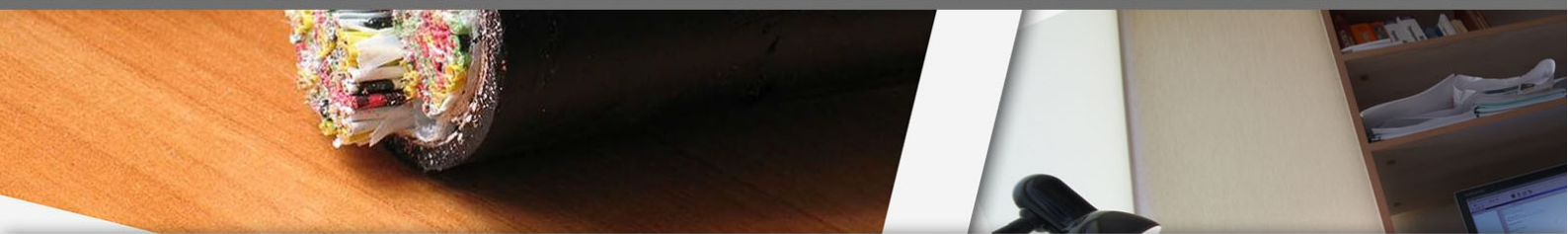


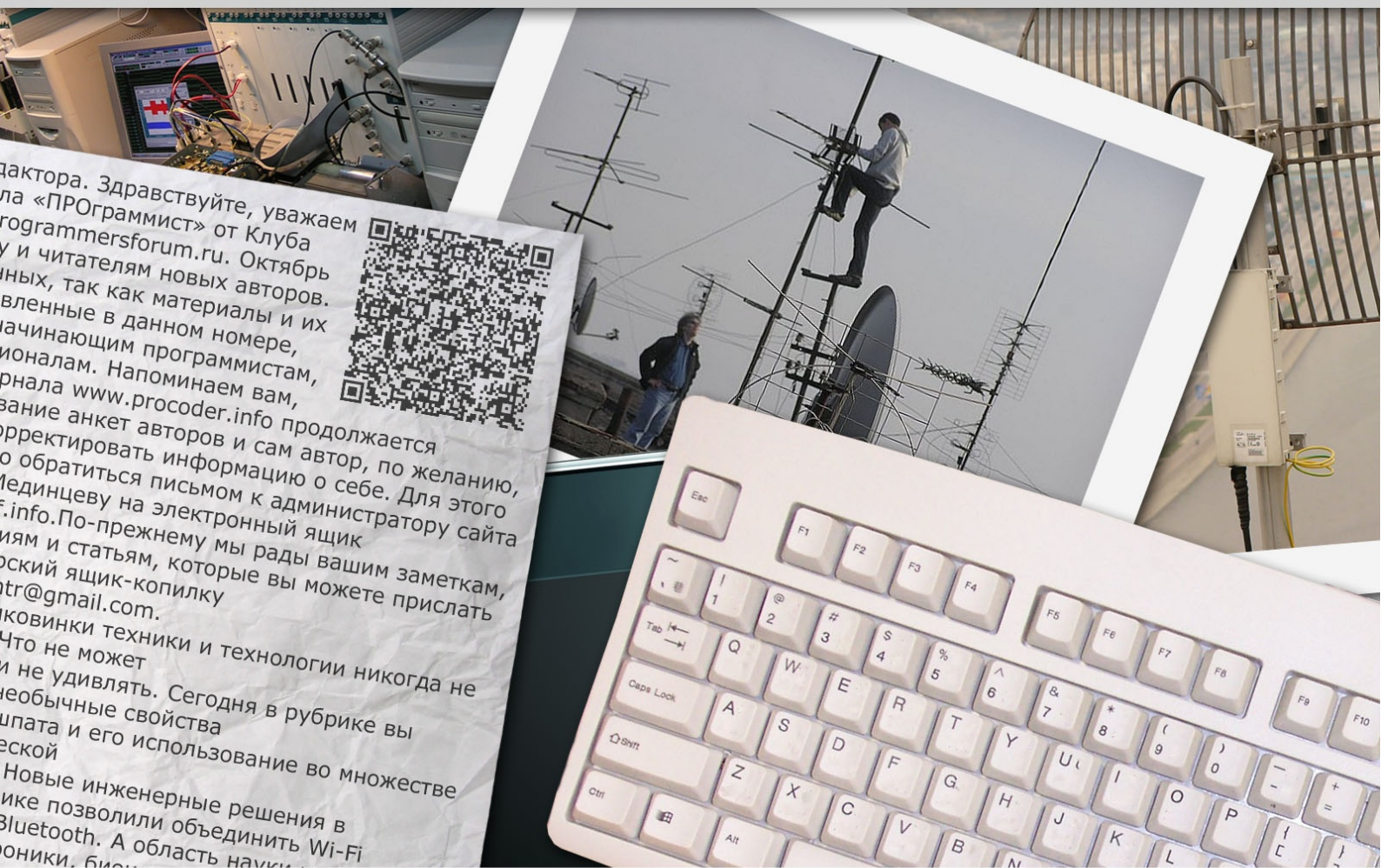
# программист

журнал клуба программистов



## Сегодня в номере:

- История одного ЛА
- Может-ли ПО работать быстрее?
- Маленькие помощники программиста
- Беспроводная сеть масштаба микрорайона
- Layered в Windows. Использование слоев
- Компилятор домашнего приготовления
- Работа с MySQL в C++ через mysql++
- И многое другое...



Издается с марта 2010. Выходит ежемесячно  
№7, октябрь 2010 г.

**Редакция:****Выпускающий редактор**

Сергей Бадло

**Литературный редактор**

Utkin

**Редакторы**

JTG, Василий Мединцев,  
Алексей Шульга, Егор Горохов

**Редактор-корректор**

Yan Liplavskiy

**Дизайн и верстка:**

Егор Горохов, Сергей Бадло

**Авторский состав:**

Utkin, Сергей Апанесевич,  
Алексей Шишкин, Александр Горский,  
Владимир Лихонос, Виталий Белик,  
Олег Кутков, Сергей Бадло

**Официальный сайт журнала:**

[www.procoder.info](http://www.procoder.info)

**Контакты:**

Авторские статьи направляйте на  
[maindatacentr@gmail.com](mailto:maindatacentr@gmail.com)  
Вопросы и предложения для редакции  
[reddatacentr@gmail.com](mailto:reddatacentr@gmail.com)  
Вопросы и предложения администратору  
[info@procoder.info](mailto:info@procoder.info)

**Информационная поддержка:**

Международная Академия Информатизации  
(МАИН) РК [www.academy.kz](http://www.academy.kz)  
Журнал «Радиолобитель»  
[www.radioliga.com](http://www.radioliga.com)  
Клуб ПРОграммистов  
[www.programmersforum.ru](http://www.programmersforum.ru)  
V.K. сайт...  
[www.kotoff.info](http://www.kotoff.info)  
Free Legal Soft Group  
[www.flsoft.ru](http://www.flsoft.ru)  
AirNet-Berdyansk  
[www.airnet.sytes.net](http://www.airnet.sytes.net)

**Примечание:**

Издание некоммерческое. Все материалы,  
товарные знаки, торговые марки и логотипы,  
упомянутые в журнале, принадлежат их  
владельцам. Статьи, поступающие в редакцию,  
рецензируются. Мнение авторов не всегда  
совпадает с мнением редакции. Перепечатка  
материалов журнала и использование их в  
любой форме, в том числе в электронных СМИ,  
возможны только с разрешения редакции.  
Тираж неограничен. Формат А4, 57 стр.

**Учредитель:**

Клуб ПРОграммистов  
[www.programmersforum.ru](http://www.programmersforum.ru)

**Обложка номера:**

Дизайн Егора Горохова

**ТЕМА НОМЕРА**

Наши разработки ..... с.0x02

**НЕВЕРОЯТНО, НО ФАКТ**

Любопытные факты ..... с.0x03

**АЛГОРИТМЫ**

История одного лексического анализатора ..... с.0x08

Может-ли ПО работать быстрее. Взгляд изнутри ..... с.0x0F

**ОТДЕЛ ТЕСТИРОВАНИЯ**

Маленькие помощники программиста. Часть 2 ..... с.0x13

**WI-FI СЕТИ**

Беспроводная сеть масштаба микрорайона. Часть 2 ..... с.0x16

**ЛАБОРАТОРИЯ**

Layered в Windows. Использование слоев ..... с.0x1C

Компилятор домашнего приготовления. Часть 2 ..... с.0x22

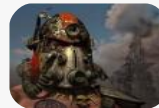
**АРХИВ**

Работа с MySQL в C++ через mysql++ ..... с.0x32

**ЮМОР**

Невыдуманные истории ..... с.0x36

**От редактора.** Здравствуйте, уважаемые читатели журнала «ПРОграммист» от Клуба ПРОграммистов [www.programmersforum.ru](http://www.programmersforum.ru). Октябрь подарил нашему проекту и читателям новых авторов. Будем надеяться, постоянных, так как материалы и их опыт, представленные в данном номере, будут полезны не только начинающим программистам, но и профессионалам. Напоминаем вам, что на официальном сайте журнала [www.procoder.info](http://www.procoder.info) продолжается формирование анкет авторов и сам автор, по желанию, может скорректировать информацию о себе. Для этого достаточно обратиться письмом к администратору сайта Василию Мединцеву на электронный ящик [info@kotoff.info](mailto:info@kotoff.info). По-прежнему мы рады вашим заметкам, предложениям и статьям, которые вы можете прислать на редакторский ящик-копилку [maindatacentr@gmail.com](mailto:maindatacentr@gmail.com).



### В этом выпуске...

Порадует вас материалом по самостоятельному построению лексического анализатора наш литературный редактор **Utkin**, что даст возможность с легкостью осуществить поддержку собственных API и скриптов в различных приложениях.

В рубрике «Отдел тестирования» Алексей Шишкин продолжит разработку маленьких, но полезных помощников в повседневной работе программиста.

Сергей Апанасевич, наш дебютант, в рубрике «Алгоритмы» покажет, как можно реализовать вычисления оптимальным и экономичным способом. Его взгляд на ПО изнутри довольно любопытен.

Александр Горский сегодня расскажет про принципы построения, учета и ведения статистики беспроводной сети масштаба микрорайона. Читаем рубрику «Wi-Fi сети».

В рубрике «Лаборатория» с материалом по созданию окон необычной формы и использованием различных эффектов выступает наш новый автор Владимир Лихонос.

А секретами домашней подготовки компиляторов продолжит делиться наш шеф-повар Виталий Белик. Пальчики оближешь!

В рубрике «Архив» эстафету от нашего форумчанина **psycho-coder** по работе с MySQL в C++, но уже в среде UNIX, принял Олег Кутков. На этот раз вы узнаете, как работать с СУБД MySQL, используя библиотеку `mysql++`.

### Рубрики журнала (плавающие)

- Новости программирования
- Отдел тестирования
- Общие вопросы (правовое использование)
- Алгоритмы
- Переводные материалы
- Разработка (проекты от этапа ТЗ до сдачи)
- Wi-Fi сети
- Лаборатория
- Архив (по материалам клуба и форума)
- Юмор (специфические хохмы программистов)

### Общие требования к материалам

У нас нет категоричных требований к оформлению, но в связи с особенностями верстки (используется свободное ПО «**SCRIBUS**») и облегчения труда редакторов, есть некоторый желательный минимум:

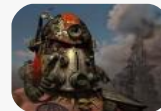
- статья должна иметь выраженную структуру с разделами и содержать название статьи, сведения об авторах, экскурс или введение, информацию об используемых средствах разработки, теоретическую и/или практическую часть, заключение и ресурсы к статье;
- текст статьи без табуляции в формате MS Word, **VK WordPad** или обычным текстовым файлом, шрифт Arial 10;
- все рисунки, таблицы должны быть подписаны и иметь упоминание в тексте;
- рисунки к статье должны прилагаться **в виде отдельных файлов** в формате PNG, TIF;
- разделы статьи отделять двумя <ENTER>.

По присланным материалам автор получает рецензию и корректирует статью согласно замечаниям. Шаблон для написания статьи можно взять [тут](#).

С уважением, Редакция



Пожалуй, диковинки техники и технологии никогда не закончатся. Что не может не радовать и не удивлять. Сегодня в рубрике вы узнаете про необычные свойства исландского шпата и его использование во множестве сфер человеческой деятельности. Новые инженерные решения в радиоэлектронике позволили объединить Wi-Fi стандарта N и Bluetooth. А область науки на стыке той же радиоэлектроники, биоинженерии и химии дала возможность создать самовосстанавливающиеся солнечные элементы, ведь, как известно, срок службы тех же пластин из монокристаллического кремния ограничен 25...30-тью годами. Но, это не все новости на сегодня. Итак...



**Сергей Бадло**

by **raxp** <http://raxp.radioliga.com>

**SSD в форм-факторе** модуля DDR3 DIMM разработала Viking Modular. Новинка предназначена для применения в корпоративном секторе в составе серверов, систем хранения данных облачных компьютерных системах.

Твердотельный накопитель SATADIMM содержит чипы флэш-памяти, которые установлены на печатную плату формата DDR3 240-pin DIMM. При этом через контакты слота памяти устройство получает питание, напряжение которого составляет 1.5 В. Для передачи данных системе используется стандартный интерфейс SATA2. Однако в будущих версиях накопителей планируется реализовать возможность передачи данных через слот памяти, чтобы исключить необходимость использования кабеля.



Дополнительно отмечается наличие на печатной плате супер конденсатора, который обеспечивает сохранность данных в случае неожиданного отключения энергии. Новинка может иметь емкость 50, 100 или 200 ГБ. При этом ее скорость чтения и записи заявлена на уровне 260 МБ/с, а количество операций ввода-вывода в секунду составляет 30000.



**Самовосстанавливающиеся солнечные элементы** создали ученые из Массачусетского технологического.

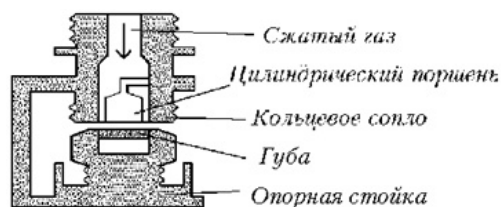
Основной недостаток солнечных панелей – это ограниченный срок службы. Например, панели из поликристаллического кремния имеют срок службы до 10 лет, а из монокристаллического до 25 лет. Плюс к этому их хрупкость и дороговизна. Конечно, ученые и просто инженеры пытаются создать панели, которые будут устойчивы к повреждениям, но до сих пор эти попытки не приносили особого успеха. Величина новых солнечных элементов невелика, не более нескольких нанометров. Однако при таких размерах эти элементы могут поддерживать сами свою работу, восстанавливаясь в случае повреждений. При этом выработка солнечной энергии идет примерно на одном и том же уровне. Панели из наноструктуры способны находить необходимые для себя элементы, используя для этого протеины, углеродные трубки и некоторые другие материалы. В результате получаются устройства с весьма продолжительным сроком эксплуатации.

Кроме того, в состав такого элемента входят ПАВ (поверхностно-активные вещества), которые разбивают некоторые типы соединений. Конечно, пройдет немало времени, пока



появятся самовосстанавливающиеся солнечные батареи, но ведь прогресс идет, правильно? Возможно, лет через 5-10 мы и получим желанные солнечные панели, которые могут служить очень долго.

**Немногие знают**, что первый ультразвуковой свисток сделал в 1883 году англичанин Ф.Гальтон. При пропускании под высоким



давлением воздуха через маленькую цилиндричес-

кую резонансную полость в результате удара цилиндрического поршня о губу (металлическую пластинку) в зазоре генерируется ультразвук частотой около 170 кГц (определяется размерами кольцевого сопла и губы). Мощность свистка Гальтона невелика, его в основном применяют для подачи команд при дрессировке собак.

**Компания Verbatim** представила накопитель MediaShare, несколько отличающийся от традиционных устройств. Этот аппарат подключается к домашней сети через Ethernet-соединение и выступает в качестве центрального хранилища для разнообразного контента. Интересная особенность – возможность доступа к хранимым на нем файлам с любого ПК, подключенного к Интернету.



С помощью MediaShare просмотреть фотографии, воспроизвести музыку, скачать или выложить файлы становится

очень легко. Это можно сделать через удаленный доступ с любого компьютера, подключенного к сети Интернет. Легко можно организовать потоковое вещание музыки для пользователей iTunes на любом подсоединенном к домашней сети ПК или Mac компьютере. Также MediaShare поддерживает показ фотографий и воспроизведение видео на телевизоре с

поддержкой DLNA, через DLNA-совместимый медиаплеер, Playstation 3 или Xbox360.



**Ученые объединили клетки мозга** и кремниевый чип. Это удалось сделать специалистам с факультета медицины Университета Калгари, работавшим в сотрудничестве с Национальным исследовательским советом Канады. Фактически им удалось создать нейрочип. В отличие от предыдущих технологических решений для исследования активности мозга, новая разработка позволяет осуществлять взаимодействие с множеством клеток мозга одновременно. Ранее ученым была доступна возможность отслеживать работу только одной или двух клеток.

**Хотите подключить USB-флешку** или устройство с COM-портом или управлять своим роботом с КПК по радиоканалу (блютуз)? Нет ничего проще. Среди прочих устройств Plug&Play стоит выделить модули BlueGIGA WT-11/12. Никакой дополнительной обвязки не требуется, достаточно подать на них питание и вы в работе. На сегодня доступны позиции по цене от 20\$ при единичном заказе [http://catalog.compel.ru/bluetooth/info/WT12AAI3%20\(BLUEGIGA\)](http://catalog.compel.ru/bluetooth/info/WT12AAI3%20(BLUEGIGA))

WT12\* – малогабаритный встраиваемый Bluetooth модуль 2-го класса, поддерживающий спецификацию Bluetooth v2.0, что позволяет передавать данные со скоростью более 2 Мбит/с. Модуль WT11 имеет предустановленное программное обеспечение iWRAP™ (разработка компании Bluegiga), отвечающее за управление работой модуля и связь с внешними устройствами. Внешнее управление основано на наборе простых команд, по аналогии с AT-командами для модемов. Поддерживается функция AFH (adaptive frequency hopping), позволяющая Bluetooth-устройствам работать в



зоне действия Wi-Fi сети. WT12 характеризуется низким энерго-потреблением и возможностью работы в промышленном диапазоне температур.

Основные характеристики:

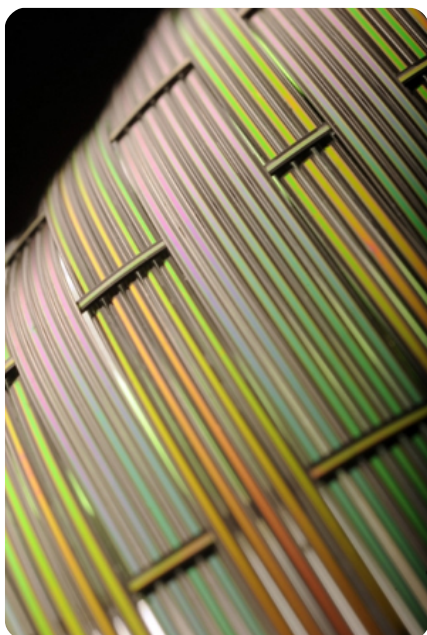
- Bluetooth® 2-го класса
- Встроенная ЧИП-антенна
- Улучшенная до 2-3 Мбит/сек скорость передачи данных (EDR)
- Поддержка адаптивной перестройки частоты (AFH) для совместимости с 802.11
- Поддержка UART и USB версии 2.0
- Встроенная Flash-память 8 МБайт
- Поддержка Bluetooth-профилей: SPP, HFP, HFP-AG, OBEX OPP and DUN, + HCI
- PCM-интерфейс для голосовых приложений
- Возможность встраиваемых приложений пользователя
- Расширенный температурный диапазон от -40C до +85C
- Совместим по выводам с модулем WT11
- Встроенное ПО iWRAPTМ firmware
- Сертифицирован на соответствие Bluetooth® 2.0+EDR, CE, FCC and IC

\* Комментарий редакции.

Документация на модули включена в архив седьмого номера.

**Пьезоэффект в волокнах** позволяет воспринимать и воспроизводить звук. Ученые Массачусетского технологического на протяжении последних лет занимаются разработкой волокон с необычными свойствами.

Из таких волокон можно создавать ткани с необычными свойствами, работающие как чувствительный микрофон, который можно использовать как для звукозаписи, так и для мониторинга параметров функционирования человеческого организма



(давления крови в капиллярах, внутричерепного давления и прочее).

Необычные свойства волокна обусловлены особым способом его производства – здесь используются несколько материалов, скомпонованных специальными образом. Основой является пластик, обычно используемый в микрофонах, при этом атомы фтора и водорода в новом волокне располагаются в специальном порядке, который не меняется даже в процессе нагревания и охлаждения. Вследствие асимметрии молекул пластик обладает пьезоэлектрическим свойством, т.е. изменяет форму при приложении электрического поля. В обычных микрофонах электрическое поле генерируется металлическими электродами, а в новом волокне используется проводящий пластик, содержащий графит.

Оказалось, что при подключении полученного в лаборатории материала к источнику переменного тока, он начинает вибрировать. Опыты по изучению вибрации проводились в водной среде. Если подобрать частоту в звуковом спектре и поднести материал к уху, новый материал можно услышать.

**Компания Toshiba заявила** о существенном технологическом прорыве в производстве жестких дисков. Вместо используемой сейчас технологии, где магнитный материал равномерно распределен по поверхности пластины, инженерам удалось создать пластины с покрытием, которое заранее разделено на отдельные магнитные зоны – каждая такая зона хранит ровно один бит информации. Toshiba развивает технологию получения структурированных магнитных сред, частицы в которых выстроены в битовый массив (Bit Patterned Media, BPM). Суть методики сводится к тому, что каждый бит информации хранится на одном магнитном «островке», не превышающем 10 нанометров, где каждый бит заранее отделен от других, а не на десятках зерен. Это позволит многократно повысить плотность записи.



**Действующий прототип** первого в мире оптического канала передачи данных с интегрированными лазерами представила компания Intel. Разработка позволит существенно сократить затраты на оптическую связь - технология позволяет снизить стоимость одного порта до доллара, тогда как существующие решения на базе дискретных лазеров стоят сотни долларов за порт.

Система на базе полупроводниковых компонентов с использованием лазера, продемонстрированная специалистами Intel, состоит из двух микросхем, в которые интегрированы все компоненты, необходимые для передачи данных по оптическим каналам со скоростью до 50 гигабит в секунду, что эквивалентно передаче всего содержимого двухслойного Blu-ray диска за одну секунду.

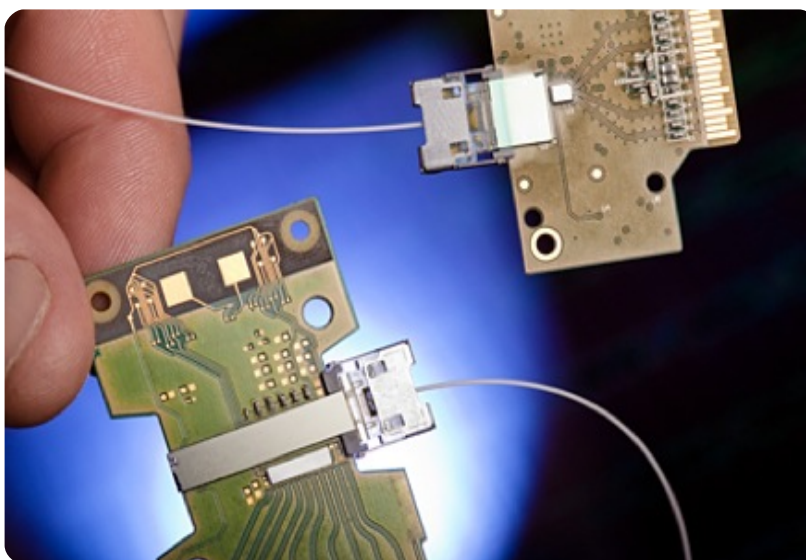
В основе решения лежат кремниевые чипы, передатчик и приемник. Чип-передатчик использует четыре гибридных лазера (discrete VCSEL - vertical cavity surface-emitting laser) с разной длиной волны, передающих данные в виде отдельных потоков со скоростью до 12.5 Гбит/с (благодаря фильтрации суммарная скорость и составляет 50 Гбит/с). Лазерные импульсы собираются в единый световой поток в модуляторе, который преобразует электрические сигналы в световые импульсы, кодирует и передает их по оптоволоконному кабелю. На другом конце соединения приемник разделяет лучи и направляет их в кремниевые-германиевые фотодетекторы, преобразующие данные в электрические сигналы.



**Компания AMD** в официальном блоге <http://blogs.amd.com/developer/2010/08/18/3dnow-deprecated> объявила, что будущие процессоры этой компании не будут иметь поддержки инструкций 3DNow!

**К запуску Google Games** готовится Google, вложившая до \$200 млн в разработчика игр для социальных сетей Zynga. Интернет-гигант и Zynga планируют заключить соглашение о стратегическом партнерстве, в рамках которого Zynga предоставит свои игры для нового сервиса Google Games.

**Ralink RT3592BC8** **одновременно** поддерживает 2x2 802.11n и Bluetooth 3.0+HS.



Это пока единственный в мире чипсет, способный одновременно поддерживать передачу со скоростью 300 Мбит/с по Wi-Fi 802.11n и работу интерфейса Bluetooth 3.0+HS. Новинка стала основой решения, оформленного в виде модуля

половинного формата MiniCard. Модуль предназначен для ПК, ноутбуков, нетбуков и других мобильных устройств. Он является развитием представленного осенью прошлого года модуля RT3090BC4, который показан на снимке.

К достоинствам RT3592BC8 компания относит поддержку конфигурации 2x2 MIMO и двух диапазонов Wi-Fi - 2.4 и 5 ГГц. Кроме того, сильной стороной новинки является эффективное обеспечение «мирного сосуществования» Wi-Fi и Bluetooth в одном





диапазоне. Для снижения взаимного влияния выполняется динамическая регулировка параметров передачи. В результате, превосходство в пропускной способности подключения к беспроводной сети по сравнению с другими решениями, объединяющими WLAN и Bluetooth, достигает 80%.

**Немногие знают, что в системах гелиоцентрирования,** авиации и морском деле давно используется прибор – поляризатор (небесный компас). Пробразом которому стал прозрачный минерал – исландский шпат\*\*.

