ХАКЕР
СПЕЦ
10|71|ОКТЯБРЬ 2006
ЕЖЕМЕСЯЧНЫЙ
ТЕМАТИЧЕСКИЙ
КОМПЬЮТЕРНЫЙ
ЖУРНАЛ

ДРЕССИРОВАННЫЙ КОД

ТОНКОЕ МАСТЕРСТВО ПРОГРАММИРОВАНИЯ

(game)land hi-fun media
9771609 102006 10>
publishing for enthusiasts


ЖУРНАЛ-БЛОКБАСТЕР / BLOCKBUSTER MAGAZINE
СИНК
SYNC
30 РУБЛЕЙ
> МЫ РАЗДЕЛИ «БЛЕСТЯЩИХ»
ЮЛЯ КОВАЛЬЧУК
АБРАМОВИЧ +
27
БАЛОВНЕЙ СУДЬБЫ
FUN
ПРОДАВЦЫ РАДОСТИ:
ШОУ-РАША О DISNEYLAND
ОРГАЗМ
И ЕЩЕ
15
ВЕЩЕЙ, ПРИНОСЯЩИХ РАДОСТЬ
СПЕЦПРОЕКТ SYNC
МУЖСКОЙ ЖУРНАЛ SYNC.*
НОВЫЕ ТЕНДЕНЦИИ ЕЖЕМЕСЯЧНО.
В ПРОДАЖЕ С 27 ОКТЯБРЯ.
ПО НОВОЙ ЦЕНЕ

intro

Я тебя знаю, дяденька. Ты — программер. Что, не программер? Админ? И все равно — в некотором роде — программер. Веб-мастер — опять-таки, программер, иначе какой же ты веб-мастер? Продаешь компьютеры на Савеловке? Ну, это не страшно, в институте же тебя заставляли кодить? Значит — с определенными скидками — тоже программер.
Я клоню к тому, что к какой бы области IT-индустрии ты не тяготел — алгоритмическое мышление в целом и новинки программерских технологий в частности помогут тебе. Совсем не обязательно перечитывать Кнута в пятый раз: открой этот номер, твоему вниманию престанут два раздела. Первый — про программерские трюки и алгоритмические мудрости, а второй — про новейшие веяния в программерском мире. В общем, самый гламур и самый актуальный фэшн :) (я не знаю, что означают эти слова, но, по-моему, что-то хорошее).

Александр Лозовский

СПЕЦ

ЕЖЕМЕСЯЧНЫЙ
ТЕМАТИЧЕСКИЙ
КОМПЬЮТЕРНЫЙ
ЖУРНАЛ

10|71|2006

WWW.XAKEP.RU

РЕДАКЦИЯ
Главный редактор
Николай «AvaLANche» Черепанов (avalanche@real.xakep.ru)
Выпускающие редакторы
Александр «Dr.Klouniz» Лозовский (alexander@real.xakep.ru)
Андрей Каролик (andrusha@real.xakep.ru)
Редактор CD
Иван «SkyWriter» Касатенко (sky@real.xakep.ru)
Литературный редактор
Настя Глухова
Арт-директор
Иван Васин (vasin@real.xakep.ru)
Дизайнер
Наталья Жукова (zhukova@real.xakep.ru)
Цветокорректор
Александр Киселев (kiselev@real.xakep.ru)

ОТДЕЛ РЕКЛАМЫ
Директор по рекламе
Игорь Пискунов (igor@gameland.ru)
Руководитель отдела рекламы цифровой группы
Ольга Басова (olga@gameland.ru)
Менеджеры отдела
Ольга Емельянцева (olgaeml@gameland.ru)
Евгения Горячева (goryacheva@gameland.ru)
Оксана Алехина (alekhina@gameland.ru)
тел.: (495) 935.70.34
факс: (495) 780.88.24

ОТДЕЛ ДИСТРИБУЦИИ
Директор отдела дистрибуции и маркетинга
Владимир Смирнов (vladimir@gameland.ru)
Оптовое распространение
Андрей Степанов (andrey@gameland.ru)
Подписка
Алексей Попов (popov@gameland.ru)
Региональное розничное распространение
Татьяна Кошелева (kosheleva@gameland.ru)
тел.: (495) 935.70.34
факс: (495) 780.88.24

ИНФОРМАЦИЯ О ВАКАНСИЯХ ИЗДАТЕЛЬСТВА «ГЕЙМ ЛЭНД»
Менеджер отдела по работе с персоналом
Марина Нахалова (nahalova@gameland.ru)
тел.: (495) 935.70.34 (доб. 454)

ИЗДАТЕЛЬСТВО «ГЕЙМ ЛЕНД»
Генеральный Директор
Дмитрий Агарунов (dmitri@gameland.ru)
Управляющий Директор
Давид Шостак (shostak@gameland.ru)
Директор по развитию
Паша Романовский (romanovski@gameland.ru)
Директор по персоналу
Михаил Степанов (stepanovm@gameland.ru)
Финансовый директор
Елена Дианова (dianova@gameland.ru)
Издатель цифровой группы
Борис Скворцов (boris@gameland.ru)
Редакционный директор цифровой группы
Александр Сидоровский (sidorovsky@gameland.ru)

ИНФОРМАЦИЯ О ПОДПИСКЕ
Бесплатный тел.: 8 (800) 200-3-999

ДЛЯ ПИСЕМ
101000, Москва, Главпочтамт, а/я 652, Хакер Спец
spec@real.xakep.ru

Отпечатано в типографии «ScanWeb», Финляндия
Зарегистрировано в Министерстве Российской Федерации
по делам печати, телерадиовещанию
и средствам массовых коммуникаций
ПИ № 77-12014 от 4 марта 2002 г.
Тираж 42 000 экземпляров.
Цена договорная.

ПРОГРАММНОЕ ЗАКУЛИСЬЕ

НОВЫЕ ФОКУСЫ

SPECIAL DELIVERY

МИХАИЛ ФЛЕНОВ

НАСТОЯЩИЙ X-ЧЕЛОВЕК.
С МОМЕНТА ОСНОВАНИЯ СПЕЦА
И ХАКЕРА — ОН С НАМИ.
ОН — ЭТО CYDSOFT.COM,
ОН ЖЕ — VR-ONLINE.RU,
И ОН ЖЕ — АВТОР МНОЖЕСТВА
КНИГ ПО ПРОГРАММИРОВАНИЮ
И КУЧИ ПЛАТНЫХ И БЕСПЛАТНЫХ
ПРОГРАММ. В ОБЩЕМ, ЭКСПЕРТ
ЭТОГО НОМЕРА

offtopic

cd

```
#include <mind>
#include <inspiration>

int main()
{
        if (!have_tools()) {
                load_cd();
                while (1) enjoy();
        }
}
```

ИНСТРУМЕНТЫ

- JDK 1.5.0 для Windows
- Borland C++ Builder 6
- JDK 1.5.0 для Linux
- OllyDbg 1.10
- APISpy32 3.0
- Набор инструментов с wasm.ru без купюр!

LOWLEVEL КОДИНГ

- LiveKd с sysinternals.com
- Masm32 8.0
- nasm-0.98.39
- TASM32
- MPLab IDE 7.42
- PIC Simulator IDE
- 8085 Simulator Addin
- Z80 Similator Addin
- Полная документация по контроллерам Microchip
- WinDriver 8.10 x32

ВОКРУГ ДА ОКОЛО

- DoxyGen 1.4.7 (Windows/*IX)
- DoxyWizard 1.4.7
- JavaDoc 1.5.0
- Элементы теории графов
- Graphviz 2.8 (Windows/Linux)

СОФТ ОТ NONAME

- ABoo 0.6
- Advanced Spyware Remover v.1.5
- GeeXboX v1
- KlipFolio 3.1
- IP Shifter v2.1
- ApexDc++ 0.2.1
- History Sweeper 2.71
- Windows Updates Downloader 2
- BitSpirit 3.2
- GRSoftware GRBackPro v6
- Eudora 7
- Dupeguru v2
- Actual Reminder 2

timeline

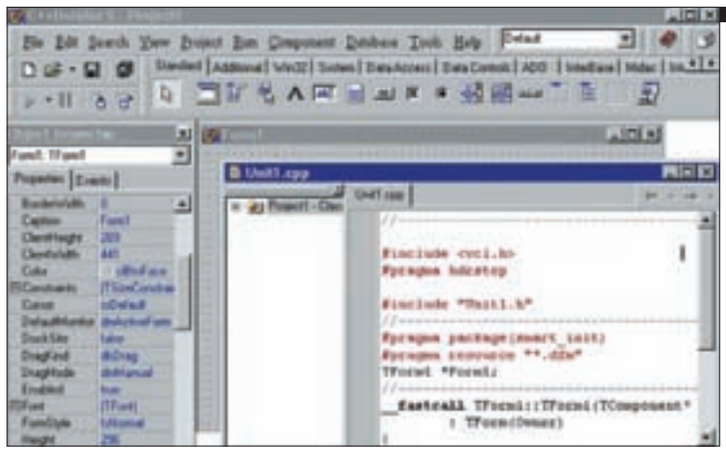
Андрей Каролик
andrusha@real.xakep.ru

1965

Впервые основные идеи структурного программирования были высказаны Эдсгером Дейкстрой. Разрабатывая идеи структурного программирования, Дейкстра решал задачу доказательства правильности программ. То есть искал ответ на вопрос, какими должны быть структуры программ, чтобы без чрезмерных усилий можно было находить доказательство их правильности. Это особенно важно при разработке больших программных систем. Правильность логической структуры системы поддается доказательству, а сама программа допускает достаточно полное тестирование. В результате, в готовой программе встречаются только тривиальные ошибки кодирования, которые легко исправляются.

1967

Simula 67 — первый объектно-ориентированный язык программирования. Он был разработан группой сотрудников Норвежского Вычислительного Центра (Norwegian Computing Center) в Осло под руководством Оле Йохана Даля (Ole Johan Dahl) и Кристена Нигарда (Kristen Nygaard) для решения задач моделирования сложных систем. Но Simula 67 опередил свое время и не выдержал конкуренции с другими языками программирования (прежде всего с Fortran). Прохладному отношению способствовало и то, что его реализация была весьма неэффективна. Но впоследствии идеи языка Simula 67 были заслуженно оценены и положены в основу современных объектно-ориентированных языков программирования — C++, Smalltalk, Eiffel и тому подобных.



1983

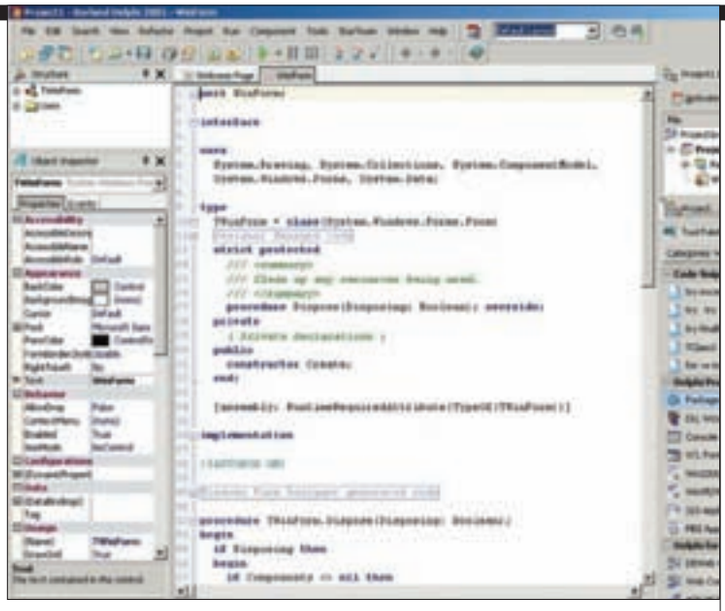
Группой исследователей во главе с Б.Страуструпом был разработан язык программирования C++ — логическое продолжение языка программирования C в направлении объектной ориентированности. Он поддерживает разные парадигмы программирования: процедурную, обобщенную, функциональную. Название C++ придумал Рик Масситти, — оно указывает на эволюционную природу перехода к нему от C, так как «++» — это операция приращения. Любимая шутка Страуструпа: «Знатоки семантики языка находят, что C++ хуже, чем ++C».

1969

Сотрудники Bell Labs Кен Томпсон и Денис Ритчи занимались развитием языка Би, а в итоге за несколько лет разработали Си — стандартизованный процедурный язык программирования. Он изначально был создан для использования в операционной системе UNIX, но с тех пор портирован на многие другие операционные системы и стал одним из самых используемых языков программирования. Си ценят за его эффективность, и для него характерны лаконичность, современный набор конструкций управления потоком выполнения, структур данных и обширный набор операций.

1993

Borland Delphi появляется в России и сразу же завоевывает широкую популярность. Новые версии выходят практически каждый год. В них реализуются все новые мастера, компоненты и технологии программирования. Таким успехом Delphi обязан процессу разработки, который в нем предельно упрощен. В первую очередь это относится к созданию интерфейса, на который уходит до 80% времени разработки программы. Delphi — результат развития языка Турбо Паскаль, который, в свою очередь, развился из языка Паскаль. Паскаль был полностью процедурным языком, Турбо Паскаль 5.5 добавил обьектно-ориентированные свойства, а Delphi уже стал полноценным объектно-ориентированным языком программирования с возможностью доступа к метаданным классов (то есть к описанию классов и их членов) в компилируемом коде.

