ПРОГРАММИРОВАНИЕ на C++ ГЛАЗАМИ ХАКЕРА

2-е издание

+CD

Михаил Фленов

Как сделать код маленьким, а программу невидимой
От простых шуточных программ до сложной манипуляции системой
Примеры работы через компоненты C++ и через Windows Sockets
Как работать с портами компьютера и получать информацию о системе
Интересные алгоритмы написания сетевых программ

bhv

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1
Ф69

Фленов М. Е.

Ф69 Программирование на C++ глазами хакера. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2009. — 352 с.: ил. + CD-ROM

ISBN 978-5-9775-0303-7

Рассмотрены нестандартные приемы программирования, а также примеры использования недокументированных функций и возможностей языка C++ при разработке шуточных программ и серьезных сетевых приложений для диагностики сетей, управления различными сетевыми устройствами и просто при повседневном использовании интернет-приложений. Во втором издании содержатся новые и переработаны старые примеры, а в качестве среды разработки используется Visual Studio 2008, хотя большинство описываемых примеров работоспособны в более старых версиях и в CodeGear C++ Builder.

Компакт-диск содержит исходные примеры из книги, а также популярные приложения компании CyD Software Labs.

Для программистов

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Елена Кашлакова*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 24.11.08.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 28,38.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.003650.04.08 от 14.04.2008 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0303-7

Оглавление

ГЛАВА 1

Оптимизация

В этой главе мы будем говорить об оптимизации, причем — с неожиданных сторон. Мы будем оптимизировать видимость окон, мы будем уменьшать размер программы, повышать скорость работы кода и даже защищать его. Какое отношение имеет защита к оптимизации? Иногда эти темы являются антиподами, и именно поэтому я решил объединить эти две темы под одной крышей, а точнее — в одной главе.

Что самое главное при написании программ-приколов? Ну, конечно же, невидимость. Программы, созданные в этой и следующих главах, будут незаметно сидеть в системе и выполнять нужные действия при наступлении определенного события. Это значит, что программа не должна отображаться на Панели задач или в списке запущенных программ, в окне, появляющемся при нажатии <Ctrl>+<Alt>+<Del>. Поэтому, прежде чем начать что-то писать, нужно узнать, как спрятать свое творение от чужого глаза.

Кроме этого, программы-приколы должны иметь маленький размер. Приложения, создаваемые Visual C++ с использованием современных технологий MFC (Microsoft Foundation Classes, базовые классы от Microsoft), достаточно "весомые". Даже простейшая программа, выводящая одно окно, займет достаточно много места на диске. Если вы захотите отослать такую шутку по электронной почте, то отправка и получение письма с вашей программой отнимут лишнее время у вас и получателя. Это не очень приятно, поэтому в этой главе мы познакомимся с тем, как можно уменьшить размер программ, создаваемых в Visual C++.

1.1. Сжатие исполняемых файлов

Самый простой способ уменьшить размер приложения — использование программы для сжатия файлов. Раньше я любил использовать ASPack (**http:/**

www.aspack.com), но в последнее время перешел на использование открытой библиотеки UPX. Эта библиотека распространяется в исходных кодах и может быть найдена по адресу **http://upx.sourceforge.net**. Для работы с библиотекой лучше иметь какую-нибудь визуальную программу. Тут я предпочитаю разработку Абдульманова Рафаэля Рахимовича (**http://rafsoft.narod.ru**) — UPX Control. В принципе, программа простая, но удобная. Главное окно программы можно увидеть на рис. 1.1.

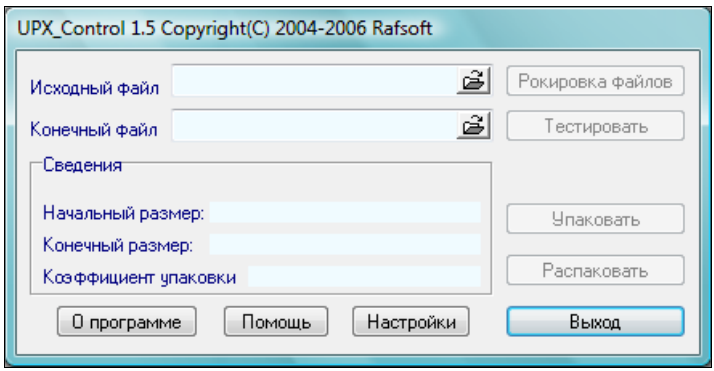

Рис. 1.1. Главное окно утилиты управления UPX

Окно простейшее, ибо нужно только выбрать исполняемый файл, который вы хотите сжать, результирующий файл — и нажать кнопку **Упаковать**. Этой же программой можно вернуться к изначальному состоянию, нажав кнопку **Распаковать**.

