линия из точек.
PS_DASHDOT – штрихпунктирная линия.

Использование перьев опасно для здоровья.
Функция **CPen::CreatePen()** выполняет инициализацию пера, но не присвоение новых значений, поэтому **перо можно создавать только один раз**.
Повторный вызов CreatePen(…) для уже созданного пера приводит к краху программы.

Поэкспериментируйте с перьями. Попытайтесь нарисовать оси координат для будущего графика. Подберите сочетание цвета фона и цвета перьев. Сделайте это красиво.

10. Вычерчивание графика функции

Войдите в закладку Resource View, разверните папку Dialog и дважды щелкните по IDD_TABFUN_DIALOG. В открывшемся окне справа щелкните левой кнопкой мыши по границе нашего холста и когда появится прямоугольник выделения, посмотрите в правый нижний угол экрана Visual C++. Найдите там нечто, похожее на рис. 21.

Рис. 21. Позиция и размер элемента диалога

В левом окошке показываются координаты **x, y** верхнего левого угла нашего холста относительно левого верхнего угла главного окна программы, а правое окошко содержит размеры нашего холста. То есть, мы должны рисовать график на поле размером 430 на 192 растровых единиц (пикселей). (Естественно, размеры Вашего холста могут быть другими). А если пользователь введет интервал табулирования, например: от -150 до 420?!? А если диапазон изменения функции на заданном интервале превысит значение 192?!? Как должна вести себя программа???

Давайте будем дружелюбны к пользователю и сделаем автоматический выбор масштаба, чтобы график функции на введенном ин-

тервале табулирования ***гарантированно*** помещался на нашем холсте.

Сначала определим максимальное и минимальное значение функции на интервале, который задал пользователь программы. Для этого надо выполнить следующие шаги.

1. Переслать данные из полей редактирования в память переменных ***m_Begin***, ***m_End*** и ***m_Step***. Для этого служит программа **UpdateData(…)** с параметром **TRUE**. Если задать параметр **FALSE**, то данные будут пересылаться из памяти для переменных в поля редактирования.
2. Задать очень большое положительное число для **Y_min**, чтобы первое же вычисленное значение функции на интервале было меньше этого числа.
3. Задать очень большое отрицательное число для **Y_max**, чтобы первое же вычисленное значение функции на интервале было больше этого числа.
4. Организовать цикл, в котором **x** принимает значения от ***m_Begin*** до ***m_End*** с шагом ***m_Step***. Внутри цикла вычислить $Y = 5.0 \times \sin(x) + 0.1 \times x^2$ и провести сравнения: 1) если **Y** < **Y_min**, то **Y_min** присвоить значение **Y**; 2) если **Y** > **Y_max**, то **Y_max** присвоить значение **Y.**

При выходе из цикла **Y_min** и **Y_max** будут равны минимальному и максимальному значению функции на интервале табулирования. Написание кода этого фрагмента программы не должно вызвать у Вас трудностей. Отметим лишь, что для корректного использования математических функций необходимо подключить заголовочный файл **math.h**, для чего нужно перейти в начало файла **TabFunDlg.cpp**, который содержит программу **OnDraw()** и после строк

```
#include "stdafx.h"
#include "TabFun.h"
#include "TabFunDlg.h"
```

добавить строку

```
#include <math.h>
```

Масштабы **Mx** и **My** соответственно по осям **x** и **y** вычисляются следующими строками программы.

```
    //Вычисление масштабов
Mx = (double)(m_Holst.right - m_Holst.left - 40) /
              (m_End - m_Begin);
My = (double)(m_Holst.bottom - m_Holst.top - 40) /
              (Y_max - Y_min);
    //Предосторожность на всякий случай
    if (Mx < 0.0) Mx = -Mx;
    if (My < 0.0) My = -My;

//*******************************************************
    if (Mx < My) My = Mx; else Mx = My; //Сделать
                        // одинаковым масштаб по осям X и Y
//*******************************************************
```

Идея вычисления проста: масштаб по оси **x** (**y**) равен отношению уменьшенной на 40 пикселей ширины (высоты) холста к диапазону изменения аргумента (значения) функции. Почему ширину или высоту холста надо уменьшать? Да просто потому, чтобы график не касался границ холста. Мы сделаем «поля» по 20 пикселей слева, справа, сверху и снизу. Одна строка программы выделена комментариями со звездочками. Она вызывает сомнение и дальше мы увидим почему.

Измените толщины перьев. Сделайте красное перо толщиной 2 пикселя, синее перо толщиной 3 пикселя и черное перо толщиной 1 пиксель. Черным пером нарисуйте оси координат: ось X расположите горизонтально на 20 пикселей *выше* ***m_Holst.bottom*** и проведите линию от (**m_Holst.left + 20**) до (**m_Holst.right – 20**); ось Y расположите вертикально на расстоянии 20 пикселей *правее* **m_Holst.left** и проведите линию от (**m_Holst.bottom – 20**) до (**m_Holst.top + 20**). Все Ваши эксперименты с перьями «удалите», добавив знак комментария в начало «удаляемой» строки.

Теперь можно писать код черчения графика. Попробуйте сделать это самостоятельно. Не получается? Ошибок компилятор не обнаруживает, но график не рисуется? Наверное, Вы забыли перенести начало координат или умножить на масштаб… Попробуйте еще. Опять не получается? Тогда читайте дальше.

Код для вычерчивания графика прост, но надо учесть, что ось Y направлена вниз и что надо совместить начало координат по горизонтали с началом диапазона табулирования **m_Begin**, а по вертикали – со значением **Y_min**. Это делается так.

```
    //Вычерчивание графика

dc.SelectObject(&m_Pen1); //Выбираем перо

for (x = m_Begin; x <= m_End; x += m_Step)
{
    Y = 5.0 * sin(x) + 0.1 * x * x;  //Функция
                                     // для табулирования

    //Масштабирование и перенос координат
xr = (int)((x - m_Begin) * Mx) + m_Holst.left + 20;
yr = (int)((Y - Y_min) * My) - m_Holst.bottom + 20;

    yr = -yr;       //Ось Y холста направлена ВНИЗ !!!

    if (x == m_Begin) dc.MoveTo(xr, yr);//Подводим
                            // перо в самую первую точку
       dc.LineTo(xr, yr);  //Рисуем отрезок прямой
                            // в текущую точку
}
```

Выполните программу. Для принятой функции получится график, показанный на рис. 22.

Не нравится? Слишком много пустого места? Зато одинаковый масштаб по осям X и Y. А если сделать его разным? Превратите в комментарий «сомнительную» строку между строками со звездочками и снова выполните программу. График изменился, см. рис. 23; эстетически он стал лучше. График заполнил все отведенное ему поле как по координате X, так и по координате Y. Но что предъявить пользователю?

График правильный, но некрасивый (рис. 22), или красивый, но неправильный (рис. 23) ?!?

А пусть пользователь решает сам!?! И мы дадим ему такую возможность.

Рис. 22. График функции Y = 5.0 * sin(x) + 0.1 * x * x.
Одинаковый масштаб по осям

Рис. 23. График функции Y = 5.0 * sin(x) + 0.1 * x * x.
Разный масштаб по осям

Войдите в закладку Resource View, разверните папку Dialog и дважды щелкните по IDD_TABFUN_DIALOG. В открывшемся окне редактирования ресурсов поднимите кнопку «Чертить график» до положения, как на рис. 23. Справа от нее разместите Radio Button (Переключатель), взяв мышью с панели управления. В окне свойств переключателя измените ID на **IDC_SCALE**, а Caption (Заголовок) на «**Одинаковый масштаб**». Если не хватает места для текста, то растяните прямоугольник выделения. В окне свойств отметьте флажки Visible и Group. Окно должно принять вид, показанный на рис. 24.

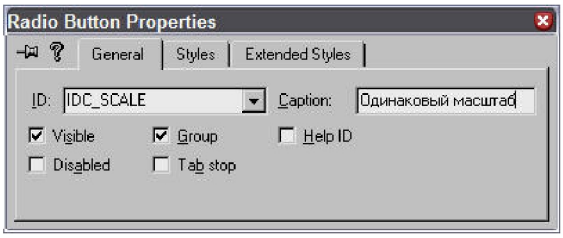


Рис. 24. Задание ведущей радио-кнопки

Перетащите с панели управления второй переключатель (Radio Button) и расположите его под первым. Войдите в окно свойств и измените *только* Caption (Заголовок) на «Разный масштаб». ***Проследите, чтобы флажок* Group *был сброшен***. Окно свойств второго переключателя должно иметь вид, показанный на рис. 25.

Щелкните правой кнопкой мыши по ***первому*** переключателю. Войдите в меню ClassWizard (Мастер классов), в закладке Member Variables (Переменные члены) выделите **IDC_SCALE** и щелкните по кнопке Add Variable (Добавить переменную). В открывшемся окне введите имя переменной **m_Scale** в поле Member variable name (Имя переменной члена) **Value** в поле Category и **int** в поле Variable type.

После того как окно примет вид, показанный на рис. 26, нажмите кнопки OK в окне свойств и в окне ClassWizard.

В рабочей области проекта Visual C++ войдите в закладку ClassView, разверните класс CTabFunDlg и убедитесь, что появилась переменная **m_Scale**. Дважды щелкните левой кнопкой мыши по

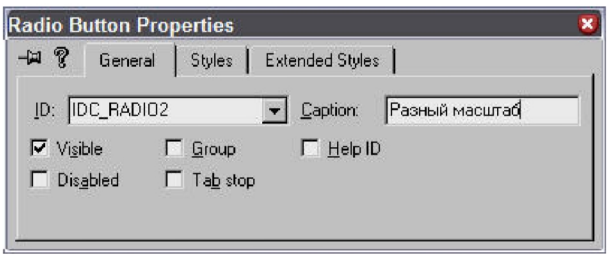


Рис. 25. Задание ведомой радио-кнопки

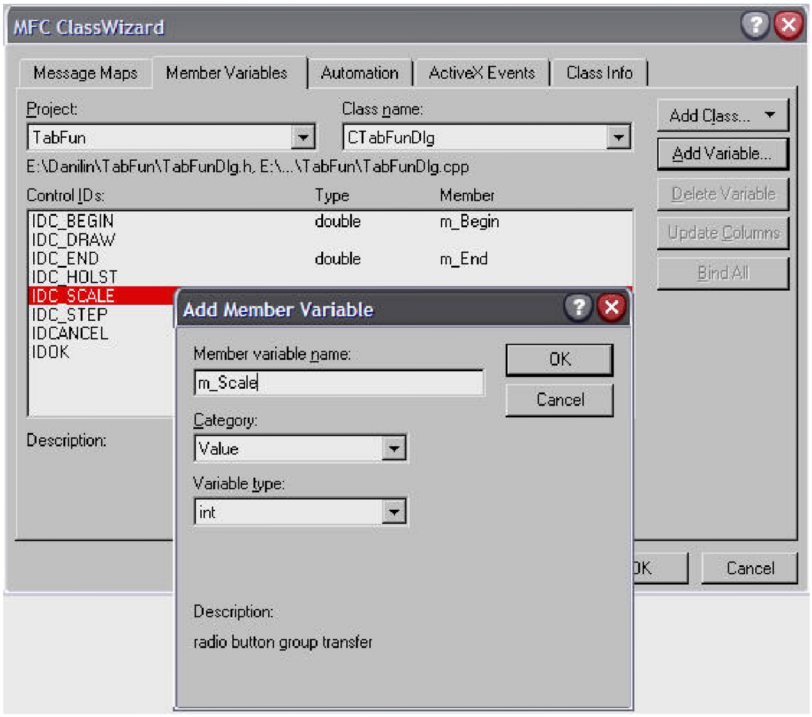


Рис. 26. Задание переменной для радио-кнопки

конструктору CTabFunDlg и перед закрывающейся фигурной скобкой введите строку

```
m_Scale = 0;
```

Войдите в программу OnDraw(). Модернизируйте «сомнительную» строку. Код между строками комментариев со звездочками должен выглядеть так.

```
//*****************************************************
if (m_Scale == 0) {        // Если утоплена кнопка
                           // "Одинаковый масштаб", то
    if (Mx < My) My = Mx; else Mx = My; //делаем его
                           // таковым по осям X и Y
}
//*****************************************************
```

Выполните программу. По кнопке «Чертить график» рисуется график либо с одинаковым, либо с разными масштабами по осям X и Y.

На этом работа над программой OnDraw() не закончена. Как она будет работать, если пользователь введет шаг табулирования равный нулю или отрицательный? Что будет, если начало интервала табулирования больше, чем его конец? А что должна сделать программа, если (Y_min == Y_max) и при выборе масштаба нужно будет делить на нуль?

Подобные ситуации, мешающие нормальной работе программы, называются ***побочными эффектами*** и они также должны проектироваться и программироваться. Напишите код для побочных эффектов, информируя пользователя о нештатной ситуации с помощью уже известной Вам программы

```
MessageBox("Мы находимся внутри OnDraw","ОТЛАДКА",
           MB_OK);
```

и инструктируя пользователя, в чем он неправ, и как должно быть. Текст из первой строки **MessageBox(…)** помещается в диалоговое окно, а из второй строки – в заголовок диалогового окна. Параметр MB_OK организует кнопку выхода из диалогового окна.

Поработайте с программой. Попробуйте разные диапазоны табулирования, разные значения шага и разный тип масштабирования. Попытайтесь послать программу в нокаут. Если это Вам удастся, то

проанализируйте ситуацию и напишите код для найденного побочного эффекта.

11. Вывод таблицы значений

Остался заключительный этап нашей программы – выдать таблицу значений функции. Давайте посмотрим на рис. 23 и определим, сколько строк будет содержать таблица для заданного интервала и шага табулирования (будем считать, что в строке содержатся два значения: X и Y). Нетрудно подсчитать, что в таблице будет 900 строк. А если шаг табулирования задать равным 0.001, то количество строк увеличивается до 9000 ! Где эту информацию хранить и в каком виде выдавать?

Пожалуй, единственным способом решения такой задачи является выдача информации порциями и предоставление пользователю возможности просмотреть следующую порцию данных или вернуться к предыдущей. Мы тоже пойдем по этому пути.

Войдите в закладку Resource View в рабочей области проекта, разверните папку Dialog и дважды щелкните по IDD_TABFUN_DIALOG. В открывшемся справа окне перетащите с панели управления элемент Button (Кнопка). Расположите ее ***под*** кнопкой «Чертить график». Войдите в окно свойств новой кнопки, измените ID на **IDC_TABLE**, а Caption (Заголовок) на «**Выводить таблицу**». Если не хватает места для текста, то растяните прямоугольник выделения и выравните кнопку по левой границе кнопки «Чертить график». Для новой кнопки вызовите ClassWizard (Мастер классов) и в закладке Message Maps (Карты сообщений) выделите **IDC_TABLE** и **BN_CLICKED**. Окно Message Maps должно принять вид, показанный на рис. 27. Щелкните по кнопке **AddFunction** (Добавить функцию) и в раскрывшемся окне Add Member Function (Добавить функцию-член) нажмите кнопку **OK**. Щелкните по кнопке **Edit Code** (Редактировать код) и Вы входите в тело созданной функции OnTable.