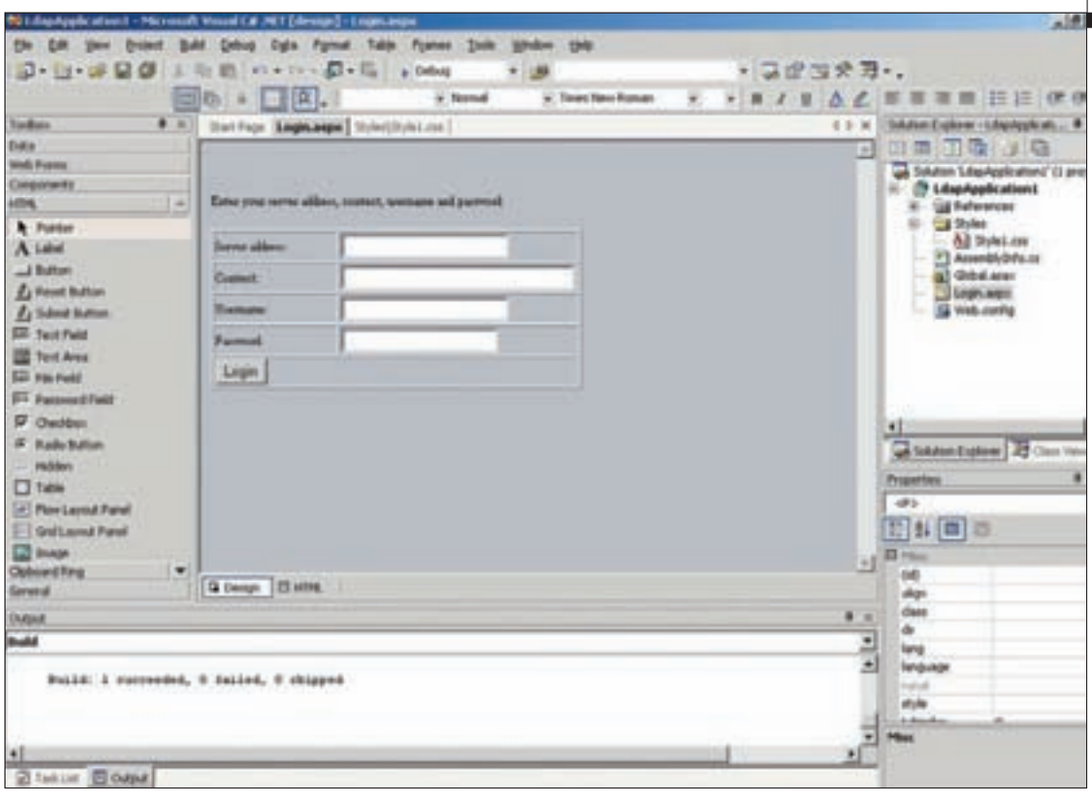


1995

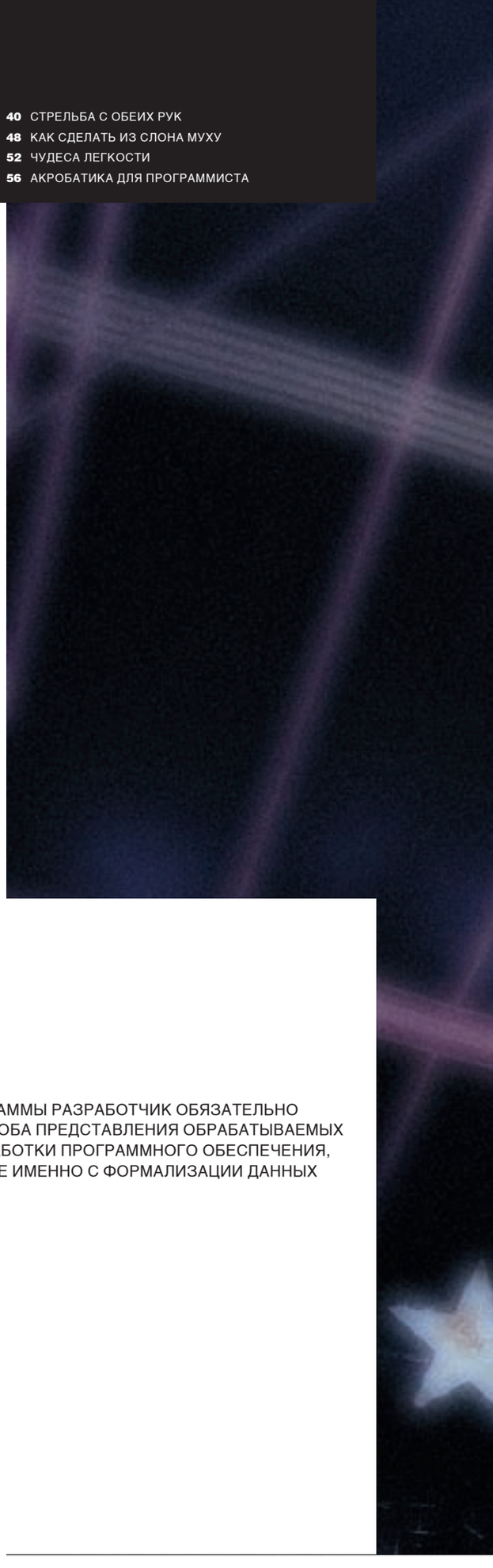
Sun официально объявила о языке Java. Хотя началось все в 1991 году, когда компания Sun Microsystems финансировала собственный исследовательский проект под кодовым названием «Green». Причиной было желание разработчиков найти универсальный язык программирования, чтобы соединить воедино все подключенные к сети приборы, будь то суперкомпьютеры или какие-нибудь холодильники с автоматическим заказом закончившихся продуктов. В результате был создан язык на основе языков C и C++ — «Oak», названый так в честь дуба, растущего за окном здания Sun. Правда, позже было обнаружено, что язык программирования с названием «Oak» уже существует. После визита в местное кафе было предложено имя Java, закрепившееся в последствии за языком.

1998

Начал разрабатываться проект, получивший кодовое название COOL (C-style Object Oriented Language). Это некий стратегический противовес Microsoft в отношении Java, так как последний создала конкурирующая фирма Sun Microsystems при поддержке двух других злейших врагов Microsoft — Oracle и IBM. Первая версия C# напоминала по своим возможностям Java 1.4, несколько их расширяя. И только в 2000 году компания Microsoft анонсировала платформу .NET и новый язык программирования, получивший название C# («си-шарп»).



ПРОГРАММНОЕ ЗАКУЛИСЬЕ



фигуры под куполом

СТРУКТУРЫ ДАННЫХ

ПРИ СОЗДАНИИ ПРАКТИЧЕСКИ ЛЮБОЙ ПРОГРАММЫ РАЗРАБОТЧИК ОБЯЗАТЕЛЬНО СТАЛКИВАЕТСЯ С ПРОБЛЕМОЙ ВЫБОРА СПОСОБА ПРЕДСТАВЛЕНИЯ ОБРАБАТЫВАЕМЫХ ДАННЫХ. СУЩЕСТВУЮТ МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, ПРЕДЛАГАЮЩИЕ НАЧИНАТЬ ПРОЕКТИРОВАНИЕ ИМЕННО С ФОРМАЛИЗАЦИИ ДАННЫХ (ТАК НАЗЫВАЕМЫЕ DATA DRIVEN ПОДХОДЫ)

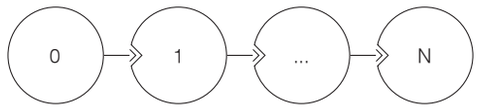
Антон Палагин aka Tony
tony@eykontech.com

➜ **массивы.** Классический способ объединений данных, который можно встретить в любом языке программирования. Массив — это последовательный набор данных. Каждый элемент массива обладает своим порядковым номером. В языках программирования, которые позволяют обратиться напрямую к оперативной памяти, элементы массива хранятся в памяти друг за другом. Из кода C/C++ к элементам массива можно обратиться с помощью оператора «i», где i — порядковый номер элемента массива. Аналогом Си-массива является контейнер STL vector, который гарантирует последовательное размещение в памяти элементов массива, обеспечивает доступ к ним с помощью оператора «», а также механизма итераторов.

Естественно, функциональность вектора этим не ограничивается, он также позволяет вставлять новые элементы в начало, середину и конец вектора. Этим занимаются функции поэлементной и интервальной вставки insert, push_front и push_back. Кроме вставки вектор также позволяет производить поэлементное и интервальное удаление своих элементов с помощью функций erase и clear.

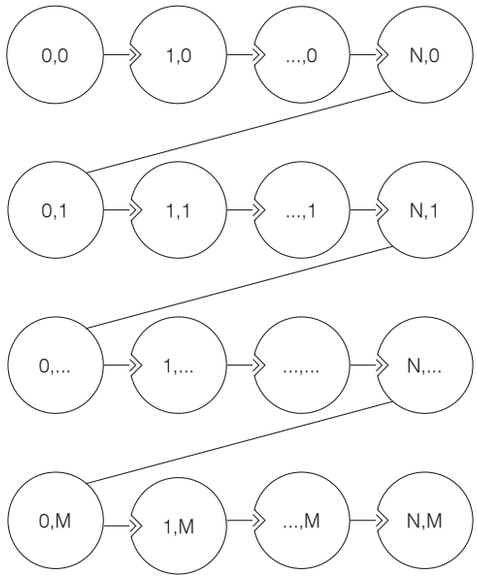
(1)

Элементы массива располагаются в памяти последовательно друг за другом. Каждый элемент массива нумеруется своим индексом. Итерироваться по по элементам можно произвольным образом.



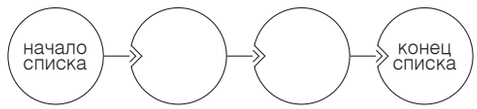
(2)

Элементы массива располагаются в памяти последовательно друг за другом, причем сначала идут элементы нулевой строки, за ними элементы первой и т.д. (как показано стрелками). Каждый элемент массива нумеруется двумя индексами по горизонтали и вертикали. Итерироваться по элементам можно произвольным образом.



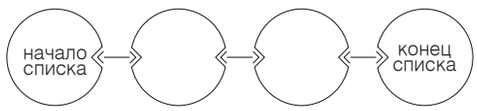
(3)

Каждый элемент односвязного списка хранит ссылку на следущий элемент. Поэтому итерирование по списку возможно только в одном направлении. Элементы списка могут храниться в произвольных областях памяти.



(4)

Каждый элемент двусвязного списка хранит ссылку на предыдущий и следущий элементы. Поэтому итерирование по списку возможно в двух направлениях. Элементы списка могут храниться в произвольных областях памяти.



Столь детальное внимание этому контейнеру уделяется неспроста, ведь правильный выбор контейнера, хранящего твои данные, гарантирует эффективное и безопасное выполнение кода. Об этом повествует книга Скотта Мейерса «Эффективное использование STL».

Строго говоря, вектор — это наиболее общий тип контейнера, который больше всего похож на стандартный массив. Существуют его аналоги, такие как deque, rope, list, map, hash_map и т.д. Их необходимо выбирать, исходя из способа обращения к данным (смотри рисунок 1).

➜ **производные массивов.** У массивов есть несколько производных — многомерные массивы и ассоциативные массивы. Многомерные массивы хранят элементы, индексируемые не одним, а несколькими «координатами» (индексами), а ассоциативные массивы хранят элементы, индексируемые неким ключом, заданным программистом. Например, таким ключом может быть строка. Еще одной производной массива является стек. Если проводить ассоциации с реальной жизнью, то стек похож на женский чулок, в котором хранятся луковицы :). Если он завязан с одного конца, то ты можешь заложить в него только одну луковицу и забрать ее же. Самую первую заложенную луковицу ты сможешь забрать только после того, как вытащишь все луковицы, заложенные после нее. Это модель стека LIFO (last in first out). Если чулок развязать и закладывать луковицы с одной стороны, а извлекать с другой, то ты получишь стек FIFO (first in first out), также называемый очередью. Если пример с чулком не очень понятен, то просто попробуй купить билет в метро в восемь утра :). Стеки используются, например, при передаче аргументов в функцию. Когда ты работаешь с функцией, принимающей переменное число аргументов, ты как раз разматываешь подобный стек с помощью стандартных функций va_arg, va_start, va_end (смотри рисунок 2).

работа с массивами

```
int array1[10]; //Определение обычного
массива
vector<int> stlArray1; //То же самое,
но с помощью STL
array1[0] = stlArray1[5] = 10;
//Присваивание значения элементам массива
float array3[10][15][30]; //Определение
трехмерного массива
vector< vector< vector<float> > >
stlArray3; //То же самое, но с помощью STL
array3[1][0][10] = stlArray3[9][2][4] =
0.1f; // Присваивание значения элементам
массива
map< string, int > mapArray; //Определение
ассоциативного массива, в котором ключом
является строка текста (имя человека),
а значением — его возраст (целое число).
```

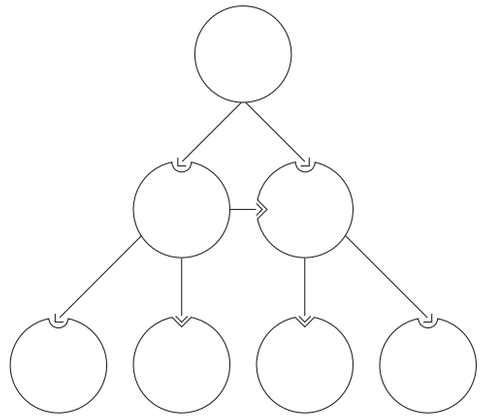
➜ **списки.** Список — это тоже производная массива, правда, здесь тебе никто не даст гарантию, что элементы списка будут лежать в памяти последовательно. Каждый элемент списка имеет ссылку на следующий элемент. Такой список называется однонаправленным связанным списком. Если элемент списка хранит ссылку на следующий и предыдущий элементы, то это двунаправленный список. Двунаправленные списки реализуются с помощью контейнера list. Все эти структуры данных являются линейными по своей сути и используются, когда обрабатываются одномерные последовательности данных — сортируются, выбираются нужные элементы и т.д. (смотри схемы 3 и 4)

работа со списками

```
//Определение элемента односвязного списка
template< class T >
struct SimpleElem
{
 T elem; //Текущий элемент
 SimpleElem * next; //Указатель на следу-
 ющий элемент
};
//Определение элемента двусвязного списка
template< class T >
struct BiElem
{
 T elem; //Текущий элемент
 BiElem * prev; //Указатель на предыдущий
 элемент
 BiElem * next; //Указатель на следующий
 элемент
};
```

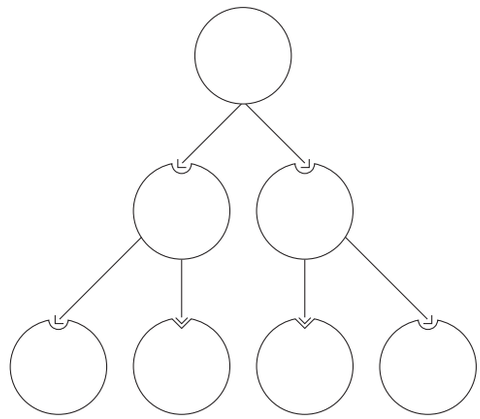
(5)

Вершины обобщенного графа (graph) могут иметь сколь угодно дочерних вершин. Любая вершина графа имеет как минимум одну дочернюю вершину, кроме конечных вершин, называемых лисьями (leaf)



(6)

Бинарное дерево обязательно имеет вершину, называемую корнем (root). Это верхний элемент на схеме. Каждая вершина может иметь только две дочерних вершины. Нижние лепестковые вершины не имеют дочерних вершин.



➔ **таблицы** — это основные единицы хранения данных в современных СУБД. Это тоже своего рода двумерный массив, но массив неоднородных данных, которые к тому же должны обладать специальными характеристиками, такими как уникальность, определенность, размер данных ячейки таблицы и т.д. Таблица состоит из столбцов и строк соответственно. Каждая ячейка однозначно определяется двумя координатами — номерами строк и столбцов.

определение таблицы

```
//Определяем столбцы таблицы
struct Row
{
std::string firstName;
std::string secondName;
int age;
};
//Определяем набор строк таблицы
std::vector< Row > Table;
//Доступ к ячейке таблицы
Table[10].age = 28;
```

➔ **графы.** Граф — наиболее общий вид древовидных структур данных. Если продолжать пищевые ассоциации, то узлы графа — это узлы ветвей виноградной лозы, а сами виноградины — это узлы нижнего уровня без дочерних узлов (так называемые leaf'ы). Узлы графа соединены между собой ребрами. Графы применяются для алгоритмов оптимизации, например, для выбора наиболее оптимального маршрута передачи пакетов между маршрутизаторами. Графы изучает целая наука — теория графов. Существует превеликое множество различных графов, но остановимся только на тех видах, которые крутятся на языке (смотри рисунок 5).