Теперь давайте разберемся, как работает сжатие. Сначала весь код программы сжимается архиватором. В программе используется алгоритм сжатия LZWA, оптимизированный для сжатия двоичного кода. В конец сжатого кода добавляется код разархиватора. И в самом конце ASPack изменяет заголовок исполняемого файла так, чтобы при старте сначала запускался разархиватор, который будет во время выполнения распаковывать программу в первоначальное состояние прямо в памяти и запускать распакованный вариант на исполнение.

В UPX алгоритм сжатия очень хороший, а разархиватор достаточно маленький (меньше 1 Кбайт), поэтому сжатие происходит очень сильно, а к результирующему файлу добавляется только один килобайт. Таким образом, программа может сжать файл размера в 1,5 Мбайт в 300–500 Кбайт.

Теперь, когда вы запускаете сжатую программу, сначала заработает разархиватор, который разожмет бинарный код программы и аккуратно поместит его в памяти компьютера. Как только этот процесс закончится, разархиватор передаст управление вашей программе.

Некоторые считают, что из-за расходов на распаковку программа будет работать медленней!!! Я бы сказал, что вы не заметите разницу. Даже если и будут какие-то потери, то они будут неощутимы (по крайней мере, на современных компьютерах). Это происходит потому, что архивация хорошо оптимизирована под двоичный код. И по сути дела, распаковка происходит только один раз и в дальнейшем никакого влияния на работу программы не оказывает. В результате, потери в скорости из-за сжатия будут неощутимы.

Программа без сжатия перед началом выполнения все равно грузится в память. В случае сжатого кода во время загрузки происходит разархивирование кода. В данном способе есть две стороны: происходят затраты времени на распаковку, но программа меньше занимает места на диске и быстрее считывается с него. Жесткий диск — одно из самых медленных звеньев персонального компьютера, поэтому чем меньше надо загружать, тем быстрее программа может приступить к выполнению. Именно вследствие этого итоговая потеря в скорости запуска программы незначительная.

При нормальном программировании с использованием всех современных возможностей типа визуальности и объектного программирования код получается большим, но его можно сжать специальным архиватором на 60–70%. А писать такой код намного легче и быстрее.

Еще одно "за" использование сжатия — заархивированный код труднее взломать, потому что не каждый дизассемблер сможет прочитать упакованные команды. Так что помимо уменьшения размера вы получаете защиту, способную остановить большинство взломщиков. Конечно же, профессионала этим не отпугнешь. Но взломщик средней руки не будет возиться со сжатым двоичным кодом.

Тут нужно быть честным и вспомнить, что и описанная ранее утилита работы с UPX умеет разархивировать, а значит, хакер тоже сможет ей воспользоваться. Декомпиляторы и отладчики тоже не из каменного века, и хорошие разработки умеют определять алгоритм сжатия и распаковывать код автоматически при открытии файла без дополнительных программ.

Коммерческие программы сжатия, такие как ASPack, имеют функции шифрования, которые могут усложнить жизнь хакерам. Опять же, только усложнить, но не испортить вовсе. Для исполнения код в любом случае будет расшифрован, а в арсенале взломщиков есть утилиты, позволяющие снимать код из памяти для дальнейшего анализа. Этот процесс не из легких, но вполне возможный.

1.2. Без окон, без дверей...

Следующий способ уменьшить размер программы заключается в ответе на вопрос, из-за чего программа, созданная в Visual C++, получается большой.

Ответ очень прост: C++ является объектно ориентированным языком. В нем каждый элемент представляет собой объект, который обладает своими свойствами, методами и событиями. Любой объект вполне автономен и многое умеет делать без ваших указаний. Это значит, что вам нужно только подключить его к своей форме, изменить нужным образом свойства — и приложение готово! И оно будет работать без какого-либо прописывания его деятельности.

Но в объектном программировании есть и свои недостатки. В объектах реализовано большое количество действий, которые вы и пользователь сможете производить с ними. Но реально в любой программе мы используем два-три из всех них. Все остальное — для программы лишний груз, который никому не нужен.

Но как же тогда создать компактный код, чтобы программа занимала минимум места на жестком диске и в оперативной памяти? Тут есть несколько вариантов:

- не использовать библиотеку MFC, которая упрощает программирование на языке C++. В этом случае придется больше писать кода вручную и работать только с Windows API (Windows Application Programming Interface, прикладной программный интерфейс). Программа получается очень маленькой и быстрой. Результирующий код будет меньше, чем при использовании MFC в сочетании с самым большим сжатием. Но таким образом вы лишаетесь простоты визуального программирования и можете ощутить все неудобства программирования с помощью чистого WinAPI.

  Для большей оптимизации размера файла можно использовать ассемблер, но он относительно сложен, да и писать программу на нем намного дольше, чем даже на чистом C. Хотя кому как. Именно поэтому данная тема не рассматривается в этой книге;

- сжимать готовые программы с помощью компрессоров. Объектный код сжимается в несколько раз, и программа, созданная с использованием MFC, может превратиться из монстра в 300 Кбайт в скромного по размерам "зверя", занимающего на диске всего 30–50 Кбайт. Главное преимущество состоит в том, что вы не лишаетесь возможностей объектного программирования и можете спокойно забыть про неудобства WinAPI.