Эта функция должна инициировать процесс выдачи таблицы, который мы организуем следующим образом.

1. Введем переменную **m_Current**, которая будет отслеживать текущее начало данных в выдаваемой таблице внутри интервала табулирования. Эту переменную придется объявить вручную.

Войдите в закладку Class View в рабочей области проекта и дважды щелкните по ***классу*** CTabFunDlg. В объявление класса добавьте «приватную» (после метки ***private:***) переменную типа ***double***. Добавьте также переменную типа ***CFont*** с именем ***m_Font*** и переменную типа ***double*** с именем ***m_InString***, которые нам понадобятся в дальнейшем.

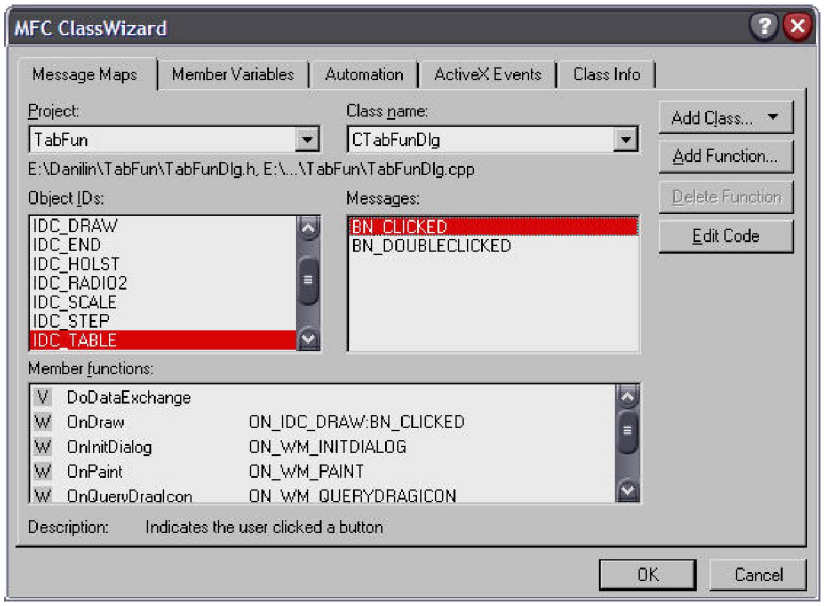


Рис. 27. Задание обработчика нажатия кнопки выдачи таблицы

Переменная ***m_Font*** будет определять шрифт, а ***m_InString*** – количество чисел в строке вывода. Фрагмент объявления класса ***CTabFunDlg***() должен выглядеть так.

```
private:
    CRect m_Holst;  //Переменная, определяющая холст
    double m_Current,//Текущая позиция порции таблицы
       m_InString; //Количество чисел в строке вывода
    CFont m_Font;   //Шрифт для вывода
```

2. В функции **OnTable()** присвоим переменной ***m_Current*** значение ***m_Begin*** и вызовем функцию выдачи на экран порции таб-

лицы, код которой мы должны написать сами. Назовем эту функцию **OutTable()**. Все функции надо объявлять, поэтому вслед за объявлением переменной ***m_Font*** в объявлении класса ***CTabFunDlg***() введите строку

```
void OutTable();     //Программа выдачи порции таблицы
```

Теперь можно писать код функции **OnTable()**. Он до примитивизма прост.

```
void CTabFunDlg::OnTable()
{
    UpdateData(TRUE);// Если пользователь захочет
                     // выдать сначала таблицу
    m_Current = m_Begin;
    OutTable();// Вызов программы выдачи порции таблицы

}
```

3. Оставим пока за кадром функцию **OutTable()** и обсудим, что будет делать наша программа после выдачи на экран порции таблицы. Очевидно, она должна давать возможность пользователю продвигаться ***вперед и назад*** по интервалу табулирования. Поэтому в нижней части диалогового поля, у правой границы, на уровне кнопки «Выход» создайте еще две кнопки: 1) с ID равным **IDC_BACK** и заголовком (Caption) «**Назад**»; 2) с ID равным **IDC_FORWARD** и заголовком (Caption) «**Вперед**». В карте сообщений для IDC_FORWARD и сообщения (Message) BN_CLICKED добавьте функцию **OnForward()**, а для IDC_BACK и также сообщения BN_CLICKED добавьте функцию **OnBack()**.

Теперь сценарий работы с нашей программой будет выглядеть так. Пользователь щелкает по кнопке «Вывести таблицу» и ему выдается первая порция данных. После анализа он щелкает по кнопке «Вперед» или «Назад» и анализирует следующую или предыдущую порцию.

Для того чтобы проверить работоспособность сделанного, не хватает тела функции **OutTable()**. Добавьте после текста программ **OnForward()**, **OnBack()** пустую функцию **OutTable()** в виде следующих четырех строк.

```
void CTabFunDlg::OutTable()
{
//Программа выдачи порции таблицы на экран
}
```

Запустите программу на выполнение, чтобы посмотреть на организацию диалога. Подправьте размеры и расположение кнопок по Вашему вкусу и наберитесь терпения перед предпоследним усилием (тяжелый вздох обязателен…).

Когда происходит выдача на экран, принтер, фотонаборное или какое-либо еще устройство визуального представления информации, то внутрикомпьютерное представление числа или символа должно быть преобразовано в код, по которому это устройство нарисует понятное нам изображение. Мы уже обсуждали, что такое контекст устройства и зачем он нужен, здесь же отметим, что ***при выдаче числовой информации в контекст устройства ее нужно предварительно преобразовать в набор символов***, который и будет изображен.

Прежде чем переходить непосредственно к выводу, введите код программ **OnForward()** и **OnBack**(). Он не должен вызывать затруднений и представлен следующими строками.

```
void CTabFunDlg::OnForward()
{
    UpdateData(TRUE);
    m_Current += m_InString * m_Step;
    if (m_Current > m_End) m_Current = m_End -
                          (m_InString-1.0) * m_Step;
       OutTable();
    }
    //================================================
    void CTabFunDlg::OnBack()
    {
       UpdateData(TRUE);
       m_Current -= m_InString * m_Step;
       if (m_Current < m_Begin) m_Current = m_Begin;

           OutTable();
    }
    //================================================
```

Выполните еще несколько подготовительных действий.

Добавьте в объявление ***класса CTabFunDialog*** «приватную» (после метки ***private:***) переменную **m_String** типа **CString**. Класс CString позволяет оперировать со строками переменной длины, динамически удлиняя или укорачивая их без участия программиста и без потери информации.

В ***конструктор*** класса **CTabFunDialog**(…) добавьте строку создания шрифта для вывода таблицы.

```
m_Font.CreateFont(16,0,0,0,800,0,0,0,0,0,0,0,0,
                  "Courier");
```

Шрифты, как известно, бывают пропорциональными (типа Arial, Times…), когда ширина изображения зависит от выводимого символа (например, символ «**:**» и «**S**» занимают разное по ширине место); и моноширинными, когда ширина, занимаемая символом, - постоянна. Если мы хотим, чтобы начало (или конец) числа в двух строках совпадали по вертикали, то нужно использовать моноширинный шрифт, типа Courier. Функция CreateFont() создает шрифт и содержит много параметров, позволяющих задать и наклон строки, и ширину символа, и тип (нормальный, курсив или жирный), и наклон буквы к наклону строки, и… Если для параметра указан нуль, то принимается значение по умолчанию. Последний параметр в виде строки указывает, ***ближе к какому стилю*** создавать шрифт. Не факт, что Вы получите непосредственно шрифт Courier, но то, что это будет моноширинный шрифт, - это точно.

Следом за созданием шрифта задайте начальное значение количества чисел в строке вывода таблицы:

```
m_InString = 7;
```

Для Вашего размера диалогового окна значение **m_InString** может быть другим.

11.1. Первый вариант вывода (простой и непрофессиональный)

Фрагмент таблицы будем рисовать в виде двух строк: верхняя содержит значения аргумента X; нижняя – значения функции Y. Эти

две строки расположим в нижней части окна, см. рис. 23, на всю ширину окна между холстом для графика и кнопкой «Выход».

Введите текст программы OutTable(), приведенный ниже. Назначение операторов ясно из комментариев.

```
    void CTabFunDlg::OutTable()
    {
    //Программа выдачи порции таблицы на экран

double Xt[10], Yt[10], dx;// Размер массивов
                          //Xt[] и Yt[] должен быть
int i;                    // не меньше значения m_InString

    CClientDC dc(this);   //Объявить и создать контекст
                          // устройства
    dc.SelectObject(&m_Font);    //Выбрать шрифт для выдачи
                                 // на экран
for(i = 0; i < 10; i++) Xt[i] = Yt[i] = 0.0; //"Свои" нули

    dx = 0.0;
    for(i = 0; i < m_InString; i++)
    {
    //Заполнение порции таблицы
       Xt[i] = m_Current + dx;
       if (Xt[i] > m_End) { Xt[i] = 0.0; break; }
    //Функция
       Yt[i] = 5.0 * sin(Xt[i]) + 0.1 * Xt[i] * Xt[i];
       dx += m_Step;
    }

    //Выдача порции таблицы в две строки
    //Строка X
m_String.Format(
" : X :%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e: ",
Xt[0],Xt[1],Xt[2],Xt[3],Xt[4],Xt[5],Xt[6]);
dc.TextOut(m_Holst.left+2, m_Holst.bottom + 22, m_String);
    //Строка Y
    m_String.Format(
" : Y :%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e: ",
Yt[0],Yt[1],Yt[2],Yt[3],Yt[4],Yt[5],Yt[6]);
dc.TextOut(m_Holst.left+2, m_Holst.bottom + 38, m_String);

    }
```

Дадим некоторые пояснения. Класс контекста устройства имеет в своем составе функцию выдачи текста **TextOut(…)**. Первые два параметра – это **x** и **y** координаты начала строки текста в диалоговом окне, а третий параметр – сама строка. В приведенном тексте программы сделана «привязка» к левой и нижней границе холста, который использовался для рисования графика. Строка **m_String**, которая выдается на экран, формируется до TextOut(…) с помощью функции **Format()** класса **CString**. Функция Format() работает аналогично функции sprintf(…) классического языка С. Краткое описание форматов для выдачи числовых данных с помощью функции Format(…) приведено в Приложении 1.

Выполните программу. Поработайте кнопками «Вперед», «Назад». Проверьте, как ведет себя программа при достижении границ интервала табулирования. Есть ли побочные эффекты? Если да, то попытайтесь устранить их.

11.2. Второй вариант вывода (профессиональный и тоже простой)

Поработав с предыдущим вариантом, можно заметить сильное неудобство, а именно: при малом шаге табулирования очень трудно добраться до числовых значений функции в районе середины интервала, хотя получаются гладкие хорошие графики. Можно, конечно, изменить интервал табулирования, но это уже «пользователь для программы, а не программа для пользователя».

Давайте так же, как в предыдущем варианте, выдадим **m_InString** пар значений аргумента и функции в двух строках, то таких пар строк выдадим двадцать пять штук и организуем вертикальную прокрутку этих строк.

Библиотека MFC предлагает для этих целей отлаженный инструмент. Войдите в закладку Resource View в рабочей области проекта, разверните папку Dialog и дважды щелкните по IDD_TABFUN_DIALOG. В окне справа перетащите с панели управления элемент **List Box** (Список) и разместите его в нижней части Вашего диалогового окна между холстом для рисования графика и кнопкой «Выход». Растяните прямоугольник выделения элемента

«список» на всю ширину окна и немного увеличьте его по высоте. Откройте свойства элемента управления «список». В закладке General (Общие) измените ID списка на IDC_LIST_TABLE. Позаботьтесь о том, чтобы флажки Tab stop и Visible были отмечены, а все остальные сброшены. В закладке Styles (Стили) выберите Single (Одиночный) из выпадающего списка Selection (Выбор) и No из списка Owner Draw (Рисование владельца). Отметьте флажки Border (Бордюр), Notify (Извещать), Vertical Scroll (Вертикальная прокрутка) и No Integral Hight (Не учитывать общую высоту).

Все остальные сбросьте. Результат установок показан на рис. 28. При помощи ClassWizard (Мастера классов) добавьте новую переменную для списка IDC_LIST_TABLE.

Присвойте этой переменной имя **m_TableString** и установите для нее Category в **Control**, а Type в **CListBox**. Visual C++ не может сам заполнять поля списка, - для этого служит функция **AddString()**, однако он берет на себя проблемы, связанные с организацией прокрутки списка. Новый код для функции **OutTable** выглядит так.

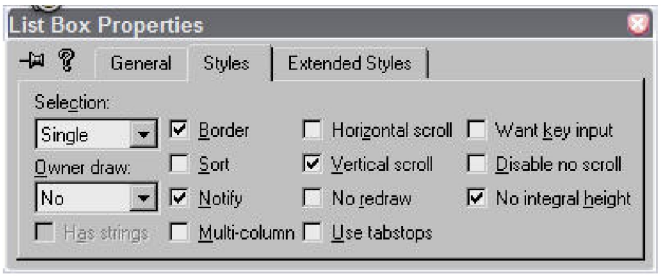


Рис. 28. Свойства списка

```
//=======================================
void CTabFunDlg::OutTable()
{
//Программа выдачи порции таблицы на экран
double Xt[10], Yt[10], dx;   // Размер массивов
                             // Xt[] и Yt[] должен быть
int i, j;              // не меньше значения m_InString

m_TableString.SetFont(&m_Font,TRUE); //Задать шрифт

//Очистить поле списка: удалить 25 раз по три строки
for(j = 0; j < 75; j++) m_TableString.DeleteString(0);
```

```
    dx = 0.0;
    for(j = 0; j < 25; j++) //Двадцать пять по три строки
    {
       for(i = 0; i < 10; i++) Xt[i] = Yt[i] = 0.0;
       for(i = 0; i < m_InString; i++)  // m_InString чисел
                                        // в строке
       {
              //Заполнение порции таблицы
              Xt[i] = m_Current + dx;
              if (Xt[i] > m_End) { Xt[i] = 0.0; break; }
    //Функция
    Yt[i] = 5.0 * sin(Xt[i]) + 0.1 * Xt[i] * Xt[i];
              dx += m_Step;
       }      //Конец цикла по i - значениям в строке

       //Выдача порции таблицы в две строки
    m_String.Format(
": X :%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e: ",
Xt[0],Xt[1],Xt[2],Xt[3],Xt[4],Xt[5],Xt[6]);
    m_TableString.AddString(m_String);  //Строка значений X

m_String.Format(
": Y :%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e: ",
Yt[0],Yt[1],Yt[2],Yt[3],Yt[4],Yt[5],Yt[6]);
    m_TableString.AddString(m_String);  //Строка значений Y
m_String.Format(
":-----------------------------------------------: ");
    m_TableString.AddString(m_String);  //Добавить строку
                                        // разделитель
       }      // Конец цикла по j - строкам

    }
    //================================================
```

Однако этих изменений недостаточно. Теперь продвижение по кнопкам «Вперед», «Назад» должно происходить на более дальнюю дистанцию. Поэтому код для функций **OnForward** и **OnBack** тоже изменяется.

```
    //================================================
    void CTabFunDlg::OnForward()
    {
       UpdateData(TRUE);
       m_Current += m_InString * m_Step * 25.0;
       if (m_Current > m_End)
    m_Current = m_End - (m_InString - 1.0)*25.0*m_Step;
       OutTable();
    }
```

```
//=============================================
void CTabFunDlg::OnBack()
{
   UpdateData(TRUE);
   m_Current -= m_InString * m_Step * 25.0;
   if (m_Current < m_Begin) m_Current = m_Begin;
   OutTable();
}
```

Выполните программу. Итоговое диалоговое окно показано на рис. 29.