Такой поляризационный компас применяют в дневное время в случаях, когда невозможно прямое наблюдение Солнца при не сплошной облачности или при положении Солнца за горизонтом (незадолго перед восходом или после заката). Поскольку распределение поляризации по небосводу однозначно для данного времени дня (изменяется от 0 до 80%), то, находя направление преимущественных колебаний в некоторой точке (например, в зените), можно определить истинное направление на север. Точность прибора порядка двух градусов.

На свойствах этого камня основано действие небесного сумеречного компаса Колльсмана, который до недавнего времени использовался на военных и гражданских самолетах, совершающих полеты в полярных широтах. При полетах около Северного

полюса возникают неудобства в пользовании магнитными компасами, и тогда в современных устройствах навигации применяют Небесный компас, содержащий исландский шпат как основной элемент системы.



**Продолжается Всеукраинский чемпионат компьютерных талантов «Золотой Байт - 2010»**, стартовавший 1 сентября в Украине. Чемпионат организован Компьютерной Академией «ШАГ» и

ориентирован на молодое поколение. К участию приглашается молодежь от 14 до 35 лет. Участие в чемпионате – бесплатное! Чемпионат предусматривает как одиночное, так и командное выступление.

Региональные финалы пройдут в 13 областных центрах и городах Украины: Киев, Запорожье, Донецк, Одесса, Днепропетровск, Харьков, Львов, Луганск, Винница, Николаев, Полтава, Ровно, Мариуполь.

Финал будет проходить в течение двух дней в период с 1 по 14 ноября 2010 г. в городе Львов.

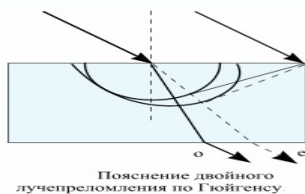
Все подробности, а также порядок участия смотрите на официальном портале чемпионата [www.goldenbyte.org](http://www.goldenbyte.org).

Телефон организатора: +38 (062) 345-24-74.

#### \*\* Комментарий редакции.

Исландский шпат – минерал, бесцветная и прозрачная разновидность кальцита ( $\text{CaCO}_3$ ), которая вызывает «раздвоение» изображений вследствие высокого двулучепреломления.

Кристаллы исландского шпата имеют форму ромбоэдра. Если положить кристалл исландского шпата на границу с напечатанным текстом, то текст представляется раздвоенным. При вращении кристалла над строчкой происходит вращение одного изображения вокруг другого.



Примечательна история применения столь замечательного материала. Сканируя небосвод с помощью солнечного камня, древние викинги могли определять направление, где интенсивность света оставалась постоянным при вращении образца, определяя этим даже через облака положение Солнца. Пользуясь таким примитивным, но эффективным оптическим устройством для навигации, викинги могли достигать берегов Америки и других северных земель. В другой саге упоминается о модификации этого навигационного устройства, выполненного в виде стола.

Если образец исландского шпата обращен к небесному своду в направлении, где нет Солнца, то к наблюдателю будет приходить свет, отраженный от атмосферных неоднородностей и обязательно, хотя бы частично поляризованный. Тогда, при повороте исландского шпата, осуществляющего поляризацию света для обыкновенного и необыкновенного лучей, относительно оси наблюдения, интенсивность проходящего света будет изменяться.

При совпадении плоскости поляризации приходящего света и образца исландского шпата – интенсивность проходящего света будет максимальной. При повороте образца исландского шпата относительно вертикальной оси – интенсивность проходящего света будет уменьшаться.

Если же образец исландского шпата направлен на Солнце, то его почти неполяризованные лучи, при любом положении образца относительно оси наблюдения, никогда не будут изменять интенсивности проходящего света.

В данной статье будет рассмотрено самостоятельное построение лексического анализатора (далее ЛА). В нем нуждаются как начинающие программисты (желая получить больше свободы и писать так, как вздумается), так и опытные (с целью поддержки собственных API, создания скриптов)...



by **Utkin** [utkin295@yandex.ru](mailto:utkin295@yandex.ru)

### Зачем это надо

Лексический анализатор – самостоятельная программа (или ее фрагмент), реализующая функцию лексического анализа. Лексический анализ – это разложение и преобразование входящего текста на определенные последовательности символов (не обязательно текстового представления), именуемые токенами. Каждый токен – это последовательность символов (не обязательно текстовых), однозначно характеризующих лексему. Лексема – это последовательность символов и слов, однозначно определяющих какую-либо составляющую языка.

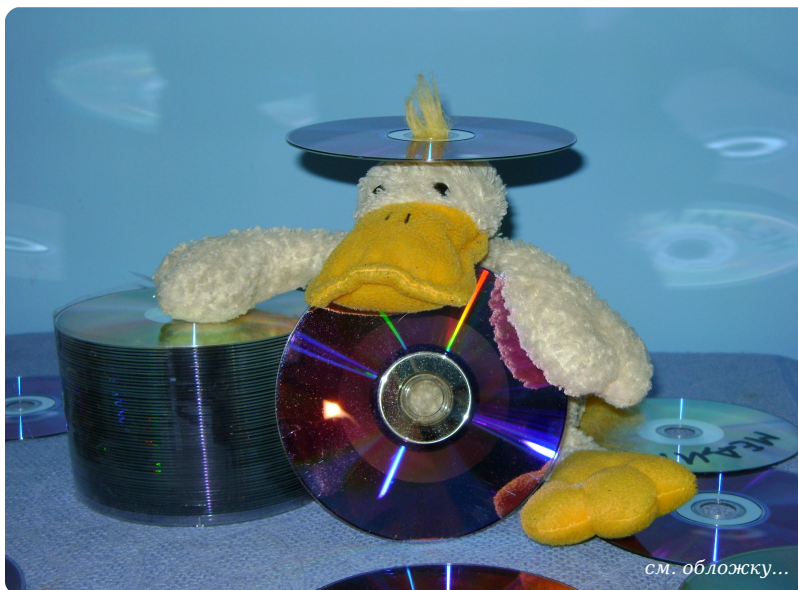
Немного проясним ситуацию. Представим, что имеется определенный язык (нам ближе язык программирования), хотя его назначение на данном этапе роли не играет. Пусть Паскаль. Вот фрагмент записи: `For i:=0 to 100 do`

Этот фрагмент программы ответственный за организацию цикла. Транслятору важно знать, когда кончается слово `for` и начинается переменная `i`. Таким образом, он дробит указанную строчку на лексемы: `for`, `i`, `to`, `:=`, `0`, `100`. Они определяют ключевые слова, имена переменных, числа и т.д. Однако внутри программы хранить данные в виде записи неудобно и медленно. Примем обозначения: для ключевых слов будем использовать букву `k` с числовым индексом, однозначно определяющим ключевое слово. Для имен переменных – букву `n`,

для знаков операций – букву `o`, для чисел же – букву `x`.

Теперь мы имеем следующую последовательность: `k1, n1, o1, x1, k2, x2, k3`. Это и есть таинственные токены. Иногда, особо дотошные и ответственные трансляторы кодируют и разделители между лексемами, но, на мой взгляд, такая информация является избыточной. А уже оперируя токенами, можно проверить порядок их следования и осуществлять поиск логических ошибок. Иными словами, лексический анализатор – это средство, с помощью которого осуществляется разбор операторов языка программирования (в нашем случае).

### Как это было сделано



см. обложку...

Стандартная методика работы следующая: берется сплошной поток символов, разбивается на токены, анализируется, и (в зависимости от типа транслятора) выполняются последующие действия – компиляция либо исполнение. Но, как это обычно бывает, современные тех-

нологии ушли далеко вперед, а методы преподносятся те же самые. Более того, сухое изложение теории данного вопроса совершенно не способствует появлению интереса в глазах студентов. Здесь же мы рассмотрим лексический анализатор, полученный немного другим способом.

Сначала выработаем некое множество абстрактных языков, с которыми будет работать наш лексический анализатор. Это важно, так

как, в конечном счете, все лексические анализаторы имеют ограничения и способны работать только под контролем жестких запретов, наложенных многочисленными правилами (грамматикой) языка. Итак, наш язык программирования, в целях упрощения изложения, предполагает построение по строковому принципу – иными словами, одна строка может содержать только одну команду.

Действительно, работать с потоком символов неудобно (но быстрее), в то же время имеется большое количество классов, ориентированных на работу с наборами строк. Кроме того, наличие нескольких команд в одной строке было вызвано историческими причинами: ранние редакторы были экраноориентированными, и большое количество строк являлось непозволительной роскошью. В то же время, мы не стеснены обязательствами совместимости с предыдущими версиями нашего языка. Далее введем такой термин, как конструкция. **Конструкция** – это символическое представление команды языка программирования. Казалось бы, зачем такое усложнение концепций? Но мы ведь не собираемся отставать от современных технологий и будем равняться на .Net (конечно, условно). Здесь для нас важнейшей чертой .Net является многообразие языков программирования, выполняющих по сути одну и ту же работу (возможно, с разных точек зрения, – сейчас это не принципиально). Хотя C#, хоть Дельфи с приставкой .Net – все они различны внешне, но построены на одной платформе, и это чувствуется, несмотря на различный синтаксис. Именно с целью легкой смены синтаксиса и вводится термин «конструкция». Итак, общий вид конструкции языка программирования у нас следующий:

```
type
  TKonstr = record
    Slovo: array [1..8] of String; // Слова конструкции
    Id: Integer;                  // Идентификатор конструкции
    Remark: String;               // Описание конструкции
  end;
```

Таким образом, в конструкции может быть не более восьми лексем. Здесь следует оговориться: разделители лексемами не являются (в других языках это может быть не так). Практика

показывает, что восьми лексем хватает для кодирования подавляющего большинства команд языка. Больше количество лексем избыточно и ведет к затруднению восприятия (но, естественно, после усвоения принципов построения лексических анализаторов каждый может подобрать оптимальное количество лексем самостоятельно) и уменьшает скорость работы анализатора. Наши лексемы представлены либо частью команды, либо командой, либо выражениями нескольких видов (последнее не оговаривается: разбор выражений – это другая большая тема). Так что все упоминания переменных и чисел относятся к выражениям. Сама команда может рассматриваться как комбинация служебных слов (в некоторых источниках литературы их называют ключевыми) и параметров. Тот же пример с циклом в новом свете выглядит так:

```
for параметр1 to параметр2 do;
```

И к тому же содержит пять лексем. То есть нужно учитывать тот факт, что команда не обязательно занимает все восемь лексем, а может и меньше (но не больше, иначе наш лексический анализатор будет непригоден). Теперь идентификатор, если хотите, можно с натяжкой назвать токеном <http://ru.wikipedia.org/wiki/Токен>, но токеном глобальным.

Он отвечает не за конкретную лексему, а за всю команду. Это вполне естественно: если в строке может быть только одна команда, то ее можно условно обозначить определенным числом. Нет смысла на каждый кусок конструкции заводить свое число. Достаточно помнить саму команду (в виде числа) и ее параметры. Теперь поле **Remark** – это просто описание команды, которое можно, например, выводить во всплывающей подсказке при наведении на команду (что полезно начинающим), в скриптах или в языках программирования, ориентированных на предметные области (математика, физика, бухгалтерия и т.д.). Это описание одной конструкции языка. Соответственно, чтобы описать весь язык, нужен массив таких вот записей. Это несложно, но есть нюансы: в описание команды нужно запихнуть параметры, а также учесть строковые константы (если они





условие – 2, математическое выражение – 3 и т. д. Соответственно, нужно решить, при каком идентификаторе, какой параметр, на каком месте находится. И все это независимо от обозначения самой конструкции во входящем тексте программы.

Промежуточные итоги вышесказанного:

- мы можем создавать различные конструкции, выполняющие одни и те же функции;
- мы можем менять порядок следования параметров в некоторых пределах без перестройки транслятора;
- поскольку описание всех конструкций – есть записи, состоящие из простых полей, мы можем их сохранять и считывать из внешних источников – файлов, получать по сети, генерировать во время работы и т.д., и все это прямо время работы транслятора;
- имея четкое и простое внутреннее представление, мы можем подготавливать различные отчеты по своему языку, например, автоматически генерировать форму Бэкуса-Наура (форма описания синтаксиса языка программирования);
- при определенной сноровке мы можем описывать синтаксис, выражение которого обычными средствами трудно, либо невозможно.

Однако на этом полезности нашего лексического анализатора не заканчиваются. Многие языки программирования вводят так называемый синтаксический сахар – это различные представления одного и того же механизма. Эффекта от них немного, но это просто удобно для восприятия человеком. В частности, наши двойные конструкции – на латинице и на кириллице – уже сам по себе синтаксический сахар, поскольку новых возможностей, позволяющих эффективнее решать задачи, программист не получил. Но представим, что после жарких дебатов было решено ввести еще одну конструкцию цикла, так как это ближе к аналогичным реализациям у конкурентов.

Например, русскоязычным вариантом Паскаля является язык программирования Глагол, наименование конструкций в котором по большей части является прямым переводом слов

с английского языка. Старые же конструкции, в силу совместимости с прошлыми версиями, мы выкинуть не можем. Кроме того, не всем понравилась конструкция с переменной в конце команды, а не в начале, как это заложено в традиционном Паскале.

Чтобы отвечать всем требованиям, введем еще пару конструкций:

```
Цикл параметр1 до параметр2
```

```
Для параметр1 до параметр2
```

На самом деле, чрезмерное обилие конструкций, выполняющих одно и то же действие, может приводить к путанице, но только в случае их безграмотного обозначения (что у нас и наблюдается). Но для нашего учебного примера подойдет и такой образец того, «как делать не надо».

Теперь взглянем на это с другой стороны – с позиций лексического анализатора. Каждая конструкция есть соответствующая строчка в массиве конструкций, которые перебираются до тех пор, пока не будет найдено совпадение строки с образцом. Процесс разбора строки есть занятие трудоемкое, и производится он каждый раз заново, пока не будет найден нужный элемент либо пока весь массив не будет просмотрен до конца (в таком случае возвращается специальный идентификатор – неизвестная конструкция). Изменить же такой порядок (в смысле разложения строки на составляющие) не представляется возможным: требуется кардинальная перестройка алгоритма, и не факт, что новый вариант будет производительней (а для нас немалую значимость представляет и простота внутреннего устройства).

Поэтому возникла идея объединять похожие конструкции. Взгляните на последние две: они отличаются только одним словом – почему бы их не рассматривать одновременно, как одну строку?

Для этого в лексическом анализаторе есть соглашение: если в лексеме есть слово, содержащее пробел, то это два слова (соответственно, сколько пробелов, столько слов

плюс еще одно):

```
TKonstr.Slovo[1]:='Цикл Для';
TKonstr.Slovo[2]:='1';
TKonstr.Slovo[3]:='до';
TKonstr.Slovo[4]:='2';
TKonstr.Slovo[5]:='';
TKonstr.Slovo[6]:='';
TKonstr.Slovo[7]:='';
TKonstr.Slovo[7]:='';
TKonstr.Id:=1;
TKonstr.Remark:='Цикл с приращением переменной';
```

Сам пробел все равно в конструкциях участвовать права не имел, так как является разделителем между лексемами. При разборе строки конструкция будет засчитана, если первое слово Цикл или Для, и далее осуществляется соответствие остальным лексемам конструкции. То есть строка будет разобрана в цикле на один раз меньше, что в целом ускоряет работу лексического анализатора (альтернативное направление – правильное расположение конструкций в массиве, чтобы часто используемые конструкции находились в начале массива).

Это совсем не излишество еще и по другой причине. Наш лексический анализатор заточен не только на классические языки программирования, но и на лексический анализ в целом. Это дает ему и другие сферы применения, например, использование в качестве составляющей бота, как для сканирования страниц на предмет наполнения определенным контентом (не по ключевым словам, которые могут завлекать случайных прохожих на сайты, содержащие совсем другую тематику), так и в роли робота, общающегося в чате с посетителями, не могущими отличить настоящего человека от программы.

Кроме того, можно из пушки стрелять по воробьям, а именно – заниматься парсингом\* различных табличных отчетов, логов и другим.

#### \* Комментарий автора.

При желании можно написать универсальный парсер, способный с помощью данного анализатора разжевывать логи с нескольких программ.

## Особенности реализации

Лексический анализатор представлен в виде класса, что дает программам, его использующим, большую гибкость:

```
// Это реализация лексического анализатора
type
  TLA=class
  protected
    Nabor: Array of TKonstr; // Набор конструкций
    Constr: TConstrParam; // Врем-е место для хран-я парам-в

    // Получение фрагмента строки до первого символа табуляции
    function GetTab (Stroka, Symb: String): String;
    // Получение строки без первого вход-я символа табуляции
    function EraseTab(Stroka, symb: String): String;
    // Проверка совпадения строки с шаблоном конструкции
    function CompareStrId(Stroka: String; Index: Integer):
        Boolean;
    // Проверка: входит-ли слово в указ-е множество слов,
    // разделенных пробелами
    function CheckInSet(Template, Value: String): Boolean;
    // Проверка: является ли строка числом
    function CheckNumberStroka(Stroka: String): Boolean;
    // Получение числа слов в строке
    function GetIndex(Stroka1, razdelitel: String):Integer;
    // Отбрасывает из строки элем-ы, пока не встр-я искомый
    function GetEnd (Stroka, Element: String): String;
    // Откидывает элементы, заключенные в указанный символ
    function DelAp (Stroka, Symb: String): String;
    // Удаление элемента строкового массива по индексу
    function DelItemMas(Stroka1, razdelitel: String;
        Index: Integer): String;
    // Возвращает часть строки, наход-ся после указ.подстроки
    Function After (Src: String; S: String ): String;
    // Получение элемента строкового массива по индексу
    function GetItemMas(Stroka1, razdelitel: String;
        Index: Integer): String;

  private
  public
    constructor Create;
    procedure Init(); // Инициализация класса
    destructor Destroy; override;
    // Добавление новой конструкции (в виде строки)
    procedure AddConstr(Constr: String);
```



```
// Получение идентификатора и параметров конструкции
procedure GetID(Stroka: String; var Id: Integer;
               var Param: TConstrParam);

// Чтение набора конструкций из списка
procedure ReadNabor (var Spisok: TStringList);

// Запись набора конструкций в список
procedure GetNabor (var Spisok: TStringList);

// Чтение конструкции в виде строки
function GetConstr(Index: Integer): String;

// Определение числа конструкций в наборе
function GetCount(): Integer;

end;
```

Пользование готовым лексическим анализатором никаких хлопот не доставляет, тем более что он хорошо документирован в виде комментариев к коду. Добавление конструкций в их набор (по сути, описание языка) осуществляется в виде строки, где каждое поле отделено символом табуляции. Сделано это также для простоты и для последующей возможности сброса конструкций в текстовый файл CSV формата (поля разделяются определенным символом). Сам класс анализатора сохранять и читать в файл не умеет (для придания универсальности), но умеет это делать в список строк `TStringList`, где список конструкций можно обработать либо просто сохранить напрямую. Читать из набора конструкции можно также в виде строк. Самая главная и ответственная функция здесь `GetID`, которая в цикле сравнивает строку с конструкциями посредством самой хитрой и интересной функции `CompareStrId`. Последняя совсем не проста, она разлагает строку с учетом многих параметров: количество лексем в конструкции, порядок следования параметров, наличие в строке строковых констант и т.д. Также в секции `protected` содержится ряд довольно-таки мощных вспомогательных функций по обработке строк, которые обязательно понадобятся для обработки строк.

### Теперь касательно лексем

В лексемах запрещены символы табуляции (поскольку вы не сможете правильно сохранить и прочесть данную конструкцию в последующем), символы пробела (они являются разделителями служебных слов внутри лексем), любые

самостоятельные целые числа (так как они обозначают местоположение параметров, но разрешены, например, шестнадцатеричные) и символ апострофа, как признак текстовой константы (в данный момент это жестко прописано в лексическом анализаторе, но можно исправить в функции `GetEnd`), да и перевод строки тоже не желателен, хотя и возможен при определенных манипуляциях со `TStringList`. Зато все остальное, включая непечатные символы, вполне допустимо. Это позволяет использовать такие конструкции, использование которых при традиционных лексических анализаторах невозможно: многократное использование точечной нотации (как для определения структур данных, так и для определения конструкций), скобочной нотации и т.д. Возможно это потому, что лексический анализатор не выделяет токены лексем, в частности, разделителей (скобки, точки, запятые и прочие спецзна-ки). При правильной расстановке конструкций в наборе допускается использование и арифметических знаков в составе лексем конструкций. Все это ведет к более широкому использованию контекстно зависимых грамматик (для справки: на данный момент большинство языков программирования подчиняется усеченному множеству контекстно зависимых грамматик – контекстно свободным грамматикам), что позволяет писать программы на языках, более приближенных к естественным (это приводит к уменьшению числа логических ошибок в программах). В целях упрощения работы с лексическим анализатором лексемы (но не параметры) сравниваются со строкой без учета регистра (кому интересно, может это исправить самостоятельно).

### Заключение

Данный пример, хоть и представляет вполне рабочий лексический анализатор, все же является учебным. Однако на нем вполне можно построить, например, интерпретатор. Читателям, в качестве самостоятельных занятий, рекомендуется для лучшего закрепления знаний о лексическом анализаторе написать интерпретатор без посторонней помощи. Дам единственную подсказку: следует серьезно отнестись к разбору математических выражений либо использовать уже имеющиеся (либо



адаптировать иные компоненты, например TParser). Что касается простых скриптов, то лексический анализатор уже сейчас можно использовать для этих целей. Например:

Ширина окна	параметр1	Идентификатор=1
Высота окна	параметр1	Идентификатор=2
Заголовок окна	параметр1	Идентификатор=3

И так далее. Сам скрипт мог бы выглядеть так:

```
Ширина окна 100
Высота окна 100
Заголовок Ну, погоди!
```

Согласитесь, такой скрипт использовать приятней, чем, скажем, .DFM файл или ему аналогичный. Обработку можно осуществлять по идентификатору через **case of**. А на вход лексического анализатора желательно также подавать обработанные строки, так как, например, строка  $x=y+1$  будет неопределенной, в то время как  $x = y+1$  легко определяется.

Все упомянутые в статье исходные коды и тестовый проект приведены в виде ресурсов в теме «Журнал клуба программистов. Седьмой выпуск» или непосредственно в архиве с журналом.

## Ресурсы

- Лексический анализ [http://ru.wikipedia.org/wiki/Лексический\\_анализ](http://ru.wikipedia.org/wiki/Лексический_анализ)
- Правила языка <http://ru.wikipedia.org/wiki/Грамматика>
- БНФ [http://ru.wikipedia.org/wiki/Форма\\_Бэкуса\\_-\\_Наура](http://ru.wikipedia.org/wiki/Форма_Бэкуса_-_Наура)
- Синтаксический сахар [http://ru.wikipedia.org/wiki/Синтаксический\\_сахар](http://ru.wikipedia.org/wiki/Синтаксический_сахар)
- Парсер <http://ru.wikipedia.org/wiki/Парсер>
- Контекстно свободная грамматика [http://ru.wikipedia.org/wiki/Контекстно\\_свободная\\_грамматика](http://ru.wikipedia.org/wiki/Контекстно_свободная_грамматика)
- Контекстно зависимая грамматика [http://ru.wikipedia.org/wiki/Контекстно\\_зависимая\\_грамматика](http://ru.wikipedia.org/wiki/Контекстно_зависимая_грамматика)

В наше время повсеместно используются компьютеры. Современный человек просто не представляет свою жизнь без компьютера. На сегодняшний день программные и технические средства ЭВМ развиваются очень быстро. Увеличивается скорость выполнения операций. В связи с этим многие пользователи и начинающие программисты считают, что с помощью современного компьютера можно «в лоб» решить любые задачи и, к сожалению, не задумываются, что при решении многих задач можно сэкономить как компьютерные, так и временные ресурсы. Конечно, при написании простых бытовых программ эти вопросы не актуальны. Но, когда речь заходит о таких серьезных отраслях как: криптография, архитектура, дизайнерские и графические приложения, то эти вопросы становятся очень актуальными и важными. Ведь при реализации этих приложений нужно проводить сложные математические вычисления. Важно, чтобы эти вычисления проходили быстро, так как таких вычислений нужно проводить очень много за малый промежуток времени. А в связи с этим важно, чтобы эти вычисления были реализованы оптимальным и экономичным способом.



**Сергей Апанасевич** [mjdel@tut.by](mailto:mjdel@tut.by)  
преподаватель информатики и математики

Эта статья и посвящена этим важным вопросам. В статье сразу рассказывается как на предварительном этапе, во время теоретической разработки ПО, можно подобрать оптимальное решение задачи и теоретически подсчитать экономию\* компьютерного ресурса. Затем рассказывается, как во время написания кода программы можно выбрать или разработать более быстродействующую конструкцию.

**Профилирование** – сбор характеристик работы программы, таких как: время выполнения отдельных фрагментов (обычно подпрограмм), число верно предсказанных условных переходов, число кэш промахов и т.д. Характеристики могут быть аппаратными (время) или вызванные программным обеспечением (функциональный запрос). Инструментальные средства анализа программы чрезвычайно важны для того, чтобы понять поведение программы. Проектировщики ПО нуждаются в таких инструментальных средствах, чтобы оценить, как хорошо выполнена работа. Программисты нуждаются в инструментальных средствах, чтобы проанализировать их программы и идентифицировать критические участки программы.

**\* Комментарий редакции.**

Для отслеживания «узких» мест в программе еще используются специальные утилиты – профайлеры <http://ru.wikipedia.org/wiki/Профайлер>. Например, ProDelphi.

Это часто используется, чтобы определить, как долго выполняются определенные части программы, как часто они выполняются, или генерировать граф вызовов (Call Graph). Обычно эта информация используется, чтобы идентифицировать те участки программы, которые работают больше всего. Эти трудоемкие участки могут быть оптимизированы, чтобы выполняться быстрее. Это – также общая методика для отладки.

### Алгоритм Евклида

Рассмотрим алгоритм Евклида. Он был впервые изложен в одном из томов 13-ти томного сочинения Евклида (3 век до н.э.), известного под названием «Начала Евклида». Алгоритм нахождения наибольшего общего делителя двух натуральных чисел ( $\text{НОД}(a,b)$ , где  $0 < b \leq a$ ) базируется на теореме (1):

$$\text{НОД}(a,b) = \text{НОД}(b,r); \quad (1)$$

где  $r$  – остаток от деления  $a$  на  $b$ . Известны ученые, такие как: Ламе, Гашков, Абрамов, Кормэн и другие, проводившие исследования, благодаря которым вывели формулы для предварительного расчета приблизительного количества шагов, за которое можно получить конечный результат. Рассмотрим эти формулы.