определение элемента (узла) графа

```
template< class T >
struct Graph
{
 T elem; //Текущий элемент
 std::list< Graph *> edges; //Указатели
 на связанные элементы графа
};
```

➔ **бинарное дерево,** оно же двоичное дерево. Часто используется в алгоритмах поиска. Каждый узел в дереве формирует свое поддерево. У каждого узла может быть только два дочерних узла — правый и левый. Соответственно, каждый узел (за исключением конечных, самых нижних узлов) такого дерева имеет одно входящее и два исходящих ребра. Главным алгоритмом для деревьев является алгоритм его обхода (Traverse), который позволяет найти наиболее оптимальный маршрут к нужным данным. Бинарное дерево лежит в основе алгоритма отбрасывания невидимых полигонов BSP (binary space partitioning), используемого в играх. Аналогично двунаправленному списку, каждая вершина бинарного дерева ссылается на правый и левый элементы, но в случае дерева эти элементы являются дочерними для узла, а не равноправными, как в случае списка (смотри рисунок 6).

определение элемента (узла) бинарного дерева

```
template< class T >
struct BiTree
{
 T elem; //Текущий элемент
 BiTree * left; //Указатель на левый
 дочерний элемент
 BiTree * right; //Указатель на правый
 дочерний элемент
};
```

➔ **октарное дерево.** Оно же octree. Это способ разбиения трехмерного пространства (space partitioning), который используется при разработке игр для отбрасывания объектов, не попадающих в поле зрения камеры (frustum camera's view). Из названия понятно, что узел такого дерева имеет восемь дочерних элементов. Суть метода заключается в том, что трехмерный куб (пространство) бьется на восемь меньших равных кубов. Каждый подкуб бьется еще на восемь меньших и т.д.

определение элемента (узла) октарного дерева

```
template< class T >
struct OcTree
{
 T elem; //Текущий элемент
 OcTree * childs[8]; //Дочерние элементы
};
```

Таким образом получается октарное дерево, каждый узел которого имеет восемь дочерних элементов. Если перейти от трехмерного пространства к двухмерному, то может быть использовано не октарное, а квадовое дерево. То есть квадрат разбивается на четыре части (как бы половинка октарного дерева) C

www.sgi.com/tech/stl/index.html
руководство для программиста по stl (на английском).

http://ru.wikipedia.org/wiki/список_структур_данных
что думает по поводу структуры данных википедия.

www.gamemaker.webservis.ru/articles/octree/octree.htm
познавательно об octree.

смертельный трюк

РАБОТА В КОМАНДЕ ПРОГРАММИСТОВ — КАК БЫТЬ?

ПОЧЕМУ-ТО ВСЕ ПРЕИМУЩЕСТВЕННО ГОВОРЯТ О ТЕХНИКЕ СОЗДАНИЯ КОМАНДЫ, ЧЕМУ ПОСВЯЩЕНО МНОЖЕСТВО СТАТЕЙ, КНИГ, СЕМИНАРОВ, И НА ВЫБОР РУКОВОДИТЕЛЯ ПРЕДЛАГАЕТСЯ РЯД МЕТОДИК, ОТРАБОТАННЫХ ГОДАМИ. НО ВОТ О ТОМ, КАК ВЫЖИТЬ В КОМАНДЕ, РЕАЛИЗОВАВ СВОЙ ТВОРЧЕСКИЙ ПОТЕНЦИАЛ, ОБЫЧНО УМАЛЧИВАЮТ

Крис Касперски
no e-mail

Времена, когда один-единственный программист мог вести весь проект целиком, безвозвратно прошли. Рынок наводнили акулы бизнеса, ожесточенно конкурирующие между собой, и требования к качеству продукта резко возросли. «Голая» программа без красивого графического интерфейса, многостраничной документации, официального сайта и службы поддержки уже никому не нужна. Исключение составляют утилиты, ориентированные на узкий круг профессионалов, которым достаточно развитого функционала, но таких — единицы, и реальные деньги крутятся там, где обитают полуграмотные пользователи, популяция которых исчисляется сотнями миллионов. Даже примитивный каталогизатор дисков, будучи достойно оформленным и поданным, может приносить неплохой доход, не говоря уже про автоматизацию предприятий или компьютерные игры. Было бы заманчиво написать такой продукт в одиночку, но, увы, это практически невозможно. Даже гениальный программист, талантливый абсолютно во всем, навряд ли захочет отвечать на идиотские письма дебильных пользователей, а писем таких будет очень много, ведь каждый покупатель считает, что вместе с продуктом он покупает еще и внимание. А ну-ка, мать вашу за ногу так, быстро разберитесь, почему ваша программа не работает у меня!

Десять лет назад можно было программировать, зная всего лишь Turbo Pascal и MS-DOS, теперь же количество обязательных языков/библиотек/технологий исчисляется десятками. Писать интерфейс на Си — это самоубийство, причем крайне болезненное и мучительное, но еще хуже пытаться реализовать вычислительное ядро на DELPHI. Поэтому неизбежно встает вопрос о создании команды, а, быть может, и целой фирмы.

Работать в команде и сложно, и просто одновременно. С одной стороны, без обмена идеями и знаниями программист никогда не сможет реа-

«every single employee understand that they are part of a whole. thus, if an employee has a problem, the company has a problem» **(c) the matrix**

лизовать свой потенциал на 100%. Великие находки и грандиозные открытия, как правило, рождаются в спорах. С другой стороны, если ты один, тебе приходится рассчитывать только на самого себя, и никто не придет на помощь в трудную минуту (про то, что некоторые пройдохи ухитряются перекладывать свои проблемы на чужие плечи, мы знаем, но скоромно промолчим).

Это были плюсы. Теперь о минусах. Командная работа требует намного большей самоотдачи, заставляя заниматься не тем, чем тебе интересно, а тем, что нужно команде (как говорится, что полезно для вида, вредно для индивида), при этом неудача одного из участников ударяет рикошетом по всем остальным, и сотни часов, проведенных за монитором, летят впустую. Сплоченные команды — большая редкость (если они вообще существуют в природе), и интересы участников чаще всего оказываются противоречащими друг другу. Кто-то рвется вперед, выдвигая одну сумасбродную идею за другой, кто-то топчется на месте, вылизывая каждую строку кода, радуясь, что ему ценой месячного траха удалось сократить длину функции на шесть байт. Как следствие — возникают расколы и конфликты. Атмосфера внутри коллектива становится душной, а работа — непроизводительной.

Хуже всего приходится творчески активным индивидуалистам — необщительным, неуживчивым, неконтактным, привыкшим работать удаленно, каким, собственно, и является мыщъх, имеющий богатый опыт командной работы (хоть большей частью и отрицательный). Но ведь тут вот какой момент… Тот, кто умеет что-то делать неосознанно, кто наделен этим талантом от рождения, никогда не сможет объяснить, как именно и что именно он делает, почему поступает так, а не эдак и т.д. Мыщъх же представляет собой довольно аутичный тип от природы и общению учился сам, действуя методом проб и ошибок, о которых и хочет рассказать, адресуя статью таким же, как он сам.

➜ **joining the team.** Примкнуть к известной команде, стать ее частью — мечта каждого второго, а, быть может, и двух третей всех программистов. Точной статистики у мыщъх'а нет, но и без нее понятно, что сильная команда — это хорошо, однако, тут все не так гладко, как кажется.

Начнем с того, что «не бойся показаться дураком, бойся показаться очень умным, потому что тогда немедленно возникнет вопрос: если ты такой умный, то почему ты нанимаешься на работу, а не нанимаешь на нее?» (с) Виктор Пелевин «Generation P». Сильные команды в своей массе довольно скептически (если не сказать брезгливо) относятся к новичкам, особенно если те ведут себя самоуверенно, как будто любая работа им по плечу (это, кстати сказать, первый признак непрофессионализма: чем больше знает человек, тем сильнее он начинает комплексовать по поводу своей тупости, потому что осознает разрыв, отделяющий его от истинных знаний; к тому же, во всякой предметной области есть куча нюансов, и ее нельзя освоить с криком «ура», бросая свое тело на амбразуру технической документации и литературы).

Лучше занизить планку своих знаний, чем завысить ее, поскольку в первом случае остается «запас прочности», а во втором — легко попасть впросак, не сумев ответить на поставленный вопрос. Опять-таки, следует различать понятия «умею» и «видел». Например, мыщъх краем глаза видел Perl, но опыта работы с ним не имеет, поэтому зачастую идет не тем путем, которым нужно, а тем, который знает. То же самое можно сказать и о DELPHI, Си++ и многих других языках.

Любой нормальный программист имеет представление как минимум о паре десятков языков, но активно программирует на двух-трех, на которые и следует делать упор, а остальные можно даже не упоминать, особенно, если работал с ними давно и уже успел подзабыть. Однажды с мыщъх'ем случился такой конфуз: желая блеснуть своим хвостом, он настрочил длинный список языков, с которыми когда-либо имел дело. Естественно, когда его начали проверять, выяснилось, что даже простейшую программу для вычисления факториала ни на Форте, ни на Лиспе мыщъх (без обращения к документации) составить не в состоянии. Доверие сразу же было подорвано, а сама компетентность поставлена под сомнение.

В идеале, следует искать только те команды, которые реально нуждаются в тебе, в твоих знаниях и умениях, а не в чем-нибудь другом, но в жизни очень часто случается так, что тебе рекомендуют подучиться, а то и переучиться полностью! «OK, ты классно программируешь на Си, но мы ведем несколько проектов на Java, поэтому ты должен быть знаком и с этим языком». Соглашаться или нет? Спорный вопрос. С одной стороны, если отвечать «да пошли вы все!», то в поисках работы можно провести всю оставшуюся жизнь, к концу которой Си рискует стать неактуальным, так что переучиваться все равно придется. Не сейчас, так потом. Но! Нормальный лидер всякой преуспевающей команды всегда стремится использовать навыки и наклонности каждого из программистов по максимуму, то есть по назначению. Собственно, именно это и обеспечивает команде процветание. Если же брать кого попало, распределяя задания хаотичным образом по принципу «не умеет — научится», то это не команда будет, а просто

сброд, с которым хорошо пить пиво и обсуждать округлости женщин, но отнюдь не выпускать конкурентоспособные продукты.

Подытоживая сказанное, сформулируем правило номер один: всегда оставайся самим собой. Если ты влюблен в ассемблер — программируй на ассемблере. Если ты небрежен, небрит и неаккуратен — вот таким и приходи на собеседование. Если у тебя рваный график, плавно перетекающий из совы в жаворонка, а потом обратно — не бери на себя обязательств приходить на работу ровно в восемь или быть в это время как штык на IRC-канале. Поверь, существует множество команд, работающих по такому же принципу. Внешний вид — не помеха. Главное — результат.

Как правило, новички панически боятся ответить «нет» на любой поставленный им вопрос, опасаясь, что за этим последует категорический отказ (и хорошо, если не пинок ногой), вот и «подписываются» на любые условия, которые им только предложат. На самом деле, в слове «нет» ничего ужасного нет. Употребляй его почаще. Оно поможет тебе продать свои умения и навыки наиболее выгодно.

→ **в команде.** Поздравляю! Ты в яйце. С этого момента твои личные интересы задвинуты на задний план и все свободное (и несвободное) время посвящено команде. Некоторые ведут сразу несколько проектов в различных командах, но, во-первых, это страшно выматывает, а, во-вторых, отсутствие концентрации на одной-единственной цели часто приводит к провалу.

В команде намного сложнее реализовать свой творческий потенциал, свои идеи (особенно рискованные и непроверенные), поскольку в случае провала страдают все участники команды (а у многих жена, дети…). Конечно, без дерзости, без творческих порывов не бывает и прорывов, но позволить себе втягиваться в работу над рискованным проектом может либо только что сформировавшаяся команда, либо команда, уже имеющая за плечами несколько успешных проектов и солидный капитал, но даже в этом случае «раскрутить» остальных членов, заинтересовать их своим проектом будет нелегко, поскольку у каждого из них имеются свои собственные идеи, и чем команда больше, тем труднее выделиться на фоне остальных.

С другой стороны, в маленьких командах никакая исследовательская деятельность невозможна в принципе. Главное здесь — получить рабочий продукт в отведенные строки, не отвлекаясь на посторонние вопросы. Ты хочешь экспериментов, чувствуешь, что можешь значительно увеличить производительность и функционал кода, если сменишь парадигму программирования, библиотеку, компилятор и т. д. Тебе интересно посидеть с Кнутом, осваивая новые алгоритмы, чтобы в конечном счете выбрать самый эффективный из них, но… Команду это не интересует! Код работает — и это главное! Навряд ли оптимизация увеличит объемы продаж. Лучше выпустить еще один минимально работающий проект. А потом еще один и еще… Постепенно это гонка превращается в настоящий марафон, требующий только тех знаний, которые уже есть, и никак не способствующий творческому развитию.

Большие команды могут позволить талантливому программисту совершенствовать свое мастерство и годами вести исследовательскую деятельность, которая, возможно, не принесет никакой отдачи. А возможно и принесет. В девятнадцатом веке лишь немногие люди верили в перспективу электричества, остальные же считали, что оно пригодно лишь для игрушек и фокусов. Графический интерфейс тоже поначалу не вызвал большого энтузиазма, а сейчас это основной способ работы с компьютером. Никто и никогда не сможет заранее сказать, какая технология окажется перспективной, а какая — нет, однако, это не значит, что любая бредовая идея будет щедро финансироваться.

В принципе, обладая достаточной напористостью (и авторитетом) можно «раскрутить» лидера команды на любой проект, убедив его, что все будет OK. Это самая легкая часть, но потом приходится сложнее. Самые замечательные и элегантные (на бумаге) конструкции зачастую рассыпаются в прах при столкновении с реальностью. На поздней стадии нередко вскрываются непредвиденные трудности, которые без дополнительных (и весьма крупномасштабных) исследований никак не удается обойти. Проект затягивается, требуя все новых вложений, но бросить его, похоронив тысячи строк кода, жалко. Команда высаживается и нервничает, теряя время и деньги, а ты — авторитет.

Правило номер два: продвигая свои идеи, никогда не выдавай желаемое за действительное и не убеждай других в том, в чем сам до конца не уверен. Если проект провалится, — это не принесет пользы ни тебе, ни команде. Конечно, от ошибок и провалов никто не застрахован, но если решение о новом проекте принималось коллегиально, то в случае досадной неудачи ответственность равномерно распределяется между всеми участниками. Если же ты агрессивно лоббировал свою позицию, тебе это обязательно припомнят в не самом лучшем виде.