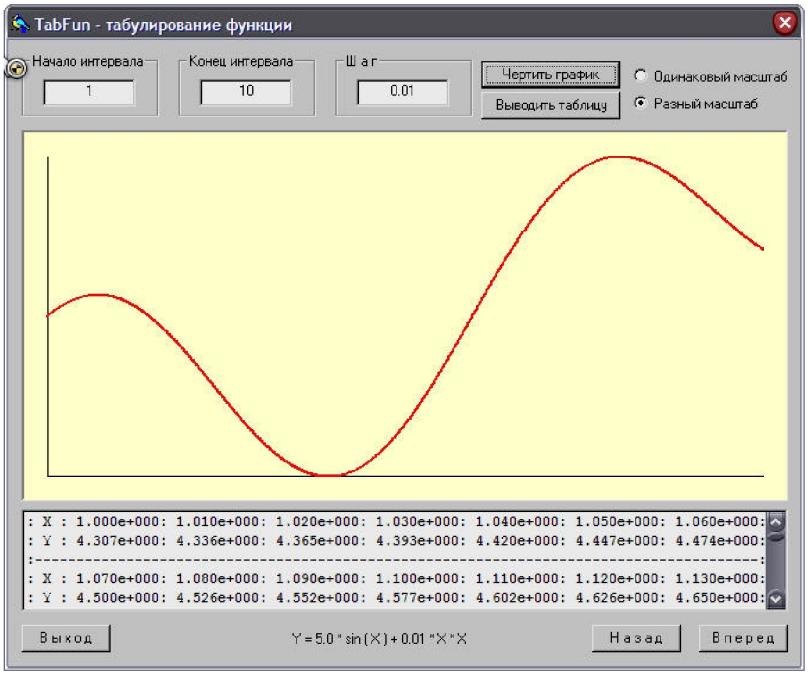


Рис. 29. Работа программы табулирования функции

Поработайте кнопками «Вперед», «Назад», полосой вертикальной прокрутки. Проверьте, как ведет себя программа при достижении границ интервала табулирования. Есть ли побочные эффекты? Если да, то попытайтесь устранить их.

Доработайте программу. Нарисуйте стрелки и обозначения осей, напишите на графике числовые значения диапазона табулирования и границы изменения функции. Не забудьте об инструкции для пользователя в окне About… (О программе…). Попытайтесь пометить на графике текущий участок интервала, который отображен в порции таблицы.

Теперь Вы все это можете…

В качестве упражнения в создании еще одного диалогового приложения создадим простейший графический редактор, чтобы прочувствовать технику программирования графики в таких распространенных программах, как PaintBrush и PhotoShop.

12. Рисуем на дисплее, как карандашом

1. Создайте диалоговое приложение в соответствии с материалом, изложенным ранее.
2. Уберите кнопку «OK». Кнопке «Cancel» дайте заголовок «Выход», оставив ID: IDCANCEL.
3. Добавьте на рабочее поле кнопку «Очистить»: ID: IDC_CLEAR. Добавьте обработчик нажатия кнопки под именем OnClear.
4. Оформите диалоговое окно About (О программе).
5. Создайте холст с IDC_HOLST и именем **m_Holst**, как Вы делали это при программировании задания по табулированию функции. Инициализируйте его, добавив вручную в программу **OnInitDialog** следующие строки (перед оператором ***return;***)

```
CWnd* pHolst = GetDlgItem(IDC_HOLST);
   pHolst->GetWindowRect(&m_Holst);
   ScreenToClient(&m_Holst);
   m_Holst.DeflateRect(2,2,1,1);
```

6. Щелкните правой кнопкой мыши по классу **C…..Dlg** и в выпадающем списке щелкните по Add Member Variable... Добавьте Private переменную типа bool с именем **m_IsDrawing**. Добавьте также Private переменную типа CPoint с именем **m_PointStart**, которая понадобится нам в пункте 12.
7. Перед кодом программы пункта 5 добавьте строку **m_IsDrawing = false;** Переменная **m_IsDrawing** будет определять, где нахо-

дится курсор, когда нажата левая кнопка мыши. Если внутри холста, то **m_IsDrawing** будет равна true, если вне холста, то false. Присвоение значений переменной **m_IsDrawing** происходит после вызова функции класса **CRect PtInRect(point)** и анализа результатов ее работы. Где и как вызывать эту функцию, мы увидим далее, в пункте 12. Сейчас важно, чтобы текст, добавляемый в **OnInitDialog,** принял окончательный вид

```
m_IsDrawing = false;
CWnd* pHolst = GetDlgItem(IDC_HOLST);
   pHolst->GetWindowRect(&m_Holst);
ScreenToClient(&m_Holst);
m_Holst.DeflateRect(2,2,1,1);
```

8. Добавьте в обработчик нажатия кнопки «Очистить» следующие строки:

```
CClientDC dc(this);
dc.SelectStockObject(NULL_PEN);
dc.Rectangle(m_Holst);
```

9. Нажмите бордовый восклицательный знак, чтобы проверить, что все работает и при нажатии на кнопку «Очистить» холст закрашивается белым цветом.

Теперь обратимся к способу работы Windows, а именно: рассмотрим как срабатывают сообщения, поступающие от мыши.

При перемещении мыши или выполнении щелчков на ее кнопках Windows генерирует одно или большее число сообщений и пересылает их в окно, внутри которого в данный момент находится курсор мыши. Существует 20 разных сообщений мыши, и столь большое их количество вызывает некоторое беспокойство. Однако сообщения мыши не являются такими сложными, какими они кажутся на первый взгляд. Windows проводит различие между действиями мыши, совершаемыми ею в клиентской области, и действиями, которые она совершает в области, отличной от клиентской, - наподобие заголовка окна или обрамления. Половина сообщений мыши соответствует событиям, имеющим место в неклиентской облас-

ти — за исключением нескольких специальных ситуаций, их можно игнорировать.

Десять оставшихся сообщений, относящихся к клиентской области, включают, в числе прочих, сообщение WM_MOUSEMOVE, которое посылается в окно при перемещении указателя мыши в клиентской области окна. Если переместить указатель мыши на элемент управления Windows, например, на какую-либо кнопку, Windows генерирует сообщения WM_MOUSEMOVE, **однако эти сообщения не пересылаются в окно приложения, они пересылаются на кнопку, которая осуществляет их внутреннюю обработку.**

Каждое из девяти оставшихся сообщений мыши, относящихся к клиентской области, генерируется при нажатии кнопки мыши. При нажатии на кнопку, отпускании кнопки и двойном щелчке по кнопке генерируются различные сообщения. С каждой из трех кнопок мыши ассоциированы три упомянутых сообщения, так что всего получается девять сообщений.

В случае нажатия какой-либо кнопки мыши генерируется сообщение WM_LBUTTONDOWN, WM_RBUTTONDOWN или WM_MBUTTONDOWN, в зависимости от того, какая кнопка нажата — Left (левая), Right (правая) или Middle (средняя). (Если у вашей мыши нет средней кнопки, вы никогда не получите набора сообщений, начинающихся с "WM_M".)

При отпускании кнопки мыши генерируется сообщение WM_LBUTTONUP, WM_RBUTTONUP или WM_MBUTTONUP. Если дважды быстро нажать и отпустить кнопку мыши, ожидаемая пара сообщений DOWN/UP не возникнет. Предположим, что этот эксперимент проводится с левой кнопкой мыши. Windows заменит второе сообщение WM_LBUTTONDOWN на сообщение WM_LBUTTONDBLCLK. Программа получит такую последовательность сообщений:

WM_LBUTTONDOWN
WM_LBUTTONUP
WM_LBUTTONDBLCLK
WM_LBUTTONUP

Когда пользователь программы нажимает кнопку мыши на холсте, это означает желание нарисовать что-либо внутри холста. Для обработки этого нажатия потребуется перехватить сообщение **WM_LBUTTONDOWN.** Выполните следующие действия.

10. В панели ***ClassView*** выберите класс **C….Dlg.** Щелкните правой кнопкой для открытия контекстного меню, затем выберите ***Add Windows Message Handler***. В результате открывается диалоговое окно ***New Windows Message And Event Handlers***, показанное на рис. 30.
11. Прокрутите список доступных сообщений, пока не найдете сообщение **WM_LBUTTONDOWN.** Выберите его и щелкните на кнопке ***Add and Edit***.
12. Замените код функции **OnLButtonDown(…),** сгенерированной ***Visual C++***, кодом, выделенным в следующем листинге.

```
void CFigureDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
// TODO: Add your message handler code here

if (m_Holst.PtInRect(point)) // Если курсор
// внутри холста, то
{
m_IsDrawing = true;        // отмечаем этот факт переменной
// m_IsDrawing,
m_PointStart = point;      //запоминаем позицию «щелчка»,
SetCapture();       //начинаем отслеживание движений мыши
}
}
```

Когда Windows вызывает функцию **QnLButtonDown(…),** она передает ей два аргумента. Первый аргумент, **nFlags**, определяет, была ли нажата клавиша Shift или Ctrl в момент выполнения щелчка на кнопке. Например, если необходимо выполнить какое-либо действие при нажатии Ctrl+щелчок, потребуется записать такой программный код:

```
if (nFlags & MK_CONTROL)
{ // Обработка Ctrl + щелчок

}
```

Для обработки щелчка при нажатой клавише Shift служит константа MK_SHIFT.

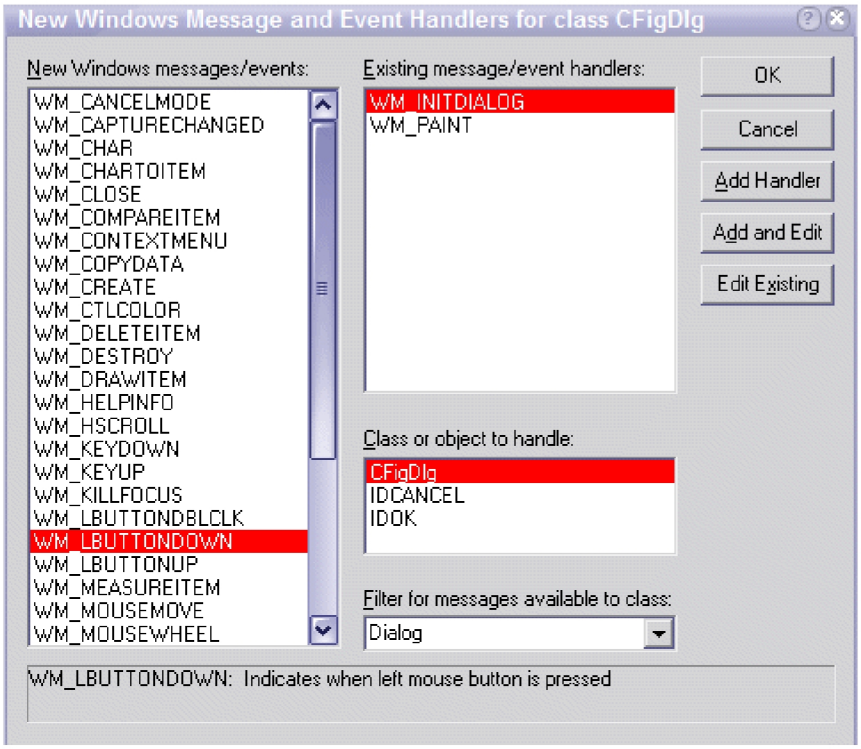


Рис. 30. Добавление обработчика нажатия левой кнопки мыши

Действия программы **OnLButtonDown(…)** приведены в комментариях. Отметим лишь, что программа **m_Holst.PtInRect(point)** возвращает **true**, если точка, заданная аргументом, лежит внутри прямоугольника **m_Holst** и **false** в противном случае.

13. Аналогично действиям пунктов 10, 11, добавьте обработчики сообщений WM_LBUTTONUP (отпустить левую кнопку мыши) и WM_MOUSEMOVE (передвинуть мышь).
14. Замените код функции **OnLButtonUp(…),** сгенерированной ***Visual C++***, кодом, выделенным в следующем листинге.

```
void CFigureDlg::OnLButtonUp(UINT nFlags, CPoint point)
{
// TODO: Add your message handler code here

m_IsDrawing = false;       //Левая кнопка отпущена, поэтому
            //местонахождение курсора для нас безразлично;
ReleaseCapture( );         //прекращаем отслеживание
                           // движений мыши
}
```

Если не прекратить отслеживание движений мыши, то Вы обнаружите, что не можете закрыть свою программу. Не паникуйте, просто переключитесь на другую программу и Windows сама освободит мышь от захвата.