Французский математик Ж. Ламе в 1844 году доказал теорему о том, что число делений с остатком, необходимых для нахождения  $\text{НОД}$  двух натуральных чисел  $a$  и  $b$ , таких, что  $0 < b \leq a$ , не превышает пятикратного числа



десятичных знаков наименьшего из этих двух чисел, т. е.  $m \leq 5p$ , где  $p$  – число разрядов числа  $b$ . Гашков предложил следующую формулу (2):

$$m \leq [\log_{\varphi}(\max(a, b) + 1/2)] - 1; \quad (2)$$

где  $\varphi = (-1)/2$ , а  $m$  – искомое число. У Абрамова получилась такая формула (3):

$$m \leq 2[\log_2 b] + 2; \quad (3)$$

Кормен подошел к этой проблеме с другой стороны. Пусть  $m$  – целое положительное число. Если  $0 < b \leq a$  и  $b \leq Fm + 1$ , то результат будет достигнут не более чем за  $m$  шагов, где  $F$  – число Фибоначчи,  $m + 1$  – порядковый номер числа Фибоначчи.

Сейчас, для сравнения результатов, посмотрим рисунок 1 и 2. На рисунках изображена таблица, в которой сравнивается быстрота нахождения НОД-а двух чисел, при этом, в столбце «Реально», указано за сколько шагов находится НОД, а в столбцах «Ламэ», «Гашков», «Абрамов» и «Кормен» указаны результаты подсчета шагов для нахождения НОД-а двух чисел по соответствующим формулам:

Числа	Реально	Ламэ	Гашков	Абрамов	Кормен
НОД(31, 3)	2	5	7	4	4
НОД(20, 15)	2	10	6	8	7
НОД(20, 16)	2	10	6	10	7
НОД(315, 15)	1	10	12	8	7
НОД(27, 5)	3	5	7	6	5
НОД(3175, 273)	10	15	17	18	13
НОД(57, 33)	5	10	9	12	8
НОД(45, 18)	2	10	8	10	7
НОД(420, 66)	4	10	13	14	10

Рис. 1. Реальное нахождение НОД-а двух чисел и теоретический расчет по формулам

На рисунке 2 отображены в виде графика значения из рисунка 1. Как показывает график, лучше всего работает формула Кормена. Конечно, для малых чисел эти формулы большой роли не играют, но, когда речь касается больших чисел, без них очень трудно обойтись. Это очень наглядно в криптографии при работе с алгоритмом RSA, где применяются числа с 20-ю и более разрядами.

### Константа Капрекаса

А сейчас рассмотрим константу Капрекаса. Напомню, что это за константа. Задано четырехзначное натуральное число, у которого

не все цифры равны между собой. Пусть  $a, b, c, d$  – четыре цифры десятичной записи этого числа. Получим из этих цифр два числа: в первом числе цифры расположены в порядке не возрастания, а во втором – в обратном порядке (неубывания). Вычтем из первого числа второе. К полученной разности применим ту же процедуру.

Оказывается, что такое преобразование для любого начального элемента обязательно приведет через некоторое число шагов к числу 6174, которое и называется константой Капрекаса. Так вот, чтобы это доказать, можно пойти путем полного перебора. Для этого нужно рассмотреть все возможные варианты. Т.е. для всех четырехразрядных натуральных чисел не считая 1111, 2222, ..., 9999 получим, что нужно проверить 8991 число. И не забывайте, что для каждого числа нужно несколько раз применить вышеописанную процедуру.

Попытаемся сократить число вариантов перебора. Если  $a, b, c, d$  – цифры исходного числа и  $a \geq b \geq c \geq d$ , то  $M1 = 1000a + 100b + 10c + d$  – наибольшее число из этих цифр, а  $M2 = 1000d + 100c + 10b + a$  – наименьшее число. Тогда их разность равна  $R = 999(a - d) + 90(b - c)$ , где  $a > d$ ,  $0 < a - d \leq 9$ ,  $b - c < a - d$ . В итоге, нужно проверить 45 вариантов, что почти в 200 раз меньше, чем при полном переборе.

Интересно, а есть ли похожая константа для чисел с другим количеством цифр? Чтобы совершить проверку нужно предварительно найти предполагаемую константу. Для этого воспользуемся следующим правилом: если на двух последовательных шагах (операция описана при рассмотрении константы Капрекаса) получается одно и то же число, то это число и есть константа для данного  $n$ , где  $n$  – число разрядов проверяемого числа.

Рассмотрим  $n = 3$ , т.е. 100, 101, 102, ..., 988 – исходная последовательность за исключением 111, 222, ..., 888. Компьютерная программа выдала следующий результат: искомое число равно 495. Если провести те же рассуждения, что и при сокращении вариантов перебора при константе Капрекаса, то получим 45 вариантов. Но, если рассмотреть оставшиеся числа окажется, что останется всего 5 вариантов, так

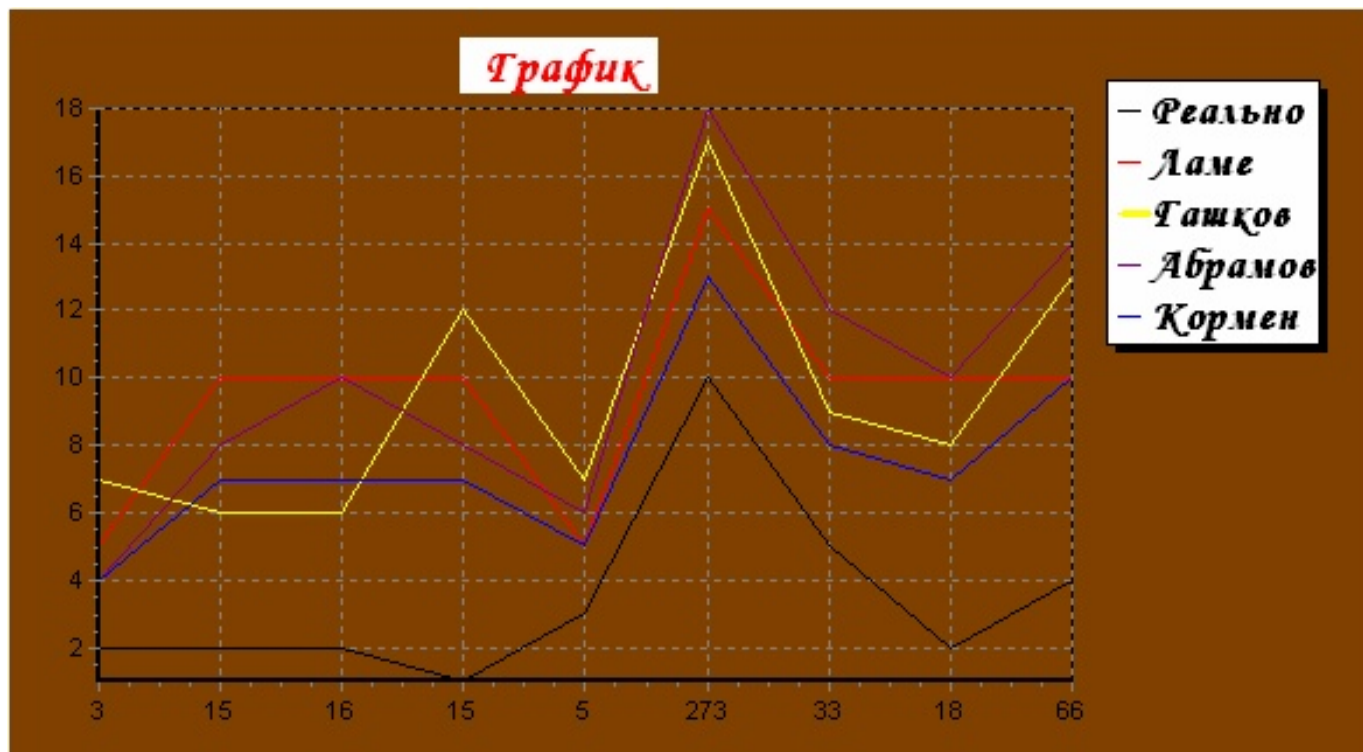


Рис. 2. Графическое изображение данных рисунка

как в остальных 40-ка числах повторяются одни и те же числа. А это почти в 180 раз меньше, чем при полном переборе. Если рассмотреть числа с количеством разрядов больше чем 4, то та же программа показывает, что константы нет, т.е. не выполняется начальное правило поиска константы.

### Способы уменьшения временных затрат

А сейчас, давайте посмотрим, как во время написания кода можно еще больше уменьшить время выполнения программы. Многие программисты сталкиваются на этапе реализации программы с проблемой: какой из альтернативных процедур, функций или операторов воспользоваться. Например, как лучше возвести число  $x$  в квадрат: воспользоваться функцией `Sqr(x)` или  $x*x$ .

Часто выбор падает на более компактную конструкцию. И это объясняется тем, что так код программы легче читается и более краткий, а раз код меньше значит и программа выполняется

быстрее. Вот тут и возникает вопрос: а будет ли более короткий код, выполняться быстрее? Чтобы ответить на этот вопрос, рассмотрим таблицы 1 и 2\*\*. В первой таблице сравниваются\*\*\* три цикла (цикл состоит из 100 млн. шагов). Во второй таблице сравниваются некоторые распространенные функции (результат получен на базе цикла `for` в 100 млн. шагов):

Табл.1. Сравнение трех циклов на быстродействие

Цикл	Время выполнения, мс
For	61
While	60
Repeat	57

Можно сказать, что циклы работают одинаково быстро\*\*\*\*, и выбор конкретного цикла зависит в основном от условия решаемой задачи, т.е. для возведения числа  $x$  в квадрат лучше использовать функцию `Sqr(x)`, которая почти в 1.1 раза быстрее работает, чем запись  $x*x$ , и в 6.5 раз быстрее, чем функция `Power(x,2)`.

Если же нужно возвести число  $x$  в третью и большую степень, то, без сомнений, нужно воспользоваться функцией `Power()`, которая почти в 4.3 раза работает быстрее, чем возводить

#### \*\* Комментарий автора.

Таблицы заполнены программой написанной на языке Delphi, выполняемой на компьютере с процессором, работающим с частотой 1.73 ГГц.

Табл.2. Сравнение распространенных операций на быстрдействие

Функция	Время выполнения, мс
$x \cdot x$	575
Sqr(x)	523
Power(x,2)	3393
Power(x,10)	2832
Возведение x в 10 степень с помощью цикла	12078
Max(x,x1)	409
Нахождение максимального числа с помощью оператора if ***	139
Min(x,x1)	351
Нахождение минимального числа с помощью оператора if ***	60

\*\*\* Комментарий редакции.

Следует констатировать, что данных недостаточно, чтобы однозначно утверждать что-либо. Во-первых, современные ОСи многозадачны и могут выполнять свои операции не связанные с решением задачи. Во-вторых, данные цифры могут существенно отличаться при других параметрах. И, в-третьих, программа это не просто цикл, но и еще куча операций, программисту напрямую не доступные. Самый простой и классический пример – дефрагментация кучи, когда места недостаточно, то менеджер кучи занимается высвобождением блока нужного объема. Операция возможна не всегда и зависит от ряда параметров, поэтому предсказать такое событие не представляется возможным. Иными словами, автору следовало-бы учитывать погрешность.

Кроме того, нахождение максимального и минимального числа с помощью оператора if разнятся между собой почти в два раза. Слишком сильное отличие. К сожалению, автор не предоставил исходники теста и не указал ресурс, где их можно найти, поэтому здесь опять же ничего нельзя сказать заранее. Все это очень условно.

число **x** в нужную степень с помощью цикла. Также, предпочтительно, при нахождении максимального и минимального числа воспользоваться оператором **if**, который работает быстрее почти в 3 и 5.85 раз, чем соответствующие функции.

Можно конечно возразить, что для современных быстродействующих компьютеров это особо не влияет на скорость выполнения программы. Да, это так. Когда речь идет о выполнении учебных задач, то разница незаметна. Но когда речь идет о реальных программах и приложениях, малая экономия времени играет огромную роль. К примеру, в дизайнерских или архитектурных программах, где нужно за малое время обрабатывать огромное количество информации.

Или в криптографии, в частности, при использовании алгоритма RSA, где приходится

оперировать простыми числами с двадцатью и более разрядами.

Сейчас подведем итоги

Во-первых, мы убедились в том, что вопросы увеличения производительности программ и экономия компьютерного ресурса очень важны и актуальны.

Во-вторых, мы убедились в том, что начинать экономить ресурсы компьютера и увеличивать производительность ПО можно на этапе теоретической разработки приложения.

В-третьих, при написании кода программ, если сделать «правильный выбор», также можно увеличить производительность ПО.

В-четвертых, когда пишется программа, нужно осознанно выбирать те или иные операторы, учитывая, в первую очередь, какие из них выполняются быстрее, и только потом на краткость записи кода. Ведь код всегда можно читабельно оформить. И не забывайте, что код программы будут видеть только единицы, а пользоваться продуктом множество людей, далеко не программистов. Их будет мало интересовать, как записан код. Главное, чтобы программа работала быстро и надежно.

\*\*\*\* Комментарий редакции.

Зависит от версии Дельфи. Сейчас все сильно оптимизируется, поэтому можно считать синтаксическим сахаром.

В статье не были затронуты вопросы длинной арифметики, а ведь речь идет о числах с количеством разрядов более 20-ти. Надеемся в следующем материале это упущение будет исправлено.



Часто самой большой проблемой программиста становится сохранение проекта в целости и сохранности. Как не допустить уничтожения проекта в процессе работы и создать сайт для любого браузера – об этом вы узнаете в статье «Маленькие помощники программиста».



**Алексей Шишкин**

by **AlexCones** <http://flsoft.ru>

В этой части нашего «путешествия» по обстановке программиста будущего я бы хотел рассказать о таком важном слове, как бекап\*, и о моем «уголке веб-девелопера».

**\* Комментарий автора.**

(англ) BackUp – резервная копия

Итак, начнем с бекапов, как самого важного в жизни программистов. Да, вы не ослышались, это важнее, чем знать в совершенстве какой-либо язык программирования, ибо, если дать двум программистам, новичку и профессионалу, в руки среду разработки и дать инструмент бекапа только в руки новичка, при повреждении данных именно он выйдет победителем. Итак, начнем. Для начала нам понадобится OpenSource\*\* программа 7zip и Delphi-образная (к примеру, Lazarus) среда разработки.

1. Для начала создадим на жестком диске папку, куда будем складывать наши бекапы (см. рисунок 1):



Рис. 1. Создаем директорию\*\*\* для копий

2. Определившись с папкой, загружаем утилиту 7z <http://www.7-zip.org>

3. Устанавливаем ее в папку по вашему усмотрению. Отныне я буду называть ее %7ZFolder%, а папку, которую мы создали для хранения бекапов, – %CopyFolder%.

4. Теперь зайдем в %7ZFolder% и создадим там \*.txt файл следующего содержания:

```
7z.exe a %DATE%.7z D:\Finder
Move %DATE%.7z D:\Copy\Finder\
Cd D:\Copy\Finder
Renamer %DATE%.7z
```

Где, в данном случае: D:\Finder – папка проекта, который мы будем бекапить, D:\Copy\Finder – %CopyFolder%.

5. Переименовываем этот файл в \*.bat

6. Но что же значит загадочный Renamer? Его созданием мы сейчас и займемся.

7. Открываем среду разработки и, создав Console Application, пишем:

```
Program Renamer;

Uses Windows, SysUtils;

Var
  Name1, Name2, S : String;
  I : Integer;
Begin
  Name1 := ParamStr(1);
  S := Name1;
  Name2 := '';
  While Pos('\', S) <= 0 Do Begin
    Name2 := Name2 + S[1];
    Delete(S, 1, 1);
  End;

  For I := 1 to Length(S) - 2 Do
    Name2 := Name2 + S[I];
  Name2 := Name2 + TimeToStr(Time) + S[Length(S)-2] +
    S[Length(S)-1] + S[Length(S)];
  For I := 1 To Length(Name2) Do
    If Name2[I] = ':' Then Name2[I] := '.';
  RenameFile(Name1, Name2)
end.
```

Думаю, не надо пояснять, что делает этот код. Компилируем его и отправляем Renamer.exe в %CopeFolder%.

**\*\* Комментарий автора.**

С открытым исходным кодом, распространяющаяся по лицензии GNU GPL.

## \*\*\* Комментарий автора.

Имена для папок желательно выбирать короткими и без кириллицы: будет меньше проблем с \*.bat файлом.

8. Вернемся к нашей основной мысли: как же нам запустить все это вместе? Создадим новый проект (как и прошлый, он будет Console Application). Обратите внимание, что в обоих случаях строк {\$ Console Application} НЕ БУДЕТ. Сделано это для того, чтобы консоль не мозолила нам глаза:

```
program Watcher;

uses ShellAPI, Windows;

Var
    Look    : Boolean = FALSE;
    Gotcha  : Boolean = FALSE;
    Hand    : Integer;
Begin
    While True Do Begin
        (* Search for Lazarus *)
        Hand := 0;
        Hand := FindWindow('Window',
            'Lazarus IDE v0.9.28.2 бета - Finder.lpi');
        Gotcha := (Hand <> 0);
        If (Look) And Not(Gotcha) Then Begin
            ShellExecute(0, 'Open', 'D:\Programs\7-Zip\Arc.bat',
                Nil, Nil, SW_HIDE);
            Look := FALSE;
        End
    Else
        If Not(Look) And (Gotcha) Then Look := TRUE;
        Sleep(10 * 1000);
    End
End.
```

Разумеется, следует заменить директории в коде на свои. Обратите внимание, что я использовал для поиска заголовков и класс окна лазаруса, так как свой проект я создаю в нем. Замените их на класс и заголовок окна своей среды разработки.

9. Итак, мы почти пришли, осталось поместить ярлык на эту программу в папку ПУСК → Все программы → Автозагрузка, и дело сделано. Теперь достаточно перезагрузить компьютер или

вручную запустить [Watcher.exe](#), и система начнет работать. Как же она работает? Очень просто:

Watcher раз в 10 секунд ищет заголовок и класс среды разработки с открытым проектом для бекапа:

- если она найдена, флаг [Gotcha](#) будет >0;
- продолжаем следить за процессом, и если окно не будет найдено, действуем;
- запускается bat файл Arc.bat;
- он бекапит папку с проектом и дает ей имя сегодняшней даты;
- затем он отправляет его Renamer`у, который переименовывает его согласно дате и времени.

Результат наших действий (см. рисунок 2):

Рис. 2. Часть бекапов в папке после часа работы (чтобы уменьшить их количество, измените число после Sleep в большую сторону)

04.09.2010.11.19.29.7z  
04.09.2010.11.52.49.7z  
04.09.2010.12.08.29.7z  
04.09.2010.12.10.09.7z  
04.09.2010.12.13.19.7z  
04.09.2010.12.20.59.7z  
04.09.2010.12.51.19.7z  
04.09.2010.13.15.00.7z

Итак, мы научились бекапить свои проекты, и теперь перебои электричества и повреждения жесткого диска нам не страшны. Только не забывайте хотя бы изредка сбрасывать свои бекапы на съемный диск или CD/DVD.

**Перейдем ко второй части** нашего рассказа, ведь я еще обещал рассказать про свой «уголок» веб-девелопера... Понадобилось мне как-то сваять в блокноте сайт для одного из своих проектов. Разумеется, я обладал некоторыми основами сайтостроения, но, разумеется, я (как и большинство веб-девелоперов) не знал заранее, как сайт будет выглядеть во всех браузерах.

И сейчас расскажу, как я избавился от этой проблемы.

Для создания этого древнего заклинания нам потребуется:

- артефакт «Флеш карта» размером от 1 Гб; доступ в Интернет на 15-20 минут;
- Delphi-образная среда разработки;
- 10 минут приготовлений.

Итак, переходим на сайт <http://torwald.ru/72> и выкачиваем ВСЕ браузеры, которые видим. По очереди запускаем инсталляторы и устанавливаем их на флешку. Некоторые из браузеров даже не потребуют установки – их просто копируем.

Далее пробуем воссоздать такую схему (см. рисунок 3):

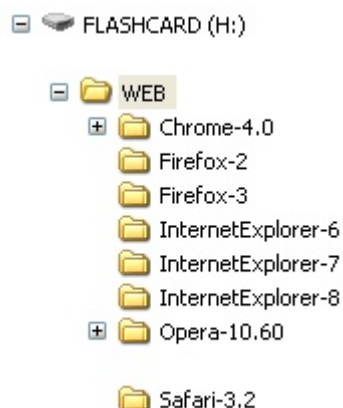


Рис. 3. Папки с программами

Теперь, когда мы все это установили, объединим все это одним заклинанием, чтобы не утруждать себя чтением древних свитков каждые пять минут:

```
Program WEBC;

Uses ShellAPI, Windows;

Var
  FileName : String;
  CurDir   : String;
  F        : TextFile;
Begin
  FileName := ParamStr(1);
  CurDir := 'H:\WEB';
  AssignFile(F, CurDir + '\Run.bat');
  Rewrite(F);
  WriteLn(F, CurDir + '\InternetExplorer-6\ie6.exe ' +
    FileName);
  WriteLn(F, CurDir + '\InternetExplorer-7\ie7.exe ' +
    FileName);
  WriteLn(F, CurDir + '\InternetExplorer-8\ie8.exe ' +
    FileName);
  WriteLn(F, CurDir + '\Chrome-4.0\Chrome.exe ' + FileName);
  WriteLn(F, CurDir + '\Safari-3.2\safari.exe ' + FileName);
  WriteLn(F, CurDir + '\Firefox-2\firefox2.exe ' + FileName);
  WriteLn(F, CurDir + '\Firefox-3\firefox3.exe ' + FileName);
  WriteLn(F, 'pause');
  CloseFile(F);
```

```
ShellExecute(HWND(0), 'Open', PChar('Run.bat'), '',
  PChar('H:\WEB'), SW_SHOWNORMAL);

End.
```

Вместо **H:\** следует подставить букву диска своей флешкарты. Поместим эту программу в папку **WEB** на флешкарте и создадим ярлык на ней, который поместим в директорию **C:\Documents and settings\%Username%\SendTo**. Не забудьте переименовать ярлык в **WEB test.lnk**

Все, заклинание готово! Запускать его будем так: клик правой кнопкой на **HTML\HTM** документ, который необходимо проверить, и выбираем **ОТПРАВИТЬ – WEB test**. Документ откроется по очереди в разных браузерах, и вы сможете увидеть, как он выглядит у пользователей.

### Заключение

При разработке использовалась среда разработки – Lazarus 0.9.28 и блокнот. Дело шло, как ни странно, под «окошками».

Итак, мы научились сохранять свои проекты в условиях глобальных катаклизмов и создавать сайты для каждого браузера. Это лишь часть знаний, которые помогут вам в жизни, но надеюсь, что когда-нибудь они вам пригодятся.

Удачи!

### Ссылки

- 7zip архиватор <http://www.7-zip.org>
- Portable версии браузеров <http://torwald.ru/72>
- Скачать среду разработки Lazarus <http://www.lazarus.freepascal.org>





В первой части нашего материала уже было сказано, что в качестве основной операционной системы для серверов сети была выбрана платформа Windows. Почему не Linux или, например, FreeBSD? Закономерный вопрос, причин такого выбора несколько...



**Александр Горский**

by **WildHunter** <http://airnet.vberdyanske.net>



### **Серверы, или «Кто кинул сапог на пульт управления?»**

Во-первых, максимальная простота в развертывании, управлении и эксплуатации, при этом достаточная гибкость.

Вторым и немаловажным критерием послужило то, что специфика сети (и ограниченный бюджет) не позволяли использовать специализированное (и дорогостоящее) оборудование, необходимое для версий Linux, «заточенных» под провайдерские задачи. Да и сами программные решения для организации сети под Linux не являются бесплатными и вполне сравнимы по цене с аналогичными решениями на базе Windows.

Третьим фактором послужила проверенная временем и большим опытом надежность Windows Server. Вижу ехидные улыбки приверженцев Linux, но это именно так и есть. За более чем 10-летний опыт мы не сталкивались с немотивированными отказами, все серверные сбои имели вполне закономерные причины, от не совсем подходящего «железа» до «человеческого фактора».

Четвертая и последняя причина – никто из организаторов проекта раньше не сталкивался с построением довольно больших сетей на базе Linux, зато кое-какой опыт использования для этих целей Windows у нас был.

### **Пара общих слов о сети**

Прежде чем перейти непосредственно к технологиям, необходимо сказать несколько слов

о том, что собой представляет наша сеть в плане услуг и правил пользования ими. Определяющее значение имеет то, что сеть некоммерческая, то есть по определению бесплатная и со свободным доступом. Но это не значит, что в сети может быть бардак и основным принципом является «что хочу, то и делаю». Единственное, что требуется от пользователей, – соблюдение правил, суть которых сводится к «пользуйся сам и не мешай другим».

Подключившийся к сети новичок сразу попадает на наш сайт, где изложены правила поведения в ней, инструкции по правильной настройке различных программ и оборудования. Задать вопрос или получить консультацию можно на нашем форуме, который также общедоступен.

Без регистрации сразу доступны и некоторые внутрисетевые сервисы, например FTP и DC++, но для выхода в Интернет необходимо зарегистрироваться, получить логин и пароль и настроить VPN-подключение. Регистрация предельно проста, главное – указать свои контактные данные, чтобы администрация смогла при необходимости оперативно связаться с пользователем.

Максимальная скорость его работы ограничена на уровне 10 Mbps, реальная скорость зависит в основном от качества беспроводного соединения «точка-клиент».

Ограничений по использованию Интернета минимум, запрещено только использование сети в криминальных целях; существуют некоторые ограничения и по количеству одновременных соединений, что требует правильной настройки некоторых программ (например, torrent-клиентов).

### **Централизация или распределение?**

Несомненно, что достаточно большая сеть должна быть структурированной, состоять из

нескольких сегментов, каждый из которых имеет свое адресное пространство и свой гейт для связи с другими сегментами и с центральным сервером (ЦС) сети. Но для эффективного контроля сеть должна быть построена по общему плану и с централизованным управлением. Именно эти функции выполняет ЦС.

Основные функции гейта сегмента:

- назначение клиентским устройствам IP-адресов, адреса шлюза и внутреннего DNS;
- маршрутизация трафика в другие сегменты и на ЦС;
- контроль и учет трафика внутри сегмента для сбора статистики и диагностики проблем.