SPECIAL МНЕНИЕ

АНТОН ПАЛАГИН
директор по развитию компании Eykon

FAQ ПО КОМАНДНОЙ РАЗРАБОТКЕ

1. Помни, что все мы люди, а людям свойственно ошибаться. Не относись к чужим ошибкам слишком категорично.
2. Уважай чужое мнение и постарайся разобраться в чужой точке зрения, вместо того, чтобы отстаивать свою с пеной у рта. Помни, что в споре рождается истина.
3. Не игнорируй командные инструменты разработки, полагаясь на свою память и кучу бумажек, наклеенных на монитор. Активно используй и систему контроля версий, и багтрек.
4. Не кати бочку на коллегу из-за того, что его «кривой» код «косячит». Если ты нашел его ошибку, опиши ее в багтреке, сопроводив куском кода, который вызвал сбой, или скриншотом, объясняющим результат.
5. Следуй стандартам кодирования, используемым остальной командой (компанией). Так твоим коллегам будет проще разбираться с кодом.
6. Не старайся объяснить дизайн своего компонента на пальцах, а используй язык uml и rational rose для создания множества других диаграмм rational.
7. Внимательно следи за развитием проекта, над которым работаешь. Это позволит тебе быть в курсе, чем занимаются твои коллеги, в курсе самых последних изменений. Вдруг то, что ты хотел сделать сегодня, уже сделано в чужом коде.

➔ **соображения о стиле программирования.** Стиль программирования — весьма щекотливый вопрос. В одних командах имеется свод правил оформления листинга (учитывающий буквально все: от расстановки фигурных скобок до системы наименования переменных), в других — ничего подобного нет. Пиши как хочешь и на чем хочешь! У каждой практики есть свои плюсы и минусы. Даже плохой стандарт кодирования, поддерживаемый всеми участниками, лучше, чем его полное отсутствие. Команда на то и команда, чтобы работать совместно, попеременно заглядывая то в свои, то в чужие листинги. И если каждый начнет следовать своим привычкам, образуется форменный балаган, в котором через некоторое время будет совершенно невозможно разобраться.

А вот и другая сторона медали (очень часто упускаемая маркетоидами из виду). Привычка — это вторая натура и от нее никуда не уйти. Кодирование в «неестественном» стиле значительно снижает производительность труда и одновременно с этим затрудняет поиск ошибок «глазами». К тому же, если программист использует фрагменты ранее написанных листингов, их придется реконструировать в соответствии с правилами кодирования, а это не только впустую потраченное время, но еще и вероятность внесения ошибок в уже отлаженный код. А что если один и тот же исходный файл используется в нескольких независимых проектах, каждый из которых велит придерживаться своих правил кодирования?! Как развивать и сопровождать такой файл? Это же сдохнуть можно, если вручную синхронизовать несколько проектов без подходящих средств оптимизации!

Правило номер три: если тебя не устраивает стандарт кодирования, принятый в команде, пиши в своем стиле, пытаясь убедить остальных в том, что от этого никому хуже не станет. Как показывает мыщъх'ный опыт, в большинстве случаев это срабатывает. Стиль программирования — это вообще-то, так, ерунда. Бюрократическая волокита. Гораздо важнее сразу определиться с используемыми языками и компиляторами. Допустим, 9 из 10 членов команды пишут на Си, а один — на DELPHI. Состыковать откомпилированные модули — не проблема, но вот для эффективной работы каждый участник проекта должен понимать любого другого. Мы не можем просто сказать — возьми DELPHI и напиши крутой интерфейс для нашей программы, поскольку сразу же возникает вопрос: а, что, собственно, писать? Какими функциями обладает программа на данный момент, и какими она (возможно) будет обладать? К тому же, интерфейс полностью отделен от вычислительной части лишь в книжках по программированию, а в реальной жизни все компоненты программы связаны друг с другом, и если DELPHI-программист абсолютно не разбирается в Си, остальные члены команды будут вынуждены постоянно отвлекаться от своей работы, объясняя ему что и как. И наоборот. Си-программисты обязаны знать DELPHI хотя бы на минимальном уровне, чтобы самостоятельно собрать проект без посторонней помощи.

Крайне нежелательно использовать в проекте языки, которые кроме тебя никто не знает (например, Haskell или Ruby), поскольку это ставит в зависимость остальных членов проекта. В результате ты становишься незаменимым носителем знания. В команде, состоящей из нескольких человек, с этим еще можно как-то смириться, но вот крупные коллективы тебя просто попрут. И правильно сделают! А вдруг ты решишь уйти или забросить проект, занявшись другими неотложными делами? Кто разберется в твоих исходных текстах, если потребуется исправить баг или добавить новую фичу? Какой смысл ставить под удар труд десятков людей только потому, что на таком-то языке задача решаться чуть-чуть более эффективно, чем на общепринятых?

Но даже в рамках одного языка (например, того же Си) необходимо согласовать используемые компиляторы, поскольку «смешанное» программирование еще никого не доводило до добра. Достаточно привести один пример: в Borland C++ тип char по умолчанию unsigned, а у Microsoft Visual C++ — signed, поэтому программа, разработанная на Borland C++, может разваливаться при компиляции под Microsoft Visual C++ и наоборот. Но с этим еще можно хоть как-то бороться. Скажем, взять за правило компилировать программу несколькими компиляторами и давить баги еще в зародыше. При переносе на другие платформы это очень помогает! Но никогда не надо собирать программу из объектных файлов, откомпилированных разными компиляторами, поскольку каждый из них завязан на свой RTL, а RTL должен быть только один! В крайнем случае, транслируйте объектные файлы, созданные различными компиляторами, в DLL-модули. Это снижает производительность (и подчас довольно значительно), но исключает конфликт RTL.

Другой больной вопрос — это хаки, то есть нестандартные приемы программирования, зачастую завязанные на малоизвестные или недокументированные возможности операционной системы, языка программирования или компилятора. Их следует избегать любой ценой! И это не обсуждается!

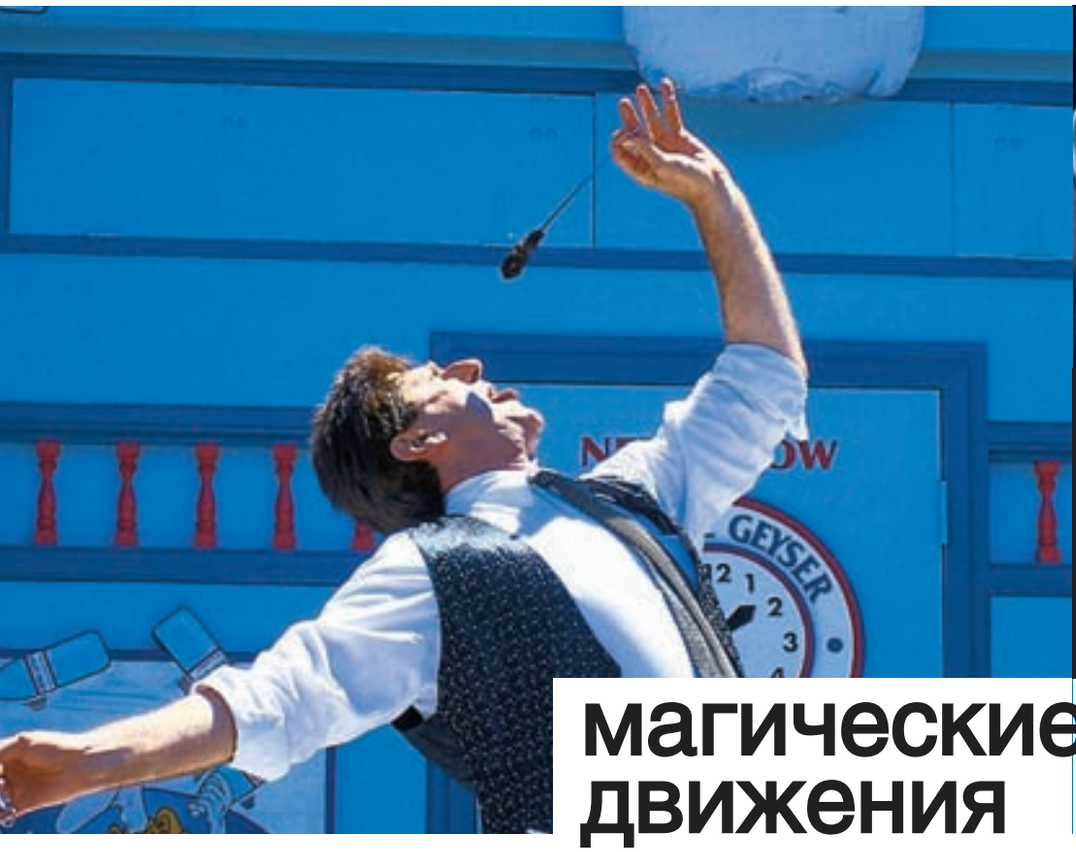
➔ **размер имеет значение, или о том, как программисты меряются письками и что из этого обычно выходит.** Как говорится, «тормозит обычно тот, кто за клавиатурой, да и то, лишь по мнению тех, кто не за ней». Критиковать чужие ошибки и просчеты всегда легко, но зачем?! Только чтобы почувствовать превосходство над остальными? «Все, мол, кретины и дураки, один я умный, весь проект держится на мне, чтобы вы, идиоты, без меня делали и т.д.». Да, в команде могут (и должны!) встречаться программисты различного уровня квалификации. Тезис о том, что все члены должны стоять на одной ступеньке, в корне неверен, и вот почему: программное обеспечение крайне неоднородно по своей структуре и состоит из модулей различной сложности. С одними справится даже девушка, закончившая двухнедельные курсы, другие же потребуют консолидации усилий нескольких матерых спецов. Какой смысл поручать спецам писать простой код? Не лучше ли использовать их навыки там, где они действительно необходимы? И потом, специалистами не рождаются. Старшие передают опыт младшим, так было испокон веков.

Профессионалы, кстати говоря, встречаются самые разные. Одни смотрят на салаг свысока, другие же общаются с ними как с равными, зачастую вступая в разборки по поводу и без. Но подробных объяснений ждать в любом случае не стоит. Ты не в институте, и за твое обучение денег никто не получает. А объяснять очередному пионеру в сотый раз одни и те же истины — скучно, неинтересно и непродуктивно.

Правило номер четыре: профессию не получают, ее воруют. Присматривайся к остальным членам команды и бери на заметку все полезные приемы программирования, обращая внимание даже на незначительные нюансы. Не бойся открыто обсуждать свои идеи с коллегами. Даже если тебя раскритикуют и поднимут на смех, по крайней мере, укажут на ошибки, а это дорого стоит! Также, не пытайся завоевать дешевый авторитет, руководствуясь принципом: кто ничего не делает — тот никогда не ошибается. Пускай авторитет позаботится о себе сам. Твоя основная задача — совершить максимум ошибок, наступив на все грабли, которые только находятся в пределах досягаемости. Шишки — заживут, ошибки — забудутся, а знания — останутся и будут работать на авторитет. На настоящий авторитет, который нельзя ни пропить, ни скомпрометировать, ни потерять, потому что он реален.

Гораздо лучше заслужить репутацию человека, не боящегося признаться в собственной некомпетентности, чем выглядеть снобом. Впрочем, в руководители высшего звена пробиваются именно снобы и губят всех тех, кто находится под ними. Сравните, какой была Microsoft при Билле Гейтсе и какой она стала при Стиве Балмере.

➔ **заключение.** Прежде чем закончить свое сумбурное «пособие по выживанию», мыщъх хотел бы напомнить, что в таких вопросах никакой истины не существует и «сколько людей — столько же и мнений». Мыщъх высказал свое. Это не означает, что все остальные мнения порочны и неверны. Вовсе нет! Главное ведь результат, а не пути его достижения. Чужой опыт редко бывает полезен остальным, поскольку зависит от психотипа личности и стоящих перед ней проблем, а проблемы можно решать по-разному. Одни предпочитают склонять голову, другие — идут напролом. Третьи же комбинируют два первых способа, помня о мудрой восточной поговорке: «не будь слишком мягким — сомнут, не будь слишком твердым — сломают» C



МАРКО БЕЛИЧ

РОДИЛСЯ В БЕЛГРАДЕ, СЕРБИЯ. ФЛЕШОМ ПРОФЕССИОНАЛЬНО ЗАНИМАЕТСЯ С 1998 ГОДА. РАБОТАЛ В НЕСКОЛЬКИХ ДИЗАЙН-СТУДИЯХ В БЕЛГРАДЕ. В 2002 ГОДУ ПЕРЕЕХАЛ В МОСКВУ. РАБОТАЛ В СТУДИИ MEMPHIS (WWW.MEMPHIS.RU), А СЕЙЧАС РАБОТАЕТ В DEFA STUDIE (WWW.DEFA.RU).

магические движения

СКРИПТОВАНИЕ ПОД ФЛЕШ

FLASH ПРОШЕЛ ДЛИННУЮ ДОРОГУ ОТ ОБЫЧНОЙ ПРОГРАММЫ ДЛЯ СОЗДАНИЯ АНИМИРОВАННЫХ ГИФОВ, ЧЕРЕЗ БАННЕРЫ И НАВИГАЦИЮ ДЛЯ САЙТОВ, ДО МОЩНОЙ ПРОГРАММЫ, КОТОРУЮ ИНОГДА НЕЧЕМ ЗАМЕНИТЬ, ОСОБЕННО ПРИ СОЗДАНИИ МУЛЬТИМЕДИЙНЫХ ПРЕЗЕНТАЦИЙ ИЛИ RIA (RICH INTERNET APPLICATIONS)

Марко Белич
marko.belic@gmail.ru

→ **процесс работы.** Пройдемся по некоторым важным моментам, которые обязательно будут встречаться при создании любого флеш-проекта, особенно когда работаешь с Actionscript.

Базовая единица в работе — timeline. Содержание меняется с помощью свойств. Для этого нужно заранее определить, как будет работать анимация и какие визуальные элементы в этом шоу будут участвовать. И уже с учетом этого создается структура movieClip'ов, отделяются подвижные элементы от статичных и т.п. Основная цель — сделать так, чтобы во всем можно было легко разобраться.

Далее следует обратить внимание на регистрационную точку movieClip. Когда работаешь с timeline tween, весь процесс основан на прямом визуальном определении позиций элементов. В случае, когда работаешь с AS tween, все иначе, а самое главное — координата movieClip на timeline. У главного timeline (_root) регистрационная точка находится в верхнем левом углу. Когда создаешь новый movieClip и продвигаешь его на координату x:100, y:100, для всего его контента локальная координата x:0,y:0 будет на самом деле координатой x:100, y:100 на _root. В этом нет ничего особо страшного, но если в процессе работы не обращать внимания на регистрационные точки movieClip, возникнет ситуация, когда локальные координаты клипов вообще не совпадут с теми, которые ты видишь на сцене. Самый лучший подход на практике — создавать все клипы так, чтобы их регистрационные точки совпадали с верхним левым углом главного _root timeline. С таким подходом ты всегда будешь работать с глобальными координатами (то есть с теми, с которыми работаешь визуально).

Последняя, но очень важная вещь, на которую нужно обратить внимание, — сам Actionscript. Рекомендуется его держать «на одном месте». Грамотно сделанный с помощью слоев timeline поможет быстрее разобраться, когда в проекте появится достаточно много контента. Создай отдельный слой (или несколько) для скрипта и отдельно слой для контента, которым будешь управлять. Старайся достаточно комментировать скрипт с помощью однострочных (//), либо в несколько строк (/* — */) комментариев. В больших скриптах комментарии помогают ориентироваться, искать потенциальные ошибки и делать изменения.