15. Замените код функции **OnMouseMove(...),** сгенерированной *Visual C++*, кодом, выделенным в следующем листинге. Наберите весь код, включая комментарии, - эти строки нам понадобятся в дальнейшем.

```
void CFigureDlg::OnMouseMove(UINT nFlags, CPoint
point)
{
      // TODO: Add your message handler code here

if(m_IsDrawing && (nFlags & MK_LBUTTON) &&
      m_Holst.PtInRect(point))
      {
CClientDC dc(this);
CPoint Point_cur;

CPen Pen;
Pen.CreatePen(PS_SOLID,4,RGB(255,0,0));

CBrush Brush;
Brush.CreateSolidBrush(RGB(255,255,0));

//HICON Pict = AfxGetApp( )->LoadIcon(IDI_FIG);
dc.SelectObject(&Pen);
dc.IntersectClipRect(m_Holst);

dc.MoveTo(m_PointStart);
```

```
//Рисуем линию
dc.LineTo(point);

//Рисуем пиктограмму
//    Point_cur.x = point.x - 32;
//    Point_cur.y = point.y - 32;
//dc.FillSolidRect(m_PointStart.x,m_PointStart.y,
// -32,-32,RGB(255,255,255));
//    dc.DrawIcon(Point_cur,Pict);

//Подготовка к рисованию фигур
//dc.FillSolidRect(m_PointStart.x+2,
//m_PointStart.y+2,-64,-64,
//RGB(255,255,255));
//dc.SelectObject(&Brush);

//Рисование квадрата
//dc.Rectangle(point.x,point.y,point.x-60,
//point.y-60);

//Рисование окружности
//dc.Ellipse(point.x,point.y,point.x-60,
//point.y-60);

//Рисование вашей фигуры


//Дальнейшее движение из текущей точки
m_PointStart = point;
      }
}
```

Соберите и выполните свою программу (бордовый восклицательный знак). Нажмите кнопку «Очистить». Установите курсор на холст, нажмите левую кнопку мыши и, **не отпуская** ее, подвигайте мышь. Нарисуйте что-нибудь. Если не понравится, то нажмите кнопку «Очистить».

Вы уже отметили одну досадную оплошность. Если не нажимать кнопку «Очистить», а сразу поместить курсор мыши на холст, нажать левую кнопку и перемещать мышь, то картинка рисуется на

сером фоне. Белый фон появляется только после нажатия кнопки «Очистить». Исправим этот недостаток.

Добавьте в класс **C…Dlg** приватную переменную **Begin** типа **int**. В программе **OnInitDialog()** перед оператором **return TRUE;** наберите код:

```
Begin = 1;
```

В самое начало программы **OnLButtonDown(…)** добавьте следующий код.

```
if (Begin) {
      Begin = 0;
      CClientDC dc(this);
      dc.Rectangle(m_Holst);
}
```

Теперь, как только Вы нажмете левую кнопку мыши, холст будет закрашен белой кистью. Закраска выполняется только один раз, в начале работы программы. Почему ?

16. Войдите на вкладку ресурсов. Добавьте новую пиктограмму: ***Insert -> Resource -> Icon -> New***. По умолчанию она имеет идентификатор IDI_ICON1. Присвойте ей идентификатор IDI_FIG и нарисуйте изображение по своему вкусу.

17. Уберите символы комментария в строке

```
//HICON Pict = AfxGetApp( )->LoadIcon(IDI_FIG);
```

сделайте комментарием строку

```
dc.LineTo(point);
```

и уберите символы комментария в следующих строках

```
// Point_cur.x = point.x - 32;
// Point_cur.y = point.y - 32;
```

```
//dc.FillSolidRect(m_PointStart.x,m_PointStart.y,
// -32,-32,RGB(255,255,255));
//dc.DrawIcon(Point_cur,Pict);
```

Соберите и выполните свою программу (бордовый восклицательный знак). Установите курсор на холст, нажмите левую кнопку мыши и, **не отпуская** ее, подвигайте мышь. Повторите это несколько раз. Если не понравится, то нажмите кнопку «Очистить».

18. Сделайте комментарием строки

```
dc.FillSolidRect(m_PointStart.x,m_PointStart.y,
                 -32,-32,RGB(255,255,255));
```

Соберите и выполните свою программу (бордовый восклицательный знак). Что изменилось ? Почему ?

Поэкспериментируйте с квадратом и окружностью.

19. Приведите текст программы в соответствие с текстом в пункте 15. Сделайте комментарием строку **dc.LineTo(point);**

Дополните текст программы таким образом, чтобы в качестве подвижного курсора использовалась какая-либо фигура.

13. Секреты резиновой нити

20. Берем за основу только что написанную программу. В программе **OnMouseMove(…)** удалите все строки, за исключением условия на входе. Программа должна принять следующий вид.

```
void CFigureDlg::OnMouseMove(UINT nFlags, CPoint
   point)
{
   if(m_IsDrawing && (nFlags & MK_LBUTTON) &&
   m_Holst.PtInRect(point))
   {

   }
}
```

21. Войдите во вкладку ресурсов, откройте диалог IDD_..._DIALOG, сместите холст в сторону и на освободившееся место добавьте дружелюбное поле со списком. Для этого выполните следующие шаги.

a. Перетащите ***Group Box*** из панели инструментов на свободное поле. Увеличьте его в размерах. Замените заголовок на «Линии и формы». Идентификатор группы IDC_STATIC можно оставить неизменным, но обязательно отметьте флажок ***Group*** в диалоговом окне ***Group Box Properties***. Это приводит к тому, что группа действует в качестве ограничителя для помещенных в нее переключателей.

b. Перетащите поле со списком ***Combo Box*** из панели инструментов и поместите его внутрь ***Group Box***. Это процесс состоит из двух шагов: 1) растяните элемент, чтобы его ширина почти равнялась ширине ***Group Box***, но не пересекала обрамление; 2) поместите курсор на **кнопку** ***Combo Box*** и щелкните левой кнопкой мыши. Появится новый прямоугольник. Растяните его вниз, чтобы задать размеры открытого поля со списком. Наш список будет состоять из четырех строк.

c. После выбора подходящих размеров поля со списком откройте диалоговое окно ***Combo Box Properties***, щелкнув правой кнопкой мыши по полю со списком и выбрав ***Properties*** из контекстного меню. В закладке ***General*** измените имя элемента управления на IDC_SHAPES и отметьте флажки ***Tab***, ***Stop*** и ***Visible***. Все остальные флажки должны быть сброшены.

d. Выберите закладку ***Data*** и введите строки «От руки», «Линии», «Эллипсы», «Прямоугольники». В конце каждой строки нажимайте **Ctrl+Enter**. Нажатие только ***Enter*** приведет к закрытию диалогового окна.

e. Выберите закладку ***Styles*** и установите ***Type*** на ***Drop List*** (Выпадающий список). Установите значение ***No*** в ***Owner Draw***. Флажок ***Vertical Scroll*** должен быть отмечен, все остальные сброшены.

f. Щелкните правой кнопкой мыши по ***Combo Box*** и войдите в ***Class Wizard***. Добавьте переменную **m_Shapes** с категорией **Control** и типом **CComboBox**.

g. В программу **OnInitDialog(...)** перед оператором **return** добавьте строку инициализации переменной **m_Shapes**.

```
m_Shapes.SetCurSel(0);
```

22. Вычерчивание линий и форм отличается от ручных рисунков. Во время рисования от руки, каждый раз, когда перемещается мышь, программа рисует небольшой сегмент линии, соединяя текущее положение мыши с позицией, зафиксированной в переменной **m_PointStart**. Затем программа обновляет значение **m_PointStart** таким образом, что она «следует» за мышью по экрану.

В случае рисования линий и форм требуется нарисовать фигуру в окончательном виде, только когда кнопка мыши отпущена. Следовательно, обновления значения переменной **m_PointStart** по мере движения мыши мы производить не будем. Вместо этого объявите в классе **C…Dlg** переменную **m_PointEnd** типа **CPoint**. Кроме того, добавьте в класс **C…Dlg** переменную **CPen Pen**; и инициализируйте ее в функции **OnInitDialog(…)**, добавив перед оператором **return** строку:

```
Pen.CreatePen(PS_SOLID, 4, RGB(255,0,0));
```

23. В функции **OnLButtonDown(…)** найдите строку, которая инициализирует переменную **m_PointStart**. *Под ней* вставьте следующую строку, инициализирующую переменную **m_PointEnd**.

```
m_PointEnd = point;
```

Кроме того, удалите вызов функции **SetCapture(…)**.

24. Измените код функции **OnLButtonUp(…)** так, чтобы он принял следующий вид.

```
void CFigureDlg::OnLButtonUp(UINT nFlags, CPoint
   point)
{
   m_IsDrawing = false;
   m_PointEnd = point;

   DrawShape();
}
```

25. Добавьте функцию **DrawShapes(bool stretch = false);** в класс ***C...Dlg***, для чего щелкните правой кнопкой мыши по классу ***C...Dlg*** в окне ***ClassView***, а затем выберите ***Add Member Function*** из контекстного меню. Возвращаемый тип должен быть **void**, а объявление должно иметь вид: **void DrawShapes(bool stretch = false);**

26. Добавьте код к сгенерированной программе **DrawShapes(...)** так, чтобы она приняла следующий вид.

```
void CFigureDlg::DrawShape(bool stretch)
{
    CClientDC dc(this);
//Ограничение области рисования
    dc.IntersectClipRect(m_Holst);

    int drawMode = m_Shapes.GetCurSel(); //Что рисовать

    dc.SelectObject(&Pen); //Выбрать перо

if (stretch && drawMode != 0) dc.SetROP2(R2_NOT);
    //Инвертировать пиксели

    switch(drawMode)
    {
    case 0: //От руки
           dc.MoveTo(m_PointStart);
           dc.LineTo(m_PointEnd);
           m_PointStart = m_PointEnd;
           break;
    case 1: //Линии
           dc.MoveTo(m_PointStart);
           dc.LineTo(m_PointEnd);
           break;
    case 2: //Эллипсы
           dc.Ellipse(CRect(m_PointStart, m_PointEnd));
           break;
    case 3: //Прямоугольники
           dc.Rectangle(CRect(m_PointStart, m_PointEnd));
           break;
    }
}
```

27. Модифицируйте программу **OnMouseMove(…)**. Она должна принять следующий вид.

```
void CFigureDlg::OnMouseMove(UINT nFlags, CPoint point)
{
      if(m_IsDrawing && (nFlags & MK_LBUTTON) &&
      m_Holst.PtInRect(point))
      {
            DrawShape(true);
            m_PointEnd = point;
            DrawShape(true);
      }
}
```

28. Нажмите бордовый восклицательный знак. Попробуйте разные фигуры. Программа обеспечивает визуальную обратную связь при помощи растягиваемой формы, которая не принимает окончательный вид, пока не будет отпущена кнопка мыши.

29. Попытайтесь нарисовать на экране свой придуманный курсор из предыдущего раздела

Пояснения к программе.

Функция **DrawShape(…)** работает во взаимодействии с функциями

OnMouseMove(…) и **OnLButtonUp(…)** для реализации эффекта резиновой нити.

Рассмотрим работу функцию **DrawShape(…)** шаг за шагом.

- Две первых строки просто устанавливают контекст устройства. Эти строки совпадают с теми, которые ранее использовались в **OnMouseMove(…)**.
- На следующем шаге (третья строка кода) определяется текущее состояние поля со списком с использованием функции **GetCurSel()**. **GetCurSel()** возвращает номер выбранного элемента в поле со списком, начиная с нуля. Это значение потребуется для принятия решения о том, какую фигуру рисовать, поэтому его следует сохранить в локальной переменной **drawMode**.

- На третьем шаге устанавливается контекст устройства. Здесь же выбирается текущее перо, как это делалось ранее. Затем, если вы не рисуете от руки, а работаете в "растягиваемом" режиме, при помощи функции **SetROP2(…)** следует установить режим рисования в **R2_NOT**, для которого характерно обращение каждого пикселя на экране. (***Чуть позже станет понятно, почему это так важно.***) Обратите внимание, что в случае режима рисования от руки вызов функции **SetROP2(…)** не выполняется.
- На четвертом, заключительном этапе выполняется оператор **switch**, примененный к значению **drawMode** (значение, полученное из поля со списком). В каждом случае рисуется конкретная фигура. В первом случае (0) используется программный код, реализующий рисование от руки, который берется из исходной функции **OnMouseMove(…)**. Во втором случае (1) рисование линии выполняется с помощью функций **MoveTo(…)** и **LineTo(…)**. Третий и четвертый случаи еще проще; вычерчивание эллипсов и прямоугольников выполняется с помощью функций **EIlipse(…)** и **Rectangle(…)**.

Для понимания метода резиновой нити при рассмотрении функции **OnMouseMove(…)** держите в уме код **DrawShape(…)**. В тексте **OnMouseMove(…)** первый оператор (при условии, что мышь попала в область холста) вызывает **DrawShape(true)**. Когда поле со списком настраивается на рисование линии или формы, передача значения **true** означает переход в режим рисования **R2_NOT** (режим обращения). В режиме обращения повторное рисование линии или формы приводит к ее стиранию — подобный прием делает возможной реализацию метода резиновой нити.

Обратите внимание на то обстоятельство, что когда функция **OnMouseMove(…)** вызывается впервые, переменные **m_PointStart** и **m_PointEnd** содержат одно и то же значение, так что ничего не рисуется. Тем не менее после вызова **DrawShape(true)** функция **OnMouseMove(…)** обновляет значение **m_PointEnd**, устанавливая его в новую позицию пера, после чего еще раз вызывает **DrawShape(true)**. На этот раз переменные **m_PointEnd** и

m_PointStart имеют разные значения, так что на экране отрисовывается "обращенная" линия или форма.

При следующем вызове функции **OnMouseMove(…)** она снова вызывает **DrawShape(true)**. Поскольку с момента последнего обращения к функции **DrawShape(…)** значение переменной **m_PointEnd** не изменялось, в результате текущего вызова рисуется новый прямоугольник или эллипс непосредственно поверх предыдущего. В качестве режима рисования установлен **R2_NOT**, поэтому цвет каждого пикселя восстанавливается таким, каким он был до рисования первого прямоугольника, ***тем самым затирая фигуру***. Далее переменная **m_PointEnd** еще раз обновляется, и весь процесс повторяется.

После отпускания кнопки мыши, в методе **OnLButtonUp(…)** функция **DrawShape(…)** вызывается в последний раз без аргументов. Поскольку по умолчанию принимается аргумент **false**, используются обычное перо и режим рисования. Форма или линия рисуются в окончательном виде и на этот раз они не затираются.

14. Мелочи, которые порождают новые идеи

Мы с Вами многому научились, однако у всех созданных программ есть одна неприятная особенность, которая бросает тень на профессионализм и компетентность программиста. Проблема заключается в следующем. Выполните, к примеру, программу табулирования функции (лучше с первым вариантом выдачи таблицы) и, не закрывая ее окна, вызовите на выполнение, допустим, стандартную программу «Калькулятор». Захватите мышью окно «Калькулятора» и перекройте им часть холста, где нарисован график функции. Затем уведите окно «Калькулятора» в сторону. Что стало с графиком ?!?

Конечно, можно легко восстановить его, щелкнув по кнопке «Чертить график». А как быть с нарисованным шедевром в программах «ручного» рисования и использования резиновой нити? Там придется все рисовать заново…

Решение проблемы для общего случая перекрытия окон приводит нас к идее разделения информации, циркулирующей в програм-

ме, на три группы, или пользуясь терминологий объектно-ориентированного программирования, на три вида объектов.