Основные функции центрального сервера:

- сервер AD (домена) для централизованного управления сетью;
- сервер VPN и маршрутизации в Интернет/из него;
- Firewall корпоративного уровня.

Все эти серверные роли (кроме Firewall) являются стандартными для Windows Server 2003, что гарантирует их совместимость друг с другом, удобство контроля и управления.

## Active Directory (AD)

Служба Active Directory [http://ru.wikipedia.org/wiki/Active\\_Directory](http://ru.wikipedia.org/wiki/Active_Directory) в нашей сети используется для централизованного управления пользователями. Именно в ней хранятся данные о пользователе, его логин и пароль, а также базовые настройки (см. рисунок 1).

Здесь также содержатся основные настройки этого пользователя для сервера VPN, такие как IP-адрес, назначаемый VPN-подключению пользователя, и дополнительные маршруты, если необходимо (см. рисунок 2).

Также AD имеет множество дополнительных возможностей, которые могут оказаться полезными в конкретных ситуациях.

## Сервер маршрутизации и удаленного доступа (VPN-сервер)

Сервер виртуальной частной сети (VPN) используется для защищенного подключения пользователей к центральному серверу сети. В первую очередь это необходимо для защиты конфиденциальных данных пользователей в открытой сети. Ни для кого не секрет, что сейчас есть множество желающих пожить за чужой

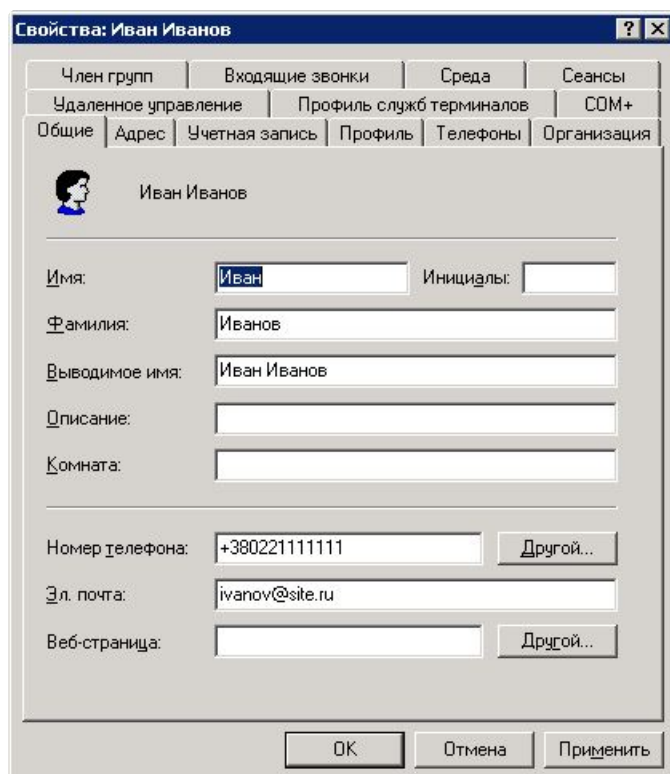


Рис. 1. Общие данные пользователя в AD

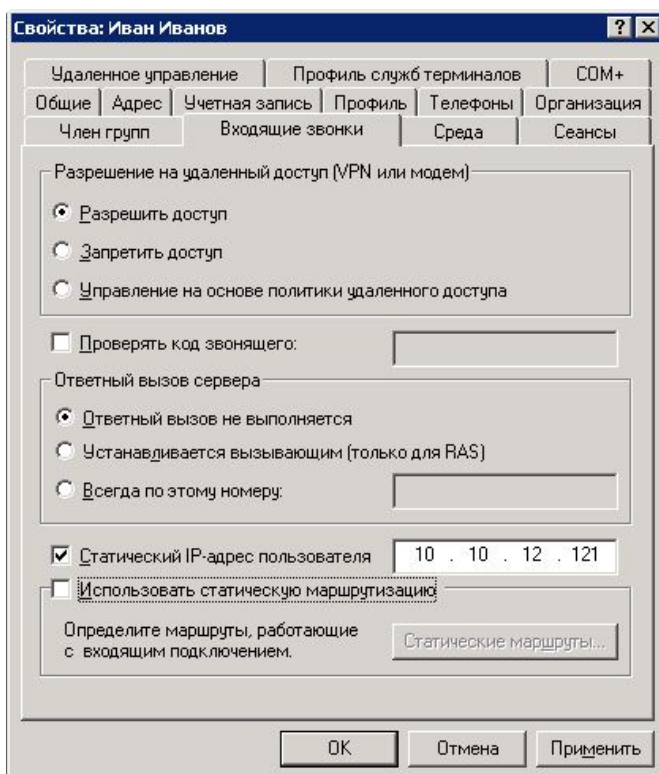


Рис. 2. Основные настройки пользователя для VPN-сервера

счет или просто сделать пакость\*. Именно для предотвращения подобного и служит подключение VPN, в котором весь трафик «клиент-сервер» защищен 128-битным шифрованием (см. рисунок 3).

Firewall и учет интернет-трафика

Перепробовав множество программ для организации доступа из локальной сети в Интернет, мы остановились на Kerio Winroute Firewall\*\*, как наиболее удовлетворяющей нашим требованиям по надежности, гибкости и удобству в эксплуатации (см. рисунок 4 и 5).

Кроме выполнения функций классического Firewall, в KWF заложена еще масса различных возможностей, позволяющих использовать эту систему как полноценный интернет-шлюз для

небольшой или средней сети. KWF имеет встроенный web-сервер, на котором пользователи могут посмотреть свою статистику, а администраторы – статистику доступа в Интернет как по сети в целом, так и по отдельным пользователям (см. рисунок 6 и 7).

К несомненным достоинствам KWF также можно отнести удобный интерфейс администрирования, который позволяет управлять шлюзом не только непосредственно с сервера, но и удаленно. Единственный обнаруженный недостаток – отсутствие встроенного шейпера <http://ru.wikipedia.org/wiki/Шейпинг>, позволяющего управлять скоростью каждого пользователя отдельно. Об этом далее...

Шейпер

В качестве простого, но достаточно гибкого и эффективного шейпера мы используем SoftPerfect Bandwidth Manager <http://www.softperfect.com> (см. рисунок 8).

**\* Комментарий автора.**  
...например, украсть номер ICQ, написать какую-нибудь гадость от чужого имени в социальной сети и т.п.

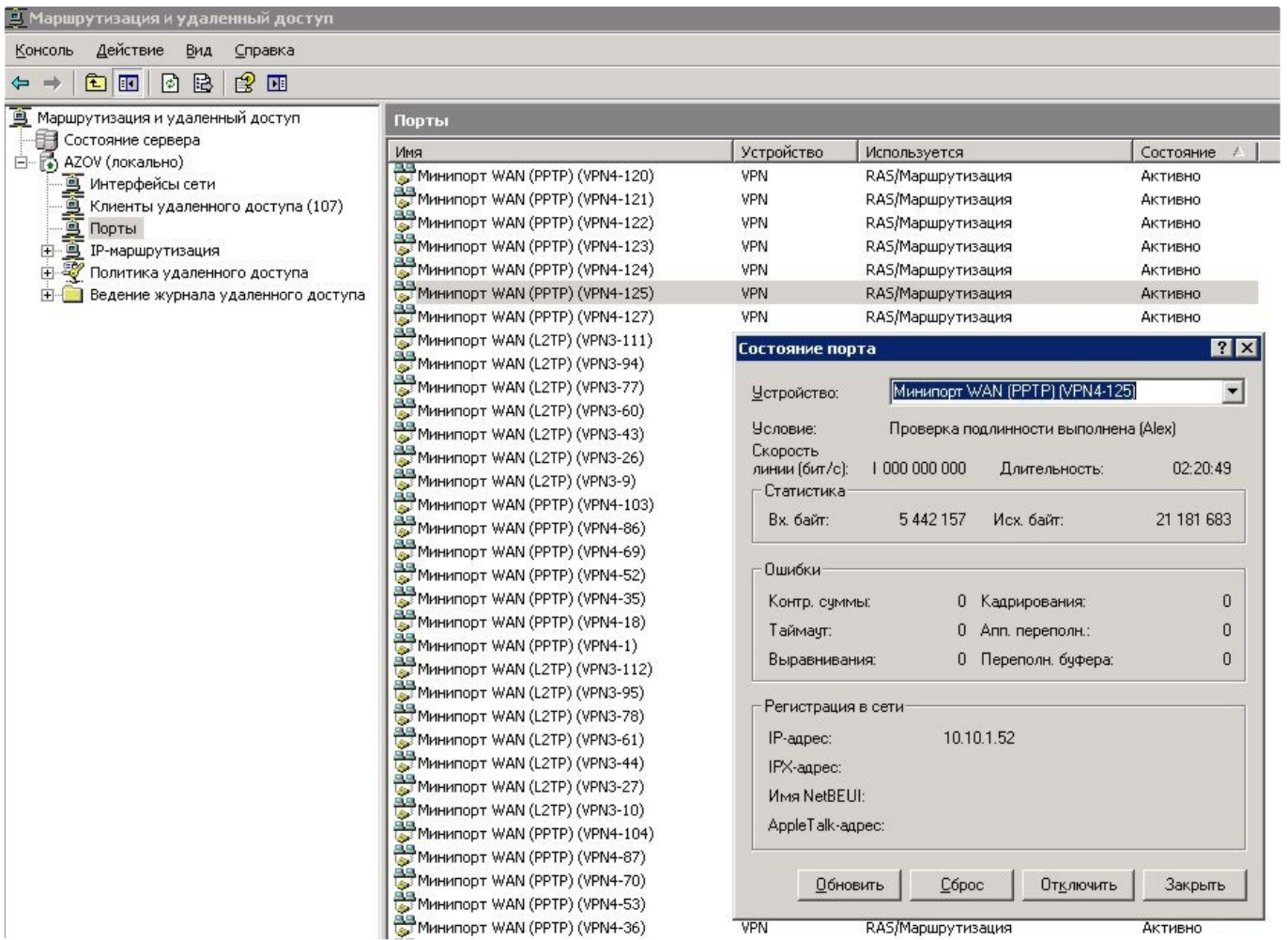


Рис. 3. Интерфейс сервера маршрутизации и удаленного доступа, используемого в качестве VPN-сервера



Ограничение скорости настраивается очень гибко, как для хоста в целом (или группы хостов), так и для отдельных портов (диапазонов портов), протоколов, интерфейсов и т.д. Возможно создание различных правил для различных временных диапазонов, например, в ночное время скорость выше, чем в дневное. Имеется также приличный набор дополнительных

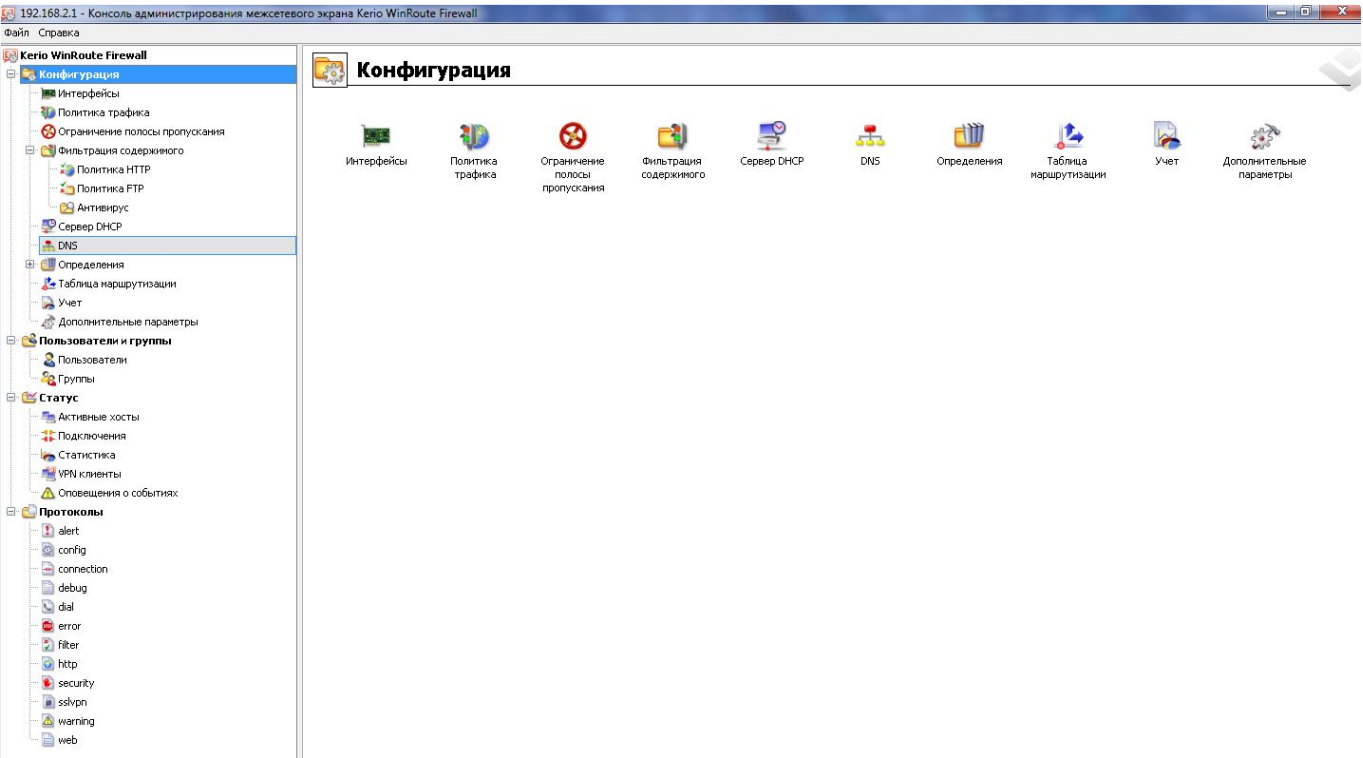


Рис. 4. Общий интерфейс KWF

Имя хоста	Пользователь	Прием [кб/с]	Передача [кб/с]	IP-адрес	Продолжительность	Время бездействия	Метод аутент.	Итоговый при.	Итоговая пер.	Подключения
192.168.1.1	Администратор	0.2	0.1	192.168.1.1	18:34:51	00:00:00	Automatic	243 082.1	492 501.8	4
192.168.1.2	Администратор			192.168.1.2	01:21:39	00:18:11	Automatic	28 697.0	1 033.6	
192.168.1.3	Администратор			192.168.1.3	00:55:13	00:08:11	Automatic	131.3	139.5	
192.168.1.4	Администратор			192.168.1.4	03:53:50	00:00:05	Automatic	6 524.4	970.8	2
192.168.1.5	Администратор	191.9	11.9	192.168.1.5	01:29:43	00:00:00	Automatic	19 622.0	3 225.4	27
192.168.1.6	Администратор			192.168.1.6	01:04:06	00:08:31	Automatic	15 967.9	673.4	
192.168.1.7	Администратор	4.4	0.3	192.168.1.7	01:27:08	00:00:05	Automatic	7 498.1	894.6	
192.168.1.8	Администратор	199.9	1.9	192.168.1.8	00:31:07	00:00:00	Automatic	185 799.6	10 344.9	50
192.168.1.9	Администратор			192.168.1.9	03:04:08	00:00:01	Automatic	5 378.9	1 269.1	2
192.168.1.10	Администратор	287.8	0.2	192.168.1.10	03:38:26	00:00:00	Automatic	24 807.1	6 568.5	32
192.168.1.11	Администратор			192.168.1.11	02:15:09	00:31:09	Automatic	266 880.0	7 019.0	
192.168.1.12	Администратор			192.168.1.12	00:53:38	00:30:18	Automatic	1 844.1	330.1	
192.168.1.13	Администратор			192.168.1.13	03:40:14	00:00:01	Automatic	250 102.0	14 102.4	24
192.168.1.14	Администратор			192.168.1.14	03:43:46	00:28:22	Automatic	13 014.9	2 911.5	
192.168.1.15	Администратор			192.168.1.15	00:36:25	00:35:04	Automatic	0.0	0.0	
192.168.1.16	Администратор			192.168.1.16	00:29:29	00:00:17	Automatic	1.7	0.0	3
192.168.1.17	Администратор			192.168.1.17	00:55:12	00:49:53	Automatic	16.5	11.2	
192.168.1.18	Администратор			192.168.1.18	01:40:53	00:57:27	Automatic	3 294.5	676.9	
192.168.1.19	Администратор			192.168.1.19	03:05:16	00:01:19	Automatic	124.7	22.5	1
192.168.1.20	Администратор			192.168.1.20	00:55:12	00:00:13	Automatic	129.0	115.9	2
192.168.1.21	Администратор			192.168.1.21	03:04:50	00:00:54	Automatic	156.9	41.9	
192.168.1.22	Администратор			192.168.1.22	01:47:50	00:21:51	Automatic	2 667.3	483.8	
192.168.1.23	Администратор			192.168.1.23	01:17:56	00:39:54	Automatic	1 541.6	335.9	
192.168.1.24	Администратор			192.168.1.24	01:22:57	00:20:42	Automatic	963.8	26.2	
192.168.1.25	Администратор			192.168.1.25	07:43:08	00:00:43	Automatic	12 553.9	2 145.6	
192.168.1.26	Администратор			192.168.1.26	01:29:59	00:00:23	Automatic	472.2	67.0	
192.168.1.27	Администратор			192.168.1.27	03:40:47	00:01:48	Automatic	1 216.0	285.7	
192.168.1.28	Администратор			192.168.1.28	02:07:08	00:25:42	Automatic	37 876.0	3 594.4	
192.168.1.29	Администратор			192.168.1.29	00:51:42	00:43:11	Automatic	6 938.0	597.6	
192.168.1.30	Администратор			192.168.1.30	00:31:11	00:00:04	Automatic	0.4	0.9	
192.168.1.31	Администратор			192.168.1.31	01:11:46	00:52:23	Automatic	22.3	4.8	
192.168.1.32	Администратор			192.168.1.32	04:00:39	00:00:07	Automatic	138.8	151.8	
192.168.1.33	Администратор	484.4	0.3	192.168.1.33	02:19:04	00:00:08	Automatic	492 689.6	287.7	7
192.168.1.34	Администратор			192.168.1.34	01:04:44	00:04:24	Automatic	1.0	0.6	

Рис. 5. Текущая активность пользователей в KWF  
(данные пользователей по понятным причинам скрыты)

**\*\* Комментарий автора.**

...сокращенно KWF <http://www.kerio.ru>

функций (определение P2P трафика, флуда мелкими пакетами, большого количества одновременных подключений и т. п.). В случае обнаружения нежелательных типов трафика могут автоматически применяться меры, например снижение разрешенной скорости.

### Немного статистики и эпилог

Для начала немного статистики, чтобы можно было получить представление о масштабах и объемах:

- 4 сегмента сети, состоящие из 19 точек доступа, уверенно «накрывающих» микрорайон средних размеров;
- около 300 постоянных и около 2000 «периодических» пользователей;
- одновременно в сети до 300 активно работающих хостов;
- среднемесячный интернет-трафик до 3 терабайт;
- среднемесячный внутрисетевой трафик до 10 терабайт.

Конечно, пришлось вложить много сил (и прилично средств) для создания такой сети, но в итоге проект успешно работает и потихоньку

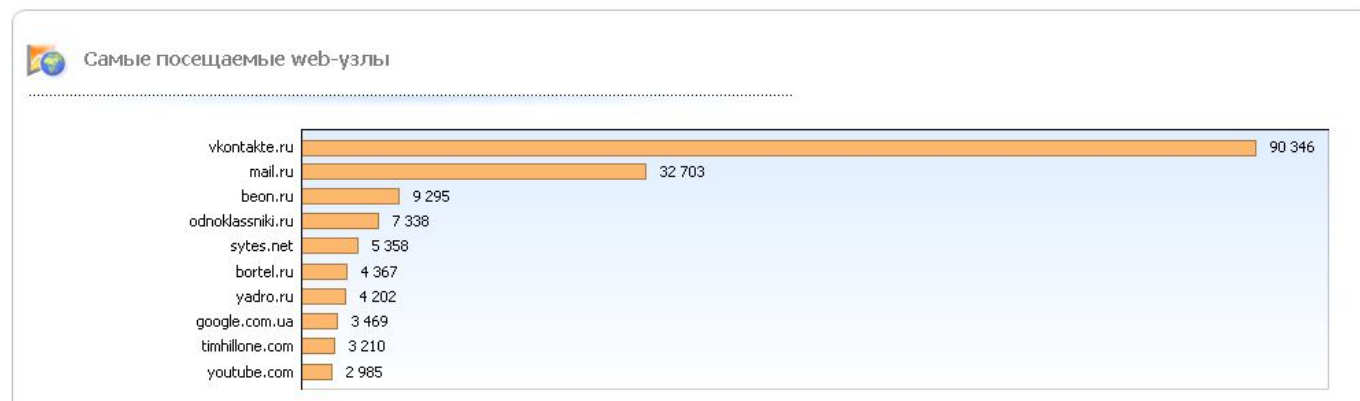


Рис. 6. Пример статистики по самым посещаемым из сети web-узлам за неделю

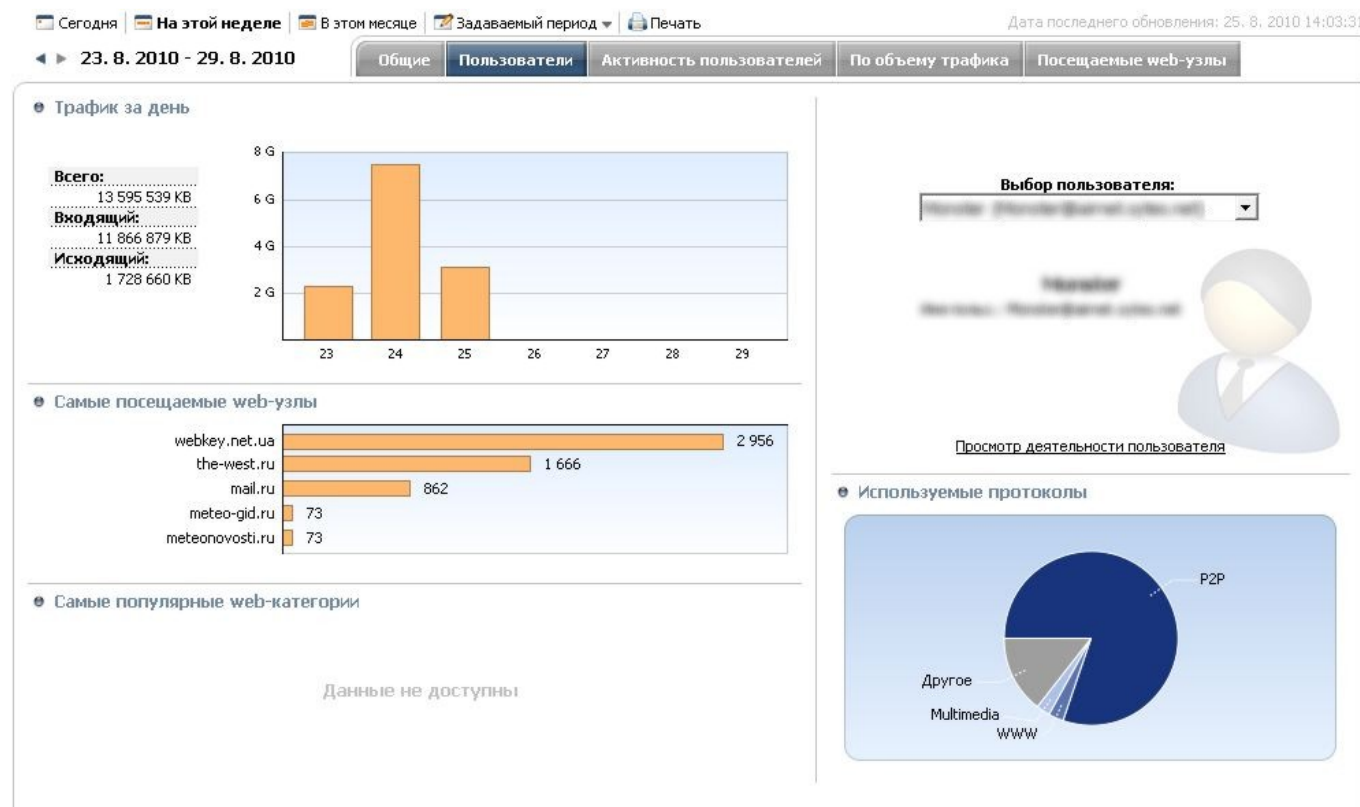


Рис. 7. Пример статистики пользователя за текущую неделю

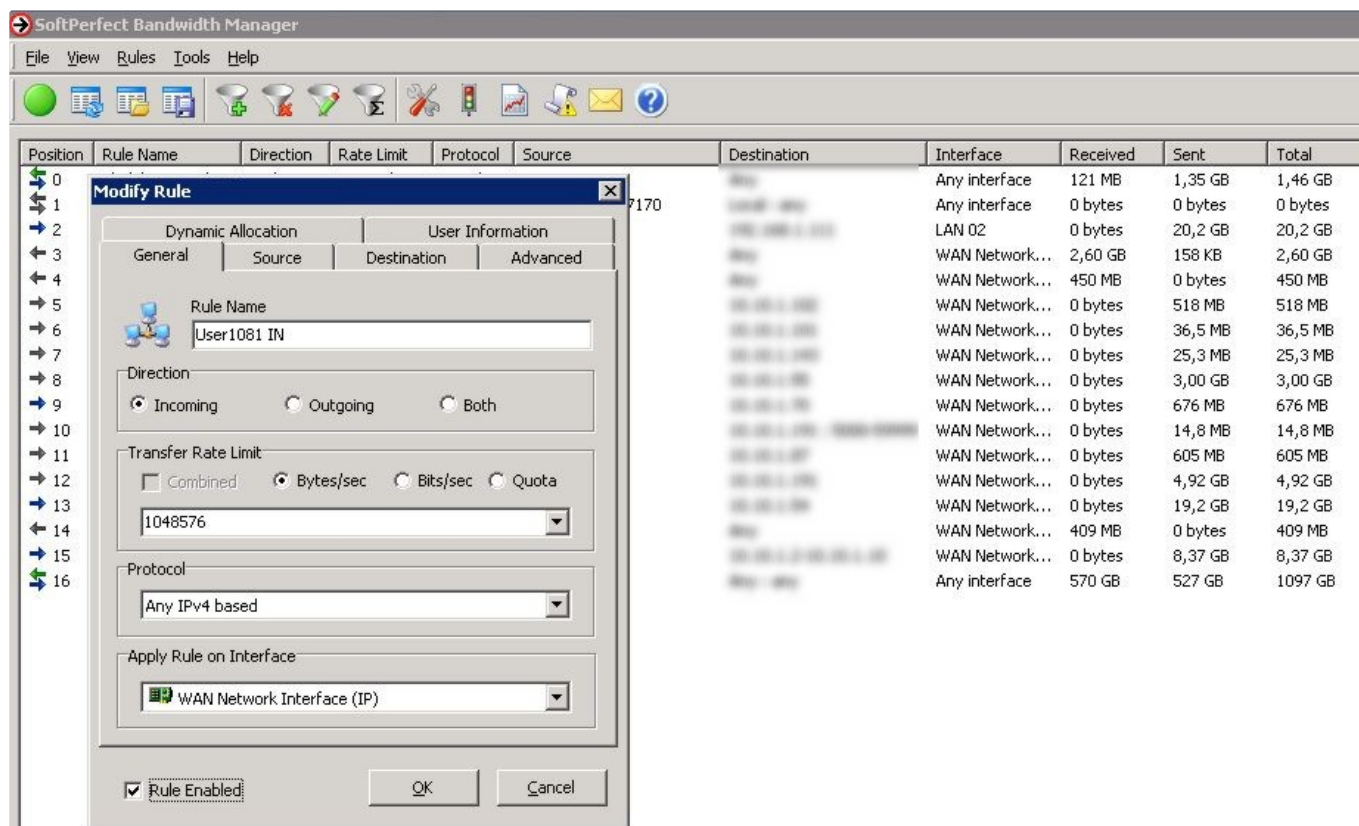


Рис. 8. Общий интерфейс SoftPerfect Bandwidth Manager

развивается. Сеть показала себя как достаточно надежная, гибкая и легко расширяемая система, соответствующая современным требованиям по скорости и качеству. В целом уровень услуг как минимум не хуже, чем у коммерческих провайдеров, работающих в нашем городе.