→ **motion tween во флеше** состоит из movieClip-объекта (можно использовать и группу), который находится на 2-х (или больше) ключевых кадрах (keyframe) в разных видах. Он может отличаться по атрибутам, месту нахождения, ротации, размеру, цвету и прозрачности. То что на самом деле делает tween, это вычисления изменений атрибутов movieClip в обычных кадрах, которые лежат между двумя ключевыми кадрами. Проще говоря, это тот же самый movieClip, который меняется с каждым кадром, пока его атрибуты не станут такими, как в последнем кадре. Вся анимация вычисляется в момент экспортирования swf-файла.

Посмотрим, как это работает на простом примере. Нарисуем на сцене что угодно и сделаем от этого movieClip. На 20-м кадре делаем другой ключевой кадр и двигаем movieClip на другое место. Между ключевыми кадрами делаем motionTween. Экспортируем и смотрим размер swf-файла. Размер должен быть достаточно маленьким, меньше 1 Кб, если, конечно, поверхность достаточно простая. Добавим теперь изменения для некоторых атрибутов. Поменяем цвет, ротацию, размер и прозрачность movieClip на втором ключевом кадре. Количество изменений выросло, и увеличился размер swf-файла, но не намного.

Давай теперь добавим еще кадров, увеличим длину tween. Подвинь последний ключевой кадр на 100-ый кадр и посмотри размер. Теперь на 200-ый, а потом и 1000-ый. Умножь это на 10 или на 20, что примерно представляет общее количество tween-кадров в одном проекте. Размер еще кажется маленьким? Конечно, даже такой размер на сегодняшний день, при серфинге интернета через высокоскоростные каналы, не представляет большой проблемы. Скорее всего, основную проблему создаст перемещение ключевых кадров. Двигать второй ключевой кадр на 1000-ый кадр было не очень приятно? А представь, что это надо делать по 20 раз и больше…

➔ **ActionScript tween.** Нам нужен один movieClip и десяток строк на AS, которые мы можем использовать для анимации любого movieClip в проекте. Не будет дополнительных tween-кадров, в которых записано, как меняются атрибуты клипов, следовательно, размер swf не будет увеличиваться. Все потому, что такой тип tween не вычисляет анимацию наперед, а только в момент ее воспроизведения. То есть компьютер в процессе прорисовки анимации должен калькулировать tween в каждом кадре. Звучит как проблема, но сегодняшние мощности персоналок глотают подобные вычисления без особых напрягов.

Достаточно написать одну строку — вызов функции, которой мы передаем название и атрибуты изменяемого movieClip, конечные числа анимации и количество кадров (длину tween). Все кажется достаточно простым. Чтобы, к примеру, поменять длину tween (подвинуть его на 1000-ый кадр), достаточно поменять одно число в его функции.

Код, который используется для tween, по размеру в финальном swf-файле занимает около 3-4 Кб. В этом коде находятся все возможные комбинации tween, которые мы, возможно, и не будем использовать. Но этот единый код мы можем использовать везде в проекте.

➔ **tween-прототипы.** Скрипт во флеше, как и все остальное, основан на проигрывании кадров. В старых версиях флеша был единый способ создания подобной анимации — с помощью пустых клипов, которые постоянно крутили два кадра, а в одном из них был скрипт, который нужно было запускать. Теперь есть возможность программно запустить скрипт на любом movieClip с помощью onEnterFrame-функции (handler), скорость же зависит от FPS-проекта.

Основу AS tween заложил Роберт Пеннер (www.robertpenner.com). Он первый, кто написал пакет основных функций. Эти функции (tween-прототипы) сделали настоящий переворот в мире AS-программирования, и в последней версии флеша Macromedia официально предложила их для AS tween.

Посмотрим несколько tween-прототипов. В первой версии, которая написана для AS1 (ее-то мы и будем использовать), все функции написаны как часть Math-объекта. Самая простая из них делает линейный tween — анимацию без ускорения и замедления.

Эта функция, как и все остальные, получает некоторые значения на входе и возвращает просчитанные значения, которые мы можем использовать для анимации. Первый параметр (t) означает текущий кадр, который считаем. Его значение начинается с единицы и увеличивается до числа, которое означает длину анимации в кадрах (d). Второй параметр (b) — это начальное значение атрибута, который обсчитываем. Так, если буем двигать movieClip просто по _x-направлению, тогда нам нужно начальное значение _x. Третий параметр (c) означает, на сколько нужно поменять атрибут movieClip. Чтобы подвинуть movieClip на 200 пикселей вправо, просто нужно поставить 200. Для движения влево нужно поставить -200. В четвертом параметре — количество кадров, то есть длина анимации. Здесь можно поставить любое число больше нуля.

Следующая функция делает более сложное движение. Сначала объект будет двигаться медленно, потом быстрее.

Все функции можно найти на странице Роберта Пеннера www.robertpenner.com. Причем функции существуют для двух форматов AS1 и AS2.

Для движения movieClip нам нужны не только tween-прототипы. Они возвращают специфические значения в зависимости от параметров, которые мы им задаем. А нужен нам еще один тип функции — tween manager. Именно tween manager упростит и автоматизирует процесс создания tween.

Нет единого способа создания tween-менеджера. По сути, это функция, которая принимает, аналогично tween-прототипам, параметры, меняет переменную (t) текущего кадра и атрибуты movieClip в зависимости от результата. В Сети есть несколько неплохо сделанных и проверенных временем «пакетов» функций, с помощью которых можно сделать любой tween. Вот некоторые из них:

[1] TWEEN MANAGER LADISLAVA ZIGO МОЖНО НАЙТИ НА HTTP://LACO.WZ.CZ/TWEEN/. НА ЕГО САЙТЕ НАХОДИТСЯ И ПРОГРАММКА ДЛЯ СОЗДАНИЯ ЛЮБОЙ TWEEN-ФУНКЦИИ, КОТОРУЮ МОЖНО СКАЧАТЬ КАК КОМПОНЕНТ ДЛЯ ФЛЕША. ЕСТЬ И ОБЗОР РАЗНЫХ TWEEN-МЕНЕДЖЕРОВ, ТАК ЧТО МОЖНО ПОСМОТРЕТЬ, КАК ОНИ РАБОТАЮТ, И ОПРЕДЕЛИТЬСЯ, КАКОЙ ТЕБЕ БЛИЖЕ.

[2] ANIMATION PACKAGE ALEX UHLMANNA ЛЕЖИТ НА САЙТЕ WWW.ALEX-UHLMANN.DE/FLASH/ANIMATIONPACKAGE/. ЭТОТ МОЩНЫЙ ПАКЕТ МОЖНО ИСПОЛЬЗОВАТЬ НЕ ТОЛЬКО ДЛЯ СОЗДАНИЯ TWEEN, НО И ДЛЯ РИСОВАНИЯ ЛИНИЙ И РАЗНЫХ ПОВЕРХНОСТЕЙ (SHAPE).

Все эти менеджеры на первый взгляд кажутся достаточно сложными, и с непривычки будет проблематично оседлать незнакомый скрипт. Поэтому вернемся к нашим тараканам и сами напишем простой tween-менеджер. А когда поймешь работу созданной функции, будет гораздо легче разобраться с принципом работы других пакетов.

➔ **наш tween-менеджер** будет основан на onEnterFrame-функции. А значит, будет зависеть от FPS флеш-документа, в котором работает. Наиболее оптимальные значения FPS — в районе 40-60, но в зависимости от случая они могут быть меньше или больше. Также эту функцию напишем как прототип movieClip-объекта. Это позволит его запускать на любом movieClip в проекте. AS1 подходит для этого как нельзя лучше.

Наша функция должна иметь следующие переменные на входе:

— ТИП TWEEN-ПРОТОТИПА ИЗ ПАКЕТА РОБЕРТА ПЕННЕРА, КОТОРЫЙ ХОТИМ ИСПОЛЬЗОВАТЬ;

— АТРИБУТ MOVIECLIP, КОТОРЫЙ ХОТИМ ПОМЕНЯТЬ — ФУНКЦИЯ ДОЛЖНА ВЗЯТЬ ЗНАЧЕНИЕ ЭТОГО АТРИБУТА ДЛЯ НАЧАЛА АНИМАЦИИ;

— ЗНАЧЕНИЕ АТРИБУТА В КОНЦЕ TWEEN;

— ДЛИНА TWEEN В КАДРАХ.

```
1 MovieClip.prototype.tweenPropertyTo =
function(tweenType, property, change,
frameDuration) {
```

КАК ЗАКАЛЯЛАСЬ СТАЛЬ

Статьей этой было весьма весело заниматься. Основная тонкость заключалась в том, что автор отлично знал тему, но плохо писал по-русски. Читал и говорил он при этом очень даже хорошо. Писать же, к примеру, на английском, а потом переводить на русский — совсем не то, так как потерялись бы некоторые языковые тонкости и непередаваемые русские обороты :). В итоге, было принято решение найти некий компромисс. Выбор пал на транслит вперемешку с английскими словами и чисто флешевыми терминами без перевода. Тогда мы подумали, что проблем особых не будет. А когда получили сам текст... Сама перепись с транслита на русский потребовала немало времени, а въехать в некоторые обороты и попробовать воссоздать все на читаемом русском... Одним словом, суммарно было убито времени как на 2-3 обычных статьи. Но результатом довольны, так как это получился, в своем роде, эксклюзив по-сербски :).

Функция называется tweenPropertyTo, декларируем ее через прототип movieClip-объекта. Все входящие параметры уже описаны.

```
2 var c = this;
3 c.vars = new Object();
```

Локальную переменную с будем использовать как замену movieClip, на который применяем tween. Потом делаем объект vars, в котором будут храниться атрибуты и значения переменных, нужных для tween.

```
4 c.vars.t = 0;
5 c.vars.pr = property;
6 c.vars.b = c[c.vars.pr];
7 c.vars.c = change-c.vars.b;
8 c.vars.d = frameDuration;
9 c.vars.tT = tweenType;
```

переменные по очереди:

- T БУДЕМ ИСПОЛЬЗОВАТЬ КАК СЧЕТЧИК;
- PR БУДЕТ ХРАНИТЬ АТРИБУТ, КОТОРЫЙ ДЕЛАЕМ STRING (ЕСЛИ НУЖНО МЕНЯТЬ _X, В ФУНКЦИЮ БУДЕМ ПЕРЕДАВАТЬ «_X»);
- B АВТОМАТИЧЕСКИ ВЫЧИСЛЯЕТ НАЧАЛЬНОЕ ЗНАЧЕНИЕ АТРИБУТА, КОТОРЫЙ МЕНЯЕМ;
- C ХРАНИТ ЗНАЧЕНИЕ ИЗМЕНЕНИЯ АТРИБУТА;
- D ХРАНИТ ДЛИНУ TWEEN В КАДРАХ;
- TT ХРАНИТ ТИП TWEEN, КОТОРЫЙ ХОТИМ ИСПОЛЬЗОВАТЬ, ЕГО ТОЖЕ НУЖНО НАЗНАЧИТЬ КАК STRING (ЕСЛИ БУДЕМ ИСПОЛЬЗОВАТЬ ФУНКЦИЮ MATH.LINEARTWEEN, НАЗНАЧИМ ПЕРЕМЕННУЮ КАК «LINEARTWEEN», ТОЛЬКО ОБРАЩАЙ ВНИМАНИЕ НА ЗАГЛАВНЫЕ БУКВЫ, ТАК КАК ОНИ ДОЛЖНЫ СТРОГО СООТВЕТСТВОВАТЬ).

```
10 c.onEnterFrame = function() {
11 this.vars.t++;
12 if (this.vars.t<=c.vars.d) {
13 this[this.vars.pr] =
Math[this.vars.tT](this.vars.t,
this.vars.b, this.vars.c, this.vars.d);
14 } else {
15 delete (this.vars);
16 delete (this.onEnterFrame);
17 }
18 };
19 };
```

Запускаем onEnterFrame event, который будет исполнять tween в каждом кадре. В строке 11 поднимаем значение t на единицу. В строке 12 проверяем, стало ли значение t больше значения d. Если стало больше, то тогда функция onEnterFrame останавливается в строке 16, а объект vars удаляется.

Обрати внимание на строку 13. В ней находится вызов tween-прототипа, и его результат автоматически применяется к атрибуту movieClip. Значит, если хотим анимировать свойство _x и используем функцию easeOutCirc, сама строка выглядит как:

```
this._x = Math.easeOutCirc(this.vars.t,
this.vars.b, this.vars.c, this.vars.d);
```

Это самый простой tween manager. Остается только его запустить. Быстрый способ — один из tween-прототипов Роберта Пеннера выложить на первый кадр нашего проекта. Делаешь movieClip, называешь его mclip и анимируешь один из его атрибутов.

скрипт, который вставляем в первый кадр:

```
Math.easeOutCirc = function (t, b, c, d) {
 return c * Math.sqrt(1 - (t=t/d-1)*t) + b;
};
MovieClip.prototype.tweenPropertyTo =
function(tweenType, property, change,
frameDuration) {
 var c = this;
 c.vars = new Object();
 c.vars.t = 0;
 c.vars.pr = property;
 c.vars.b = c[c.vars.pr];
 c.vars.c = change-c.vars.b;
 c.vars.d = frameDuration;
 c.vars.tT = tweenType;
 c.onEnterFrame = function() {
  this.vars.t++;
  if (this.vars.t<=c.vars.d) {
   this[this.vars.pr] =
   ath[this.vars.tT](this.vars.t,
   this.vars.b, this.vars.c, this.vars.d);
  } else {
   delete (this.vars);
   delete (this.onEnterFrame);
  }
 };
};
mclip.tweenPropertyTo("easeOutCirc","_x"
,350,50);
```

Последняя линия кода — сущность tween. То, ради чего мы все это делали. Вся история создания и добавления кадров, передвижения клипов и изменения их атрибутов — все это можно поместить в одну строку кода. Если не нравится длина — 50 кадров, — меняешь на 1000. Все просто. Исходник можно найти по адресу: www.shockwaver.net/swf/tween/tweentest.fla.

Но это самая простая версия tween-менеджера, и она не позволяет менять несколько атрибутов одновременно, например _x и _y вместе, чтобы получить движение по диагонали.

Полную же версию можно найти здесь: www.shockwaver.net/swf/tween/tweenfull.fla. Основной принцип одинаковый. Несколько атрибутов и их значения можно передать функции как массив.

Еще одна возможность — исполнение дополнительной функции по завершению tween:

```
rf = function () {
 trace (this);
}
mclip.tweenPropertyTo("easeOutBounce",
["_x","_y","_alpha"],[350,55,60],150,rf);
```

Спустя некоторое время работы со скриптами ты увидишь, что скриптовая анимация не только не сложна, но и незаменима.

➔ **когда нужен frame tween.** Есть определенные ситуации при работе с флешем, когда frame motion или shape tween необходимы. К примеру, какое-то очень сложное движение по заранее определенной траектории или покадровая анимация, 3d… В любом случае, анимацию, которая у нас лежит в movieClip, можно контролировать с помощью tween-менеджеров и tween-прототипов.