- Объекты-модели (Model Objects) – предназначены для хранения компьютерного представления реально существующих объектов окружающего мира, с которыми имеет дело программа: сводок погоды, списка студентов в группе, расписания авиарейсов, распределения напряжений в конструктивных элементах самолета, ведомостей зарплаты, чертежей деталей и т.п.
- Объекты-представления (View Objects) – предназначены для отображения информации, заключенной в объектах-моделях. Один объект-модель может иметь множество представлений, таких как таблица, график, диаграмма, чертеж, рисунок, шифрограмма и т.п.
- Объекты-контроллеры (Controller Objects) – реагируют на ввод пользователя. Это различного рода диалоги, меню, обработчики движения мыши, нажатия клавиш на клавиатуре, прикосновений к сенсорным панелям и т.п.

Библиотека MFC имеет в своем составе реализацию архитектуры «документ-представление», известной под именем ***DocView***. Архитектура ***DocView*** работает с документами: ее модельная часть сохраняет и обновляет документы, а представленческая отображает их. Примерами являются программы Microsoft Office, CorelDraw, Adobe Illustrator и другие. Роль контроллера играют компоненты Windows и сама Windows: функции выборки входных сообщений и сообщений из очередей выполняются автоматически, см. Приложение 2.

Изучение архитектуры ***DocView*** является длительным и кропотливым процессом с множеством ответвлений, поэтому сейчас мы рассмотрим предшественника ***DocView*** из ***Windows***, а именно: класс **CMetaFileDC**, который позволит нам удерживать нетронутыми наши картины, когда на них наезжает «паразитное» окно.

Класс **CMetaFileDC** предназначен для создания и хранения метафайла. ***Метафайл*** – это просто структура данных в памяти, содержащая команды GDI-интерфейса (см. с. 42-43), такие как **LineTo()** или **FillRect()**. Воспользовавшись классом **CMetaFileDC**, можно автоматически сохранять линии и формы такими, какими Вы их нарисовали, причем неважно как, - то ли программным путем, то

ли от руки. Если картинку требуется нарисовать заново, программе нужно лишь воспроизвести этот метафайл в методе **OnPaint()**.

Для включения в приложение метафайл нужно выполнить всего три шага.

1. В программу необходимо добавить переменную типа указатель на объект класса **CMetaFileDC**. Потребуется также написать код, отводящий и освобождающий память под объект **CMetaFileDC**.
2. Необходимо записать в метафайл каждую операцию GDI. Напомним, что это операции созданного нами объекта **dc**.
3. Внутри функции **OnPaint()** воспроизвести метафайл. После отображения рисунка следует выполнить повторную инициализацию метафайла, так чтобы он продолжал записывать каждую деталь рисунка.

Сначала рассмотрим программу табулирования функции с простым «непрофессиональным» выводом таблицы. Излишне говорить, что из Visual C++ надо загрузить проект программы **TabFun**, восстановить текст программы **OutTable()** для первого варианта выдачи таблицы и войти в закладку *ClassView*.

Непосредственная работа с контекстом устройства **dc** ведется в программах **OnDraw()** и **OutTable()**. Кроме того, некоторая подготовительная работа выполняется в программе **OnInitDialog()**. Ими мы и займемся.

14.1 Создание объекта CMetaFileDC

1. Откройте диалоговое окно *Add Member Variable*, щелкнув правой кнопкой мыши на классе *CTabFunDlg* и выбрав из контекстного меню пункт *Add Member Variable*. В появившемся диалоговом окне добавьте приватный указатель **m_pMF** на **CMetaFileDC**. Диалоговое окно должно принять вид, показанный на рис. 31.

Рис. 31. Добавление переменной для метафайла

2. Воспользовавшись закладкой ClassView, войдите в функцию OnInitDialog() и в конце, перед оператором **return TRUE;** добавьте следующие строки.

```
//Добавить метафайл
    m_pMF = new CMetaFileDC;
    m_pMF->Create();
```

3. Щелкните правой кнопкой мыши на классе ***CTabFunDlg*** и выберите из контекстного меню пункт ***Add Windows Message Handler*** (Добавить обработчик сообщений Windows). В появившемся диалоговом окне выберите сообщение WM_DESTROY, как показано на рис. 32.
Щелкните левой кнопкой мыши на ***Add and Edit*** (Добавить и редактировать).

В появившемся окне редактирования добавьте выделенные строки.

```
void CTabFunDlg::OnDestroy()
{
        CDialog::OnDestroy();

        // TODO: Add your message handler code here
        m_pMF->Close(); //Закрыть метафайл
        delete m_pMF;   //Удалить метафайл
}
```

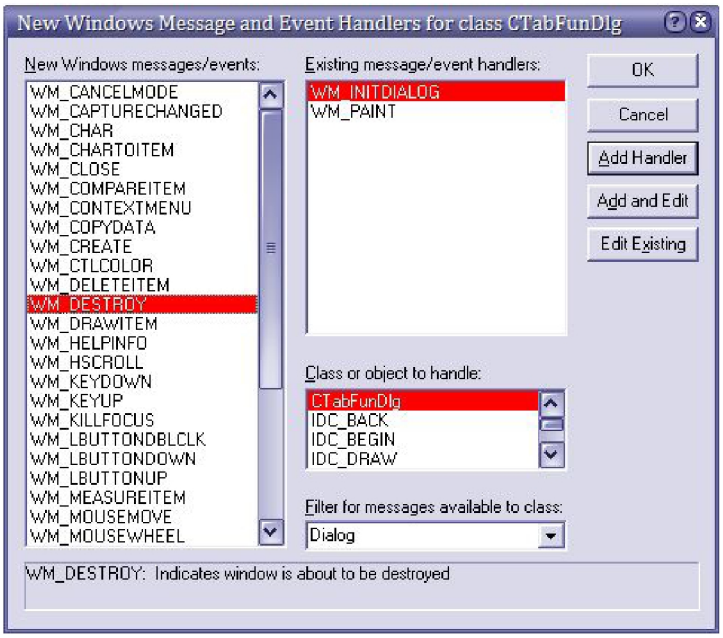


Рис. 32. Добавление обработчика закрытия окна

Видно, что код, необходимый для создания метафайла, достаточно простой. Для инициализации метафайла используются оператор ***new*** и функция **Create()**, а для освобождения перед завершением программы используемой им памяти– функция **Close()** и оператор ***delete***. Отметим, что программа **OnDestroy()** вызывается автоматически библиотекой MFC ***перед*** выходом из программы **TabFun()** для уничтожения диалогового окна, в котором она рисовала график и выдавала таблицу значений.

14.2 Запись в метафайл графических данных

Мы уже обсуждали, что такое контекст устройства (**DC** – device context) и как осуществляется вывод информации в среде Windows. Более того, мы достаточно уверенно работали с ним во всех наших программах. Однако за кулисами остались некоторые подробности, без которых не обойтись, работая с метафайлами.

Базовый класс контекстов устройств **CDC** с ***каждым*** порожденным от него контекстом-потомком (**CPaintDC** или **CClientDC**) связывает два внутренних контекста устройства. Один из них, известный как *контекст устройства вывода (output device context)*, использует методы **DC**, непосредственно генерирующие вывод, - такие как **LineTo(...)** или **FillRect(...).** Второй, известный как *контекст атрибутов устройства (attribute device context)*, используется для изменения атрибутов DC, таких как цвет фона, толщина пера, и т.п.

Работая со стандартным **DC**, мы не должны следить за **DC** вывода и **DC** атрибутов, поскольку оба они относятся к одному и тому же логическому устройству. Напротив, у класса ***CMetaFileDC*** обычно нет **DC** атрибутов (его значением является NULL). Тогда в случае использования функций, изменяющих атрибуты DC, например, SetBkMode(...) или SelectObject(...), класс ***CMetaFileDC*** будет попросту игнорировать эти вызовы, если не принять специальных мер. Эти меры заключаются в принудительном установлении адреса DC атрибутов класса ***CMetaFileDC*** равным адресу DC атрибутов неметафайлового контекста устройства. Для этого в программы **OnDraw()** и **OutTable()** после строки

```
CClientDC dc(this);//Объявить и создать контекст устройства
```

нужно добавить строку

```
m_pMF->SetAttribDC(dc);//Взять в метафайл атрибуты экрана
```

Теперь можно записывать метафайл. Для этого после каждого оператора обращения к DC экрана, который начинается в нашем случае с «**dc.**», нужно добавить точно такой же оператор, но с первыми буквами «**m_pMF->**». Код программы **OnDraw()** будет выглядеть следующим образом (добавленные строки выделены).

```
void CTabFunDlg::OnDraw()
{
// Программа черчения графика функции на заданном интервале

CClientDC dc(this);//Объявить и создать контекст устройства
m_pMF->SetAttribDC(dc);//Взять в метафайл атрибуты экрана
```

```
CBrush m_Brush;                          //Объявить кисть
m_Brush.CreateSolidBrush(RGB(255,255,200));//Создать кисть

dc.FillRect(m_Holst,&m_Brush);     //Замазать холст кистью
m_pMF->FillRect(m_Holst,&m_Brush); // Замазать холст
                                    // и в метафайле

CPen m_Pen1, m_Pen2, m_Pen3;             //Объявить перья

    //Красное перо
m_Pen1.CreatePen(PS_SOLID, 2, RGB(255, 0, 0));

    //Синее перо
m_Pen2.CreatePen(PS_SOLID, 3, RGB(0, 0, 255));

    //Черное перо
m_Pen3.CreatePen(PS_SOLID, 1, RGB(0, 0, 0));

UpdateData(TRUE);   //Взять данные из полей
                    // редактирования в переменные

double Mx, My;              //Масштаб по x, y
double Y_max, Y_min;        //Максимальное и минимальное
                            // значение функции
double Y, x;          //Рабочие переменные
int xr, yr;           //"Рисовальные" координаты

//Определение максимального и минимального значения функции
    Y_max = -1.0e50;
    Y_min = 1.0e50;
for (x = m_Begin; x <= m_End; x += m_Step)
{
    //Функция для табулирования
    Y = 5.0 * sin(x) + 0.1 * x * x;

       if (Y > Y_max) Y_max = Y;
       if (Y < Y_min) Y_min = Y;
}

    //Вычисление масштабов
Mx = (double)(m_Holst.right - m_Holst.left - 40) /
             (m_End - m_Begin);
My = (double)(m_Holst.bottom - m_Holst.top - 40) /
             (Y_max - Y_min);

    //Предосторожность на всякий случай
       if (Mx < 0.0) Mx = -Mx;
       if (My < 0.0) My = -My;
```

```
//*********************************************************
    if (m_Scale == 0) {    // Если утоплена кнопка
                           // "Одинаковый масштаб", то
       //делаем одинаковый масштаб по осям X и Y
       if (Mx < My) My = Mx; else Mx = My;
    }
//*********************************************************

    //Рисуем оси координат
dc.SelectObject(&m_Pen3);        //Выбираем перо для экрана
m_pMF->SelectObject(&m_Pen3);    //Выбираем перо в метафайл
    //Ось X на экране
dc.MoveTo(m_Holst.left+20, m_Holst.bottom - 20);
dc.LineTo(m_Holst.right-20, m_Holst.bottom - 20);

    //Ось X в метафайле
m_pMF->MoveTo(m_Holst.left+20, m_Holst.bottom - 20);
m_pMF->LineTo(m_Holst.right-20, m_Holst.bottom - 20);

    //Ось Y на экране
dc.MoveTo(m_Holst.left+20, m_Holst.bottom - 20);
dc.LineTo(m_Holst.left+20, m_Holst.top + 20);

    //Ось Y в метафайле
m_pMF->MoveTo(m_Holst.left+20, m_Holst.bottom - 20);
m_pMF->LineTo(m_Holst.left+20, m_Holst.top + 20);

    //Вычерчивание графика
    dc.SelectObject(&m_Pen1);    //Выбираем перо для экрана
    m_pMF->SelectObject(&m_Pen1);//Выбираем перо в метафайл

    for (x = m_Begin; x <= m_End; x += m_Step)
    {
       //Функция для табулирования
       Y = 5.0 * sin(x) + 0.1 * x * x;

    //Масштабирование и перенос начала координат
    xr = (int)((x - m_Begin) * Mx) + m_Holst.left + 20;
    yr = (int)((Y - Y_min) * My) - m_Holst.bottom + 20;

    yr = -yr;       //Ось Y холста направлена ВНИЗ !!!

    if (x == m_Begin)//Подводим перо в самую первую точку
       {
         dc.MoveTo(xr, yr);
         m_pMF->MoveTo(xr, yr);//Подводим перо в метафайле
       }
```

```
    dc.LineTo(xr, yr);            //Рисуем на экране
    m_pMF->LineTo(xr, yr);        //Рисуем в метафайле
    }

}
```

Аналогичная техника используется и для программы **OutTable()**, код которой примет следующий вид (как и раньше, добавленные строки выделены).

```
void CTabFunDlg::OutTable()
{
    //Программа выдачи порции таблицы на экран
double Xt[10], Yt[10], dx;// Размер массивов Xt[] и
int i; // Yt[] должен быть не меньше значения m_InString
CClientDC dc(this);       // Объявить и создать
                          // контекст устройства
m_pMF->SetAttribDC(dc);   //Взять в метафайл атрибуты
dc.SelectObject(&m_Font);
m_pMF->SelectObject(&m_Font); //Взять в метафайл шрифт

for(i = 0; i < 10; i++) Xt[i] = Yt[i] = 0.0; //"Свои" нули
dx = 0.0;

for(i = 0; i < m_InString; i++)
{
    //Заполнение порции таблицы
    Xt[i] = m_Current + dx;
    if (Xt[i] > m_End) { Xt[i] = 0.0; break; }

    //Функция для табулирования
    Yt[i] =  5.0 * sin(Xt[i]) + 0.1 * Xt[i] * Xt[i];
       dx += m_Step;
}
    //Выдача порции таблицы в две строки
m_String.Format(
" : X :%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e: ",
    Xt[0],Xt[1],Xt[2],Xt[3],Xt[4],Xt[5],Xt[6]);
       //X на экран
dc.TextOut(m_Holst.left+2, m_Holst.bottom + 22, m_String);
       // X в метафайл
m_pMF->TextOut(m_Holst.left+2,m_Holst.bottom + 22,m_String);

m_String.Format(
" : Y :%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e:%11.3e: ",
    Yt[0],Yt[1],Yt[2],Yt[3],Yt[4],Yt[5],Yt[6]);
```

```
    //Y на экран
dc.TextOut(m_Holst.left+2, m_Holst.bottom + 38, m_String);
     // Y в метафайл
m_pMF->TextOut(m_Holst.left+2,m_Holst.bottom + 38,m_String);

}
```

14.3. Воспроизведение метафайла

Теперь, когда программный код, обеспечивающий запоминание графических команд GDI, добавлен, можно записать код, обеспечивающий воспроизведение этих команд. Как всегда, сначала нужно решить, куда его поместить. Если Вы внимательно и вдумчиво читали данное пособие, то решение отыскивается легко: этот код должен находиться в программе **OnPaint()**. Почему ?!?