В перспективе – расширение на весь город (а возможно, и за его пределы), полный переход на стандарт 802.11n и много других интересных задумок.

*Продолжение следует...*

**AirNet** – это некоммерческий проект, организованный сообществом OpenHosts Group. Целью проекта является исследование востребованности беспроводных сетей, отработка технологий, тестирование оборудования и т.д. Проект <http://airnet.vberdyanske.net> города Бердянск поддерживается исключительно силами добровольцев, не предоставляет никаких платных и гарантированных услуг.

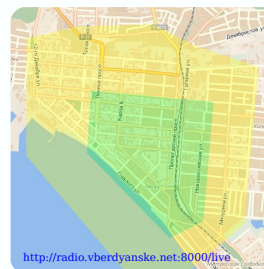


Контактный e-mail: [ailab@i.ua](mailto:ailab@i.ua)

ICQ: 48400568

Тел.: +380-664473731, +380-992757779

Группа в контакте <http://vkontakte.ru/club20515243>



На настоящий момент зона уверенного покрытия от Приморской площади до 3-го пляжа, от берега моря вглубь города примерно на 800 метров. Зона неуверенного покрытия практически вся нижняя часть города. Покрытие обеспечивают точки доступа (хотспоты) стандарта 802.11g с идентификаторами AirNet-01 - AirNet-20 и хотспоты стандарта 802.11n с идентификаторами AirStream-01 - AirStream-05. В пределах «Зеленой зоны» возможно уверенное подключение к сети практически в любых местах WiFi-устройств. За пределами «Желтой зоны» возможно подключение с помощью спецоборудования.



Многие профессионалы и новички в программировании под Windows задаются вопросом, как создавать окна необычной формы с использованием различных эффектов, таких как тени, к примеру. Такие окна часто можно встретить в виде виджетов/дополнений на рабочем столе Windows 7/Vista или же заставки при запуске приложений – к примеру, Photoshop (все мы помним то перышко, выходящее за границу прямоугольника). Также имеет смысл создавать подобные окна с полноценными элементами управления: данная технология позволяет форматировать вывод окна попиксельно, то есть можно настроить прозрачность и цвет любого пикселя окна вашего приложения – разве не здорово?



**Владимир Лихонос**

by **ВОВАН13** [www.programmersforum.ru](http://www.programmersforum.ru)



## Введение

Обычно после осознания преимуществ технологии мы понимаем, что есть смысл использовать ее. Основная проблема в том, что нет возможности

использовать стандартные компоненты той же VCL (Delphi), Win API Controls и другое, т.к. после настройки окна на использование данной технологии окно более не получает сообщение WM\_PAINT, соответственно пользователь более не увидит элементы управления.

## Основная часть

Давайте рассмотрим простой пример: отображение PNG иконки в окне, причем будем рисовать загруженную иконку с тенью. Нам понадобятся:

- Microsoft Visual Studio 2010 / MFC / C++;
- Иконка: android.png (см. ресурсы к статье в папке Images).

Вот и все. Будем идти, рассматривая детально весь процесс подготовки и корректной организации подобных окон (оговорюсь, что подобный подход я лично вывел для себя в течение нескольких лет практики; пишите как вам удобней, решений может быть множество).

Итак, если кто-то не знаком с MFC, давайте создадим пустой проект **MS VS** → **File** → **New** → **Project** → **Visual C++** → **General** → **Empty Project**, назовем его **LayersExample**. Теперь создадим иерархию файлов подобным образом. **Header Files**:

```
app.h
dib.h
layersExample.h
stdafx.h
```

## Source Files:

```
app.cpp
dib.cpp
layersExample.cpp
main.cpp
```

Теперь рассмотрим **app.h/.cpp**:

```
#ifndef APP_H
#define APP_H

#include <afxwin.h>
#include <gdiplus.h>
#include "stdafx.h"
#include "layersExample.h"

class CApp: public CWinApp
{
public:
    CApp();
    ~CApp();

    virtual BOOL InitInstance();
    virtual int ExitInstance();
public:
```

```
Gdiplus::GdiplusStartupInput gdiplusStartupInput;

ULONG_PTR gdiplusToken;

};

#endif /* APP_H */
```

Принципиально нового здесь ничего нет: наследуем класс `CWinApp` и основные его методы `InitInstance` и `ExitInstance` для обработки запуска и завершения нашего приложения соответственно. Для работы с графикой в приложении будет отвечать GDI+. Следовательно, его надо подключить `#include <gdiplus.h>` и инициализировать. Для этого объявим переменные `gdiplusStartupInput` и `gdiplusToken`. Давайте рассмотрим `app.cpp` файл:

```
#include "app.h"

CApp::CApp()
{
}

CApp::~CApp()
{
}

BOOL CApp::InitInstance()
{
    Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput,
        NULL);

    m_pMainWnd = new CLayersExample();
    m_pMainWnd->ShowWindow(SW_SHOW);

    return CWinApp::InitInstance();
}

int CApp::ExitInstance()
{
    Gdiplus::GdiplusShutdown(gdiplusToken);

    return CWinApp::ExitInstance();
}
```

Как вы уже заметили, для инициализации и деинициализации GDI+ используются функции

`GdiplusStartup` и `GdiplusShutdown` соответственно. Как основное окно используется класс `CLayersExample` – он содержит весь основной код (рассмотрим его чуть позже). Также был замечен файл под странным названием `stdafx.h`, Microsoft предлагает в нем писать различные декларации (дополнения) для приложения.

К примеру, наш `stdafx.h` будет содержать следующее:

```
#ifndef STDAFX_H
#define STDAFX_H

#pragma comment(lib, "gdiplus.lib")

#ifdef _UNICODE
#if defined _M_IX86
#pragma comment(linker, "/manifestdependency:\
type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='x86' publicKeyToken='6595b64144ccf1df'
language='*'\")
#elif defined _M_IA64
#pragma comment(linker, "/manifestdependency:\
type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='ia64' publicKeyToken='6595b64144ccf1df'
language='*'\")
#elif defined _M_X64
#pragma comment(linker, "/manifestdependency:\
type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='amd64'
publicKeyToken='6595b64144ccf1df' language='*'\")
#else
#pragma comment(linker, "/manifestdependency:\
type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='*' publicKeyToken='6595b64144ccf1df'
language='*'\")
#endif
#endif

#endif /* STDAFX_H */
```

Принципиально здесь ничего сложного, `#pragma comment(lib, "gdiplus.lib")`, как можно догадаться, используется для подключения GDI+ к проекту, иначе функции, которые продекларированы в `gdiplus.h`, не будут найдены при линковке

проекта. То, что ниже, – обычный манифест; в принципе, для нашего проекта он не особо нужен. Прежде чем перейдем к самому вкусному, надо рассмотреть `main.cpp`, иначе наше приложение попросту не запустится. Как мы помним, мы описали класс `Capp`. Теперь надо где-то объявить его экземпляр:

```
#include "app.h"

static Capp app;
```

Ничего сложного, статически создается экземпляр, и наше приложение запустится. Т.е., по сути, произойдет вызов `CApp::InitInstance`, где его поджидает инициализация GDI+ и создание главного окна. Теперь давайте рассмотрим `CLayersExample`:

```
#ifndef LAYERSEXAMPLE_H
#define LAYERSEXAMPLE_H

#include <afxwin.h>
#include <gdiplus.h>
#include "stdafx.h"
#include "dib.h"

class CLayersExample: public CFrameWnd
{
public:
    CLayersExample();
    ~CLayersExample();

public:
    void LayerDraw(CDIB *dib);
    void LayerUpdate(CDIB *dib);

private:
    CString applicationPath;

    CDIB *dibAndroid;
    CDIB *dib;

public:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnDestroy();
    afx_msg LRESULT OnNcHitTest(CPoint point);
```

```
DECLARE_MESSAGE_MAP();

};

#endif /* LAYERSEXAMPLE_H */
```

Сразу оговорюсь: вы, возможно, уже заметили `#include "dib.h"`, этот модуль я написал для связки `HBITMAP + HDC + Scan0 + Gdiplus::Bitmap` и добавил некие методы для обработки и изменения данных. Я счел ненужным описывать и обсуждать методы класса `CDIB`: в принципе, все наглядно и открыто, понять назначение того или иного метода не составит труда. За формирование нужного нам изображения (напомню, это иконка с диска и уже нарисованная к ней тень) отвечает `void LayerDraw(CDIB *dib)`:

```
void CLayersExample::LayerDraw(CDIB *dib)
{
    if (!dib->Ready() || !dibAndroid->Ready())
    {
        return;
    }

    Graphics g(dib->dc);
    g.SetCompositingMode(CompositingModeSourceCopy);
    g.SetInterpolationMode(InterpolationModeHighQualityBicubic);

    g.DrawImage(dibAndroid->bmp,
        RectF(AndroidShadowSize, AndroidShadowSize, (REAL)AndroidSize -
            AndroidShadowSize, (REAL)AndroidSize - AndroidShadowSize),
        0, 0, (REAL)dibAndroid->Width(), (REAL)dibAndroid->Height(),
        UnitPixel);

    DIB_ARGB *p = dib->scan0;
    int size = dib->Width() * dib->Height();
    for(int i = 0; i < size; i++, p++)
    {
        p->r = 0;
        p->g = 0;
        p->b = 0;
    }

    dib->Blur(dib->Rect(), CRect(0, AndroidShadowSize, 0, 0),
        AndroidShadowSize);
    dib->AlphaBlend(dib->Rect(), 160, DrawFlagsPaint);
```



```
g.SetCompositingMode(CompositingModeSourceOver);

g.DrawImage(dibAndroid->bmp, RectF(AndroidShadowSize / 2, 0,
AndroidSize, AndroidSize), 0, 0, (REAL)dibAndroid->Width(),
(REAL)dibAndroid->Height(), UnitPixel);
}
```

Рассмотрим подробнее. В начале кода мы видим `if (!dib → Ready() || !dibAndroid → Ready())`, метод `CDIB::Ready()` возвращает булево значение, т.е. если `TRUE`, то `CDIB` содержит корректно сформированный либо загруженный битмап, иначе что-то не так и доступ к любому из полей `CDIB` может вызвать исключение. В нашем случае, `dib` – это `CDIB` экземпляр, который будет хранить готовое изображение нашего будущего окна, в то время как `dibAndroid` – это загруженная иконка (`Images\android.png`) с диска:

```
Graphics g(dib->dc);

g.SetCompositingMode(CompositingModeSourceCopy);

g.SetInterpolationMode(InterpolationModeHighQualityBicubic);
```

Данный код связывает контекст GDI+ с нашим `HDC`, а именно `dib → dc`. Это необходимо для рисования нужной нам иконки на `dib`. `SetCompositingMode` устанавливает режим рисования, в нашем случае это `CompositingModeSourceCopy`, т.е. рисование с перекрытием. `SetInterpolationMode` необходим для настройки качества масштабирования изображений при рисовании их на контекст GDI+. `InterpolationModeHighQualityBicubic` – для установки лучшего качества.

На выходе мы желаем получить `dib`, внутри которого была бы иконка с тенью. Так как наша иконка без тени, алгоритм будет выглядеть приблизительно так:

- рисуем уменьшенную копию иконки; делаем изображение черным – тень будет черной;
- применяем эффект `Blur` для создания тени;
- применяем эффект `AlphaBlend` для задания тени большей прозрачности;
- рисуем оригинальную иконку поверх полученного изображения.

В принципе, ничего сложного. Однако замечу,

что эффекты `AlphaBlend` и `Blur` реализованы собственноручно и будут выполняться на CPU, а не на GPU, как бы хотелось. Следовательно, если задать большой радиус размытия для `Blur`, то это займет много времени (обычно 10-20 вполне хватает, и работает относительно быстро). Приступим к воплощению мечты в реальность по заданному алгоритму.

1. Рисуем уменьшенную копию иконки:

```
g.DrawImage(dibAndroid->bmp,
RectF(AndroidShadowSize, AndroidShadowSize,
(REAL)AndroidSize - AndroidShadowSize,
(REAL)AndroidSize - AndroidShadowSize), 0, 0,
(REAL)dibAndroid->Width(),
(REAL)dibAndroid->Height(), UnitPixel);
```

Как видим, используется метод GDI+ `DrawImage`, где применяем следующие методы:

- `dibAndroid → bmp` – `Gdiplus::Bitmap`, что рисуем;
- `RectF(AndroidShadowSize, AndroidShadowSize, (REAL)AndroidSize - AndroidShadowSize, (REAL)AndroidSize - AndroidShadowSize)` – это `Gdiplus::RectF`, куда рисуем, т.е. координаты `x` и `y`, а за ними следуют ширина и высота. О константах поговорим позже;
- `0, 0, (REAL)dibAndroid → Width(), (REAL)dibAndroid → Height()` – далее 4 параметра, а именно `x`, `y`, ширина и высота части изображения `dibAndroid->bmp`, которое будет взято для рисования на контексте `g`;
- `UnitPixel` – это формат значений, которые используются в параметрах вызова метода `DrawImage`.

2. Делаем изображение черным, тень будет черной:

```
DIB_ARGB *p = dib->scan0;

int size = dib->Width() * dib->Height();

for(int i = 0; i < size; i++, p++)
{
    p->r = 0;
    p->g = 0;
    p->b = 0;
}
```

Здесь вообще все просто: получили указатель

scan0 на первый пиксель, рассчитали общее количество пикселей в нашем dib и в цикле каждому пикселю присвоили черный цвет RGB, не затрагивая альфа-составляющую.

3. Применяем эффект Blur для создания тени:

```
dib->Blur(dib->Rect(),
          CRect(0, AndroidShadowSize, 0, 0),
          AndroidShadowSize);
```

Вызов прост, 1-й параметр **CRect** – рабочая область (по которой будет проходить размытие), 2-й параметр **CRect** – область смещения от краев изображения, которая будет игнорироваться для размытия, далее радиус размытия.

4. Применяем эффект AlphaBlend для задания тени большей прозрачности:

```
dib->AlphaBlend(dib->Rect(), 160, DrawFlagsPaint);
```

Здесь 3 параметра: 1-й – область, к которой будет применен эффект (в нашем случае вся область), далее 2-й параметр в диапазоне от 0 до 255 (уровень прозрачности) и 3-й параметр – константа. Так как мы уже рисовали с помощью GDI+, используем **DrawFlagsPaint**.

5. Рисуем оригинальную иконку поверх полученного изображения:

```
g.SetCompositingMode(CompositingModeSourceOver);
g.DrawImage(dibAndroid->bmp,
            RectF(AndroidShadowSize / 2, 0, AndroidSize,
                  AndroidSize), 0, 0, (REAL)dibAndroid->Width(),
            (REAL)dibAndroid->Height(), UnitPixel);
```

Мы уже знакомы с данными методами, остановимся лишь на новом значении **CompositingModeSourceOver**, которое отвечает за то, что иконка будет нарисована поверх уже имеющегося на нашем **dib**.

Итак, в итоге у нас готовый **dib**, содержащий нужное нам изображение. Теперь дело за малым: отобразить **dib** на нашем окне.

Для этого используется определенный нами метод **void LayerUpdate(CDIB \*dib)**:

```
if(!dib->Ready())
{
    return;
}

CRect rect;

GetWindowRect(&rect);

CWnd *wndDst = GetDesktopWindow();
CDC *hdcDst = wndDst->GetDC();

CDC *dc = new CDC();
dc->Attach(dib->dc);

BLENDFUNCTION blend = {AC_SRC_OVER, 0, 255, AC_SRC_ALPHA};

CPoint zp(0, 0);

CPoint dstPt(rect.left, rect.top);
CSize dstSize(rect.Width(), rect.Height());

UpdateLayeredWindow(hdcDst, &dstPt, &dstSize, dc, &zp, NULL,
                    &blend, ULW_ALPHA);

dc->Detach();
delete dc;

wndDst->ReleaseDC(hdcDst);
```

Для начала формирования нужных переменных необходимо определить размер и положения нашего окна. Воспользуемся API функцией **GetWindowRect**, сохраним результат в переменную **rect**. Сразу могу сказать, что данный метод в основном сводит все к MFC.





Перевести на чистый Win API, я думаю, никому не составит труда.

Теперь необходимы два CDC\*: 1-й – экран, 2-й – наш dib/изображение окна. Получим окно экрана `CWnd *wndDst = GetDesktopWindow();`, а затем и его CDC\*, `CDC *hdcDst = wndDst → GetDC()`. Так как у нас в CDIB есть только HDC, создадим новый CDC\* – `CDC *dc = new CDC()`. И свяжем его с нашим HDC `dc → Attach(dib → dc);`

Теперь необходимо подготовить структуру, описывающую детали отображения:

```
BLENDFUNCTION blend = {AC_SRC_OVER, 0, 255, AC_SRC_ALPHA};
```

Здесь мы видим также 255, это значение прозрачности нашего изображения на окне. Сразу скажу: менять его нежелательно, намного лучше перерисовать само изображение, так как (по моим наблюдениям) происходит уменьшение производительности при изменении данного параметра.

Далее сформируем позиции и размеры области обновления окна:

```
CPoint zp(0, 0);

CPoint dstPt(rect.left, rect.top);

CSize dstSize(rect.Width(), rect.Height());
```

А именно: **Zero Point** – указатель на левый верхний угол окна и его размеры в данный

#### \* Комментарий автора.

Более детально о каждом параметре смотрите MSDN.

момент. Теперь мы готовы обновить окно, вызовем Win API функцию:

```
UpdateLayeredWindow(hdcDst, &dstPt, &dstSize, dc, &zp, NULL,
                    &blend, ULW_ALPHA);
```

В конце, обычная ситуация для каждого из нас (попользовались ресурсами – пришло время освободить их):

```
dc->Detach();

delete dc;

wndDst->ReleaseDC(hdcDst);
```

## Заключение

Я опустил детали создания объектов CDIB и окна. Если интересно, можете попробовать в прилагающихся материалах. Насчет констант **AndroidSize** и **AndroidShadowSize**: это просто размеры иконки исходной и размер тени соответственно. Сначала код кажется громоздким, но в итоге все компактно и процесс формирования нового splashscreen'а или виджета превращается в удовольствие, когда смотришь на уже «живой» результат на экране монитора.

Надеюсь, пример и статья помогут вам понять основы создания Layered окон в Windows. Всего наилучшего и успехов в практике!

Все упомянутые в статье модули приведены в виде ресурсов в теме «Журнал клуба программистов. Седьмой выпуск» или непосредственно в архиве с журналом.





Ветер... У вас тоже на улице ветерок? Возможно, он принесет дождь, по крайней мере, разгонит эту изнурительную жару. Но пока этого не случится, людям стоит спрятаться от палящих лучей солнца в помещения с кондиционером и холодильником, в котором охлаждается квасок или живое пиво. Ну, а пока это счастье свежее от фреонного холода, есть время продолжить разработку своего компилятора...



**Виталий Белик**

by **Stilet** [www.programmersforum.ru](http://www.programmersforum.ru)

### Краткий экскурс

В прошлый раз нам удалось поковыряться во внутренностях Win32 программы. Даже удалось написать простенький «плагиатор», генерирующий экзешник со строкой, которую мы заранее определили. Можно ли остаться на этом этапе написания ядра компилятора и перейти непосредственно к разработке своего языка? Конечно можно: DOS эмуляторы все еще живы, даже если не в виде NTVDM, который, скорее всего, скоро будет исключен из стандартного комплекта Windows, то в виде популярного DosBox, более приспособленного под работу с DOS программами. Одно только маленькое «но», которое вряд ли подсластит пилюлю работы с DOS, – под него нет удобных инструментов отладки (разве что кроме «Иды», которая IDA). Популярный Turbo Debugger вполне мог бы подойти для отладки, и, собственно, это лучшее средство отлаживать DOS программы, которое было на вооружении программистов до появления Windows.

Теперь же мы, избалованные красивым графическим интерфейсом, смотрим на него совершенно другими глазами. Я уверен, что максимум 1 из 100 пользователей Windows с уверенностью и без тени сомнения может отложить мышку в сторону и станет все делать в консоли только клавиатурой. Отвыкли мы от старого стиля работы с вычислительной техникой. Раньше ведь и мышка-то не везде поддерживалась, а сейчас некоторые ОС могут и не запускаться без нее. Мне удалось застать времена, когда в ходу были DOS 5 на процессорах 80286(386), а 40-мегабайтные жесткие диски приходилось сжимать архиватором типа Stacker. Тогда средства отладки Debug, Soft Ice, Turbo Debugger казались пределом совершенства и

удобства (кто еще помнит – пусть сравнит Turbo Pascal 5 и Turbo Pascal 7, в первом мышка не работала, зато второй так жутко тормозил 286-е, что на него не сразу перешли).

Сейчас все эти инструменты ушли на почетную пенсию. И использовать их не то что невыгодно, но даже опасно: операционная система нашего времени может запротестовать, и накроется программа «медным тазиком».

Так что и нам нужно будет сделать еще один шаг вперед\*, чтобы писать программы под еще популярную, но уже уступающую свои позиции платформу Win32.

### Run on the run

Итак, достаточно лирики. Начнем, пожалуй. С чего начать-то? Как обычно, со стратегии и планирования. Многие исполняемые программы под Win32 хранятся в файлах, описанные PE заголовком, в котором содержится информация для загрузчика: как содержимое файла разместить в памяти, что считать кодом, что считать данными, сколько отвести памяти и прочее. В прошлой статье «Оля» (Olly Debugger)

#### \* Комментарий автора.

То, что описано в статье, всего лишь начало. Я не хочу читать книги о компиляторах по одной простой причине: я хочу самостоятельно прийти к тому, к чему (возможно) пришли авторы тех книг, именно поэтому я сделал акцент на том, что было бы, если бы компилятор писал студент, которому совершенно неинтересны всякие умные авторы умных книг. Что получится, если иметь фантазию и уметь искать ответы на свои вопросы. Боже упаси меня конкурировать с кем-то или идти по чьей-то протоптанной дорожке.

В статье (этой и последующих) я хочу показать, что любой, даже тот, кто не захочет читать много скучной теории, сможет без проблем добиться желаемого, идя своим оригинальным путем. Я не ссылаюсь и буду стараться избегать ссылок на кого бы то ни было из ветеранов этой области. Для них есть Википедия и прочие порталы. В конце концов, литературы по этому делу много. Главное – не бояться пробовать. Подумаешь, не по правилам. Подумаешь, чего-то не хватает. Подумаешь, непривычно...

показала нам этого зверя изнутри. Теперь же наша задача - приготовить ядро нашего компилятора, способное формировать эту информацию.

Поверив «Оле», мы разделим ядро на блоки. Каждый блок (класс) будет отвечать за свою

часть формирования файла, а главный класс будет, управляя этими классами, получать скомпилированные части экзешника, которые поступят непосредственно в файл, так сказать, открытым текстом.