Второй метод мы уже обсудили *(см. разд. 1.1)*, поэтому остается только описать первый, и самый интересный, вариант. Полученный код уже получается маленьким, и его можно еще дополнительно сжать с помощью утилиты UPX. Результат будет потрясающим.

Если вы хотите создать действительно компактную программу, то необходимо забыть про все удобства. Вы не сможете подключать визуальные формы или другие удобные модули, написанные фирмой Microsoft для упрощения

жизни программиста. Нельзя использовать классы или компоненты ActiveX. Только функции API самой ОС Windows — и ничего больше.

Теперь переходим к практическим занятиям. Запустите Visual C++ и создайте новый проект. Для этого нужно выбрать команду меню **File | New | Project** (Файл | Новый | Проект). Перед вами откроется окно создания нового проекта (рис. 1.2). Слева расположено дерево типов проектов. Нас интересует C++, поэтому выберите пункт **Visual C++**. Этот пункт мы будем выбирать при написании абсолютно всех примеров из данной книги. С правой стороны в списке **Templates** (Шаблоны) варианты создания различных проектов. Выберите пункт **MFC Application** (Приложение MFC).

Внизу окна расположены две строки ввода. В первой вы должны указать имя создаваемого проекта. Оно будет использоваться в качестве имени запускаемого файла и имени файла, который вы будете в дальнейшем открывать для редактирования. Давайте здесь укажем: `TestMFC`.

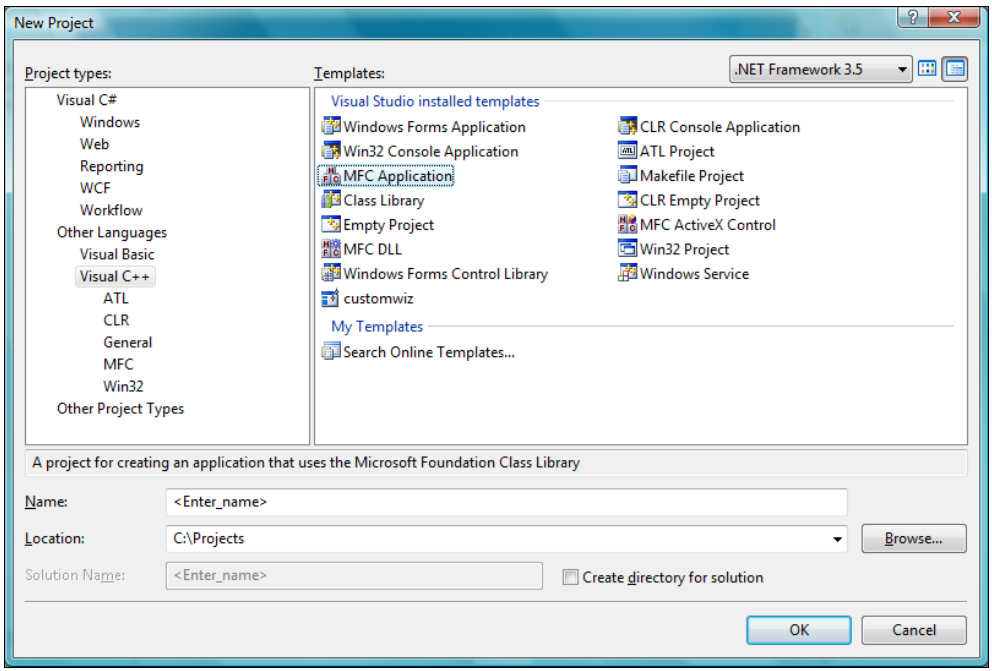

Рис. 1.2. Окно выбора типа проекта

В строке **Location** (Расположение) нужно указать путь к папке, в которой среда разработки создаст необходимые проекту файлы. Я рекомендую завести свою папку с именем, например, My C++ Projects (или любым другим на

выбор, но понятным по смыслу), где будут размещаться все проекты. Выберите эту папку и нажмите **ОК**. По окончании работы мастера у вас в папке My C++ Projects появится еще одна папка с именем TestMFC, в которой и будут находиться все файлы данного проекта.

Как только вы нажали **ОК** в окне создания нового проекта (см. рис. 1.2), перед вами откроется окно Мастера создания нового MFC-приложения (рис. 1.3).