Если Вы затрудняетесь ответить на этот вопрос, то Вы читали данное пособие невнимательно и бездумно и Вам следует прочитать еще раз с.37 и Приложение 2. В любом случае мы будем модернизировать программу **OnPaint().**

Для воспроизведения метафайла MFC предлагает единственную программу **PlayMetaFile(…)**, которая в качестве аргумента принимает дескриптор (описатель) метафайла типа **HMETAFILE**, но не **CMetaFileDC**. Поэтому придется покопаться в недрах MFC.

В результате раскопок обнаруживается функция **Close()** класса **CMetaFileDC**, которая немедленно прекращает запись GDI команд в метафайл и возвращает дескриптор метафайла типа **HMETAFILE**. Вслед за закрытием метафайла нужно воспроизвести его, вызвав программу **PlayMetaFile(…),** после чего продолжить запоминать в метафайле команды рисования. К сожалению, в MFC нет средств, позволяющих открыть для дополнения закрытый метафайл. Поэтому потребуется создать новый экземпляр класса **CMetaFileDC**, скопировать содержимое старого метафайла в новый, затем удалить старый **HMETAFILE**, *а также* старый экземпляр класса **CMetaFileDC**. И наконец, новый экземпляр класса **CMetaFileDC** необходимо активировать.

Проделайте следующие шаги.

1. Удалите весь код, содержащийся в программе **OnPaint()**.

2. Создайте DC для экранной выдачи информации программой **TabFun()**. Этот DC должен иметь тип **CPaintDC**, а не **CClientDC**, как было раньше. Он будет воспроизводить графику, накопленную к этому моменту в метафайле **m_pMF**.

```
CPaintDC dc(this);
```

3. Создайте дескриптор **HMETAFILE** с именем **hmf**, воспользовавшись функцией **Close()** из методов класса **CMetaFileDC**:

```
HMETAFILE hmf = m_pMF->Close();
```

4. Вызовите функцию **PlayMetaFile(...)** контекста устройства **dc** для выполнения записанных ранее графических команд GDI. В качестве параметра передается дескриптор метафайла **m_pMF**, который мы только что закрыли:

```
dc.PlayMetaFile(hmf);
```

   Этим оператором на экране воспроизводится картинка, которая была изображена до перекрытия окна программы **TabFun()** окном другой программы. Но на этом нельзя останавливаться, - надо позаботиться о будущем.

5. Создайте новый экземпляр класса **CMetaFileDC** для записи дополнительных графических команд GDI. Для этого наберите следующий код:

```
CMetaFileDC *temp = new CMetaFileDC;
temp->Create();
```

   Нами создан указатель на объект **temp** типа **CMetaFileDC**. Теперь надо наполнить этот объект предшествующими графическими командами. Для этого служит тот же оператор **PlayMetaFile(...)**, но в качестве получателя нужно указать объект **temp**, а в качестве параметра – дескриптор предыдущего метафайла **hmf**:

```
temp->PlayMetaFile(hmf);
```

Теперь новый метафайл **temp** содержит все операции рисования, записанные в исходном метафайле **m_pMF**.

6. Удалите метафайл **HMETAFILE** с именем **hmf**, воспользовавшись функцией **DeleteMetaFile()**, освободите память для старого метафайла **m_pMF** типа **CMetaFileDC**, а затем присвойте адрес нового метафайла **temp** указателю **m_pMF**:

```
DeleteMetaFile(hmf);
delete m_pMF;
m_pMF = temp;
```

Итоговый код программы **OnPaint()** должен принять следующий вид.

```
void CTabFunDlg::OnPaint()
{
CPaintDC dc(this); //Контекст для воспроизведения метафайла

    HMETAFILE hmf = m_pMF->Close(); // Дескриптор метафайла
    dc.PlayMetaFile(hmf);          // Воспроизвести метафайл

    CMetaFileDC *temp = new CMetaFileDC;
    temp->Create();
    temp->PlayMetaFile(hmf);

    DeleteMetaFile(hmf);
    delete m_pMF;
    m_pMF = temp;
}
```

Оттранслируйте и выполните программу. Вызовите «Калькулятор» и «походите» его окном по Вашей программе. Все восстанавливается ? Или что-то не так ?

Оказывается, еще не все огрехи устранены. «Чужое» окно оставляет следы на границах холста. Связано это со стилем холста, на котором происходит вычерчивание графика. С целью улучшения внешнего вида для ресурса **IDC_HOLST** мы задали стиль **Sunken** (Вдавленный) и **Client Edge** (Клиентская кромка). Такие стили приводят к утолщению кромок окна для вычерчивания графика и наша компенсация этого в **OnInitDialog()** путем вызова

```
m_Holst.DeflateRect(2, 2, 1, 1); // Исключить из холста
                                 // толщину рамки
```

недостаточна. Для устранения следов на границах холста можно либо снять отметки на стилях **Sunken** (Вдавленный) и **Client Edge** (Клиентская кромка) ресурса **IDC_HOLST**, либо увеличить значения параметров в **DeflateRect(...)**, например так:

```
m_Holst.DeflateRect(3, 3, 3, 3); //Исключить из холста
                                 // толщину рамки
```

Соберите и выполните программу. Поработайте с ней. Теперь она вполне профессиональна...

А как быть со вторым вариантом выдачи таблицы?

Здесь все несколько проще. Поскольку для выдачи на экран таблицы значений мы использовали ресурс **List Box** (Список) непосредственно с фабрики MFC, то о перерисовке собственного окна заботится сам список. Поэтому программу **OutTable()** дополнять нам не надо.

14.4. Различные детали

Аналогичная техника восстановления изображений может быть использована в рассмотренной программе рисования на дисплее как карандашом и в методе резиновой нити. Однако в методе резиновой нити в метафайле необходимо запоминать только окончательный вариант линии или фигуры, но никак не промежуточные шаги. К счастью, можно без труда определить, находится форма в процессе растяжения, или уже зафиксирована. Если аргумент **stretch** имеет значение **true**, то форма находится в процессе растяжения и в метафайл записываться не должна. Напротив, если аргумент **stretch** имеет значение **false**, либо установлен режим рисования от руки, который не предусматривает режима резиновой нити, то каждое обращение к DC экрана должно сопровождаться обращением к DC метафайла.

Прежде чем переходить к программному коду, отметим, что, как и ранее, нужно объявить переменную **m_pMF** и создать обработчик сообщения **WM_DESTROY**, см. с. 81, 82.

Необходимый программный код, обеспечивающий рисование в методе резиновой нити представлен далее. Как всегда, добавленные или модернизированные строки выделены.

```
void CFigureDlg::OnDestroy()
{
	CDialog::OnDestroy();

	m_pMF->Close();
	delete m_pMF;

}
//=================================================

//Концовка файла OnInitDialog()

	// TODO: Add extra initialization here

	m_IsDrawing = false;

	CWnd* pHolst = GetDlgItem(IDC_HOLST);
	pHolst->GetWindowRect(&m_Holst);
	ScreenToClient(&m_Holst);

	m_Holst.DeflateRect(3,3,3,3);

	m_Shapes.SetCurSel(0);
	Pen.CreatePen(PS_SOLID,4,RGB(255,0,0));

	Begin = 1;

m_pMF = new CMetaFileDC;
m_pMF->Create();

	return TRUE;
}
//=====================================================

void CFigureDlg::OnPaint()
{
	CPaintDC dc(this);  // Контекст устройства
	                    // для рисования

	HMETAFILE hmf = m_pMF->Close();
	dc.PlayMetaFile(hmf);
```

```
        CMetaFileDC *temp = new CMetaFileDC;
        temp->Create();
        temp->PlayMetaFile(hmf);

        DeleteMetaFile(hmf);
        delete m_pMF;
        m_pMF = temp;
}

//=================================================

void CFigureDlg::DrawShape(bool stretch)
{
        CClientDC dc(this);
        m_pMF->SetAttribDC(dc);

        dc.IntersectClipRect(m_Holst);
        m_pMF->IntersectClipRect(m_Holst);

        int drawMode = m_Shapes.GetCurSel();

        dc.SelectObject(&Pen);
        m_pMF->SelectObject(&Pen);

        if (stretch && drawMode != 0) {
                dc.SetROP2(R2_NOT);
        }

        switch(drawMode)
        {
        case 0: //От руки
                dc.MoveTo(m_PointStart);
                dc.LineTo(m_PointEnd);

                m_pMF->MoveTo(m_PointStart);
                m_pMF->LineTo(m_PointEnd);

                m_PointStart = m_PointEnd;
                break;
        case 1: //Линии
                dc.MoveTo(m_PointStart);
                dc.LineTo(m_PointEnd);
                if (! stretch) {
                        m_pMF->MoveTo(m_PointStart);
                        m_pMF->LineTo(m_PointEnd);
                }
                break;
```

```
        case 2: //Эллипсы
                dc.Ellipse(CRect(m_PointStart, m_PointEnd));
                if (! stretch) {
        m_pMF->Ellipse(CRect(m_PointStart, m_PointEnd));
                }
                break;
        case 3: //Прямоугольники
                dc.Rectangle(CRect(m_PointStart, m_PointEnd));
                if (! stretch) {
        m_pMF->Rectangle(CRect(m_PointStart,
m_PointEnd));
                }
                break;
        }
}
```

Соберите и выполните программу. Вызовите «Калькулятор» и «походите» его окном по Вашему холсту. Что получается ? Вы этого ожидали ?

Нарисованные фигуры восстанавливаются, но фон свободного пространства закрашивается серым цветом !?! А если нажать кнопку «Очистить» и повозить чужим окном по холсту, то, оказывается, можно восстановить все, что когда-либо было Вами нарисовано, правда, на сером фоне (случай беспримерного злопамятства…).

Дело в том, что программа пока не обеспечивает корректного рисования фона как в программе **OnLButtonDown()**, так и при нажатии на кнопку «Очистить». Что же должно происходить ?

Когда очищается холст, программа должна начать *новый* метафайл. Новый метафайл не должен получать старые команды GDI – он должен начинаться заново, с новым контекстом устройства DC. Решение этой задачи не содержит особых трудностей. Просто измените код программ **OnClear()** и **OnLButtonDown()**. Как всегда, выделены добавленные или модифицированные строки. Смысл операторов дан в комментариях.

```
void CFigureDlg::OnClear()
{
        CClientDC dc(this);
```

```
        HMETAFILE hmf = m_pMF->Close();  // Закрыть старый
                                         // метафайл
        ::DeleteMetaFile(hmf);     // Удалить содержимое
                                   // метафайла
        delete m_pMF;              //Удалить метафайл
                                   // из памяти
        m_pMF = new CMetaFileDC;   //Разместить новый
                                   // экземпляр
        m_pMF->Create();           //Создать ресурсы GDI
        m_pMF->SetAttribDC(dc); //Подключиться к текущему dc

        dc.SelectStockObject(NULL_PEN);  //Выбрать перо
        dc.Rectangle(m_Holst);           //Замазать холст

m_pMF->SelectStockObject(NULL_PEN);//Взять перо в метафайл
m_pMF->Rectangle(m_Holst);       //Взять холст в метафайл

}

//=========================================================

void CFigureDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
if (Begin)
{
        Begin = 0;
        CClientDC dc(this);
        HMETAFILE hmf = m_pMF->Close();  // Закрыть старый
                                         // метафайл
        ::DeleteMetaFile(hmf);     // Удалить содержимое
                                   // метафайла
        delete m_pMF;              //Удалить метафайл
                                   // из памяти
        m_pMF = new CMetaFileDC;   //Разместить новый
                                   // экземпляр
        m_pMF->Create();           //Создать ресурсы GDI
        m_pMF->SetAttribDC(dc); //Подключиться к текущему dc

        dc.SelectStockObject(NULL_PEN);  //Выбрать перо
        dc.Rectangle(m_Holst);           //Замазать холст

m_pMF->SelectStockObject(NULL_PEN);//Взять перо в метафайл
m_pMF->Rectangle(m_Holst);       //Взять холст в метафайл
}

        if(m_Holst.PtInRect(point))
        {
              m_IsDrawing = true;
```

```
            m_PointStart = point;
            m_PointEnd = point;
        }

}

//=====================================================
```

Соберите программу и запустите на выполнение. Нарисуйте на холсте очередной свой шедевр. Вызовите «Калькулятор» и «походите» его окном по Вашему холсту. Теперь все в порядке. Ведь и в самом деле – «рукописи не горят…»

Заключение

На этом мы завершаем рассмотрение диалоговых приложений. Многие базовые идеи и методы их реализации в рассмотренных программах не лишены изящества, однако рассмотренными вопросами не исчерпываются возможности Visual C++. Это только начало, и следующей большой темой для рассмотрения будет архитектура «документ-представление», которая является фундаментальной для всей операционной среды Windows. Мы познакомимся со способами сохранения и воспроизведения в разных форматах как текстовых, так и графических данных, создания выпадающих и пристыкованных меню, печатью с предварительным просмотром образа печати и многими другими приемами.

Библиографический список

1. Гилберт, Стивен. Самоучитель Visual C++ 6 в примерах: пер. с англ./ Стивен Гилберт, Билл Маккартни – Киев: Издательство «ДиаСофт», 2000. -496с.

2. Олафсен, Юджин. Visual C++ 6 и MFC. Энциклопедия пользователя: пер. с англ./ Олафсен Юджин, Скрайбнер Кенн, Уайт К. Дэвид и др. – Киев: Издательство «ДиаСофт», 2000. -720с.

3. Грегори, Кейт. Использование Visual C++ 6. Специальное издание: пер. с англ./ Кейт Грегори – М.; СПб.; Киев: Издательский дом «Вильямс», 2000. -864с.

4. Луис, Марк. Visual C++ 6. / Марк Луис – М.: Лаборатория базовых знаний, 1999. -720с. (Справочник).

5. Шилдт, Герберт. Справочник программиста по C/C++: пер. с англ./ Герберт Шилдт. -М.: Издательский дом «Вильямс», 2000. -448с.

6. Фролов, А.В. Операционная система Microsoft Windows 3.1 для программиста. / А.В.Фролов, Г.В.Фролов – М.: "ДИАЛОГ-МИФИ", 1994. -269с. (Библиотека системного программиста; Т.11).

7. Баженова, И.Ю. Visual C++ 6.0 (Visual Studio 98). Уроки программирования. / И.Ю.Баженова – М.: "ДИАЛОГ-МИФИ", 1999. -416с.