Вспомним, из каких же частей\*\* состоит

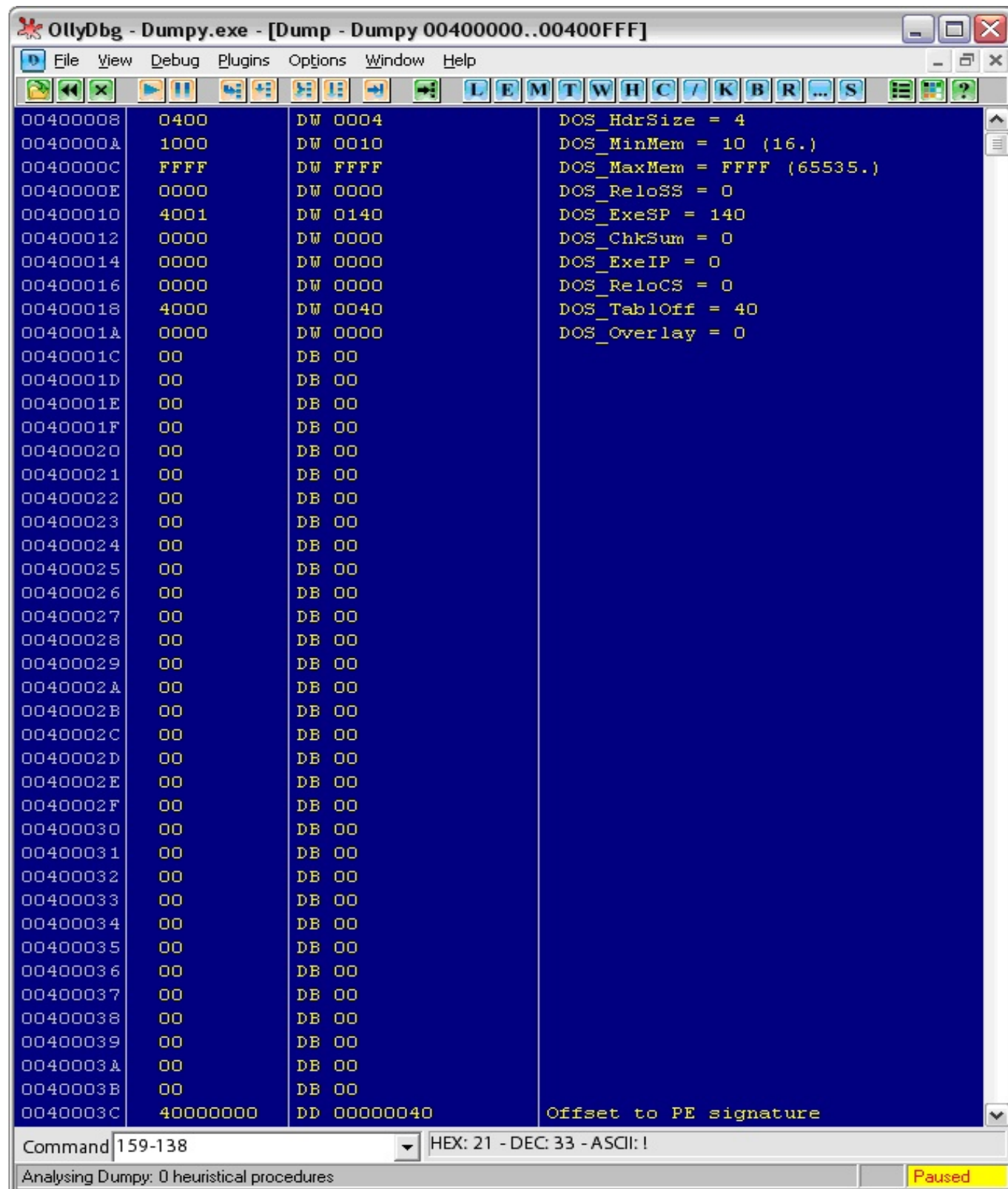


Рис. 1. Начало PE заголовка

**\*\* Комментарий автора.**

Пока этого нам хватит за глаза. Не будем встречать в работу с импортом-экспортом. Потом, когда пойдет работа с вводом-выводом, дополнительно разберемся с Win API функциями. Отсюда вывод: нам понадобится в общем случае 4-5 классов. Давайте оговорим их.

эксешник win32:

- MZ заголовок – 64 байта минимум;
- какой-то обработчик события, если программа не запущена под Windows (ну кому он нужен, а? предлагаю обойтись без него);
- непосредственно PE заголовок, имеющий 246 байт в общем случае;
- массив описания секций. Каждая секция размером 32 байта.

**The Favels**

Пойдем по порядку. Помните, «Оля» показывала нам PE заголовок (см. рисунок 1), вспомнили? Вот так и пойдем описывать класс, отвечающий за MZ часть заголовка:

```
TAlisaMZHeader = class                                     ЛИСТИНГ 1

private                                                    // Класс-описатель MZ заголовка

    DOS_PartPag,      // Размер всего MZ заг-ка до начала PE

    DOS_PageCnt,

    DOS_Relocnt,

    DOS_HdrSize,

    DOS_MinMem,

    DOS_MaxMem,

    DOS_RelocSS,

    DOS_ExeSP,

    DOS_ChkSum,

    DOS_ExeIP,

    DOS_RelocCS,

    DOS_Tabloff,

    DOS_Overlay:word;

    OffsetToPE:DWord;    // Смещение на начало PE заголовка

public

    Constructor Create;

    Function Compile:String; // Компиляция

end;
```

Надеюсь, не забыли\*\*\*, что мы пишем на Delphi? Хотя я и перечислил сюда все поля, можно было

этого и не делать. Но если мы все не учтем это, то впоследствии можем сильно запутаться. Из всего этого нам важны два поля: [DOS\\_PartPag](#) и [OffsetToPE:DWord](#). В принципе, это будут константы, т. е. я имею в виду, что мы забудем в эти поля значения при создании экземпляра класса.

**\*\*\* Комментарий автора.**

Пока не забыл, хочу напомнить, что наш компилятор должен компилировать значения этих полей не так, как мы определим их, а в обратном порядке. Т.е. нам придется вносить их в хранилище компиляции с конца.

Функция Compile будет компилировать эти поля в поток байт. Заранее договоримся, что в качестве хранилища компиляций будут переменные типа String. Поскольку в Delphi этот тип весьма развит (с ним еще с Паскаля было очень удобно обрабатывать массивы байтов), пускай и понимаются они в виде символов. Так что в качестве потока будет String во всей «Алисе» ([TAlisaMZHeader](#)).

Поэтому приготовим пару функций, которые помогут нам переворачивать байты числа, превращая их в строку:

```
// Функция, переворачивающая 2-х байтное слово          ЛИСТИНГ 2

Function WordToStr(w: word):String;

begin

    Result:= chr(lo(w))+chr(hi(w));

end;

// Функция, переворачивающая 4-х байтное значение

Function DWordToStr(w: dword):String;

var l,h: word; s: string[4];

begin

    asm

        mov eax,[w]

        mov [l],ax

        shr eax,16

        mov [h],ax

    end;

    Result:= WordToStr(l)+WordToStr(h);

end;
```

Первая функция сначала записывает младший байт, а после старший, как и должно быть, а



вторая помимо этого еще и разбивает на младшие и старшие части двойное слово. Теперь можем приступить к реализации:

```
constructor TALisaMZHeader.Create;
var i:integer;
begin
    DOS_PartPag := $80;
    DOS_PageCnt := $1;
    DOS_ReloCnt := $0;
    DOS_HdrSize := $4;
    DOS_MinMem := $10;
    DOS_MaxMem := $FFFF;
    DOS_ReloSS := $0;
    DOS_ExeSP := $140;
    DOS_ChkSum := $0;
    DOS_ExeIP := $0;
    DOS_ReloCS := $0;
    DOS_Tabloff := $40;
    DOS_Overlay := $0;
    OffsetToPE := $40;
end;
```

ЛИСТИНГ 3

Это наш конструктор. Здесь мы, как и договаривались, заведем константно значения в поля. Опять-таки, не обязательно было так досконально их описывать, но, дабы не запутаться, это стоило сделать. Следующим будет реализация метода компиляции заголовка:

```
function TALisaMZHeader.Compile: String;
var i:integer;
begin
    Result := 'MZ';
    Result := Result+WordToStr(DOS_PartPag);
    Result := Result+WordToStr(DOS_PageCnt);
    Result := Result+WordToStr(DOS_ReloCnt);
    Result := Result+WordToStr(DOS_HdrSize);
    Result := Result+WordToStr(DOS_MinMem);
    Result := Result+WordToStr(DOS_MaxMem);
    Result := Result+WordToStr(DOS_ReloSS);
    Result := Result+WordToStr(DOS_ExeSP);
    Result := Result+WordToStr(DOS_ChkSum);
    Result := Result+WordToStr(DOS_ExeIP);
    Result := Result+WordToStr(DOS_ReloCS);
    Result := Result+WordToStr(DOS_Tabloff);
    Result := Result+WordToStr(DOS_Overlay);
```

ЛИСТИНГ 4

```
for i:=1 to 32 do Result :=Result+#0;

Result :=Result+WordToStr(OffsetToPE);

for i:=length(Result) to OffsetToPE-1 do Result:=Result+#0;
end;
```

Тут, я думаю, тоже все понятно: каждое поле по порядку мы приводим к строке и выкатываем как поток байт в результат метода. Здесь непонятным моментом могут быть разве что циклы.

for i:=1 to 32 do Result:=Result+#0; дополняет MZ заголовок 32 байтами до поля описания смещения на PE, а for i:=length(Result) to OffsetToPE-1 do Result:=Result+#0; призван заполнить пустоту в том месте, где у порядочных компиляторов должен быть обработчик события запуска под другую операционку. Мы договорились, что его не будет, так что в принципе второй цикл можно было не писать, но он не помешает. Соответственно обратите внимание: я определил OffsetToPE так, чтобы смещение заголовка PE было непосредственно под MZ.

Так, с MZ разобрались. Перейдем к следующему – PE заголовку. Здесь ситуация чуть сложнее. Микрософт постарался и понавыдумывал кучу параметров, в которых легко запутаться, особенно при расчете их значений, так что описание класса будет побольше:

```
TALisaPEHeader=class
private
    FHeader: String;

    //*****
    Machine:word;           // Семейство процессоров
    NumberOfSections:word;  // Кол-во секций, считается
    TimeDateStamp:dword;    // Время компиляции.
    PointerToSymbolTable:dword;
    NumberOfSymbols:dword;
    SizeOfOptionalHeader:word; // Размер опци-го заголовка
    Characteristics:word;    // Тип загружаемого файла
    MagicNumber:word;
    MajorLinkerVersion:byte;
    MinorLinkerVersion:byte;
    SizeOfCode:dword;
```

ЛИСТИНГ 5

```

SizeOfInitializedData:dword;

SizeOfUninitializedData:dword;

AddressOfEntryPoint:dword;    // Адрес точки начала кода

BaseOfCode:dword;

BaseOfData:dword;

ImageBase:dword;              // Базовый адрес

SectionAlignment:dword;      // Размер секций в памяти

FileAlignment:dword;         // Размер секций в файле

MajorOSVersion:word;

MinorOSVersion:word;

MajorImageVersion:word;

MinorImageVersion:word;

MajorSubsystemVersion:word;

MinorSubsystemVersion:word;

Reserved1:dword;

{ Размер всего образа, размещаемого в памяти
  Получается, что это размер=PE+кол-во секций*Размер
  секций в памяти, учитывая, что PE тоже выравнивается до
  размера секций в памяти. Он считается такой же
  равноправной секцией, но служит для описания всех
  остальных секций, такой себе Interface
}

SizeOfImage:dword;

SizeOfHeaders:dword;    // Общий размер всех заголовков

{ Контрольная сумма. Вообще ходят неоднозначные слухи о
  ее надобности. Некоторые HEX редакторы в ней нуждаются,
  но для запуска скомпилированной программы ее никто не
  проверяет. Мы не будем ее рассчитывать, но в принципе
  на будущее необходимо знать, что это за поле
}

Checksum:dword;

Subsystem:word;          // Тип приложения,
                          // консолька, оконка или DLL

DLLCharacteristics:word;

SizeOfStackReserve:dword; // память, требуемая для стека
                          // объем памяти, отводимой в
                          // стеке немедленно после
                          // загрузки

SizeOfStackCommit:dword;

SizeOfHeapReserve:dword;  // макс.возможный размер кучи

SizeOfHeapCommit:dword;

LoaderFlags:dword;

NumberOfRvaAndSizes:dword;

Export_Table_address:dword;

Export_Table_size:dword;

```

```

Import_Table_address,

Import_Table_size,

Resource_Table_address,

Resource_Table_size,

Exception_Table_address,

Exception_Table_size,

Certificate_File_pointer,

Certificate_Table_size,

Relocation_Table_address,

Relocation_Table_size,

Debug_Data_address,

Debug_Data_size,

Architecture_Data_address,

Architecture_Data_size,

Global_Ptr_address,

Must_be_0,

TLS_Table_address,

TLS_Table_size,

Load_Config_Table_address,

Load_Config_Table_size,

Bound_Import_Table_address,

Bound_Import_Table_size,

Import_Address_Table_address,

Import_Address_Table_size,

Delay_Import_Descriptor_address,

Delay_Import_Descriptor_size,

COM_Runtime_Header_address,

Import_Address_Table_size2,

Reserved2,

Reserved3:d word;

//*****

public

Function Header:String;

Constructor Create;

end;

```

Из всей этой «лесопосадки» нам понадобятся только несколько полей (я к ним в листинге 5 комментария приписал). Все остальные, поверив «Оле», проинициализируем константными значениями.

Здесь же в классе присутствует функция `Header`, которая сформирует выходную строку, содержащую скомпилированный PE заголовок. Посмотрим на код, в котором будет описана

инициализация этих полей:

```
constructor TALisaPEHeader.Create;                                ЛИСТИНГ 6
begin
    /*******
    Machine := $014C; // IMAGE_FILE_MACHINE_I386
    //NumberOfSections := $2;
    TimeDateStamp := $0;
    PointerToSymbolTable := $0;
    NumberOfSymbols := $0;
    SizeOfOptionalHeader := $E0;
    Characteristics := $010E;
    MagicNumber := $010B;
    MajorLinkerVersion := $0;
    MinorLinkerVersion := $0;
    SizeOfCode := $0;
    SizeOfInitializedData := $0;
    SizeOfUninitializedData := $0;
    //AddressOfEntryPoint := $0;
    BaseOfCode := $0;
    BaseOfData := $0;
    ImageBase := $400000;
    //SectionAlignment := $0;
    //FileAlignment := $200
    MajorOSVersion := $1;
    MinorOSVersion := $0;
    MajorImageVersion := $0;
    MinorImageVersion := $0;
    MajorSubsystemVersion := $3;
    MinorSubsystemVersion := $A;
    Reserved1 := $0;
    //SizeOfImage := $3000;
    SizeOfHeaders := $200;
    //Checksum := $ 7FAB
    Subsystem := $3; // IMAGE_SUBSYSTEM_WINDOWS_CUI
    DLLCharacteristics := $0;
    SizeOfStackReserve := $1000;
    SizeOfStackCommit := $1000;
    SizeOfHeapReserve := $10000;
    SizeOfHeapCommit := $0;
    LoaderFlags := $0;
    NumberOfRvaAndSizes := $10;

    Export_Table_address := $0;
    Export_Table_size := $0;
    Import_Table_address := $0;
```

```
    Import_Table_size := $0;
    Resource_Table_address := $0;
    Resource_Table_size := $0;
    Exception_Table_address := $0;
    Exception_Table_size := $0;
    Certificate_File_pointer := $0;
    Certificate_Table_size := $0;
    Relocation_Table_address := $0;
    Relocation_Table_size := $0;
    Debug_Data_address := $0;
    Debug_Data_size := $0;
    Architecture_Data_address := $0;
    Architecture_Data_size := $0;
    Global_Ptr_address := $0;
    Must_be_0 := $0;
    TLS_Table_address := $0;
    TLS_Table_size := $0;
    Load_Config_Table_address := $0;
    Load_Config_Table_size := $0;
    Bound_Import_Table_address := $0;
    Bound_Import_Table_size := $0;
    Import_Address_Table_address := $0;
    Import_Address_Table_size := $0;
    Delay_Import_Descriptor_address := $0;
    Delay_Import_Descriptor_size := $0;
    COM_Runtime_Header_address := $0;
    Import_Address_Table_size := $0;
    Reserved2 := $0;
    Reserved3 := $0;

    /*******
end;
```

Обратите внимание, я закомментировал те поля, значения которых придется рассчитывать. И не забудем описать реализацию компиляции этого класса в выходную строку:

```
function TALisaPEHeader.Header: String;                            ЛИСТИНГ 7
begin
    Result := 'PE' #0 #0;
    Result := Result + WordToStr(Machine);
    Result := Result + WordToStr(NumberOfSections);
    Result := Result + DWordToStr(TimeDateStamp);
    Result := Result + DWordToStr(PointerToSymbolTable);
    Result := Result + DWordToStr(NumberOfSymbols);
    Result := Result + WordToStr(SizeOfOptionalHeader);
```

```

Result :=Result+ WordToStr(Characteristics);

Result :=Result+ WordToStr(MagicNumber);

Result :=Result+ chr(MajorLinkerVersion);
Result :=Result+ chr(MinorLinkerVersion);

Result :=Result+ DWordToStr(SizeOfCode);

Result :=Result+ DWordToStr(SizeOfInitializedData);
Result :=Result+ DWordToStr(SizeOfUninitializedData);
Result :=Result+ DWordToStr(AddressOfEntryPoint);
Result :=Result+ DWordToStr(BaseOfCode);
Result :=Result+ DWordToStr(BaseOfData);


Result :=Result+ DWordToStr(ImageBase);
Result :=Result+ DWordToStr(SectionAlignment);
Result :=Result+ DWordToStr(FileAlignment);


Result :=Result+ WordToStr(MajorOSVersion);
Result :=Result+ WordToStr(MinorOSVersion);
Result :=Result+ WordToStr(MajorImageVersion);
Result :=Result+ WordToStr(MinorImageVersion);
Result :=Result+ WordToStr(MajorSubsystemVersion);
Result :=Result+ WordToStr(MinorSubsystemVersion);


Result :=Result+ DWordToStr(Reserved1);
Result :=Result+ DWordToStr(SizeOfImage);
Result :=Result+ DWordToStr(SizeOfHeaders);
Result :=Result+ DWordToStr(CheckSum);


Result :=Result+ WordToStr(Subsystem);
Result :=Result+ WordToStr(DLLCharacteristics);


Result :=Result+ DWordToStr(SizeOfStackReserve);
Result :=Result+ DWordToStr(SizeOfStackCommit);
Result :=Result+ DWordToStr(SizeOfHeapReserve);
Result :=Result+ DWordToStr(SizeOfHeapCommit);
Result :=Result+ DWordToStr(LoaderFlags);
Result :=Result+ DWordToStr(NumberOfRvaAndSizes);
Result :=Result+ DWordToStr(Export_Table_address);
Result :=Result+ DWordToStr(Export_Table_size);
Result :=Result+ DWordToStr(Import_Table_address);
Result :=Result+ DWordToStr(Import_Table_size);
Result :=Result+ DWordToStr(Resource_Table_address);
Result :=Result+ DWordToStr(Resource_Table_size);
Result :=Result+ DWordToStr(Exception_Table_address);
Result :=Result+ DWordToStr(Exception_Table_size);
Result :=Result+ DWordToStr(Certificate_File_pointer);

```

```

Result :=Result+ DWordToStr(Certificate_Table_size);
Result :=Result+ DWordToStr(Relocation_Table_address);
Result :=Result+ DWordToStr(Relocation_Table_size);
Result :=Result+ DWordToStr(Debug_Data_address);
Result :=Result+ DWordToStr(Debug_Data_size);
Result :=Result+ DWordToStr(Architecture_Data_address);
Result :=Result+ DWordToStr(Architecture_Data_size);
Result :=Result+ DWordToStr(Global_Ptr_address);
Result :=Result+ DWordToStr(Must_be_0);
Result :=Result+ DWordToStr(TLS_Table_address);
Result :=Result+ DWordToStr(TLS_Table_size);
Result :=Result+ DWordToStr(Load_Config_Table_address);
Result :=Result+ DWordToStr(Load_Config_Table_size);
Result :=Result+ DWordToStr(Bound_Import_Table_address);
Result :=Result+ DWordToStr(Bound_Import_Table_size);
Result :=Result+ DWordToStr(Import_Address_Table_address);
Result :=Result+ DWordToStr(Import_Address_Table_size);
Result :=Result+ DWordToStr(Delay_Import_Descriptor_address);
Result :=Result+ DWordToStr(Delay_Import_Descriptor_size);
Result :=Result+ DWordToStr(COM_Runtime_Header_address);
Result :=Result+ DWordToStr(Import_Address_Table_size2);
Result :=Result+ DWordToStr(Reserved2);
Result :=Result+ DWordToStr(Reserved3);

end;

```

Здоровый код, не так ли? Мне лично неприятно иметь дело с такой щепетильностью. Будь программизм Микрософта продуманнее, они могли бы все эти поля сделать одного размера, тогда и нам было бы попроще: работа с массивом однородных элементов все-таки не требует большой писанины кода. Ну, да ладно. Скажу по секрету: это описание будет единственным огромным, далее рутинного кода будет не так много.

Еще нам понадобится класс, описывающий секции и управляющий ими. То есть нам нужно сосредоточить в нем не только описание секций, но и саму ее реализацию. Описание будет в виде списка полей плюс поле содержимого секции, которое будет представлено в виде строки:

```

TAlisaSection=class // Класс, описывающий секции    ЛИСТИНГ 8
private
    Name:string[8];           // (0h) Имя секции
    { (8h) Размер выделяемой под секцию памяти. Фактически это

```



```

размер данных, которые компилируются. Если речь идет о
коде, то это размер чистого кода в байтах без выравнивания
до размера секции
}
VirtualSize,          // (0Ch) Адрес, с которого начнется
                      // секция относительно базы

VirtualAddress,

SizeOfRawData,        // (10h) Размер секции в файле.
                      // (14h) Смещение в файле, с которого
                      // начинается секция

PointerToRawData,

PointerToRelocations,

PointerToLineNumbers:dword;

NumberOfRelocations,

NumberOfLineNumbers:word;

Characteristics:dword; // Доступы секции

public

_Data:String;          // Это поле для чистых данных или кода
                      // Функция будет возвращать
                      // скомпилированное описание заголовка
                      // секции

Function Header:String;

end;

```

Здесь, как видим, все попроще. Тоже описываются все поля, чтобы не запутаться, но только некоторые для нас будут важны – те, которым сопутствуют комментарии. Напомню их:

- Поле имени. Оно важно только для программиста, чтобы удобно было различать и группировать содержимое программы. Фактически нам понадобится только две-три секции: главная секция кода (для точности она будет идти всегда первой) и одна секция данных. Вполне возможно, что потом понадобится выделить еще одну секцию для описания функций, но время покажет: если функции небольшие, их вполне можно будет оставить вместе с главным кодом, не выделяя лишней секции (вообще можно натолкнуть все в одну, но выглядеть это будет некрасиво: неудобно для анализа).
- **VirtualSize** – размер выделяемой под секцию памяти. Получается, что это размер чистого кода, столько байт, сколько накопировано в опкодах или (если речь идет о секции данных) столько инициализированных байт,

сколько мы впишем. Если, предположим, программа будет состоять из двух операторов `Mov Eax, некое значение`, то это поле будет содержать число десять. Потому что размер этой операции – 5 байт (учитывая, что оперируем переменной `DWord`) \* 2, ибо их два. А поскольку у нас все содержимое секции будет храниться в виде строки, высчитать это поле труда не составит – функцию `Length` в Delphi еще никто не отменял.

- **VirtualAddress**. Адрес, с которого будет начинаться секция. Первая наша секция будет начинаться с `1000h` – это будет секция кода, следующие секции будут соответственно `2000h`, `3000h` и т. д. Это не займет много места: как показывает практика, для двух секций (код и данные) хватит 2 кбайт.

Так вот давайте посмотрим на свойства, которые нам покажет Windows на рисунке 1. Ничего не смущает? Откуда там аж 4 кбайта, если мы заказывали всего две секции+заголовок? Это что за особенности NTFS? Кстати, нужно взять на заметку, что секцию можно описать таким образом, что ее реализация отсутствует в файле. Это так называемая секция неинициализированных данных. Вот смотрите, на рисунке 3 наглядно показано в шаблоне для MASM из пакета MASM Builder. Видите секцию «.data»? Тело этой секции в файл не компилируется, но ее описание в PE заголовке присутствует, говоря загрузчику: «Эй, приятель, зарезервируй мне в памяти секцию. Не ищи ее в файле, она нужна будет только для работы и не содержит начальных значений». Загрузчик послушно кивает и резервирует в памяти местечко для пикника на обочине, не выискивая в файле, чем бы эту полянку наполнить. И даже в этом случае компилятор инициализирует **VirtualSize** для

#### \*\*\*\* Комментарий автора.

Кстати, что интересно. Я до сегодняшнего дня не сталкивался с программами менее мегабайта, редко писал на Ассемблере и еще реже обращал внимание на файлы, которые потом компилировались. Теперь же, когда начал вникать в то, как работают компиляторы, обнаружил интересный и незнакомый мне эффект. Помните, в предыдущей статье мы компилировали в Ассемблере программу-пустышку, чтобы принести ее в жертву во имя Ее Величества Науки?

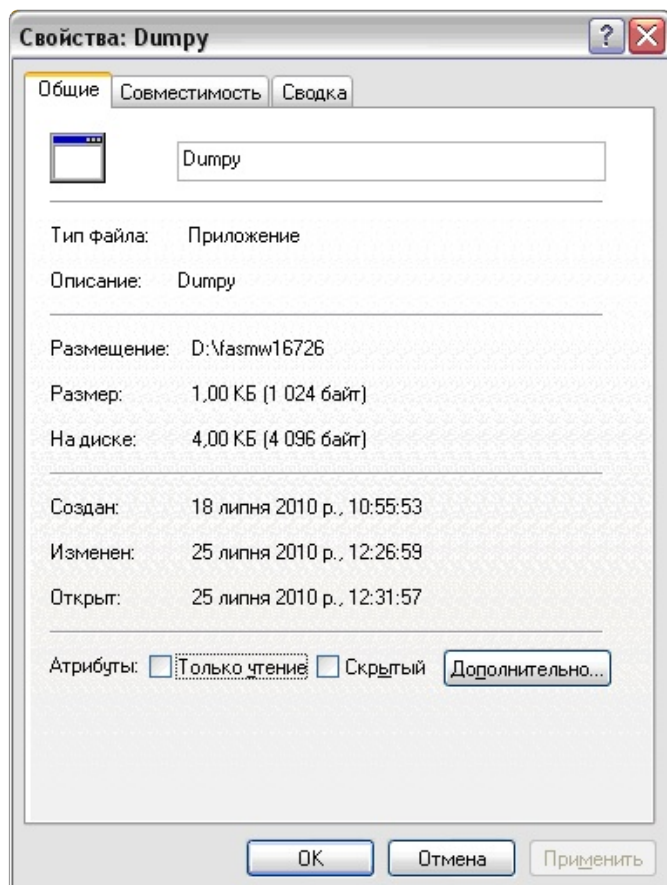


Рис. 2. Свойства файла-пустышки

неинициализированных секций. Ну, компилятору Ассемблера хорошо, он заранее знает, какого размера будут переменные, даже не инициализированные. Мы условимся следующим образом: пусть наш компилятор сам определяет, инициализированная секция или нет. Если в ней все ячейки будут пустыми, то резервировать в файле для нее место не надо. Поскольку реализация секции также будет представлять строку, проверка будет проста: если строка пуста, то и формировать тело секции в файл не стоит, но раз уж мы ее заказали – загрузчик должен зарезервировать место в памяти. Потом, экспериментируя, можно даже будет вообще обойтись одной-единственной секцией – кода, куда можно впихнуть и данные, но тогда придется менять поле доступа (*characteristics*) для этой секции.