У каждого movieClip есть атрибуты _currentframe и _totalframes, но их можно только читать. То есть текущий кадр movieClip мы не можем контролировать с помощью этих атрибутов. Для этого используем методы gotoAndStop() и gotoAndPlay(). Поскольку наш tween-менеджер может анимировать только атрибуты, посмотрим, как можно сделать новый атрибут и контролировать его (у AS2 есть более удобный способ для создания новых атрибутов, но мы работаем в AS1).

```
MovieClip.prototype.getFrame =
function () {
 return (this._currentframe);
}
MovieClip.prototype.setFrame =
function (fr) {
 this.gotoAndStop(Math.round(fr));
}
with (MovieClip.prototype) {
 addProperty ("_fr", getFrame, setFrame);
}
```

Здесь три функции. Первая и вторая называются «получатель» и «установщик». Их задача — применить существующие методы, чтобы поменять наш movieClip. Последняя функция инициализирует новый атрибут всем movieClip через их прототип. Теперь можно вместо mclip.gotoAndStop(frame) написать:

```
mclip._fr=frame;
```

У нас теперь есть новый атрибут, и мы можем его анимировать tween-менеджером:

```
mclip.tweenPropertyTo("easeOutQuad","_fr",
50,120);
```

Эта строка анимирует mclip с его текущего кадра до 50-го, используя прототип easeOutQuad длиной в 120 кадров.

Обрати внимание, что некоторые tween-прототипы (например, easeOutElastic и easeOutback) не могут работать с предельными значениями кадров. Значения, которые эти функции возвращают, выходят за рамки, в которых работают остальные tween. Если у нашего клипа есть 50 кадров, и мы в качестве финального значения используем 50 и функцию easeOutElastic, tween-менеджер попробует подвинуть клип на кадры за 50-ым. А поскольку их нет, это не будет отображаться.

➔ **еще несколько полезных функций,** которые мы можем использовать вместе с нашим tween-менеджером. Опять нам поможет Роберт Пеннер с собранием функций для контролирования цвета movieClip. Поменять цвет movieClip — не настолько простой процесс, но эти функции сделают это легко. С помощью функций можно менять цвет в hex, инвертировать цвета клипов, контролировать яркость и т.п. Качаешь их тут: www.robertpenner.com/tools/color_toolkit.zip. Включаешь в проект (уже присутствуют в tweenfull.fla) и получаешь новые атрибуты, которые можно использовать в tween-менеджере. Некоторые из этих атрибутов: _brightness, _contrast, _rgb и другие.

Сделаем с помощью этих функций tween цвета. Для этого нужна еще одна функция, которая подготовит movieClip для tween:

```
MovieClip.prototype.tweenHex =
function (tweenType, startHex, endHex,
frames, callBack) {
 var c = this;
 c.setRGBStr(startHex);
 var stR = parseInt(startHex.substr(0,2),
 16);
 var stG = parseInt(startHex.substr(2,2),
 16);
 var stB = parseInt(startHex.substr(4,2),
 16);
 var enR = parseInt(endHex.substr(0,2),
16);
 var enG = parseInt(endHex.substr(2,2),
16);
 var enB = parseInt(endHex.substr(4,2),
16);
 var rFunc = function () {
   c.setRGBStr(endHex);
   callBack();
 }
 c.tweenPropertyTo(tweenType,
["_red","_green","_blue"],[enR,enG,enB],
frames, rFunc);
}
```

а потом запускаешь функцию:

```
mclip.tweenHex("easeOutBounce","000000",
"FF3366",250);
```

Возможности tween'a ActionScript'ом позволяют сделать практически все и на порядок больше, нежели обычный tween, который создается при помощи множества кадров.

Надеемся, что уже в следующем своем проекте ты попробуешь что-нибудь сделать с помощью ActionScript tween'a ☻

наводнение в цирке

БОРЬБА С УТЕЧКАМИ РЕСУРСОВ И ПЕРЕПОЛНЯЮЩИМИСЯ БУФЕРАМИ НА ЯЗЫКОВОМ И ВНЕЯЗЫКОВОМ УРОВНЕ

С УТЕЧКАМИ РЕСУРСОВ И ПЕРЕПОЛНЯЮЩИМИСЯ БУФЕРАМИ ПРОГРАММИСТСКОЕ СООБЩЕСТВО БОРЕТСЯ УЖЕ ЛЕТ ТРИДЦАТЬ, НО ВСЕ БЕЗУСПЕШНО. В ЭТОЙ СТАТЬЕ МЫЩЪХ ДЕЛИТСЯ СВОИМИ СОБСТВЕННЫМИ ТРЮКАМИ И НАХОДКАМИ. НА «СЕРЕБРЯНУЮ ПУЛЮ» ОНИ НЕ ПРЕТЕНДУЮТ, НО ЛИКВИДИРУЮТ ЗНАЧИТЕЛЬНОЕ КОЛИЧЕСТВО ОШИБОК С МИНИМАЛЬНЫМИ НАКЛАДНЫМИ РАСХОДАМИ

Крис Касперски ака мыщъх
no e-mail

→ **введение.** Проблемы утечки ресурсов и переполнения буферов главным образом относятся к Си (и в меньшей степени к его преемнику — Си++). Это не покажется удивительным, если вспомнить, что Си — самый низкоуровневый из всех высокоуровневых языков, всего лишь на одну ступень отстоящий от ассемблера. Си делает только то, о чем его просят, вынуждая программиста самостоятельно выделять и освобождать память, контролировать границы буферов и заботиться о массе других вещей, которые на Java, C# и многих других языках автоматически выполняются компилятором (интерпретатором). Однако ни один из них не завоевал такой популярности, как Си, и навряд ли завоюет ее в дальнейшем. Почему? На Си написано огромное количество программ, которые необходимо развивать и поддерживать, Си обеспечивает максимальную производительность и переносимость, в то время как его конкуренты предлагают либо производительность, либо переносимость.

Что же касается Си++, то это гибрид, вобравший в себя множество концепций и парадигм программирования, позаимствованных из таких языков как, например, ADA или Smalltalk, но все равно оставшийся «ручным». В нем нет ни сборщика мусора, ни динамических стековых массивов, ни автоматического контроля границ, ни многих других вещей, которые по-прежнему приходится делать руками. А нет их там потому, что вся эта «автоматизация» заметно снижает производительность и превращает Си++ в пародию на Visual Basic, ярчайшим примером которой является C#. Ходят устойчивые слухи, что значительная часть Longhorn'a была написана на C#, но, несмотря на все усилия разработчиков, достичь приемлемой производительности и стабильности им так и не удалось (а что еще можно ожидать от Бейсика, пускай и продвигаемого под видом Си?). В конечном счете, компания была вынуждена похоронить миллионы строк, и начать разработку заново. Если не ошибаюсь, текущая версия Windows Vista базируется на коде Server 2003, написанном на смеси Си с Си++, а это значит, что отказ от Си/Си++ (по крайней мере, в крупных проектах) невозможен, и вместо того, чтобы жаловаться на судьбу, лучше придумать пару-тройку новых методов б орьбы с утечками и переполняющимися буферами, о чем мы сейчас и поговорим.

→ **переполняющиеся буферы.** Ошибок переполнения не избежала практически ни одна чуть более сложная, чем «hello, world!», программа, а все потому, что ошибки переполнения в Си носят

фундаментальный характер, и язык не предоставляет никаких механизмов для их преодоления.

Контроль границ буфера приходится осуществлять вручную. К тому же, если говорить по существу, в Си вообще нет никаких буферов. Получив указатель на «буфер», функция не может определить его длину. Оператор sizeof возвращает размер указателя (как правило, равный DWORD), но отнюдь не размер блока, на который он указывает. Небольшую лазейку оставляет нестандартная функция _msize, сообщающая размер динамического блока, но она требует указателя на его начало, отказываясь работать с серединой.

Простейшая тестовая программа все это наглядно подтверждает:

```
программа, демонстрирующая невозможность
определения размера блока по указателю
// функция, принимающая указатель
// и пытающаяся определить размер
соответствующего ему блока
foo(char *p){printf("sizeof says =
%Xh\n_msize says =
%Xh\n",sizeof(p),_msize(p));}

main()
{
  char buf[0x666]; char *p =
malloc(0x999);

  // передаем указатель на начало блока
  foo(buf);foo(p);

  // передаем указатель на середину блока
  foo(&buf[6]);foo(p+9);
}
```

После запуска мы получим следующие весьма неутешительные данные:

результат работы программы, определяющий размер блока по указателю

```
// указатель на начало стекового буфера
sizeof says = 4h, _msize says =
12E8CEh // sizeof и _msize провалились

// указатель на начало динамического
буфера
sizeof says = 4h, _msize says =
9A0h   // sizeof провалилась, _msize —
почти OK

// указатель на середину стекового буфера
sizeof says = 4h, _msize says =
FFFFFFFFh        // sizeof и _msize
провалились

// указатель на середину динамического
буфера
sizeof says = 4h, _msize says =
D200h  // sizeof и _msize провалились
```

Мы видим, что _msize ведет себя очень странно и когда не может определить размер блока, возвращает какой-то мусор, никак не сигнализируя об ошибке. Поэтому выполнять контроль должна вызывающая функция, передавая вызываемой размер буфера как аргумент. Отсюда и появились char *fgets(char *string, int n, FILE *stream); char *strncpy(char *strDest, const char *strSource, size_t count) и другие подобные функции. Теоретически, они на 100% застрахованы от переполнения, но вот практически… значение n приходится рассчитывать вручную, а, значит, существует риск ошибиться! К тому же, если длина строки превышает n, в буфер копируется лишь «огрызок», что само по себе является нехилым источником проблем и вторичных ошибок. Приходится навешивать специальный обработчик, выделяющий дополнительную память и считывающий оставшийся «хвост», что значительно усложняет реализацию, делает код более громоздким и менее наглядным. Обычно стараются выбрать n так, чтобы его значение превышало размер наибольшей строки, выделяя память с запасом и не обращая внимания на то, что большинство строк использует лишь малую часть буфера…

Нет! Память лучше всего выделять динамически, по мере возникновения в ней потребности! В идеале, строка должна представлять собой список (желательно двухсвязный), что не только ускорит операции удаления и вставки подстрок, но и ликвидирует проблему переполнения. О контроле границ заботиться уже необязательно, поскольку память под новые символы выделяется автоматически.

В простейшем случае каждый элемент списка выглядит приблизительно так:

```
struct slist
{
  unsigned char c;
  struct slist *prev;
  struct slist *next;
  struct slist *first;
  struct slist *last;
};
```

Это просто реализуется, но имеет дикий оверхид, требующий для хранения каждого символа 17 байт, поэтому на практике приходится использовать комбинированный способ, сочетающий в себе строковые буферы со списками:

```
#define STR_SIZE256
struct slist
{
  unsigned int len;
  unsigned char buf[STR_SIZE];
  struct slist *prev;
  struct slist *next;
  struct slist *first;
  struct slist *last;
};
```

Размер буфера может быть как фиксированным, так и динамическим. Хорошей стратег выделяет под первый элемент списка 64 байта, под второй 128 байт и так далее, вплоть до 1000h, что позволяет обрабатывать как длинные, так и короткие сроки с минимальным оверхидом.

Списки на 100% защищены от ошибок переполнения (исключение составляют попытки обработать строку свыше 2 Гб, вызывающую исчерпание свободной памяти, но, во-первых, исчерпание — это все-таки не переполнение, и заслать shell-код злоумышленник не сможет, а во-вторых, это явная ошибка, которую легко обработать, установив предельный лимит на максимально разумную длину строки).

Хуже другое. Реализовав свою библиотеку для работы со «списочными строками», мы будем вынуждены переписать все остальные библиотеки, создавая обертки для каждой строковой функции, включая fopen, CreateProcess и т. д., поскольку все они ожидают увидеть непрерывный массив байт, а вовсе не список! Это чрезвычайно утомительная работа, но зато когда она будет закончена, о переполнениях можно забыть раз и навсегда. Правда, производительность (за счет постоянного преобразования типов) падает весьма значительно…

А вот более быстрое, но менее надежное решение. Отказываемся от стековых буферов, переходя на динамическую память. Выделяем каждому блоку на одну страницу больше и присваиваем последней странице атрибут PAGE_NOACCESS, чтобы каждое обращение к ней вызывало исключение, отлавливаемое нашим обработчиком, который в зависимости от ситуации либо увеличивал размер буфера, либо завершал работу программы с сообщением об ошибке. На коротких строках оверхид весьма значителен, но на длинных он минимален. Но к сожалению, такая защита страхует лишь от последовательного переполнения, но бессильна предотвратить индексное (подробнее о видах переполнения можно прочитать в моей книжке «Shellcoders's programming uncovered», которую можно найти в Осле), к тому же, переход на динамические массивы порождает проблему утечек памяти и получается так, что одно лечим, а другое калечим.

Тем не менее, лишний раз подстраховаться никогда не помешает! Чтобы защититься от переполнения кучи (которое в последнее время приобретает все большую популярность) после вызова любой функции, работающей с динамической памятью, необходимо защищать служебные данные кучи атрибутом PAGE_NOACCESS, а перед вызовом функции — снимать их. Для этого нам, опять-таки, потребуется написать обертки вокруг всех функций менеджера памяти, что требует времени. К тому же, в реализации кучи от Microsoft Visual C++ служебные данные лежат по смещению -10h от начала выделенного блока, а защищать мы можем только всю страницу целиком. Поэтому, во-первых, необходимо увеличить размер каждого

блока до 512 Кбайт, чтобы начальный адрес совпадал с началом страницы, а во-вторых, использовать блок только со второй страницы. В результате, при работе с мелкими блоками мы получаем чудовищный оверхид, но зато компенсируемый надежной защитой от переполнения. Так что данный метод, при всех его недостатках, все-таки имеет право на жизнь.

➜ **утечки ресурсов** возникают всякий раз, когда функция выделяет блок памяти, открывает файл, но при выходе забывает его освободить/закрыть. Чаще всего это происходит при преждевременном выходе из функции. Рассмотрим следующий пример:

```
foo()
{
 FILE *ff
 char *p1, *p2;
 p1 = malloc(XXL);
 ff = fopen(FN,"r");
 …
 if (bar()==ERROR) return -1;
 …
 p2 = malloc(XXL);
 …
 free(p1);
 free(p2);
 fclose(ff);
 return 0;
}
```

Функция foo намеревается выделить два блока памяти p1 и p2, но реально успевает выделить лишь один из них, после чего bar завершается с ошибкой, делающей дальнейшее выполнение foo невозможным, — вот программист и совершает возврат по return, забывая о выделенном блоке памяти.

Проблема в том, что в произвольной точке программы очень непросто сказать, какие ресурсы уже выделены, а какие — еще нет и что именно нужно освобождать! Ну ведь не поддерживать же ради этого транзакции?! Разумеется, нет. Проблема имеет весьма простое и элегантное решение, основанное на том, что Стандарт допускает освобождение нулевого указателя. Правда, к файлам и другим объектам это уже не относится, но проверку на нуль легко выполнить и вручную.