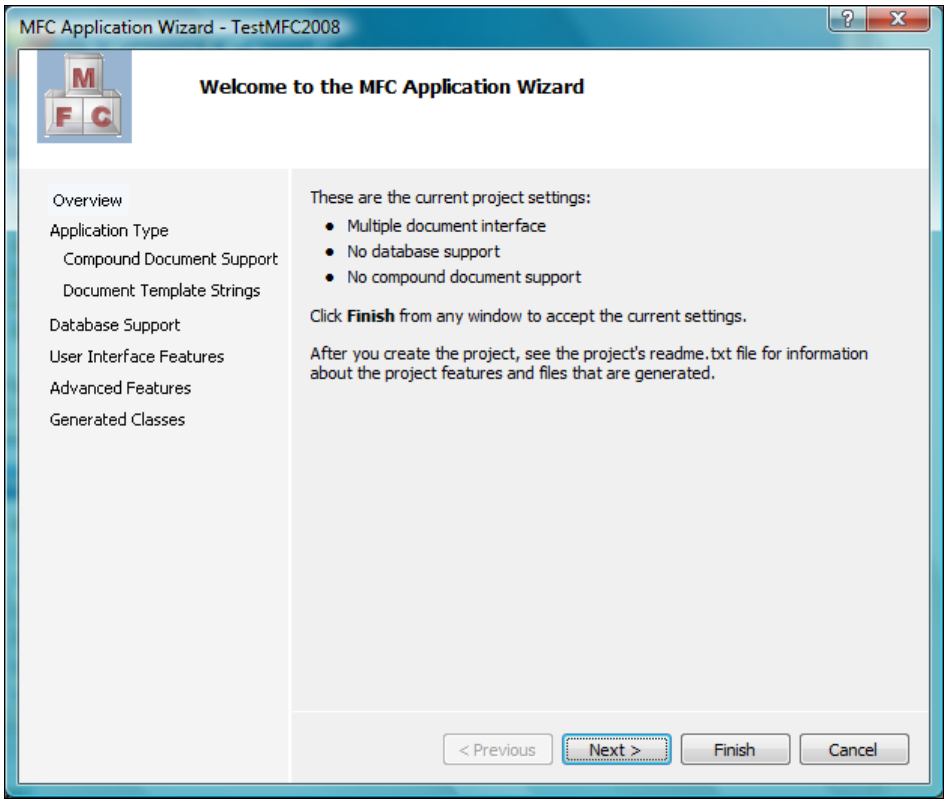

Рис. 1.3. Окно Мастера создания проекта

Вы можете сразу нажать кнопку **Finish**, чтобы завершить его работу с параметрами по умолчанию, или предварительно указать свои настройки. Наша задача — создать маленькое приложение, поэтому на данном этапе постараемся оптимизировать то, что может создать для нас Мастер.

С левой стороны окна расположены разделы. Выделяя их, вы можете настраивать соответствующие параметры. Давайте просмотрим все разделы и установим необходимые значения, а заодно познакомимся с теми параметра-

ми, которые будут использоваться для создания приложений при рассмотрении последующих примеров:

- **Application Type** (Тип приложения) — в этом разделе мы задаем тип создаваемого приложения. Давайте укажем здесь следующие значения:
  - **Single Document** (Документ с одним окном) — нам достаточно будет только одного окна. Многодокументные приложения мы не рассматриваем, поэтому большинство примеров с использованием MFC будет основываться на этом типе приложений или на основе диалоговых окон (**Dialog based**);
  - **Project style** (Стиль проекта) — во всех приложениях будем использовать стиль по умолчанию (**MFC стандарт**);
  - **Document | View architecture support** (Поддержка архитектуры Документ | Просмотр) — это значение нас пока не интересует, поэтому оставим по умолчанию;
- **Advanced Features** (Дополнительные возможности) — в этом разделе в будущем нас будет интересовать только параметр **Windows socket** (поддержка сокетов Windows), который позволит нам писать примеры для работы с сетью.

Во всех остальных разделах оставляем значения по умолчанию, потому что мы не будем использовать базы данных или документы. В большинстве случаев нам будет достаточно окна и меню. А первое время постараемся обходиться вообще без MFC.

Нажмите кнопку **Finish**, чтобы завершить работу мастера. По завершении его работы вы увидите главное окно среды разработки Microsoft Visual C++, представленное на рис. 1.4. Это окно мы будем использовать достаточно часто, поэтому в процессе работы уточним все детали.

Сейчас нас интересуют параметры проекта. Мы должны отключить все, что нам будет мешать. При сборке проекта в Visual C++ по умолчанию могут использоваться два вида настроек: debug и release. Первый необходим на этапе разработки, и в этом режиме Visual C++ создает запускаемый файл, который содержит слишком много дополнительной информации. Она будет необходима вам в среде разработки в дальнейшем при отладке программы и поиске ошибок. Во втором режиме эта информация отключается, и запускаемый файл будет меньшего размера.

В верхней части окна на панели с кнопками найдите выпадающий список, в котором написано **Debug**. Измените это значение на **Release.**

Среда разработки Visual C++ может создавать запускаемые файлы, использующие MFC-библиотеки двух типов: статические и динамические. По умолчанию используется динамическая сборка. В таком режиме запускаемый

файл получается меньшего размера, но он не будет работать без динамических библиотек, таких как mfcXXX.dll, где XXX — это номер версии среды разработки.