- **SizeOfRawData.** Поле, говорящее, сколько байт в файле займет секция (при неинициализированной секции это значение равно 0).
- **PointerToRawData.** Поле, указывающее смещение, по которому находится начало

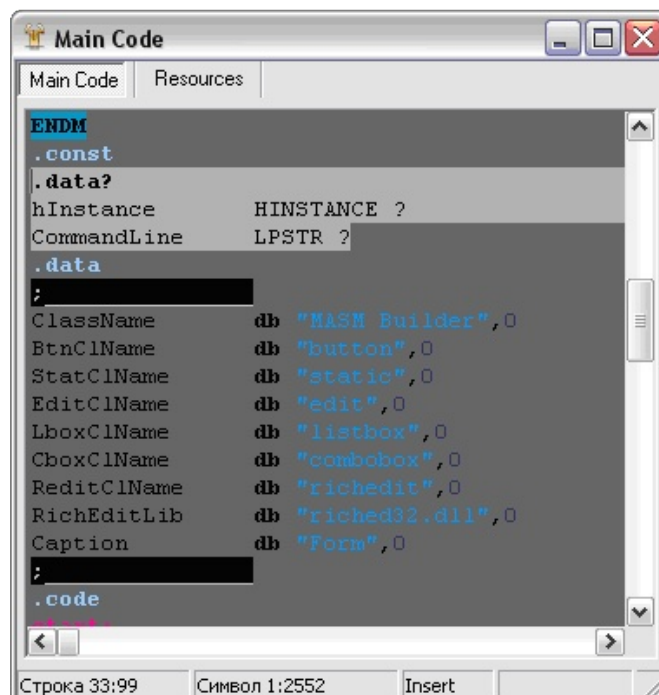


Рис. 3. Пример секции неинициализированных данных

секции. Фактически оно считается по формуле  $\text{fileAlignment} * (\text{Номер секции} + 1) + 1$ , потому что PE заголовок мы тоже считаем секцией. Для неинициализированной секции имеет значение 0.

- **Characteristics.** Это поле доступов. Указывает, что можно делать с этой секцией: только читать, писать и читать – или только выполнять. Значения мы подсмотрим в «Оле» и установим их стандартно. Пока что нам хватит только двух значений: чтения и выполнения (для секции кода – \$60000020) и чтения и записи (для секции данных – \$C0000040).
- Здесь же опишем поле **\_Data**, которое будет хранить собственно тело секции. Для секции кода здесь будут опкоды команд, для секций данных – данные. Компилировать описание секции будет функция «Header».

Ничего не забыли? Вроде нет. В таком случае перейдем к реализации методов этого класса. А собственно, у нас только одна функция, ее и реализуем:

```
function TALisaSection.Header: String;
begin
    Result:='';
```

ЛИСТИНГ 9

```
// Если тело ее пусто, будем считать, что это
// неинициализированная секция, и она не нуждается в
// компиляции в файл.
if _Data='' then begin
    SizeOfRawData:=0;
    PointerToRawData:=0;
    VirtualSize:=2;
end;
VirtualSize:=length(_Data);
if VirtualSize=0 then VirtualSize:=1;

// В остальном действия такие же – переворачиваем значение и
// конвертируем в строку
Result :=Result+ Name;
Result :=Result+ DWordToStr(VirtualSize);
Result :=Result+ DWordToStr(VirtualAddress);
Result :=Result+ DWordToStr(SizeOfRawData);
Result :=Result+ DWordToStr(PointerToRawData);
Result :=Result+ DWordToStr(PointerToRelocations);
Result :=Result+ DWordToStr(PointerToLineNumbers);
Result :=Result+ WordToStr(NumberOfRelocations);
Result :=Result+ WordToStr(NumberOfLineNumbers);
Result :=Result+ DWordToStr(Characteristics);
end;
```

Видите, все просто. Банально поле за полем выдаем в строку поток скомпилированных данных по порядку, не забывая переворачивать положение байт в поле.

## Master of puppets

У нас есть заготовленные детали механизма. Простые шестеренки, неплохо бы собрать из них редуктор. Именно этим и займемся – напомним главный класс, который будет компилировать, используя вышеописанные классы-шестеренки и формировать выходной файл. Определимся, что туда должно входить:

1. Поля классов для PE, MZ и массив секций.
2. Функции компиляции описаний всех этих секций и тел секций.
3. Метод создания секций.
4. Функции расчета размера скомпилированного заголовка, расчета выравнивания секции в файле. Хотя мы и не будем писать огромные программы, но неплохо бы подумать на

перспективу о расчете размера выравниваний согласно содержимому секции. А вдруг данных в секции больше, чем 512 байт (это минимальное значение для инициализированной секции или PE заголовка)? Или количество секций, например, пять, а их описание раздувает заголовок в размер поболее 512 ( $372+40*5 = 572$ . Здесь 372 – минимальный размер PE заголовка, 40 – размер описания секций, 5 – их количество)? Значит, придется резервировать для секций уже не 200h (512 в десятиричной).

Определились? Приступим. Смотрим описание класса:

ЛИСТИНГ 10

```
TAlisaCompiler=class
Private
    // Наши компиляторы заголовков
    PE:TAlisaPEHeader;
    MZ:TAlisaMZHeader;
    Sections:TObjectList;
    // Это переменная-коллектор скомпилированного файла
    _Data:String;

    // Функции, компилирующие описания заголовков, и их реализации
    Function CompilePE:string;
    Function CompileSectionsHeader(i:integer):string;

    // Вспомогательные функции
    // Функция вычисления, сколько нужно для стандартного PE
    // заголовка без обработчика запуска программы не из-под
    // Windows
    Function LengthHeader:Integer;

    // Функция расчета выравнивания в файле согласно максимальному
    // размеру тел секций
    Function CalcFileAlignment:Integer;

    // Функция, проставляющая для каждой секции рассчитанное
    // выравнивание секций в файле
    procedure FixAlignSectionInFile;

public
    // По умолчанию мы обязательно будем иметь как минимум одну
    // секцию – секцию кода. Она же будет в списке первой и
    // начинаться будет по смещению 1000h
    _Code:TAlisaSection;

    // Функция, создающая новую секцию.
    Function NewSection(AName:string;Type_:Dword):TAlisaSection;

    // функция общей компиляции в поле _Data. Его содержимое и
```

```
// будет поступать в файл как есть.

Function Compile:String;

Constructor Create;

Destructor Free;

end;
```

Начнем по порядку реализовывать каждое звено цепочки. Первым пойдет конструктор. Здесь мы проинициализируем основные поля заголовка и создадим секцию кода:

```
constructor TALisaCompiler.Create; ЛИСТИНГ 11
begin
    MZ:=TALisaMZHeader.Create;
    pe:=TALisaPEHeader.Create;
    //*****
    pe.AddressOfEntryPoint:=$1000;
    pe.ImageBase:=$400000;
    pe.SectionAlignment:=$1000;

    { TODO -oUser -cConsole Main : С этим параметром аккуратнее }
    pe.FileAlignment:=$200;

    pe.SizeOfHeaders:=pe.FileAlignment;
    pe.Subsystem:=3;
    //*****
    Sections:=TObjectList.Create;
    _Code:=NewSection('Код',codeSec);
end;
```

Это стандартные настройки для основных параметров нашего компилятора, позаимствованные из компиляции FASM. Их описания я давал выше. Конструкторы MZ и PE мы уже описали и реализовали выше, поэтому перейдем к реализации метода создания секции:

```
function TALisaCompiler.NewSection; ЛИСТИНГ 12
begin
    // создадим секцию и внесем ее в список секций
    Result:=TALisaSection.Create;
    Sections.Add(Result);
    // Назовем ее и обязательно дополним до 8 байт
    Result.Name:=AName;
    while length(Result.Name)<8 do Result.Name:=Result.Name+#0;
    // Определим параметры секции. PE заголовок хоть и
    // считается секцией, но в список их не заносится. Здесь
```

```
// находятся только настоящие секции

Result.Characteristics:=Type_;

Result.VirtualAddress:=PE.SectionAlignment*Sections.Count;

Result.VirtualSize:=1;

Result.SizeOfRawData:=PE.FileAlignment;

end;
```

Теперь посмотрим\*\*\*\*\* реализацию метода компиляции, чтобы располагать последовательностью вызовов методов:

```
function TALisaCompiler.Compile: String; ЛИСТИНГ 13
var i,csp:integer; HeaderSum,Checksum:dword;s,e:string;
begin
    // Посчитаем количество секций
    PE.NumberOfSections:=Sections.Count;
    // и выясним общий размер заголовка с описаниями секций
    pe.SizeOfImage:=(PE.NumberOfSections+1)*pe.SectionAlignment;
    // Проведем расчет выравнивания секций в файле
    pe.FileAlignment:=CalcFileAlignment;
    // Поправим на основе расчета поля секций
    FixAlignSectionInFile;
    // Скомпилируем заголовки
    _Data:=CompilePE;

    // Теперь присоединим тела секций
    for i:=0 to Sections.Count-1 do
        _Data:=_Data+CompileSection(i);

    // Теперь нужно посчитать контрольную сумму
    //ChecksumMappedFile(@s[1],length(s),@HeaderSum,@Checksum);

    Result:=_Data;
end;
```

В листинге 4 приведена функция расчета длины заголовка в байтах. Помните, мы условились, что не будем использовать обработчик события запуска под не Windows систему? Поэтому

#### \*\*\*\*\* Комментарий автора.

Обратите внимание, я на всякий случай привел реализацию расчета контрольной суммы.. Вдруг потребуется. Но на данный момент функция расчета контрольной суммы закомментирована. Как я уже говорил, Windows на нее плюет с высокой колокольни. Для работы этого метода нужно реализовать шесть функций: LengthHeader, CalcFileAlignment, FixAlignSectionInFile, CompilePE, CompileSectionsHeader и CompileSection. Они не все вызываются в нем, но мы и вложения перечислим. Сделаем это ниже, в листингах 14-19.



```
function TALisaCompiler.LengthHeader: Integer;    ЛИСТИНГ 14
begin
    // Это длина MZ+PE без описания секций
    Result:=MZ.OffsetToPE+248;

    //Эта общая чистая длина PE с секциями
    Result:=Result+Sections.Count*40;

end;
```

можем смело говорить, что `OffsetToPE` является крайней точкой MZ заголовка, за которым, не теряя ни одного байта, начнется PE заголовок. Кто-то скажет: «А зачем нужно было избавляться от этого обработчика, если все равно выравнивать до `FileAlignment` придется? В чем тут потери?» Верно, можно было и не отказываться от него, но, во-первых, лень писать еще и код этого обработчика, а во-вторых, при анализе так

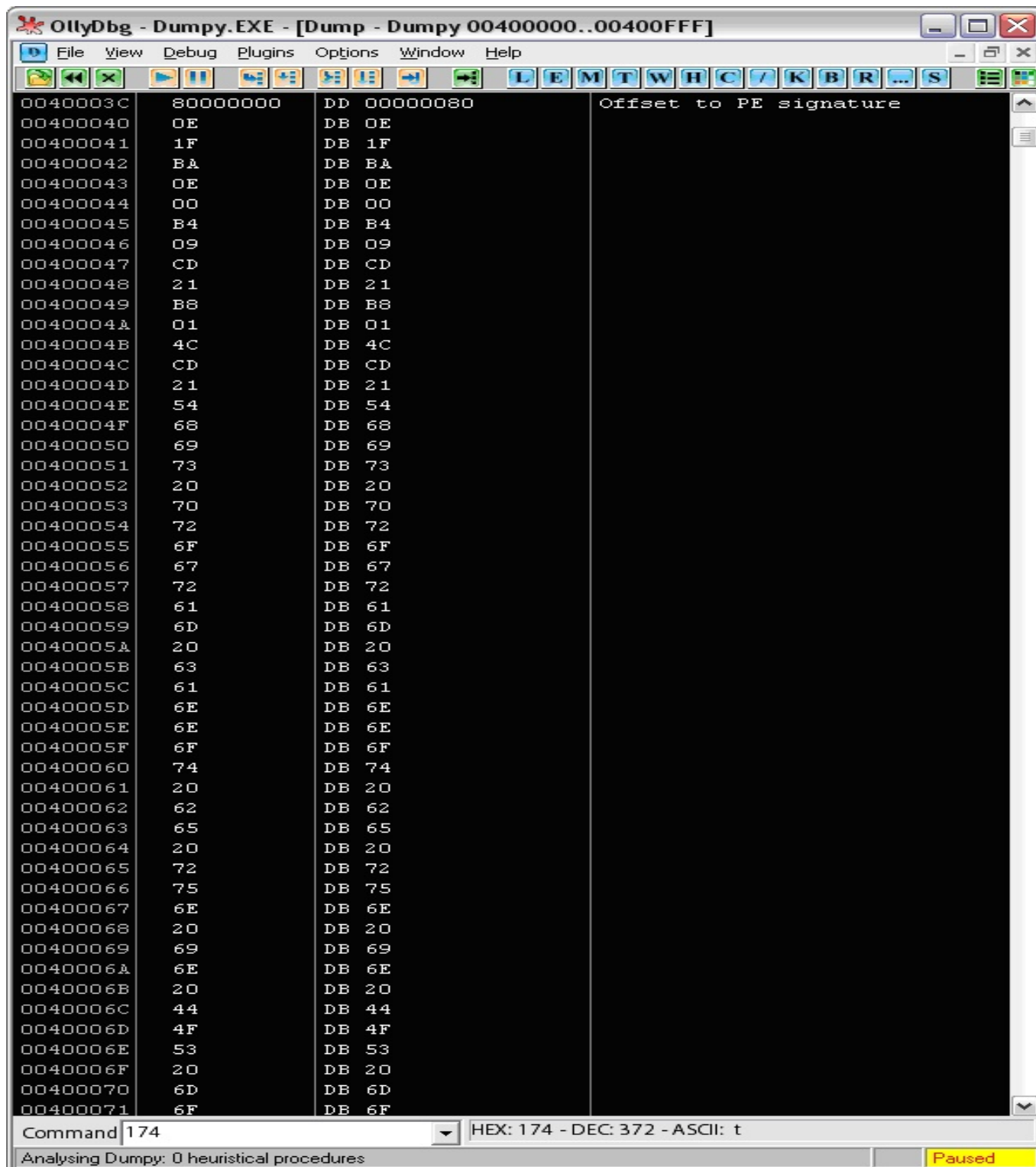


Рис. 4. Вид с мусором обработчика

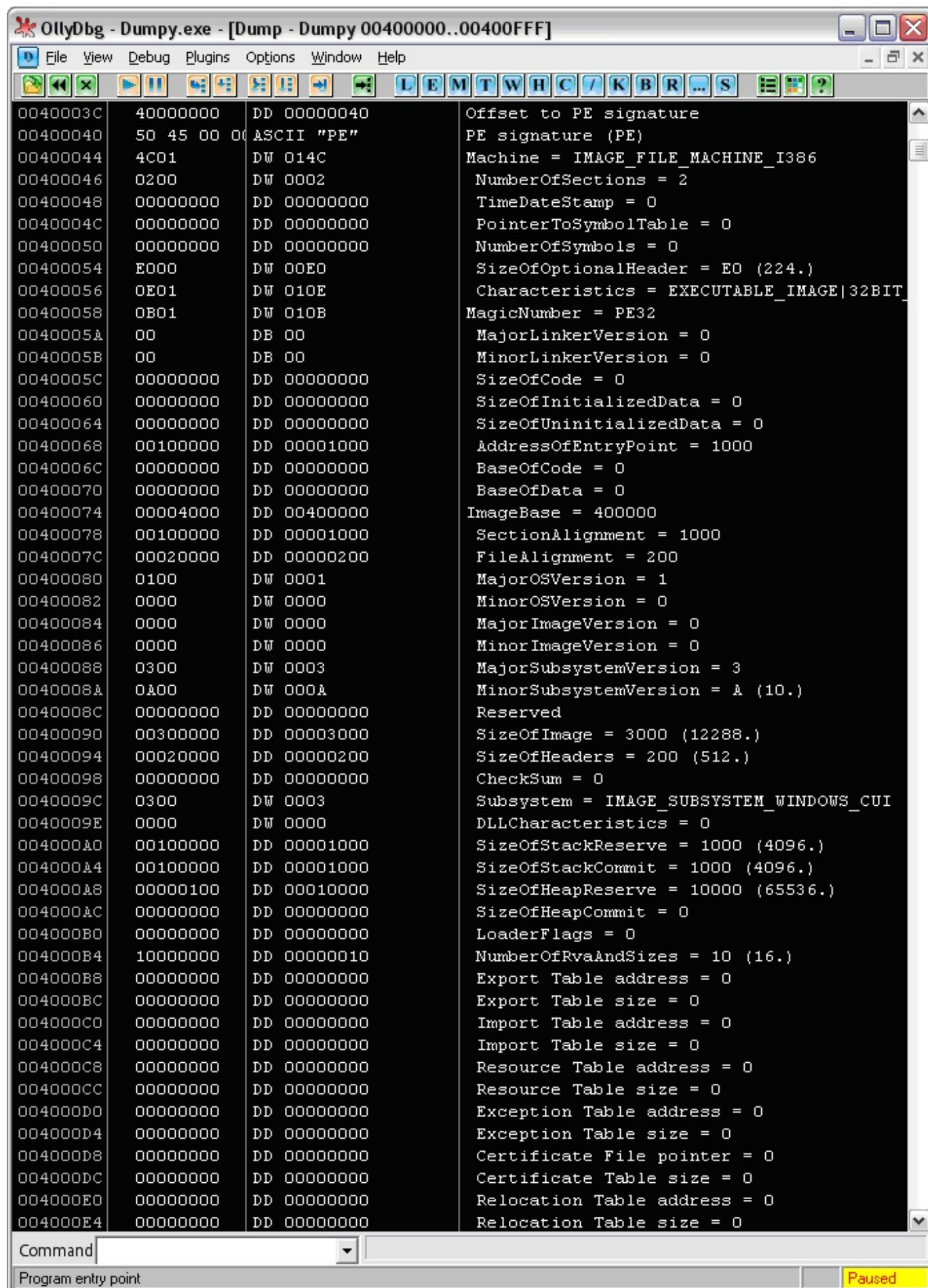


Рис. 5. Вид без мусора обработчика

быстрее можно найти части заголовка. Давайте сравним вид «Оли» с обработчиком и без него, начиная с точки смещения на PE часть. Видите? На рисунках 4 и 5 разница очевидна. На 2-м мы сразу добрались до PE, а на 1-м его еще не видать, хотя на размер PE экзешника это не повлияло. Ладно, идем далее. Рассмотрим расчет выравнивания тел секций в файле:

```
function TALisaCompiler.CalcFileAlignment: Integer;
var ls,i,l: integer;
begin
    // Получаем размер PE
    l:= LengthHeader;

    // Если этот размер не делится нацело на 512, значит, его
    // лучше выравнивать считая, что появилась еще какая-то секция
    if (l mod $200) <> 0 then l:=(l div $200+1)*$200;

    // И соответственно точно так же проанализировать тела
    // секций. Вдруг какая-то из них больше минималки, тогда
    // равнять будем по ней
    for i:=0 to Sections.Count-1 do begin
        ls:=length(TALisaSection(Sections[i])._Data);
        if ls>l then
            if (ls mod $200) <> 0 then l:=(ls div $200+1)*$200;
    end;
    Result:= l;
end;
```

И проставление полей в секциях согласно расчету:

```
procedure TALisaCompiler.FixAlignSectionInFile;
var i:integer;
begin
    for i:=0 to Sections.Count-1 do
        if TALisaSection(Sections[i]).SizeOfRawData <> 0 then
            TALisaSection(Sections[i]).PointerToRawData:=
                pe.FileAlignment*(i+1)
        else
            TALisaSection(Sections[i]).PointerToRawData:=0;
    end;
```

Следующий метод призван компилировать PE заголовок (все поля должны быть рассчитаны корректно до вызова этой функции):

```
function TALisaCompiler.CompilePE: string;
var i:integer;
begin
    // Компилируем PE заголовок
    Result:= MZ.Compile+PE.Header;

    // Компилируем описания секций
    for i:=0 to Sections.Count-1 do begin
        Result:=Result+TALisaSection3(Sections[i]).Header;
    end;

    // выравниваем скомпилированное по минимальному размеру
    // тела секции в файле.
    while length(result)<pe.FileAlignment do Result:=Result+#0;
end;
```

Здесь всплыла функция Header (метод класса TALisaPEHeader), описанный выше. Идем далее. Компиляция тел секций:

```
function TALisaCompiler.CompileSection(i: integer): String;
begin
    Result:='';
    if (i>=0) and (i<Sections.Count) then
        with TALisaSection(Sections[i]) do begin
            Result:=_Data;
            if Result <> '' then
                while length(result)<pe.FileAlignment do
                    Result:=Result+#0;
        end;
    end;
```

Здесь тела секций выравниваются до **FileAlignment**, дополняясь нулями. Вот оно – ядро компилятора, формирующее интерьер файла-экзешника. Далее содержимое поля **\_Data** пойдет непосредственно в файл. А что – попробуем? Поместим эти классы в отдельный модуль, назовем <Alisa.pas>. На всякий случай напомним, что описание классов вешается в раздел **interface**, а реализация – в раздел **implementation**. Также в модуле понадобятся использование модулей **types** и **contnrs**. Я не описал деструктор **TALisaCompiler.Free**, но он не так важен. В нем просто нужно освобождать задействованные в классе объекты – это вы можете написать сами.

Создадим также консольный проект, где напомним собственно программу, вызывающую

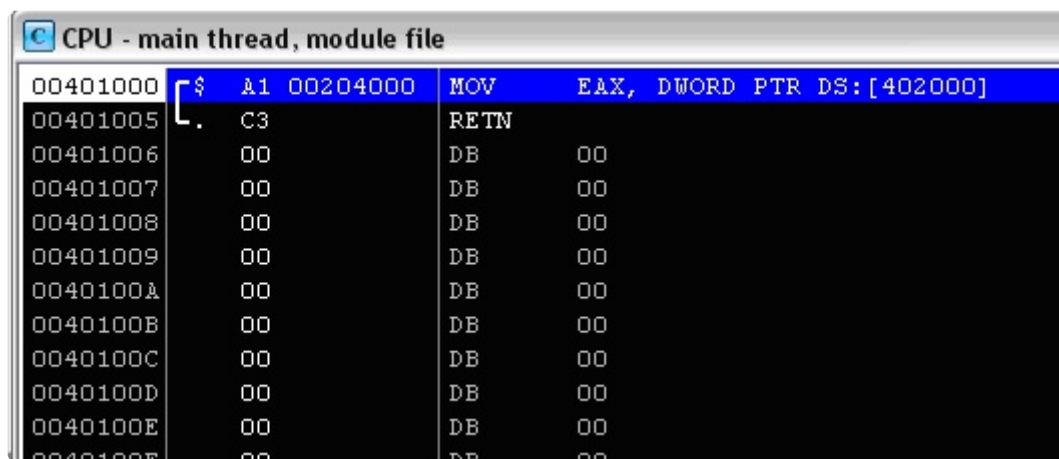


Рис. 6. Вид реализации задания

компилятор и передающую ему опкоды команд. Пока что придется заглядывать в книгу и help по Ассемблеру, чтобы определить, какие опкоды нам нужны. Допустим, задача «поместить значение переменной в регистр» выглядит в реализации так (см. рисунок 6). Видите опкод? **A100204000** в 16-тиричном виде. Вот так мы его и опишем в строку (обратите внимание, «Оля»

переменной? Не будем же мы заставлять в коде указывать адрес?» Нет, конечно, но проверить нам наше творчество как-то надо, а мы помним, что у нас секция данных идет после кода и размер секций в памяти 1000h, плюс база 400000h, вот и получается, что 400000h+размер секции кода дадут 402000h. А поскольку у нас только одна переменная (для проверки хватит),

показывает нам, как нужно правильно написать опкод в уже перевернутом виде: сначала младшие, потом старшие байты), не забыв про retn (C3h) в листинге 19.

Кстати, кто-то спросит: «Откуда мы знаем адрес

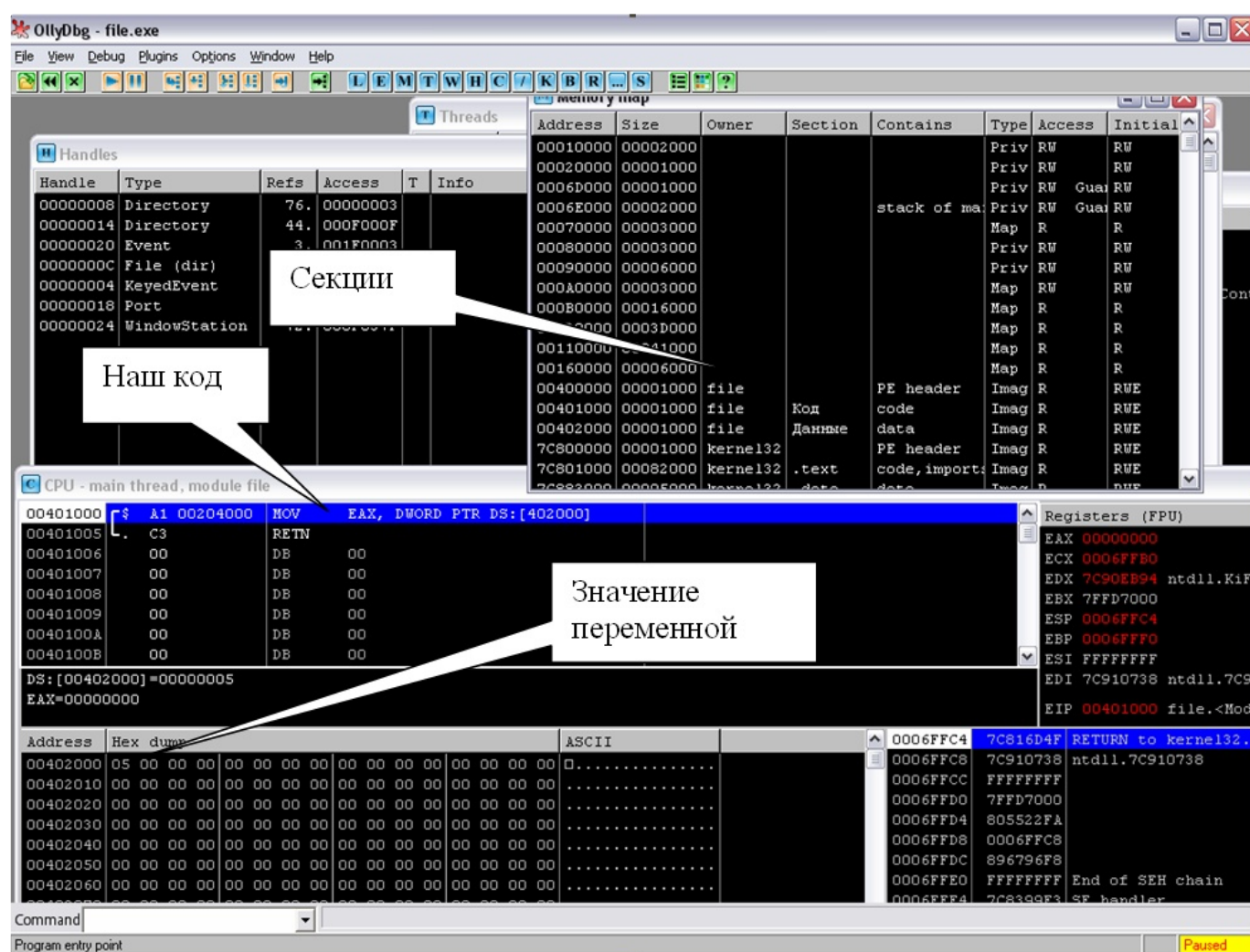


Рис. 7. Результат компиляции



```

program Project1;                                     ЛИСТИНГ 19

{$APPTYPE CONSOLE}

uses

  SysUtils,
  Unit1 in 'Unit1.pas';

var a: TALisaCompiler;
    d: TALisaSection;
    f: file of byte; s: string;

begin
  // Зарядим компилятор
  a:=TALisaCompiler.Create;

  // Создадим секцию данных
  d:=a.NewSection('Данные',dataSec);

```

```

// Положим в секцию данных число 5
d._Data:=#5;

// положим в секцию кода опкод команды, помещающей в регистр
// EAX число из секции данных и выход из программы
a._Code._Data:=$A1#$00#$20#$40#$00#$C3;

// Скомпилируем
s:=a.Compile;

{ TODO -oUser -cConsole Main : Insert code here }

// Сохраним в файл
AssignFile(f,'file.exe'); rewrite(f);
BlockWrite(f,s[1],length(s));
CloseFile(f);

a:= nil;
end.