Правильно спроектированный код должен выглядеть приблизительно так:

```
foo()
{
 int error = 0;
 FILE *ff = 0;
 char *p1 = 0; char *p2 = 0;
 {
   p1 = malloc(XXL);
   ff = fopen(FN,"r");
   …
   if ((bar()==ERROR) && (error=-1))
break;
   …
   p2 = malloc(XXL);
   …
 } while(0);
 free(p1); free(p2);
 if (ff) fclose(ff);
 return error;
}
```

Что изменилось? Абсолютно все! Теперь для внепланового выхода из программы (который осуществляется по break), нам уже не нужно помнить, что мы успели выделить или открыть! По завершении цикла while (который на самом деле никакой не цикл, а просто имитация критикуемого оператора goto), мы освобождаем (или, точнее, пытаемся освободить) все ресурсы, которые потенциально могли быть выделены. Структура программы значительно упрощается, и главное тут — не забыть освободить все, что мы выделили, но программисты об этом все равно забывают.

Решение заключается в создании своей собственной обертки вокруг функции malloc (условно назовем ее my_malloc), которая выделяет память, запоминает указатель в своем списке/массиве и перед возвращением в вызываемую функцию подменяет адрес возврата из материнский функции на свой собственный обработчик (конечно, без ассемблерных вставок тут не обойтись, но они того стоят). Как следствие — при выходе из foo управление получает обработчик my_malloc, читающий список/массив и автоматически освобождающий все, что там есть, снимая тем самым эту ношу с плеч программиста.

Если же выделенная функцией память по замыслу разработчика не должна освобождаться после ее завершения, на этот случай можно предусмотреть специальный флаг, передаваемый my_malloc, и сообщающий, что этот блок освобождать не надо и программист освободит его сам.

Одним их самых мерзких недостатков языка Си является отсутствие поддержки динамических стековых массивов. Стековая память хороша тем, что автоматически освобождается при выходе из функции, однако, выделение стековых массивов «на лету» невозможно, и мы должны заранее объявить их размеры при объявлении переменных. В C++ сделаны небольшие подвижки в этом направлении, и теперь мы можем объявлять массивы, размер которых задается аргументом, передаваемым функции, но это не решает всех проблем, и к тому же C++ поддерживают далеко не все компиляторы.

В частности, компилятор GCC 2.95 нормально «переваривает» следующий код, а Microsoft Visual C++, увы, нет:

```
f(int n)
{
 char buf[n];
 return sizeof(buf);
}
```

На самом деле, выделять динамические массивы все-таки возможно, но только в том случае если компилятор, во-первых, адресует локальные переменные через EBP, а во-вторых, в эпилоге использует конструкцию MOV ESP, EBP вместо ADD ESP, n. К таким компиляторам, в частности, относится Microsoft Visual C++, автоматически переходящий на адресацию локальных переменных через регистр EBP, если в теле функции присутствует хотя бы одна ассемблерная вставка.

Фрагмент одной из таких функций приведен ниже (компилировано Microsoft Visual C++ с максимальной оптимизацией (ключ /Ox):

```
text:00000010            push    ebp
text:00000011            mov     ebp, esp
text:00000013            push    esi
…
text:0000002B            sub     esp,
400h
…
text:00000034            mov     eax,
[ebp+var_4]
…
text:00000048            pop     esi
text:00000049            mov     esp, ebp
text:0000004B            pop     ebp
text:0000004C            retn
```

Выделение памяти на стеке осуществляется путем «приподнимания» регистра-указателя стека на некоторую величину, что можно сделать командой «SUB ESP, n», где n — количество выделяемых байт. Поскольку компилятор адресует локальные переменные через регистр EBP, то изменение ESP не нарушит работы функции и все будет OK, но… так будет продолжаться недолго. При выходе из функции она попытается восстановить регистры, сохраненные на входе (в данном случае — это регистр ESI), но на вершине перемещенного стека их не окажется! В регистр ESI попадет мусор, и материнская функция рухнет.

Существует, по меньшей мере, два решения проблемы: либо вручную опускаем регистр ESP при выходе из функции (если, конечно, не забудем это сделать), либо копируем на вершину выделенного блока порядка 20h байт памяти с макушки старого стека (обычно этого более чем достаточно: даже если функция сохраняет все регистры общего назначения, ей требуется всего лишь 1Ch байт). В этом случае о ручном освобождении выделенной памяти можно не заботиться. Это сделает машинная команда MOV ESP, EBP, помещенная компилятором в эпилог функции.

Ниже приведена пара макросов для динамического выделения освобождения стековой памяти (только для Microsoft Visual C++):

```
#define stack_alloc(p,n,total)
{ __asm{sub esp,n};\
    __asm{mov dword ptr ds:[p],esp};\
```

```
      total += n;}
#define stack_free(total)
{__asm{add esp,total};}
```

А вот пример использования макросов stack_alloc и stack_free:

```
foo()
{
 char* p; int n;
 int total = 0;

 n = 0x100;
 stack_alloc(p, n, total);

 strcpy(p,"hello, world!\n");printf(p);

 stack_free(total);
}
```

Естественно, о вызове stack_free программист может забыть (и ведь наверняка забудет!), поэтому лучше выделять память так, чтобы при выходе из функции она освобождалась автоматически.

Ниже приведен исходный текст макроса auto_alloc, который именно так и работает:

```
#define auto_alloc(p,n)
{__asm{add n,20h};\
      __asm{mov eax,esp};\
      __asm{sub esp,n};\
      __asm{mov p,esp};\
      __asm{push 20h};\
      __asm{push eax};\
      __asm{mov eax,p};\
      __asm{push eax};\
      __asm{call memcpy};\
      __asm{add esp,0Ch};\
      __asm{add p,20h};}
```

Как же его можно использовать на практике? Хотя бы так:

```
foo()
{
 char* p; int n; n = 0x100;
 auto_alloc(p, n);

 strcpy(p,"hello, world!\n");printf(p);
}
```

При работе со стековой памятью следует учитывать три обстоятельства. Во-первых, по умолчанию каждый поток получает всего лишь 1 Мбайт стековой памяти, что по современным понятиям очень мало. Стековую память первичного потока легко увеличить, передав линкеру ключ «/STACK:reserve[,commit]», где reserve — зарезервированная, а commit — выделенная память. Размер стековой памяти остальных потоков определяется значением аргумента dwStackSize функции CreateThread.

Во-вторых, при старте потока Windows выделяет ему минимум страниц стековой памяти, размещая за ними специальную «сторожевую» страницу (PAGE GUARD), при обращении к которой возбуждается исключение, отлавливаемое системой, которая выделяет потоку еще несколько страниц, перемещая PAGE GUARD наверх. Если же мы попытаемся обратиться к памяти, лежащей за PAGE GUARD — произойдет крах. Поэтому, при ручном выделении стековой памяти необходимо последовательно обратиться хотя бы к одной ячейке каждой страницы: #define stack_out(p,n) for(a=0;a<n;a+=0x100)t=p[a];.

В-третьих, размер выделяемых блоков памяти должен быть кратен четырем, иначе многие API и библиотечные функции откажут в работе.

Но что делать, если, несмотря на все усилия, память продолжает утекать? На этот случай у мыщъх'а припасено несколько грязных, но довольно эффективных трюков. Вот один из них: когда количество потребляемой приложением памяти достигает некоторой, заранее заданной отметки, мы «прогуливаемся по куче» API-функцией HeapWalk, сохраняя все выделенные страницы в специальный файл (устроенный по принципу файла подкачки) и возвращаем память системе, оставляя страницы зарезервированными и назначая им атрибут PAGE_NOACCESS. После чего нам остается только отлавливать исключения и подгружать содержимое реально используемых страниц, восстанавливая оригинальные атрибуты доступа (PAGE_READ или PARE_READWRITE). В результате, утекать будет только адресное пространство, которое, между прочим, не бесконечно, и при интенсивной течи довольно быстро кончается. И что же тогда? Можно, конечно, просто завершить программу, но лучше рискнуть и попробовать освободить блоки, к которым дольше всего не было обращений. Разумеется, мы не можем гарантировать, что именно они ответственны за утечку памяти. Быть может, программа в самом начале выделила несколько блоков, планируя обратиться к ним при завершении процесса, но… риск благородное дело!

➔ **заключение.** Помимо рассмотренных нами, существуют и другие методы борьбы с утечками и переполняющимися буферами. Для мира BSD/LINUX характерны run-time верификаторы, встраиваемые непосредственно в сам компилятор (ведь его исходные тексты доступны). Под Windows более популярны статические анализаторы — потомки древнего LINT. Но всем им свойственны недостатки, поэтому настоящие программисты никогда не останавливаются на достигнутом, а неуклонно движутся вперед, выдумывая все новые приемы и трюки. Одни сметаются временем, другие получают широкое распространение, попадая в учебники и справочные руководства.

Но лучшее руководство — это свой собственный опыт, который и описал мыщъх, впрочем, не претендуя на новизну и новаторство

болевой прием

АССЕМБЛЕР ПРОТИВ СИ

ДЕЙСТВИТЕЛЬНО ЛИ ЭФФЕКТИВНЫ (НЕЭФФЕКТИВНЫ) СИ-КОМПИЛЯТОРЫ, И НАСКОЛЬКО БОЛЬШЕ МОЖНО ВЫИГРАТЬ, ПЕРЕПИСАВ ПРОГРАММУ НА ЯЗЫКЕ АССЕМБЛЕРА? КАКУЮ ЦЕНУ ЗА ЭТО ПРИДЕТСЯ ЗАПЛАТИТЬ? ПОСТАРАЕМСЯ ДАТЬ ПРЕДЕЛЬНО ОБЪЕКТИВНЫЙ И НЕПРЕДВЗЯТЫЙ ОТВЕТ

Крис Касперски ака мыщъх
no e-mail

Любовь хакеров к ассемблеру вполне понятна и объяснима. Разве не заманчиво знать язык, которым владеют немногие? Ассемблер окружен мистическим ареолом — это символ причастности к хакерским кругам, своеобразный пропуск в клан системных программистов, вирусописателей и взломщиков. Ассемблер теснее всех других языков приближен к железу, и ниже его находятся только машинные коды, уже вышедшие из употребления, а жаль!

Программисты каменного века с презрением относились к ассемблеру, поскольку для тех времен он был слишком высокоуровневым языком, абстрагирующимся от целого ряда архитектурных особенностей. Программируя на ассемблере, можно не знать последовательность байт в слове, систему кодирования машинных инструкций; ассемблер скрывает тот факт, что команда «ADD AL, 6h» может быть закодирована и как «04h 06h», и как «80h C0h 06h». Хуже того: ассемблер не предоставляет никаких средств выбора между этими вариантами. Хорошие трансляторы автоматически выбирают наиболее короткий вариант, но никаких гарантий, что они это сделают, нет, а в самомодифицирующемся коде это весьма актуально! Да что там самомодифицирующийся код (или код, использующий коды мнемоник как константы) — на ассемблере невозможна эффективная реализация выравнивания команд! Тупая директива align, вставляющая NOP'ы, не в счет. В частности, «ADD AL, 6h», закодированная как «80h C0h 06h», намного эффективнее, чем «04h 06h + 90h (NOP)».

Кто-то, наверняка, скажет: «Ассемблер позволяет закодировать любую команду через директиву DB, следовательно, на нем можно делать все». Весьма спорное утверждение. Си также позволяет объявлять массивы вида unsigned char buf[] = «\x04\x06\x90» и умеет преобразовывать указатели на данные в указатели на функции. Рассуждая по аналогии, можно сказать, что на Си легко сделать то же самое, что и на ассемблере, даже не используя ассемблерных вставок (которые, на самом деле, не часть языка, а самостоятельное расширение). Но вряд ли программу, полностью состоящую из «\x04\x06\x90», можно назвать программой на языке Си. Точно так же и с ассемблером. Это вовсе

не язык неограниченных возможностей, каким его иногда представляют. Ассемблер — всего лишь средство выражения программистской мысли, рабочий инструмент. А выбор инструмента всегда должен быть адекватен. Не стоит рыть траншею лопатой, если под рукой есть экскаватор, но и строить собачью конуру из бетонных блоков с помощью крана — не верх инженерной культуры, а признак ее отсутствия.

Считается, что программа, написанная на ассемблере, по определению компактнее и производительнее аналогичной программы, написанной на языке высокого уровня. Действительно, человек всегда в состоянии обогнать даже самый совершенный компилятор, потому что компилятор действует по строго заданному шаблону (точнее, нескольким шаблонам), а человек способен на принципиально новые решения. Однако, ассемблерные программы, написанные начинающими программистами, как правило, значительно хуже кода, сгенерированного компилятором. Распределение переменных по регистрам, устранение зависимостей по данным, переупорядочивание инструкций с целью предотвращения простоев конвейера — это слишком нудная работа, отнимающая кучу сил и времени. И хотя человек потенциально способен добиться намного лучшего распределения, чем компилятор, этот разрыв не настолько велик и с коммерческой точки зрения ничем не окупается.

Ассемблерная программа, оптимизированная вручную, становится совершенно немобильной. Если потребуется внести в код даже незначительные изменения или выйдет процессор с новыми правилами оптимизации — всю работу придется начинать заново. А на языке высокого уровня просто перекомпилировал, и все! Большинство программных комплексов, написанных на Си/Си++, никогда бы не увидели свет, если бы в качестве основного языка разработки был выбран ассемблер, требующий неимоверной концентрации труда. Механизация придумана как раз для того, чтобы облегчать человеку жизнь и воплощать его грандиозные замыслы. Никто же не спорит, что на дачном участке ручной уход за растениями дает намного больший урожай, чем тракторист на колхозном поле, но обработать колхозное поле вручную практически невозможно!

Несколько десятилетий тому назад, когда счет памяти шел на килобайты и приходилось экономить каждый такт, ручная оптимизация еще имела смысл, поскольку откомпилированные программы на массовом железе тормозили со страшной силой, но сейчас все изменилось. Системные требования уже давно перестали быть основным потребительским фактором. Теперь в ассемблере нуждаются лишь критичные к быстродействию модули, связанные с обработкой огромного количества данных в реальном времени. Во всех остальных случаях лучше воспользоваться качественным оптимизирующим компилятором (например, Microsoft Visual C++, GCC 2.95).