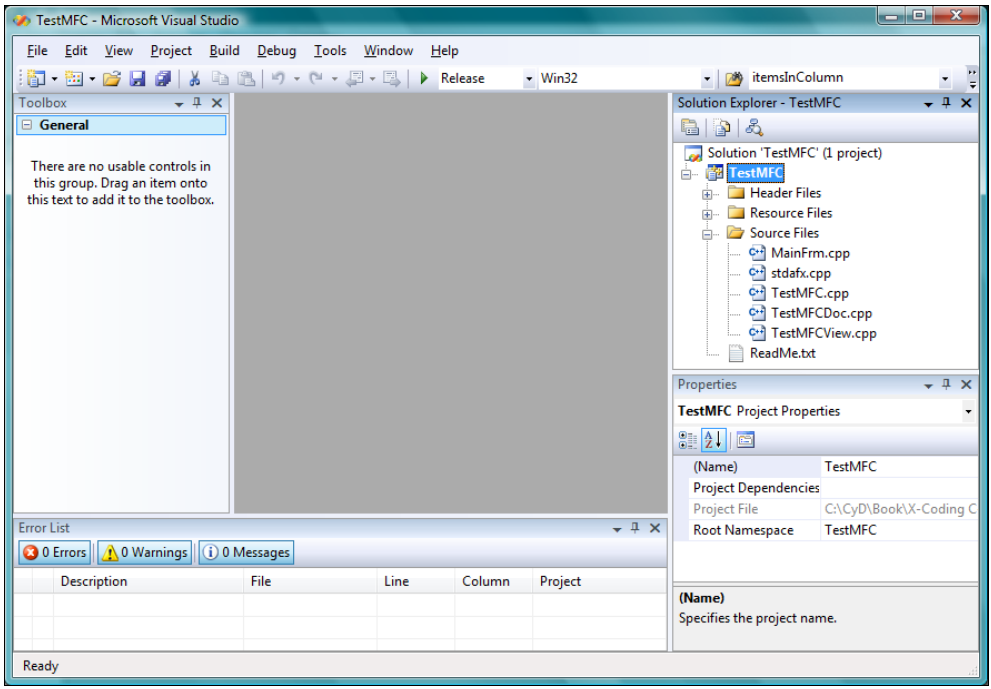

Рис. 1.4. Окно среды разработки

В этом случае, чтобы кто-то смог запустить наш проект, мы должны отослать ему не только запускаемый файл, но и библиотеки. Это неудобно и неприлично. Лучше использовать статическую компиляцию, при которой результирующий файл будет намного больше, зато все будет содержать внутри себя. При таком подходе не потребуются дополнительные библиотеки.

Чтобы изменить тип использования MFC, в окне **Solution Explorer** сначала выберите имя вашего проекта, а затем в меню команду **Project/Properties**. На рис. 1.5 представлено окно свойств проекта.

Примечание

После выхода первой книги я получил много вопросов, когда пользователи не смогли скомпилировать проекты. Это случилось у тех, у кого по умолчанию в IDE выбирается Multibyte-строки. Я не использовал Unicode-строки, поэтому если компилятор будет ругаться, то в параметре **Character Set** выберите значение **No Set**.