```

она лежит в самом начале секции данных – вот вам и адрес (см. листинг 19).

Набрали? Сохранили проект? Жмем <F9>. И пытаемся открыть полученный file.exe в отладчике или дизассемблере («Оля» рулит, так что покажу в ней).

Посмотрим, что получилось (см. выше рисунок 7). Нажмите <F8>. «Оля» выполнит команду, поместив в EAX число из памяти.

### Close to seven

О! А вот и дождик пошел. Посвежело, исчезла изнуряющая жара. Это хорошо. И у нас все получилось удачно. Согласитесь, приятно собственноручно создавать что-то. Чувствуешь себя на порядок выше. Жаль, что многие начинающие хакеры этого не понимают. Хоть и хорошо знают систему, но кидают все силы на то, чтобы ее поломать, но «ломать не строить». Есть, конечно, люди, взламывающие не ради развлечения, а чтобы выявить слабые места, но таких гораздо меньше. Теперь можно подумать о том, как дальше все эти наработки использовать. И воплотить все свои замыслы в код. Но это уже совсем другая история...

Все упомянутые в статье исходные коды и тестовый проект приведены в виде ресурсов в теме «Журнал клуба программистов. Седьмой выпуск» или непосредственно в архиве с журналом.



Под впечатлением от предыдущей статьи [1] нашего форумчанина **psycho-coder**, я решил написать свой небольшой мануал по работе с СУБД MySQL, используя библиотеку `mysql++`. Данная библиотека является кроссплатформенным решением, написанным на C++, и предоставляет богатый набор классов, позволяя создавать эффективные приложения...



**Олег Кутков**

by **Oleg Kutkov** [elenbert@gmail.com](mailto:elenbert@gmail.com)

## Краткий экскурс

Так как моей основной операционной системой является Linux, я буду рассматривать процесс создания приложения под Unix платформу. Под платформу Windows все будет практически точно так же, за исключением собственно подключения библиотеки и ее заголовочных файлов к IDE (MS Visual studio, Dev-c++). Для создания приложения в среде Unix наличие IDE, как и собственно графической оболочки, не необходимо, процесс написания и компиляции может проходить в голой консоли.

Для начала необходимо скачать саму библиотеку, это можно сделать по следующим ссылкам:

- Исходный код [2]
- Бинарные RPM сборки (следует попробовать разные версии, т. к. некоторые могут не установиться):
- RPM ver1 [3]
- RPM ver2 [4]
- RPM ver3 [5]
- Версия для MS Visual C++ [6]

Я не буду описывать установку библиотеки для каждой платформы, т.к. это выходит за рамки данной статьи. Лучше познакомиться с библиотекой, рассмотреть ее классы и создать простое приложение, которое будет подключаться к базе данных и запрашивать данные из таблицы, добавлять новые, модифицировать существующие записи. Для того, чтобы воспользоваться возможностями библиотеки, следует подключить заголовочный файл `mysql++.h` и пространство имен `mysqlpp`:

```
#include <mysql++.h>
using namespace mysqlpp;
```

Далее воспользуемся классом `Connection`, который обеспечивает подключение и аутентификацию на интересующей нас базе данных. Кратко рассмотрим этот класс и наиболее интересные его методы. Класс имеет два конструктора:

- `Connection(bool te=true)` – создание экземпляра класса, без подключения к БД
- `Connection (const char *db, const char *server=0, const char *user=0, const char *password=0, unsigned int port=0)` – создание экземпляра класса с подключением к указанной БД по указанным параметрам. Думаю, что названия параметров говорят сами за себя и не требуют пояснений
- `client_version()` – возвращает строку типа `string`, содержащую версию библиотеки.
- `connect (const char *db=0, const char *server=0, const char *user=0, const char *password=0, unsigned int port=0)` – то же самое, что и описанный выше конструктор, применяется в случае использования конструктора `Connection(true)`.
- `connected()` – возвращает `true`, в случае если выполнено подключение к базе, и `false`, если подключение не выполнено.
- `disconnect()` – выполняет отключение от базы данных.
- `error()` – возвращает последнюю ошибку.
- `query (const char *qstr=0)` – возвращает объект типа `query`, позволяющий выполнить запрос (об этом ниже).
- `count_rows (const std::string &table)` – возвращает результат типа `unsigned long`, количество строк в указанной таблице `table`.

### \* Комментарий редакции.

MySQL++ – это специализированная библиотека так называемых «оберточных» (wrapper) методов для прикладного программного интерфейса C API для СУБД MySQL. Главная цель этой библиотеки – сделать работу с SQL-запросами такой же простой, как работа с STL-контейнерами.

MySQL++ обеспечивает поддержку большинства разнообразных способов и приемов работы с базами данных.

Класс `Connection` имеет еще ряд методов, но они нам пока неинтересны. Теперь рассмотрим класс `query`. В нем нас интересуют следующие методы:

- `store()` – возвращает результат запроса, типа `StoreQueryResult`.
- `execute()` – выполняет запрос, не требующий возвращения данных, метод возвращает результат `SimpleResult`.

Класс `StoreQueryResult` – именно он позволяет обратиться к запрошенной таблице, представленной как двумерный массив, и получить количество возвращенных строк:

- `num_rows()` – возвращает количество строк
- `empty()` – возвращает `true`, в случае, если запрос ничего не вернул, и `false`, если запрос вернул данные.

```
// подключаем необходимые заголовочные файлы,
// пространства имен
#include <mysql++.h>
#include <iostream>
using namespace mysqlpp;
using namespace std;

// создаем экземпляры необходимых объектов
Connection conn;
StoreQueryResult queryres;
string querystring;

int main()
{
    try
    {
        conn.connect("database", "dataserver", "datauser",
                    "password"); // пробуем подключиться к базе
    }

    // перехватываем возможное исключение типа ConnectionFailed
    catch (ConnectionFailed err)
    {
        cout << "Не удалось подключиться к базе данных,
                причина: " << err.what() << endl;
        return 1;
    }

    // проверяем, подключены ли мы к базе данных
```

Класс `SimpleResult`. Здесь нас может интересовать всего один-единственный метод `rows()` – он возвращает количество строк, подвергшихся изменению, во время вызова `execute()`. Данное приложение представляет собой очень простой и детально расписанный пример (см. листинг ниже), который поможет разобраться новичкам.

Как видно, программа очень проста и легка для понимания. Вначале мы пробуем подключиться к базе данных `database`, которая находится на сервере `dataserver`, как пользователь `datauser` с паролем `password`. В случае ошибки перехватываем исключение типа `ConnectionFailed**`, которое имеет метод `what()`, возвращающий текстовое описание проблемы.

После успешного подключения проверяем еще

```
if(conn.connected())
{
    // инициализируем строку запроса
    querystring = "SELECT * FROM Datatable";
    // выполняем запрос
    queryres = conn.query(querystring.c_str()).store();

    // если что-то вернулось
    if(!res.empty())
    {
        // построчно выводим на экран
        for(int rc = 0; rc < (int)queryres.num_rows(); ++i)
            cout << queryres[rc]["colname_one"] <<
                    queryres[rc]["colname_sec"] << endl;
    } else
    {
        // иначе сообщаем, что ничего не вернулось
        cout << "Запрос не вернул данных" << endl;
    }

    // отключаемся от базы данных
    conn.disconnect();
} else
{
    // иначе сообщаем, что коннект отвалился
    cout << "Подключение к базе данных потеряно..." <<
            endl;
    return 1;
}
```

**\*\* Комментарий автора.**

Также у класса `ConnectionFailed` есть метод `errnum()`, возвращающий номер ошибки.

раз, подключены ли мы, т.к. в случае нестабильной связи или по иным причинам соединение может успеть отвалиться. В случае успешной проверки связи инициализируем строку запроса `querystring`, наш запрос имеет вид: `"SELECT * FROM Datatable"`, что означает «вернуть абсолютно все записи из некой таблицы `Datatable`». После этого вызываем метод `query` класса соединения, и у возвращенного объекта вызываем метод `store()`, возвращенный результат запроса сохраняем в `queryres`. Далее проверяем, не пуст ли результат. Если нет – построчно выводим записи на экран.

В классе `StoreQueryResult`, экземпляром которого является `queryres`, строки хранятся в виде двумерного ассоциативного массива, это дает возможность обратиться к соответствующему столбцу по его имени. Как в данном случае: `queryres[rc]["colname_sec"]` – обращаемся к `rc`-й строке и столбцу под названием `colname_one`. После окончания работы с БД следует обязательно закрыть соединение, что мы и делаем, вызывая метод `disconnect()`.

**В предыдущем примере** мы научились получать интересующие нас записи, теперь мы научимся добавлять и удалять их. Я не буду повторять код всей программы, а просто опишу способы, с помощью которых достигается нужный нам результат. В примере ниже показано, как можно добавить строку в нашу таблицу `Datatable`. SQL запрос в данном случае выглядит как `INSERT INTO Datatable(colname_one, colname_sec) VALUES('data1', 'data2')`. А для выполнения этого запроса будем использовать метод `execute()`:

```
querystring = "INSERT INTO Datatable(colname_one,
                                   colname_sec) VALUES('data1', 'data2')";
// выполнение запроса с выводом результата
cout << "Добавлено строк: " <<
      conn.query(querystring).execute().rows() << endl;
```

Как видно, все очень просто, инициализируем

строку запроса и вызываем метод `execute()`, который возвращает нам объект класса `SimpleResult`. Используя метод `rows()` последнего, мы получаем, сколько строк изменено, т.е. в данном случае добавлено. Полученный результат сразу же шлем в поток вывода. Я написал все без введения дополнительных, промежуточных, переменных – для компактности и экономичности. Все прочие операции, модификация, удаление строк выполняются аналогично, отличия будут только в самом тексте SQL запроса.

Модификация:

```
querystring = "UPDATE Datatable SET colname_sec = 'data3'
              WHERE colname_one = 'data1'";
// выполнение запроса с выводом результата
cout << "Изменено строк: " <<
      conn.query(querystring).execute().rows() << endl;
```

Удаление:

```
querystring = "DELETE FROM Datatable
              WHERE colname_one = 'data1'";
// выполнение запроса с выводом результата
cout << "Удалено строк: " <<
      conn.query(querystring).execute().rows() << endl;
```

Случается, что запросы могут некорректно работать с кириллическими символами; для устранения этой возмутительной ошибки следует выполнить особый запрос:

```
// задаем кодировку
querystring = "SET CHARSET UTF8";
// выполняем запрос
conn.query(querystring).execute();
```

В данном примере указана кодировка UTF8, как основная на современных Linux системах. Вам следует задать кодировку вашей системы для корректной обработки кириллицы. Для компиляции программы следует выполнить команду:

```
c++ -o progа -I/usr/include/mysql
-I/usr/include/mysql++ -lmysqlpp -L/usr/lib/mysql
-L/usr/local/lib/mysql++ main.cpp
```



В этой команде файл с исходным кодом `main.cpp` компилируется, подключая необходимые библиотеки.

## Заключение

Выше были описаны примеры, позволяющие взаимодействовать с базой данных MySQL на различных платформах. Были затронуты лишь общие методы работы с библиотекой `mysql++`, список же классов и их методов намного обширнее; полностью ознакомиться с ними можно на этой странице.

Все упомянутые в статье исходные коды приведены в виде ресурсов в теме «Журнал клуба программистов. Седьмой выпуск» или непосредственно в архиве с журналом.

## Ресурсы

- Psycho-coder. Работа с MySQL в C++. – Тема на форуме <http://programmersforum.ru/showthread.php?t=59147>
- Библиотека MySQL++. Исходный код для самостоятельной компиляции и документация <http://tangentsoft.net/mysql++/releases/mysql++-3.0.9.tar.gz>
- Бинарная RPM сборка ver1 <http://tangentsoft.net/mysql++/releases/mysql++-devel-3.0.9-1.el4.i386.rpm>
- Бинарная RPM сборка ver2 <http://tangentsoft.net/mysql++/releases/mysql++-devel-3.0.9-1.el3.i386.rpm>
- Бинарная RPM сборка ver3 <http://tangentsoft.net/mysql++/releases/mysql++-devel-3.0.9-1.el5.i386.rpm>
- Бинарная RPM сборка для MS Visual C++ <http://tangentsoft.net/mysql++/releases/mysql++-1.7.1-win32-vc++.zip>
- Список классов и их методов библиотеки MySQL++ <http://tangentsoft.net/mysql++/doc/html/refman/annotated.html>
- Полная документация на английском языке с различными примерами <http://tangentsoft.net/mysql++/doc/html/userman>

Впервые данная статья опубликована на форуме Клуба ПРОграммистов [www.programmersforum.ru](http://www.programmersforum.ru) 20/09/2009.

### \*\*\* Немного истории.

История MySQL начинается в 1979 г., у ее истоков стояла небольшая компания во главе с Monty Widenius. В 1996 г. появился первый релиз 3.11 под солярис с публичной лицензией. Затем MySQL была портирована под другие операционные системы, появилась специальная коммерческая лицензия. В 1998 г. Кевин Аткинсон [Kevin Atkinson] начал разработку библиотеки, которая по его первоначальному замыслу обеспечивала бы выполнение SQL-запросов и обработку их результатов без привязки к какой-либо конкретной СУБД. Это было отражено даже в оригинальном названии - SQL++. Все версии, предшествующие 1.0, являются плодом индивидуальной работы Кевина. В 1999 г. библиотекой занялась компания MySQL AB. Сначала некоторую работу проделал Майкл "Монти" Видениус [Michael "Monty" Widenius], затем он передал полномочия другому сотруднику MySQL AB Синише Миливоевичу [Sinisa Milivojevic]. Были выпущены версии 1.0 и 1.1, после чего Аткинсон официально передал все функции сопровождения библиотеки в руки Миливоевича и полностью устранился от какого бы то ни было участия в данном проекте. Миливоевич довел библиотеку до версии 1.7.9, которая была выпущена в середине 2001 г. К этому моменту стала очевидной невозможность реализации универсальной библиотеки SQL-запросов, независимой от конкретных реализаций СУБД. Ориентация на MySQL стала неизбежной. После выпуска версии 1.7.9 в работе над MySQL наступил период некоторого зстоя, который продолжался три года, до августа 2004г., когда ситуацию под контроль взял Уоррен Янг [Warren Young]. Уоррен сразу же выпустил версию 1.7.10, устранил все проблемы с компиляцией при использовании GCC, исправил большое количество ошибок, добавил новые возможности. В общем, как говорится «процесс пошел» ...

**MySQL** имеет два слоя – внешний слой `sql` и внутренний набор движков, из которых наиболее часто используется движок `InnoDB`, как наиболее полно поддерживающий `ACID`. Реализован стандарт `SQL92`. С модульной точки зрения код MySQL можно разделить на следующие составляющие:

- инициализация сервера;
- менеджер коннектов;
- менеджер потоков;
- обработчик команд;
- аутентификация;
- парсер;
- кеш;
- оптимизатор;
- табличный менеджер;
- движки (MyISAM, InnoDB, MEMORY, Berkeley DB);
- логирование;
- репликация;
- сетевое API;
- API ядра.

**MySQL** и **PostgreSQL** – две наиболее популярные open-source базы данных в мире. Каждая база имеет свои особенности и отличия. Если вам нужно быстрое хранилище для простых запросов с минимальной настройкой, то идеально подходит MySQL. Если вам нужно надежное хранилище для большого объема данных с возможностью расширения, то стоит посмотреть в сторону PostgreSQL.

**Очередная пятая тема-клон** от одного автора на форуме нашего Клуба [programmersforum.ru...](http://programmersforum.ru...)

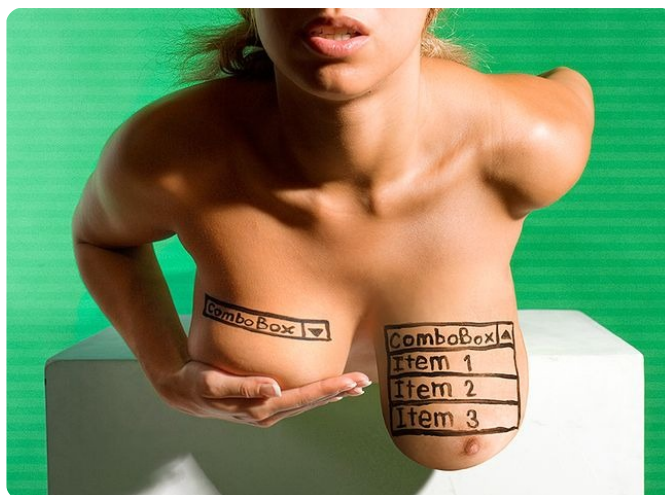
**сор08:** Дан двумерный числовой массив. Найти сумму максимальных элементов в строках

**ВОВАН13:** В принципе, можно еще раз 50 тем создать.

**Вопрос на собеседовании:** Назовите основные принципы ООП?

Ответ: Наследование, полиморфизм и... третий.

**Админы, скажите пожалуйста** есть ли антивирусы на подобе «Dr.web» которые сами производят очистку вирусов нужно делать проверку компьютера потому что меня уже 3 тию неделю подряд аблезают вирусы и меня отключают, пожалуйста ответить мне на вопрос...



**Рекордная скорость** передачи информации на малые расстояния была достигнута сегодня в офисе одной из компьютерных фирм при падении со стеллажа высотой в один метр коробки, содержащей 20 накопителей Seagate Barracuda 300 GB. Общий объем переданной на расстояние 1 метра информации составил  $300 \cdot 20 \cdot 8 = 48\,000$  гигабит, время передачи составило  $t = \sqrt{2h/g} = 0.4$  секунды. Таким образом, средняя скорость передачи информации составила 120 терабит/с!

**Работники одного НИИ** радиоэлектроники занимались, в частности, пайкой. Для которой им в бутылочки граммов на 150 выливали спирт по предъявлению каждым небольшой кучки канифольной крошки. Эту канифоль они в присутствии разливающего спирт растворяли в нем, получая на руки безнадежно испорченный для принятия внутрь продукт. У одного же товарища эта самая канифоль на спирту кончалась удивительно быстро, и он занимал ее у товарищей или приходил к разливающему за добавкой.

Причем, по его довольной физиономии всем было понятно, что спирт он как-то спас и принял в себя. Но как?

Вот канифольная крошка, вот спирт, вот насыпает ее, размешивает, растворяет. Вот снова пустая бутылочка. Опрокинул, мол,

локтем, извиняйте. Ситуация такая продолжалась несколько месяцев, дядька секрета не раскрывал, при этом очень огорчал сослуживцев здоровым и жизнерадостным перегаром.

Пока однажды его не засекли в раздевалке за странным делом: дядя перебирал леденцы монпансье. Желтенькие и зеленые он откладывал в кулек, а оранжевые – в кучку. Первые шли на закуску и отбитие запаха. Ну, а вторые он и использовал в качестве канифольной крошки для растворения...

Сегодня, 10:29

Сообщение #11 |

Цитата(m1x! @ 14.12.2009, 20:27)

Вот решил купить автомагнитоу, но денег на хорошую новую не хватит. Остановился на Alpine blu, но так как их blu мало видел, вопрос следующий: Кто разбирается в автомагнитолах и посоветует альтернативную замену? Нужно mp3, USB выход, выход на саб... Денег 800-1200 украинских баксов.

Цитата(m1x! @ 13.10.2010, 0:13)

Вообщем хочу купить магнитоу. Без всяких лишний наворотов. По деньгам маловато выходит, но всё же, 800-1000грн. Склоняюсь к Sony, Pioneer. Но есть советчики, на Kenwood, Blaupunkt. Кто в это деле разбирается, что лучше взять? (ну во всех, за эти деньги, есть юсб, линейный, 4 канала, mp3, то что нужно)

ТС, а вы каждый год новую покупаете ☹ сорри за оффтоп ...высока вероятность действия советов, что уже даны выше.

**Сказ про Федота-младчика,**  
удалого разработчика (отрывок)...

*Шеф:*

К нам сегодня прилетел  
Весь японский техотдел  
А у нас девайс на лампах  
Это что за беспредел?  
Собирайся, брат, вставай

*Федот:*

На твой код на Си Плюс Плюс  
Я без слез смотреть боюсь,  
Мой ассемблер побыстрее,  
Я не зря ему учусь...

**Севшие батарейки** «дюрасел» лежат в ящике письменного стола в 10 раз дольше, чем обычные солевые батарейки.



Да девайс перепай.  
Хочешь - в схеме ковыряйся,  
Хочешь - код рекомпилай!

*Федот:*

Получается опять  
Не придется ночью спать...  
Ты ж вчера кричал, что лампы  
Будут скоро все юзать!

Чтоб японский техотдел  
Из-за ламп не поседел  
Переделаю устройство,  
Раз я в этом преуспел.

*Программер:*

Ты скажи мне, друг Федот,  
отчего на асме код?  
Али память маловата,  
Али герц недостает?

**Реальные истории опен Wi-Fi сети AirNet** -  
г. Бердянск <http://airnet.sytes.net>

И снова звонок в ТП ...и снова девушка:

**Д:** Здравствуйте, это Эйрнет?

**ТП:** Здравствуйте, да. Слушаю вас.

**Д:** Помогите пожалуйста, у меня такая проблема, у меня есть (заминка) роутер (медленно) Asus WL-500W. Как мне через него подключить клавиатуру и мышку?

**ТП:** Ну это не совсем к нам вопрос, обратитесь в техподдержку производителя (тут в мыслях начинает возникать вопрос, зачем роутеру клавиатура и мышь). А зачем вам к нему подключать клавиатуру и мышь?

**Д:** Понимаете... у меня очень большой монитор, это подарок моего... друга. Сидеть перед ним неудобно, а провода у мышки и клавиатуры короткие. Мой... друг сказал, что их можно подключать без проводов и принес этот

приборчик, вот я и решила так сделать.

**ТП:** Стоп... вы что, хотите подключить мышь и клавиатуру к роутеру и чтобы через него они были подключены к компьютеру?

**Д:** (радостно) Ну да! Вы не беспокойтесь, провода в сам этот... роутер я нашла куда воткнуть, но не работает почему-то. Мне знакомый админ сказал, что (заминка) драйвера нужны...

**ТП:** (слегка обалдев) А мы чем можем помочь?

**Д:** Ну вы же занимаетесь всякими беспроводными штучками, наверняка знаете, как это сделать. Я заплачу, пришлите мне пожалуйста специалиста, который все сделает. Только не присылайте мальчика, а то у меня... друг ревнивый очень. У вас ведь есть специалисты-девочки?

**ТП:** (обалдев окончательно) Девушка, к сожалению девочек-специалистов у нас нет. Вам наверняка поможет фирма \*\*\*\*, запишите телефончик.

- Antiquity
- Зеленый телефон
- Tormozzz
- Гастроном 411
- DeadFish
- rama
- Not sure
- Хлам
- морозко
- Чемберлен
- airbase (вот это точно военный объект)
- Лыбидь
- teplonet
- domik
- IBM XT
- warmouse
- Хам
- ДМБ2011

...я думал,	msi	Usef@airnet.sytes.net	8.1	2.3	10.10.	...и гвоздь
что такие	10.10.1.79	ostanki@airnet.sytes.net	0.1	0.3	10.10.	номера, сеть
только в	ats44	softwer@airnet.sytes.net			10.10.	с шифрова-
анекдотах	10.10.1.126	Cabrio@airnet.sytes.net			10.10.	нием WPA2-
бывают :)	10.10.1.156	bizon@airnet.sytes.net			10.10.	Enterprise и
	10.10.1.163	Alex tiron@airnet.sytes.net	86.2	1.4	10.10.	идентифика-

тором «FreePhone».

Наш автор **WildHunter** поделился наблюдениями, что в сети периодически появляются компьютеры с довольно «неординарными» именами. Вот некоторые из них:

**Появилась помеха** в районе одного из хотспотов... далее приводится хронология событий по мере периодического сканирования сети:

- loser
- ёпт
- Error
- Железяка
- fuck\_myself
- Маша и медведи
- Device\_not\_ready
- SexUslugiBesplatno
- Маман-ПК
- Sibarit
- blathata
- Irka-dura
- Бункер Бормана
- эй на барже
- Дорогая
- Oba-na