Приложение 1

Некоторые библиотечные функции

1. Тригонометрические функции

```
#include <math.h>
double  acos(double arg);
double  asin(double arg);
double  atan(double arg);
double  atan2(double arg1, double arg2);
double  cos(double arg);
double  sin(double arg);
double  tan(double arg);
```

acos() возвращает значение арккосинуса от аргумента ***arg***. Значение аргумента должно находиться в диапазоне от -1 до 1; в противном случае возникает ошибка нарушения границ.

asin() возвращает значение арксинуса от аргумента ***arg***. Значение аргумента должно находиться в диапазоне от -1 до 1; в противном случае возникает ошибка нарушения границ.

atan() возвращает значение арктангенса от аргумента ***arg***.

atan2() возвращает значение арктангенса от ***arg1/arg2***. Для вычисления квадранта возвращаемого значения используются знаки аргументов функции.

cos() возвращает значение косинуса от аргумента ***arg***. Значение аргумента должно быть выражено в радианах.

sin() возвращает значение косинуса от аргумента ***arg***. Значение аргумента должно быть выражено в радианах.

tan() возвращает значение косинуса от аргумента ***arg***. Значение аргумента должно быть выражено в радианах.

2. Степенные функции

```
#include <math.h>
double  cosh(double arg);
double  exp(double arg);
double  sinh(double arg);
double  tanh(double arg);
double  pow(double arg1, double arg2);
```

cosh() возвращает значение гиперболического косинуса от аргумента ***arg***.

exp() возвращает значение экспоненты от аргумента ***arg***. Число ***e*** возводится в степень, заданную аргументом ***arg***.

sinh() возвращает значение гиперболического синуса от аргумента ***arg***.

tanh() возвращает значение гиперболического тангенса от аргумента ***arg***.

pow() возвращает значение аргумента ***arg1***, возведенного в степень, равную значению ***arg2***. Если значение аргумента ***arg1*** равно нулю, а ***arg2*** меньше или равно нулю, то возникает ошибка нарушения границ. Ее возникновение также вероятно, если значение ***arg1*** отрицательно, а ***arg2*** не является целым значением. При переполнении возникает ошибка пределов диапазона.

3. Логарифмические функции

```
#include <math.h>
double  log(double arg);
double  log10(double arg);
```

log() возвращает значение натурального логарифма от аргумента ***arg***. Если значение аргумента ***arg*** отрицательно, то возникает ошибка нарушения границ, а если равно нулю, то ошибка пределов диапазона.

log10() возвращает значение логарифма по основанию 10 от аргумента ***arg***. Если значение аргумента ***arg*** отрицательно, то возникает ошибка нарушения границ, а если равно нулю, то ошибка пределов диапазона.

4. Разные полезные функции

```
#include <math.h>
int     abs(int arg);
double  fabs(double arg);
long    labs(long arg);

double  ceil(double arg);
double  floor(double arg);
```

```
double  sqrt(double arg);
double  _hypot(double arg1, double arg2);
```

abs(),fabs(),labs() возвращает абсолютное значение аргумента **arg**. Функции различаются только типом аргумента.

ceil() возвращает наименьшее целое, представленное в виде с плавающей точкой, которое больше (алгебраически) значения аргумента ***arg*** или равно ему. Например, при ***arg*** равном ***1.02*** **ceil()** вернет **2.0**, а при ***arg*** равном ***-1.02*** **ceil()** вернет **-1.0**.

floor() возвращает наибольшее целое, представленное в виде с плавающей точкой, которое меньше (алгебраически) значения аргумента ***arg*** или равно ему. Например, при ***arg*** равном ***1.02*** **floor()** вернет **1.0**, а при ***arg*** равном ***-1.02*** **floor()** вернет **-2.0**.

sqrt() возвращает значение квадратного корня от аргумента ***arg***. Если значение аргумента ***arg*** отрицательно, то возникает ошибка нарушения границ.

_hypot() возвращает значение квадратного корня от (***arg1 * arg1 + arg2 * arg2***). Возвращаемое значение можно считать гипотенузой прямоугольного треугольника со сторонами ***arg1*** и ***arg2***.

5. Функция **Format** класса **CString**

```
void Format( LPCTSTR lpszFormat, arg1, arg2... );
```

Параметры:

lpszFormat – строка , в которой заданы форматы для вывода значений аргументов.

arg1, arg2... – аргументы , значения которых выводятся.

Функция выполняет форматированный вывод значений аргументов в строку, определенную классом **CString.** Каждый аргумент ***arg1, arg2...***преобразуется в строку символов соответствии со своим заданным форматом ***lpszFormat*** в и добавляется в ***CString***.

Функция завершается непредсказуемо, если одна и та же строка используется как аргумент и как получатель вывода. Например, следующий код не является корректным.

```
CString str = "Некоторые данные";
str.Format("%s%d", str, 123);// Внимание: str - это и
                             // аргумент и получатель
```

Спецификация формата, которая содержит необязательные и обязательные поля, имеет следующую форму:

```
%[флажки] [общее количество позиций]
[.точность] [{h | l | I64 | L}]тип
```

Обратите внимание, что перед точностью должна быть десятичная точка. Квадратные скобки **[ ]** указывают, что поле необязательное и при записи формата в программе присутствовать не должны. Знак | означает выбор из нескольких возможностей. Простейшая спецификация формата содержит только знак процента и символ типа, например: **%d**. Если в строку вывода надо явно записать знак процента, то его надо указать дважды без пробела.

Таблица П1. Тип форматов

Тип формата	Тип аргумента для вывода	Что выдается в строку вывода
c	char	Символ
C	char	Символ
d	int	Десятичное целое со знаком.
i	int	Десятичное целое со знаком.
o	int	Восьмеричное целое без знака.
u	int	Десятичное целое без знака.
x	int	Шестнадцатеричное целое без знака, использующее символы “abcdef.”
X	int	Шестнадцатеричное целое без знака, использующее символы “ABCDEF.”

Продолжение табл. П1

`e`	`double`	Значение со знаком в форме [–]*d.dddd* **e** [*знак*]*ddd,* где *d* – отдельная десятичная цифра, *dddd* – одна или более десятичных цифр, *ddd* – точно три десятичных цифры, и *sign* – это + или –. Символ **e** обозначает экспоненту основания 10
`E`	`double`	Идентичен формату **e**, но для обозначения экспоненты используется заглавная буква **E**
`f`	`double`	Значение со знаком в форме [–]*dddd.dddd*, где *dddd* – одна или более десятичных цифр. Количество цифр перед десятичной точкой зависит от величины числа, а количество цифр после десятичной точки определяется заданием поля «точность»
`g`	`double`	Значение со знаком, выданное по формату **e** или **f**, в зависимости от того, где оно будет короче при заданной точности. Однако, если значение аргумента для вывода меньше 10^{-4}, то используется формат **e**
`G`	`double`	Идентичен формату **g**, но для обозначения экспоненты вместо буквы **e** используется заглавная буква **E**
`n`	**Указатель на целое**	Количество символов, записанных в строку вывода. Это значение сохраняется в указанном аргументе целого типа и может быть также выдано в строку вывода
`p`	**Указатель**	Выдает адрес памяти, на который указывает аргумент в форме *xxxx*:*yyyy*, где *xxxx* – это сегмент, а *yyyy* представляет собой смещение внутри сегмента. Цифры *x* и *y* представляют собой шестнадцатеричные цифры, выданные в верхнем регистре (заглавные)

Окончание табл. П1

s	**Строка**	Строка символов. Символы строки выдаются до достижения нулевого символа в строке, либо когда их количество достигает значения, заданного в поле «точность»
S	**Строка**	Строка символов. Символы строки выдаются до достижения нулевого символа в строке, либо когда их количество достигает значения, заданного в поле «точность»

Флажки

Дополнительный символ или символы, которые управляют выравниванием символов в заданном поле вывода, выдачей знаков (±), пробелов, десятичной точки, восьмеричных и шестнадцатеричных префиксов. В спецификации формата могут указываться один или более флажков.

Таблица П2. Символы флажков

Флажок	Действие	По умолчанию
–	Выравнивание по левой границе внутри заданного поля вывода	Выравнивание по правой границе
+	Перед выдаваемым числовым значением вставляется его знак (+ или –), если выдается число со знаком	Знак (–) появляется только перед отрицательным числом
0	Если 0 указан перед числом, задающим **[общее количество позиций],** то будут вставлены ведущие нули. Знак (–) подавляет ведущий 0. Если 0 указан перед символом формата (i, u, x, X, o, d), то он игнорируется	Нет вставки

Окончание табл. П2

Пробел	Вставляет пробел перед выдаваемым значением, если оно положительное и со знаком. Пробел игнорируется, если он указан вместе с флажком +	Нет вставки пробела
#	Когда флажок # используется с форматами o, x или X, то для ненулевого значения для вывода вставляет перед ним соответственно 0, 0x или 0X	Нет вставки
	Когда флажок # используется с форматами e, E или f то принудительно вставляется десятичная точка	Десятичная точка появляется только перед дробной частью
	Когда флажок # используется с форматами g или G , то принудительно вставляется десятичная точка. Кроме того, не усекаются хвостовые нули. Флажок # игнорируется с форматами c, d, i, u или s	Десятичная точка появляется только перед дробной частью. Хвостовые нули усекаются

[общее количество позиций]

Определяет количество позиций для вывода и представляет собой неотрицательное целое десятичное число. Если реальное количество выводимых символов меньше заданного количества позиций, то слева или справа (в зависимости от соответствующего флажка) вставляются пробелы. Если количество позиций начинается с 0, то вместо пробелов вставляются нули.

Спецификация **[общее количество позиций]** никогда не приводит к усечению числа. Если реальное количество выводимых символов больше заданного количества позиций, то выдаются все символы.

[.точность]

Это число определяет максимальное количество символов, выдаваемых во все поле, или в часть поля для вывода, или минимальное количество цифр для целых значений, см. таблицу П3.

Таблица П3. Как точность влияет на тип аргумента

Тип	Действие	По умолчанию
c, C	Не влияет.	Выдается символ.
d, i, u, o, x, X	Точность определяет минимальное количество цифр для вывода. Если количество цифр в аргументе меньше, чем точность, то выводимое значение дополняется слева нулями. Значение аргумента не усекается, когда количество цифр превышает точность	По умолчанию значение точности равно 1
e, E	Точность определяет количество цифр после десятичной точки. Последняя цифра округляется	Значение по умолчанию равно 6. Если точность задана равной 0, или (.) указана без следующего за ней значения, то десятичная точка не выводится
f	Точность определяет количество цифр после десятичной точки. Если число дробное, то, по крайней мере одна десятичная цифра выводится перед точкой. Выводимое значение округляется до заданного количества цифр	Значение по умолчанию равно 6. Если точность задана равной 0, или (.) указана без следующего за ней значения, то десятичная точка не выводится
g, G	Точность определяет максимальное количество выводимых значащих цифр	Выводится шесть значащих цифр. Хвостовые нули усекаются

Окончание табл. П3

s, S	Точность определяет максимальное количество выводимых символов. Символы в позициях, превышающих точность, не печатаются.	Символы печатаются до тех пор, пока не будет обнаружен нулевой символ.

```
[{h | l | I64 | L}]
```

определяют размер аргумента.

Таблица П4. Префиксы размеров для типов формата

Определить	Префикс	С описателем типа аргумента
long int	l	d, i, o, x, или X
long unsigned int	l	u
short int	h	d, i, o, x, или X
short unsigned int	h	u
__int64	I64	d, i, o, u, x, or X
Одиночный символ	h	c или C

Пример записи функции **Format**.

```
CString stroka;
stroka.Format(
" Значения переменных: int k = %-5d, double Sd = %-12.3g",
k, Sd);
MessageBox(stroka, "Пример", MB_OK);
```

В диалоговом окне появится, к примеру:

```
Значения переменных: int k = 15    , Sd = -1.2863e-05
```

Приложение 2

Принципы работы операционной системы Windows

Материал этого раздела взят из очень внятной и поэтому очень дефицитной книги [1], но скомпонован по-другому.

При создании Windows учитывались недостатки операционной системы DOS и предполагалось, что они будут преодолены. Рассмотрим подробнее четыре характерных свойства Windows:

- Общий интерфейс пользователя.
- Общесистемный ввод, использующий очередь, и система передачи сообщений.
- Не зависящая от внешних устройств архитектура ввода-вывода.
- Многозадачность на уровне приложений и взаимодействие между процессами.

Общий интерфейс пользователя

Интерфейс пользователя Windows — это "видимая часть айсберга", которая наиболее подвержена критике. Это и естественно, поскольку это единственная видимая часть Windows! Работа с общим интерфейсом пользователя Windows значительно упрощает освоение компьютера пользователем, облегчая ему жизнь.

Однако для программистов эффект применения Windows более весом. Во-первых, Windows предлагает обширную библиотеку общих, встроенных подпрограмм пользовательского интерфейса. Причем эти подпрограммы будут гарантированно присутствовать в каждой системе, в которой выполняется Windows. Во-вторых, Windows поддерживает встроенный диспетчер, управляющий окнами и меню. Поэтому можно не уделять особого внимания таким деталям интерфейса пользователя, как изменение размеров окон или обработка меню. Появляется возможность больше времени посвятить функциональным аспектам программы.

Архитектура передачи сообщений в Windows

Кроме обеспечения пользователя встроенным диспетчером окон, Windows также реализует систему связи, известную как queued input (очередизованный ввод) или передача сообщений в Windows. Рассмотрим, что это означает.

Если создается традиционная программа DOS и необходимо осуществить ввод при помощи мыши или клавиатуры, то часто прибегают к методу, известному как polling (опрос). Программа использует функции опроса в виде бесконечного цикла, который непрерывно контролирует специфическое устройство (в данном случае, мышь или клавиатура), а затем предпринимает какое-либо действие в тот момент, когда пользователь перемещает мышь или нажимает клавишу.

При использовании опроса компьютер большую часть времени не выполняет никаких функций — "выполняет цикл", как говорят. Конечно, находчивые программисты DOS знают, как устранять циклы опроса, непосредственно перехватывая аппаратные прерывания, сгенерированные устройствами типа мыши, клавиатуры и системного таймера. К сожалению, подобная находчивость часто приводит к появлению своих проблем, поскольку программы конкурируют в процессе перехватывания тех же самых аппаратных прерываний. При состязании невероятной удачи и сверхчеловеческого предвидения возможны всякие неожиданности.

Windows отказывается от такой соревновательности, принимая на себя исключительные права по управлению всеми аппаратными устройствами. Windows непосредственно получает каждое событие, возникающее при системном вводе, которые генерируются мышью, клавиатурой и таймером, и записывает события. Эти записи затем передаются приложениям, которые в них нуждаются.