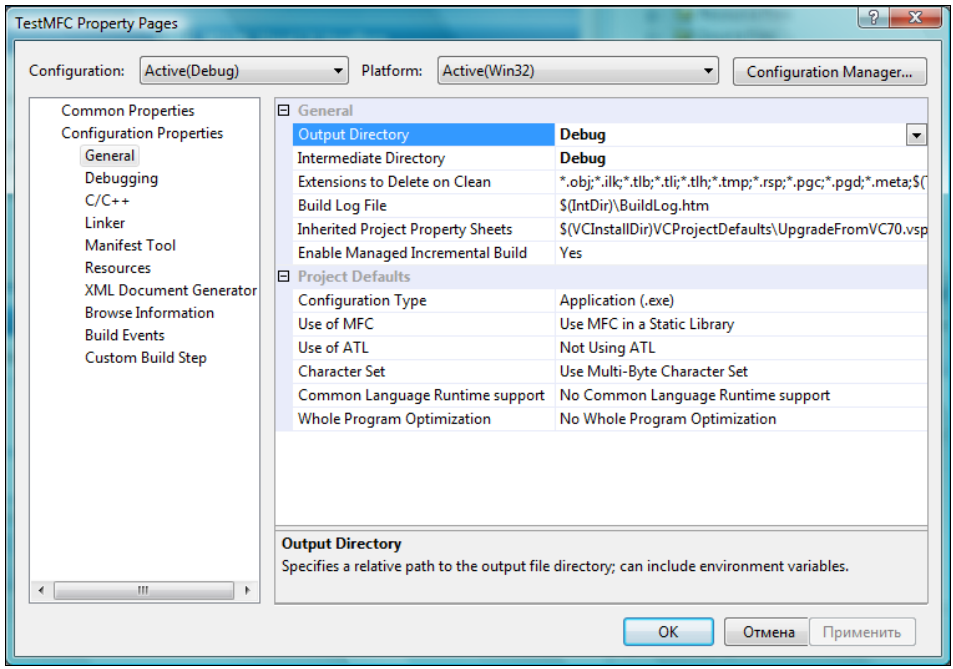

Рис. 1.5. Окно свойств проекта

Слева в окне расположены разделы свойств. Нас будет интересовать раздел **General** (Основные). Выделите его, и в основном окне появится список соответствующих свойств. Найдите свойство **Use of MFC** и измените его значение на **Use MFC in a Static Library**. Нажмите кнопку **ОК**, чтобы закрыть окно и сохранить изменения.

Теперь соберем наш проект в готовый исполняемый файл. Для этого нужно выбрать команду меню **Build | Build solution** (Построить | Построить проект). Внизу главного окна, в панели **Output** (Вывод), будет появляться информация о ходе сборки. Дождитесь, пока не появится сообщение типа:

```
---------------------- Done ----------------------
    Build: 1 succeeded, 0 failed, 0 skipped
```

Теперь перейдите в папку, которую вы выделили под хранение проектов, и найдите там папку TestMFC. В ней расположены файлы с исходным кодом нашего проекта, сгенерированные мастером. Тут же должна быть папка Release, в которой среда разработки создала во время компиляции промежуточные и исполняемый файлы. Выделите файл TestMFC.exe и посмотрите его свойства (надо щелкнуть правой кнопкой мыши и выбрать в появившемся меню пункт **Свойства**). Размер нашего пустого проекта у меня в Visual

Studio 2008 составил 446 Кбайт. От такого размера не только звезда будет в шоке, но и простые программисты, как мы! Это очень много. В Visual Studio 2003 это было около 380 Кбайт.

Попробуйте открыть его в программе UPX и сжать. У меня сжатый исполняемый файл составил 209 Кбайт. Сжатие составило более 50%, и это уже более или менее приемлемый размер для шуточной программы.

Примечание

Пример этой программы вы можете увидеть на компакт-диске в папке /Demo/Chapter1/TestMFC. Чтобы открыть этот пример, выберите команду меню **File | Open solution**. Перед вами появится стандартное окно открытия файлов. Перейдите в нужную папку и выберите файл с именем проекта и расширением .vcproj или .sln.

Чтобы сделать программу еще меньше, необходимо отказаться от MFC и писать на чистом C. Это немного сложнее и не так удобно, но для небольших проектов вполне приемлемо.

Для того чтобы создать маленькую программу без использования MFC, нужно снова использовать меню **File | New | Project** и здесь выбрать уже тип создаваемого проекта **Win32 Project**. В качестве имени давайте укажем `ctest`, а путь оставим тот же.

Если у вас все еще открыт предыдущий проект, то под строкой ввода пути для проекта есть переключатели: **Add to solution** (Добавить в решение) и **Close solution** (Закрыть решение). Если вы выберете первый из них, то текущий проект будет добавлен в уже открытый. Если выбрать закрытие, то текущий проект будет закрыт, и для вас будет создано новое рабочее поле.

После нажатия кнопки **ОК** перед вами откроется окно Мастера. Первый шаг чисто информационный, поэтому выберите раздел **Application Settings** (Настройки приложения). Перед вами откроется окно, как на рис. 1.6.

Нас интересует простое приложение Windows, поэтому вы должны выбрать в разделе **Application type** (Тип приложения) переключатель **Windows application**. Больше ничего, чтобы мастер не добавлял ничего лишнего. Нам необходим только самый минимум. Нажмите кнопку **Finish**, и будет создан новый проект.

Здесь также нужно изменить **Debug** на **Release**, чтобы создать проект без дополнительной информации. В настройках проекта ничего менять не надо, потому что созданный мастером шаблон не использует MFC, и ему не нужны динамические библиотеки. Можете зайти в свойства проекта и убедиться, что в свойстве **Use of MFC** стоит **Standard Windows Libraries** (Использовать стандартные библиотеки Windows). Это значит, что нет MFC, и ничего до-

полнительного программе не надо, только стандартные библиотеки Windows и прямой вызов API операционной системы.

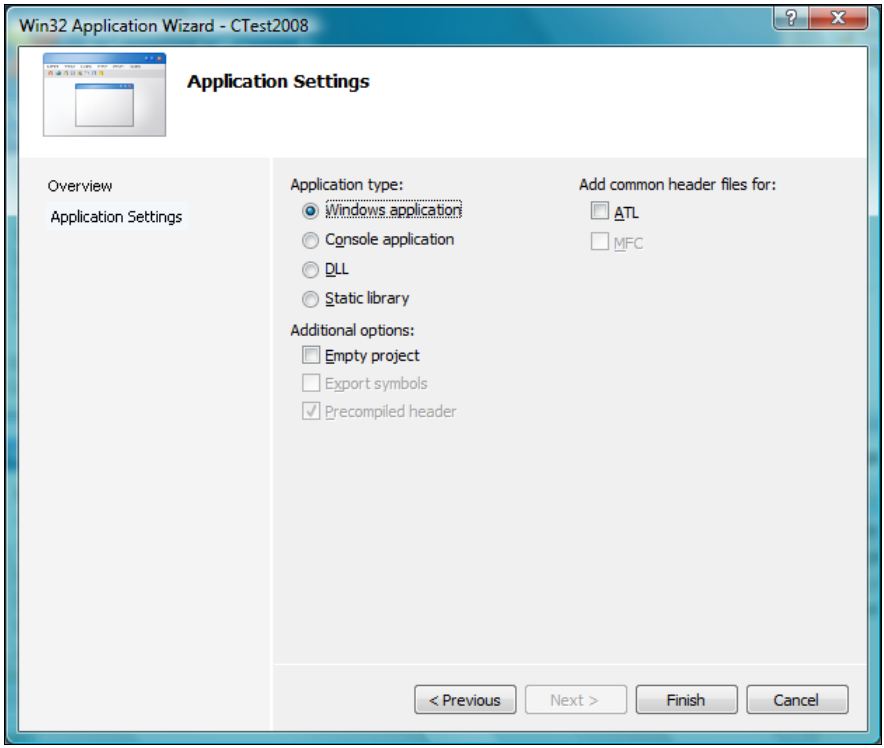

Рис. 1.6. Параметры приложения Win32

Попробуйте откомпилировать проект. Для этого выберите команду меню **Build | Build solution**. По окончании сборки перейдите в папку ctest/Release каталога, где вы создавали проекты, и посмотрите на размер результирующего файла. У меня в Visual Studio 2008 получилось 86 Кбайт. В Visual Studio 2003 размер составлял 81 Кбайт. Если сжать файл, то получится менее 70 Кбайт. Вот такая программа уже скопируется по сети достаточно быстро.

Программу можно еще больше оптимизировать и убрать некоторые вещи, которые не используются, но отнимают драгоценные килобайты, однако мы этого делать уже не будем. Сделаем только пару выводов: размер файла зависит от используемых функций и может отличаться в зависимости от используемого компилятора. Каждый компилятор может использовать разные функции и методы оптимизации.

Если вы не имеете опыта программирования, то я бы порекомендовал сейчас отложить книгу и прочитать GetStarted.doc на компакт-диске в каталоге Doc и

содержимое Doc\For Beginer, где я выложил дополнительную информацию для начинающих программистов, чтобы материал книги был понятен всем и интересен программистам с разным уровнем подготовки.

1.3. Оптимизация программ

Вся наша жизнь — это борьба с тормозами и нехваткой времени. Каждый день мы тратим по нескольку часов на оптимизацию. Каждый из нас старается оптимизировать все, что попадает под руку. А вы уверены, что вы это делаете правильно? Может быть, есть возможность что-то сделать еще лучше?

Я понимаю, что все сейчас разленились и выполняют свои обязанности спустя рукава. Лично я до такой степени привык, что за меня все делает компьютер, что даже забыл, как выглядит шариковая ручка. Недавно мне довелось писать заявление на отпуск на простой бумаге, так я долго вспоминал, как пишется буква "ю". Я помню, что эта клавиша находится возле правой клавиши <Shift>, но как ее писать... Пришлось подглядывать, как она выглядит на клавиатуре. Это не шутка. Это прогресс, благодаря которому я все делаю на компьютере.

Даже для того, чтобы написать текст из двух строк, мы включаем свой компьютер и загружаем MS Word, тратя на это драгоценное время. А может, легче было бы написать этот текст на бумаге?

Программисты считают, что раз их творение (в виде исходного кода) никто не увидит, то можно писать что угодно. Так это они ошибаются. С этой точки зрения программы с открытым исходным кодом имеют большое преимущество, потому что намного чище и быстрей. Создавая код, мы ленимся его оптимизировать не только в плане размера, но и в плане скорости. Глядя на такие вещи, хочется ругаться непристойными словами, только программа от этого лучше не станет.

Хакеры далеко не ушли. Если раньше, глядя на программиста или хакера, возникал образ прокуренного, заросшего и немытого молодого человека, то сейчас это цифровое существо, за которое все выполняют машины.

Все это деградация! Мы берем в руки мышку и начинаем тыкать ею, где попало, забывая про клавиатуру и горячие клавиши. Мышка была придумана в качестве помощника, а не заменителя клавиатуры. Я считаю, что надо бороться с этим. Меня самого некоторое время назад иногда разбирала такая лень, что я убирал клавиатуру, запускал экранную клавиатуру и начинал работать только мышкой. Осталось только покрыть мое тело шерстью и посадить в клетку к таким же ленивым шимпанзе.