Вследствие реализации этой процедуры отпадает необходимость в том, чтобы индивидуальные программы контролировали устройства ввода данных системы. В этом и нет необходимости. Вместо этого Windows направляет сообщения, поступающие от таймера, клавиатуры и мыши, соответствующим программам. Все, что Ваша программа должна теперь предпринять, — это ответить соответствующим образом.

Также в результате осуществления этого процесса Windows устраняет конкуренцию среди устройств ввода данных. При работе под управлением DOS, если несколько программ были установлены с подпрограммой обработчика прерываний мыши, часто возникали затруднения при определении того, какая программа должна обрабатывать какое-либо сообщение мыши. Однако когда Windows обрабатывает сообщения мыши, то она отслеживает расположения объектов на экране и проверяет корректность передачи событий мыши нужной программе. Таким образом, совместное использование аппаратного устройства программами становится тривиальным. Если в Вашей программе нужно, чтобы курсор мыши отображался в виде руки, в то время как все другие программы хотят использовать курсор в виде стрелки, то Windows гарантирует отображение соответствующего вида курсора.

Традиционные программы организованы иерархическим образом. В языке С, например, функция **main()** находится наверху пирамиды других функций. Она действует подобно генеральному директору, предоставляя работу нескольким подпрограммам верхнего уровня. Аналогично, каждая из этих подпрограмм управляет другими подпрограммами низшего уровня. Все упорядочено, и можно просмотреть всю программу, например, используя отладчик, от начала до конца. Подобный сорт программ большинство из нас изучает с целью их дальнейшего программирования.

Однако программа для системы Windows не организована иерархическим способом в чистом виде. Программа все еще содержит функции, но эти функции были разработаны таким образом, чтобы генерировать отклик (немного в автономном режиме) в ответ на внешние события. Поэтому говорят, что программы Windows управляются событиями.

События могут быть самими разнообразными. Например, один вид события генерируется, когда пользователь щелкает мышью или нажимает клавишу. Другой сорт события может быть сгенерирован системным таймером Вашего компьютера. Совершенно другие типы событий генерируются различными модулями собственного программного обеспечения внутри Windows. События могут даже быть

сгенерированы функциями внутри Вашей программы или других программ (вспомните про реакцию на щелчки по кнопкам).

Может вызывать затруднения тот факт, что нет необходимости вызывать функции в ответ на эти события. Вместо этого посылаются сообщения объектам Windows, запрашивая у них необходимость ответить на события предопределенным образом. И даже, что более важно, функция доставки сообщений возлагается на Windows.

На аппаратном уровне каждое устройство ввода Windows управляется прерываниями. Когда нажимается клавиша на клавиатуре, возникает аппаратное прерывание, обычное функционирование Windows приостанавливается и управление передается фрагменту кода, называемому **I**nterrupt **S**ervice **R**outine — ISR (Программа обработки прерываний). ISR действует как драйвер устройства для мыши, клавиатуры и устройств, управляемых таймером.

При получении управления со стороны ISR происходит форматирование специфических данных с дальнейшим их сохранением в предопределенных регистрах. Затем вызывается специальная внутренняя подпрограмма Windows. Эта подпрограмма извлекает данные из регистров и помещает запись о событии в аппаратную очередь Windows. Обычно каждое событие создает новую запись события, но иногда несколько событий записываются в виде единственной записи. Например, подобное действие вызывает удержание клавиши до срабатывания автоповтора.

Аппаратные события ставятся в аппаратную очередь, форматируемую в виде обычных, преобразованных в последовательную форму сообщений Windows и помещенных во входную очередь приложения при помощи вторичного потока, выполняющегося внутри Windows. (Более ранние версии Windows выполняли подобную операцию только тогда, когда приложение запрашивало новое сообщение. Это позволяло приложению Windows, не поддерживающему взаимодействие, блокировать получение и обработку сообщений о событиях для других приложений.) Таким образом, даже при том, что аппаратный ввод является управляемым прерываниями, сообщения события обработано Вашим приложением в порядке "первым пришел — первым вышел" (FIFO — **f**irst-**i**n, **f**irst-**o**ut).

Сообщения — это стандартный механизм связи внутри программ Windows, точно так же, как вызовы функций — стандартный механизм связи в традиционно организованной программе DOS. Сообщения обычно генерируются одним из трех различных источников:

- События аппаратного ввода генерируются клавиатурой, мышью или таймером. Эти события иногда называются событиями, поставленными в очередь (enqueued events), поскольку они проходят через очередь аппаратного ввода.
- События диспетчера окон генерируются Windows непосредственно в ответ на действия пользователя, выполняющего операции типа перемещения или изменения размеров окна либо выбора пункта меню. Наиболее часто возникающее сообщение подобно типа — WM_PAINT, которое посылается для обновления содержимого области окна.
- Индивидуальные окна могут посылать сообщения другим окнам. Например, для указания текстовому полю об удалении текста посылается сообщение WM_SETTEXT. Текстовое поле также использует этот механизм для сообщения об изменении текста путем посылки программисту сообщения EN_CHANGE.

Сообщения Windows идентифицируются набором мнемонических идентификаторов. Как можно было бы предположить из факта использования символов верхнего регистра, каждое имя замещает целочисленную константу, используемую для идентификации специфического сообщения. Сообщения, непосредственно сгенерированные Windows, обычно начинаются с набора символов "WM", который означает сообщение Windows (**W**indows **M**essage). Сообщения, сгенерированные специфическими видами средств управления Windows, используют различные префиксы. Например, сообщения, сгенерированные средствами редактирования, имеют префикс "EN_", который означает уведомление редактирования (**E**dit **N**otification).

Архитектура ввода-вывода, независимая от внешних устройств

Во времена DOS при покупке высококлассного 21-дюймового монитора продавец должен был также снабдить вас набором про-

граммного обеспечения, необходимого для работы таких приложений, как Lotus или WordPerfect. В противном случае работа с этими приложениями оказалась бы просто невозможной.

С другой стороны, если вы создаете программное обеспечение, то ситуация еще более усложняется. Для каждого поддерживаемого приложения необходимо было знать специфику аппаратных средств, используемых в компьютере с тем, чтобы создавать программу с учетом обеспечения прямого доступа к аппаратным средствам.

Конечно, это справедливо не для каждого аппаратного устройства. Например, DOS обеспечивает очень хорошую поддержку различных файловых систем. Нет необходимости знать что-либо относительно числа дорожек и секторов на Вашем жестком диске для чтения и записи файлов. Помимо этого, можно использовать одну и ту же программу для чтения и записи файлов на жестком или гибком диске (устройство которого отличается от устройства жесткого диска). Операционная система (DOS в данном случае) поддерживает определенный уровень абстракции (файлы и каталоги), поэтому не нужно иметь дело с аппаратными средствами на физическом уровне (дорожки и секторы).

Таким образом, под управлением DOS запись файлов не зависит от внешних устройств, но вывод на дисплей или принтер таковым не является. Операционная система Windows намеревается устранить этот недостаток, предлагая **G**raphics **D**evice **I**nterface — GDI (Интерфейс графических устройств). Если программист желает осуществить вывод изображения на монитор, используя Windows, то он не будет иметь дело с монитором как с физическим устройством. Вместо этого нужно создать программу в терминах GDI; она должна работать с каждым новым монитором.

Связь и взаимодействие между процессами

В заключение Windows решила устранить недостаток DOS, заключающийся в невозможности одновременного выполнения различных программ и создания программ, взаимодействующих друг с другом. Более ранние версии Windows обеспечивали неполную многозадачность, например, особо "эгоистичное" приложение могло бы монополизировать время CPU. В 32-разрядных версиях Windows

(Windows 95, Windows 98 и Windows NT) Microsoft реализовала более строгую форму многозадачности.

Под управлением Windows приложения могут взаимодействовать двумя способами. Во-первых, системный буфер обмена (Clipboard) позволяет совместно использовать простые данные разными приложениями, даже если эти приложения были созданы различными компаниями. Второй способ задействует технологию **C**omponent **O**bject **M**odel (COM), также известную под названием OLE (**O**bject **L**inking and **E**mbedding). Программы, поддерживающие COM, могут выступать в качестве компонентов других приложений или как вспомогательные программы, обеспечивающие взаимодействие других программ.

Архитектура приложения Windows

Не удивительно, что программа Windows является большей частью коллекцией окон. Каково же точное определение окна? Хороший вопрос.

Окно — это прямоугольная область на экране, которая включает различные свойства и области.

Рассмотрим некоторые термины, связанные с окнами.

Клиентские и неклиентские области окон (client and non-client areas) — Каждое окно включает части, о которых заботится сама Windows, и части, которые должна обрабатывать Ваша программа. Часть, которую обрабатывает Ваша программа, называется клиентской областью. В этой области осуществляется пользовательский ввод и вывод. Оставшаяся часть окна — это неклиентская область, которую поддерживает Windows.

Стиль окна (Windows style) — стиль окна определяется при его создании. Существует три основных стиля окна:

WS_OVERLAPPEDWINDOW — используется для "основного" окна, окна верхнего уровня Вашего приложения. Такие окна могут перемещаться и изменять размер, появляясь при этом в любом месте экрана. Стиль WS_OVERLAPPEDWINDOW используется для окон верхнего уровня; для таких окон нет включающих их родительских (parent) окон, которые могут ограничивать движение по экрану.

WS_POPUPWINDOW — используется для диалоговых окон. Всплывающее окно имеет родительское окно, но может "плавать" в области перед родительским окном и может перемещаться вне экранной области родительского окна.

WS_CHILDWINDOW — используется для оконных элементов управления. Командная кнопка (типа кнопки ОК в типичном диалоговом окне) — дочернее окно, подобно текстовому полю или полосе прокрутки. Дочернее окно получает возможную область перемещения от родительского окна — невозможно выполнять его перемещение вне области перемещения родительского окна.

Как и следовало ожидать, окна являются самым важным объектом в программах Windows. В программе Windows все коды нажатых клавиш, информация о перемещениях мыши и прерывания таймера посылаются в качестве сообщений некоему выделенному окну, отвечающему за обработку этого действия. Даже элементы пользовательского интерфейса, типа кнопок и текстовых полей, являются специализированными типами дочерних окон.

Программирование приложения Windows по существу сводится к созданию кода, выполняющего следующие четыре задачи:

1. Инициализация.
2. Реализация.
3. Вызов цикла сообщений.
4. Ответ на сообщения.

Первые три задачи всегда выполняются функцией под названием WinMain(). Функция WinMain() является точкой входа для каждой программы Windows, точно так же как функция main() является точкой входа для каждой традиционной программы, написанной на языке С. Четвертая задача выполняется функцией, традиционно именуемой WndProc(), хотя ее можно называть как угодно.

Ваша программа может иметь дополнительные функции, если это необходимо, но каждая программа Windows должна включать, по крайней мере, эти две функции: WinMain() и WndProc().

После того как Windows загружает Вашу программу в память, она вызывает функцию WinMain(), которая определяет оконный класс для главного окна Вашего приложения. В определении оконного класса наиболее важные поля – это имя класса и адрес функции WndProc(), которая будет создана для формирования ответов на сообщения. Как только определены все атрибуты Вашего оконного класса, управление передается Windows, которая сохраняет эту информацию. Где? Да в системном регистре, - больше негде. Этот процесс называется *регистрацией*. После регистрации класса основного окна создается и отображается экземпляр основного окна приложения. Если Ваше окно активно, то все сообщения от действий пользователя направляются Вашему окну и обрабатываются зарегистрированной процедурой WndProc(). Подведем итог.

- Программа Windows — это коллекция окон, каждое из которых связано с оконной процедурой, реагирующей на события, сгенерированные Windows.
- Каждая программа Windows в обязательном порядке запускает функцию WinMain(), которая является ответственной за определение характеристик главного окна приложения. WinMain() сначала отображает главное окно, затем выполняет цикл для получения сообщений Windows и возвращает их обратно для последующей обработки.
- Каждое окно в программе Windows связано с процедурой, созданной для организации ответа на сообщения Windows. Эта процедура, по традиции именуемая WndProc(), всегда вызывается самой Windows и никогда не вызывается Вашей программой.

Программа MFC — это программа Windows. Это означает, что каждая программа MFC также включает функцию WinMain() и функцию WndProc(), обрабатывающую сообщения. Тем не менее в программе MFC Вы никогда не встретите ни одну из этих функций (при попытке их обнаружения в исходном коде MFC). Эти функции скрыты внутри каркаса приложения, чтобы не обременять программиста возможностью «уронить» операционную систему Windows вследствие неправильной или некорректной инициализации главного окна приложения и/или обработки сообщений.

Оглавление

Введение..... 3
1. Что такое Visual C++?..... 3
2. Проект программы: предварительное обсуждение... 6
3. Работаем с AppWizard..... 8
4. Встреча с панелью управления..... 14
5. Персонализация программы..... 15
6. Трансляция, сборка, выполнение программы..... 19
7. Проектирование программы и создание диалога.... 22
8. Знакомство с каркасом приложения..... 29
9. Начинаем работать с графикой..... 36
10. Вычерчивание графика функции..... 46
11. Вывод таблицы значений..... 54
11.1. Первый вариант вывода (простой и непрофессиональный)..... 59
11.2. Второй вариант вывода (профессиональный и тоже простой)..... 60
12. Рисуем на дисплее как карандашом..... 64
13. Секреты резиновой нити..... 72
14. Мелочи, которые порождают новые идеи..... 78
14.1. Создание объекта CMetaFileDC..... 80
14.2. Запись в метафайл графических данных.... 82
14.3. Воспроизведение метафайла..... 87
14.4. Различные детали..... 90

Заключение..... 95

Библиографический список..... 96

Приложение 1. Некоторые библиотечные функции MFC.... 97
Приложение 2. Принципы работы операционной системы Windows..... 106

Учебное издание

Данилин Александр Иванович

ПРОГРАММИРОВАНИЕ В СРЕДЕ MICROSOFT VISUAL C++ С ИСПОЛЬЗОВАНИЕМ MFC. ДИАЛОГОВОЕ ПРИЛОЖЕНИЕ

Учебное пособие

Технический редактор В. Д. Еленев
Редакторская обработка Н. С. Куприянова
Корректорская обработка Т. К. Кретинина
Доверстка Т. К. Кретинина

Подписано в печать 25.09.07. Формат 60x84 1/16.
Бумага офсетная. Печать офсетная.
Усл. печ. л. 7,25.
Тираж 120 экз. Заказ . ИП- 62/2007

Самарский государственный
аэрокосмический университет.
443086, Самара, Московское шоссе, 34.

Изд-во Самарского государственного
аэрокосмического университета.
443086, Самара, Московское шоссе, 34.