Но теперь я работаю с ноутбуком, его клавиатуру не убрать. Работать с Touch Pad не так удобно, поэтому про клавиатуру никак не забыть.

Не надо тратить большие деньги на модернизацию компьютера. Лучше начните изменения с себя. Давайте оптимизируем свою работу и свои творения, и тогда компьютер заработает намного быстрее.

Изначально эта часть книги задумывалась как рассказ об оптимизации кода программ. Но впоследствии объединил советы с реальной жизнью, потому что оптимизировать надо все. Я буду говорить про теорию оптимизации, а ее законы действуют везде. По одним и тем же законам вы можете оптимизировать свой распорядок дня, чтобы успевать все сделать, и свою ОС, чтобы она работала быстрей. Но основное все же будет относиться к коду программ.

Как всегда, я постараюсь давать больше реальных примеров, чтобы вы могли применить все сказанное на практике.

Начну я с законов, которые работают не только в программировании, но и в реальной жизни. Ну, а напоследок оставлю только то, что может пригодиться при оптимизации кода.

Закон № 1

Оптимизировать можно все. Даже там, где вам кажется, что все и так работает быстро, можно сделать еще быстрее.

Это действительно так. И этот закон очень сильно проявляется в программировании. Идеального кода не существует. Даже простую операцию сложения 2 + 2 тоже можно оптимизировать. Чтобы достичь максимального результата, нужно действовать последовательно и, желательно, в том порядке, в котором описано далее.

Помните, что любую задачу можно решить хотя бы двумя способами (если не больше), и ваша задача — выбрать наилучший метод, который обеспечит желаемую производительность и универсальность.

Закон № 2

Первое, с чего нужно начинать, — это поиск самых слабых и медленных мест.

Зачем начинать оптимизацию с того, что и так работает достаточно быстро! Если вы будете оптимизировать сильные места, то можете встретить неожиданные конфликты. Да и эффект будет минимален.

Тут же я вспоминаю пример из своей собственной жизни. Примерно в 1996 году меня посетила одна невероятная идея — написать собственную игру в стиле Doom. Я не собирался ее делать коммерческой, а хотел только потренировать мозги на сообразительность. Четыре месяца невероятного труда, и нечто похожее на движок уже было готово. Я создал один голый

уровень, по которому можно было перемещаться, и с чувством гордости побежал по коридорам.

Никаких монстров, дверей и атрибутики в нем не было, а тормоза ощущались достаточно значительные. Тут я представил себе, что будет, если добавить монстров и атрибуты, да еще и наделить все это AI... Вот тут чувство собственного достоинства поникло. Кому нужен движок, который при разрешении 320×200 (тогда это было круто!) в голом виде тормозит со страшной силой? Вот именно...

Понятное дело, что мой виртуальный мир нужно было оптимизировать. Целый месяц я бился над кодом и вылизывал каждый оператор моего движка. Результат — мир стал прорисовываться на 10% быстрей, но тормозить не перестал. И тут я увидел самое слабое место — вывод на экран. Мой движок просчитывал сцены достаточно быстро, а пробоиной был именно вывод изображения. Тогда еще не было шины AGP, и я использовал простую PCI-видеокарту от S3 с 1 Мбайт памяти. Пара часов колдовства, и я выжал из PCI все возможное. Откомпилировав движок, я снова загрузился в свой виртуальный мир. Одно нажатие клавиши "вперед", и я очутился у противоположной стены. Никаких тормозов, сумасшедшая скорость просчета и моментальный вывод на экран.

Как видите, моя ошибка была в том, что вначале я неправильно определил слабое место своего движка. Я месяц потратил на оптимизацию математики, и что в результате? Мизерные 10% прироста производительности. Но когда я реально нашел слабое звено, то смог повысить этот параметр в несколько раз.

Именно поэтому я говорю, что надо начинать оптимизировать только со слабых мест. Если вы ускорите работу самого слабого звена вашей программы, то, может быть, и не понадобится ускорять другие места. Вы можете потратить дни и месяцы на оптимизацию сильных сторон и увеличить производительность только на 10% (что может оказаться недостаточным) или несколько часов на совершенствование слабой части и получить улучшение в 10 раз!

Слабые места компьютера

Меня поражают люди, которые гонятся за мегагерцами процессора и сидят на доисторической видеокарте от S3, жестком диске на 5400 оборотов и с 32 Мбайт памяти. Посмотрите в корпус своего компьютера и оцените его содержимое. Если вы увидели, что памяти у вас не более 256 Мбайт, то встаньте и громко произнесите: "Уважаемый DIMM, команда выбрала вас. Вы сегодня самое слабое звено и должны покинуть мой компьютер". После этого покупаете себе гигабайт (а лучше — 2) памяти и наслаждаетесь ускорением работы Visual C++, Photoshop и других "тяжелых" программ.