Результат трансляции crc() компилятором MS VC++ 6.0 **(1)**

```
.text:00000000                  _CRC    proc near
.text:00000000
.text:00000000                  var_1   = dword ptr -1
.text:00000000                  arg_0   = dword ptr 7
.text:00000000                  arg_4   = dword ptr 0Bh
.text:00000000
.text:00000000 51                       push  ecx
.text:00000001 8B 54 24 0C              mov   edx, [esp+1+arg_4]
.text:00000005 32 C9                    xor   cl, cl
.text:00000007 33 C0                    xor   eax, eax
.text:00000009 88 4C 24 00              mov   byte ptr [esp+1+var_1], cl
.text:0000000D 85 D2                    test  edx, edx
.text:0000000F 7E 16                    jle   short loc_27
.text:00000011 53                       push  ebx
.text:00000012 56                       push  esi
.text:00000013 8B 74 24 10              mov   esi, [esp+9+arg_0]
.text:00000017
.text:00000017          loc_17:
.text:00000017 8A 1C 30 mov             bl, [eax+esi]
.text:0000001A 02 CB                    add   cl, bl
.text:0000001C 40                       inc   eax
.text:0000001D 3B C2                    cmp   eax, edx
.text:0000001F 7C F6                    jl    short loc_17
.text:00000021 5E                       pop   esi
.text:00000022 88 4C 24 04              mov   byte ptr [esp+5+var_1], cl
.text:00000026 5B                       pop   ebx
.text:00000027
.text:00000027          loc_27:
.text:00000027 8B 44 24 00              mov   eax, [esp+1+var_1]
.text:0000002B 25 FF 00 00+             and   eax, 0FFh
.text:00000030 F7 D8                    neg   eax
.text:00000032 59                       pop   ecx
.text:00000033 C3                       retn
.text:00000033 _CRC     endp
```

Ручная ассемблерная реализация crc() **(2)**

```
00000000: 51                            push  ecx
 00000001: 8B4C240C                     mov   ecx,[esp+arg_p]
 00000005: 8B542408                     mov   edx,[esp+arg_n]
 00000009: 03CA                 add     ecx,edx
 0000000B: 33C0                 xor     eax,eax
 0000000D: EB03                 jmps    000000012
 0000000F: 0201                 add     al,[ecx]
 00000011: 41                           inc   ecx
 00000012: 3BCA                 cmp     ecx,edx
 00000014: 72F9                 jb      00000000F
 00000016: 59                           pop   ecx
 00000017: C3                           retn
```

Более новые версии компиляторов в основном пекутся о качестве поддержки очередной редакции Си++-стандарта, оставляя оптимизирующий движок без изменений, поскольку новых идей ни у кого нет. Единственное исключение составляет Intel Fortran/C++, реализующий режим глобальной оптимизации (остальные компиляторы оптимизируют код только в пределах одной функции) и проложивший мост между профилировщиком и компилятором. Используя данные профилировки, компилятор, в частности, может размещать в регистрах наиболее интенсивно используемые переменные, а остальные — гнать в память. К сожалению, эта методика далека от совершенства, и хотя «официально» Intel C++ обгоняет GCC, с этим согласны далеко не все: поклонники GCC демонстрируют множество программ, на которых Intel C++ показывает преимущества.

➜ **эффективность кодогенерации си-компиляторов.** Желая продемонстрировать превосходство ассемблера над Си, обычно берут программы типа «hello, world!» и сравнивают размеры откомпилированных файлов, причем ассемблер использует прямые вызовы API-функций GetStdHandle()/WriteFile(), а программу на Си заставляют обращаться к printf(), которая тащит за собой библиотеку времени исполнения (Run Time Library или сокращенно RTL). Ну, и где же здесь честность?! Как будто на Си нельзя программировать без RTL! Можно! Для этого достаточно переименовать функцию main() во что-то другое, указав линкеру точку входа в файл вручную. Размер откомпилированного файла сразу же сократится, вплотную приближаясь к ассемблированному файлу (или даже совпадая с ним). Но 99% пространства будет занимать служебная информация PE-формата и место, оставленное линкером для выравнивания секций. На этом фоне различия между компилятором и ассемблером становятся совершенно незаметными!

Мы же поступим иначе. Напишем небольшую программку, вычисляющую CRC8 (большая просто не уместилась бы в статье), а затем откомпилируем ее и, прогнав полученный файл через дизассемблер, посмотрим, насколько эффективно компилятор справился со своей задачей и какой простор он оставил нам для ручной оптимизации.

Исходный текст подопытной функции выглядит так (ключевой фрагмент демонстрационной программы CRC.c):

```
CRC(unsigned char *p, int n)
{
  int a; unsigned char crc=0;
  for (a=0;a<n;a++) crc += p[a];
  return 0 – crc;
}
```

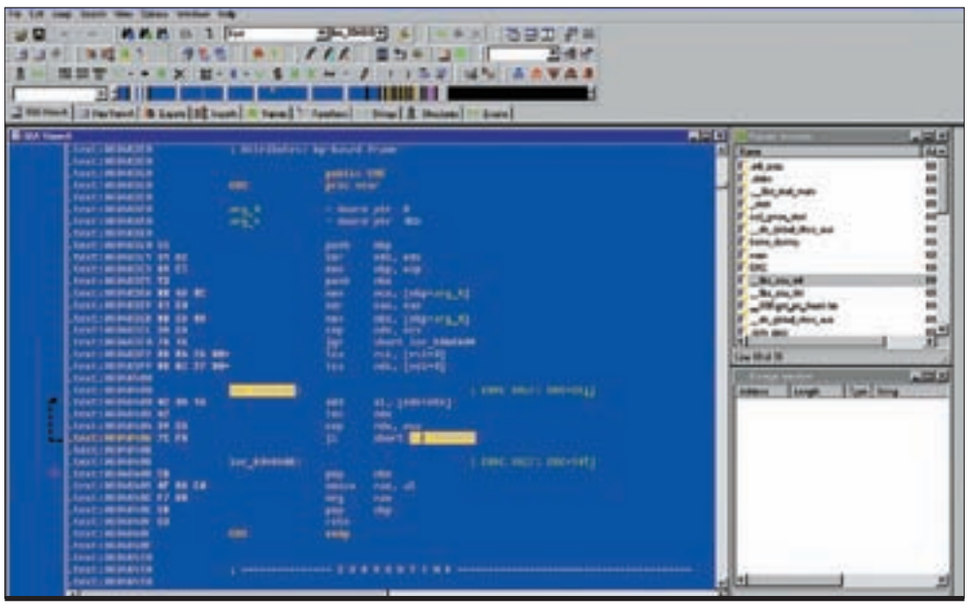
Компилируем ее Microsoft Visual C++ в режиме максимальной оптимизации (ключ Ox — «cl.exe /Ox crc.c») и загружаем полученный obj в дизассемблер (смотри листинг 1).

Сразу бросается в глаза, что компилятору не хватило регистров (!) и счетчик контрольной суммы зачем-то задвинулся в локальную переменную. Впрочем, это не сильно сказалось на производительности, поскольку «задвижение» произошло после выхода из цикла, но сам факт! А ведь чтобы исправить ситуацию, всего-то и требовалось заменить «MOV byte ptr [ESP+5+var_1], CL/MOV EAX,[ESP+1+var_1]/AND EAX, 0FFh» на «MOVZX EAX,CL», что гораздо короче и намного быстрее, особенно если n невелико и функция вызывается большое количество раз. Другое замечание — компилятору потребовалось целых 5 (!) регистров, в то время как для решения данной задачи вполне достаточно 3-х: один — на сумматор CRC, один — на указатель, и еще один — на адрес конца. Ассемблер-пример с ручной оптимизацией приведен на листинге 2.

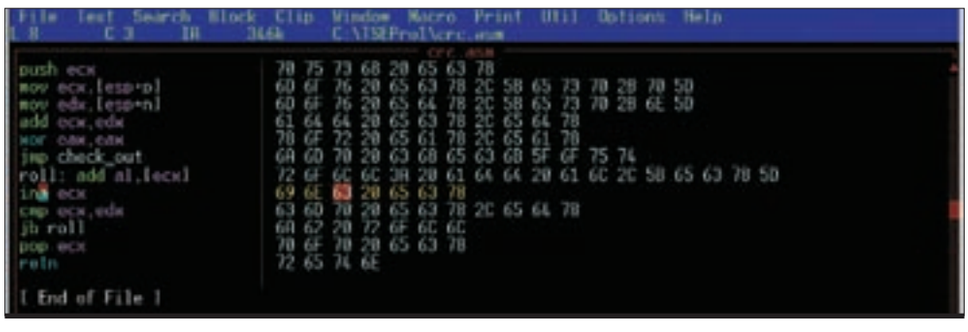
18h ассемблерных байт (и 12 команд) против 34h откомпилированных байт (и 23 команд) — для классического цикла это хороший результат. Что же тогда говорить о нетривиальных вещах: сложных структурах данных, циклах с высоким уровнем вложенности, многомерных массивах и так далее. С другой стороны, не все так плохо. Компилятор успешно распознал конструкцию «return 0 — crc» и вместо тупого вычитания из нуля подобрал адекватную машинную команду NEG, означающую «дополнение до нуля».

А что на счет GCC? Для начала возьмем древнюю, но все еще широко используемую версию 2.95, отличающуюся стабильностью и высокой скоростью трансляции. На самом высоком уровне оптимизации (ключ -O3: «gcc crc.c -O3 -o crc»), компилятор генерирует код, показанный на листинге 3.

В отличие от MS VC, компилятору GCC хватило всего 4-х регистров без обращения к локальным переменным, а сам код уложился в 30h байт, что на 4 байта короче, чем у конкурента. Но до ручной ассемблерной оптимизации все равно еще далеко. Однако, если присмотреться к телу цикла повнимательнее, можно обнаружить, что GCC, в отличие от MS VC, совместил счетчик цикла с инкрементом указателя, то есть откомпилированный цикл исполняется практически с той же самой степенью эффективности, что и ручной. «Практически» — потому что в отличие от нас



IDA Pro за работой



Набор ассемблерной программы в редакторе TSE Pro

компилятор использовал сложную адресацию «ADD AL, [EDX+EBX]», напрягающую процессор и требующую нескольких экстра-тактов на декодирование (впрочем, к последним версиям P-4 и Athlon это уже не относится).

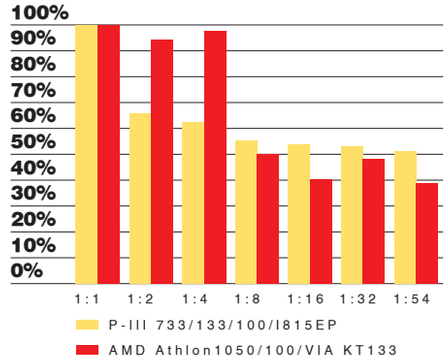
Причем цикл выровнен по адресам, кратным 10h (команды «LEA ESI,[ESI+0]» и «LEA EDI,[EDI+0]» используются для выравнивания), что, с одной стороны, хорошо, а с другой — не очень. Начиная с процессоров поколения P6 (к которым, в частности, принадлежит Pentium Pro и Penium-II) и AMD K5, выравнивание циклов требуется только тогда, когда целевой переход попадает на команду, расщепленную двумя линейками кэш-памяти первого уровня, длина которых, в зависимости от процессора, составляет 32, 64 или даже 256 байт. В противном случае, наблюдается существенное снижение быстродействия.

Компилятор MS VC вообще не выравнивает циклы, поэтому их быстродействие зависит от воли случая — попадет ли первая инструкция цикла на «расщепляющий» адрес или не попадет. Компилятор GCC выравнивает циклы, но по слишком ретивой стратегии, всегда подтягивая их к адресам, кратным 10h. Это хорошо работает на «первопнях», но вот на более современных процессорах команды, расходующиеся на выравнивание, занимают лишнее место и впустую съедают производительность (особенно на вложенных циклах, где они выполняются многократно).

Зато, в отличие от своего конкурента, GCC «догадался» использовать команду «MOVZX EAX, AL», только вот зачем она ему понадобилась — непонятно. Команда «MOVZX» пересылает значение из источника в приемник, дополняя его нулями до 16- или 32-бит (в зависимости от разрядности). Но в нашем случае старшие биты регистра EAX уже равны нулю, поскольку компилятор сам обнулил их инструкцией «XOR EAX,EAX». Следовательно, команда «MOVZX EAX,AL» совершенно не нужна и избыточна.

Интересно, изменилось ли что-нибудь в новых версиях? Компилируем программу с помощью GCC 3.4.2 и смотрим полученный результат на листинге 4.

Ага! Программа сократилась до 20h байт, что всего на 8 байт длиннее ассемблерной программы, но цикл практически не изменился. Попрежнему используется сложная адресация и никому не нужная команда «MOVZX». Изменилась только точка входа в цикл. Вместо того, чтобы проверять значение аргумента n до входа в цикл, компилятор сформировал условный переход на проверку, выполняемую перед выходом из цикла. То есть использовал тот же самый трюк, что и мы в нашей ассемблерной программе, однако, в отличие от нас, компилятор использовал 4 регистра, а не 3, и к тому же сгенерировал стандартный пролог/эпилог, который, при желании, можно подавить ключами командной строки, но это все равно не поможет, поскольку цикл остается неразвернутым (под «разворотом» в общем случае понимается многократное дублирование цикла). На таком крохотном «пяточке» процессору просто негде развернуться, поэтому все будет очень жутко тормозить — как при ручной оптимизации, так и при машинной. Компилятор MS VC вообще не умеет разворачивать циклы. По жизни. А GCC умеет, но по умолчанию не делает этого даже на уровне оптимизации O3, разве что его специально попросить, указав ключ -funroll-all-loops в командной строке. Циклы с известным количеством итераций, где const <= 32 разворачиваются полностью, при const > 32 — на ~4-х (точное значение зависит от количества инструкций в теле цикла), но циклы с неизвестным количеством итераций (то есть такие циклы, параметр которых — переменная, а не константа) не разворачиваются вообще! И в этом есть свой резон.



Влияние кратности разворота цикла на производительность на различных типах процессоров

При малых количествах итераций цикл лучше не разворачивать, поскольку выигрыш в скорости не окупится увеличением размера программы и усложнением логики, особенно если цикл расположен внутри редко вызываемой функции. А вот разворот часто вызываемого цикла с большим количеством итераций дает, по меньшей мере, двухкратный прирост производительности. Главное — не переборщить и не развернуть цикл сильнее, чем требуется. На большинстве процессоров рост скорости прекращается при развороте на 8 итераций, и дальнейшее дублирование тела цикла лишь увеличивает его размер.

Выполнить разворот цикла можно как на ассемблере (на MASM'е, за счет поддержки развитой системы макрокоманд, он реализуется особенно легко), так и на любом языке высокого уровня, действуя в обход компилятора.

После «ручной» оптимизации исходный текст нашей программы будет выглядеть таким

SPECIAL МНЕНИЕ

ДМИТРИЙ КОВАЛЕНКО
Системный программист, разработчик в Infopulse Ukraine

Сравнивать два языка программирования — дело неблагодарное, особенно такие мощные, как ассемблер и C++.

У C++ свои плюсы. Во-первых, это объектно-ориентированный подход к написанию программ, что позволяет без проблем разрабатывать и сопровождать большие проекты — в отличие от ассемблера, на котором написать что-либо большое не так-то просто. Во-вторых, программы, написанные на C++, проще переносятся на другие платформы. Кроме того, хотим мы этого или нет, но C++ является стандартом «де факто» во многих областях программирования — например, в программировании 3d, разработке драйверов и т.п.

У ассемблера свои плюсы. Во-первых, это размер и скорость кода. Во-вторых, есть программы, которые можно написать только на ассемблере — например, полиморфные вирусы, переносимый код и т.п. Ну, и кроме того, ассемблер считается истинно хакерским языком ;). Окончательный выбор за тобой…

Результат трансляции crc() компилятором GCC 2.95 **(3)**

```
.text:080483E0     CRC          proc near
.text:080483E0
.text:080483E0          arg_0    = dword ptr 8
.text:080483E0          arg_4    = dword ptr 0Ch
.text:080483E0
.text:080483E0 55               push      ebp
.text:080483E1 31 D2            xor       edx, edx
.text:080483E3 89 E5            mov       ebp, esp
.text:080483E5 53               push      ebx
.text:080483E6 8B 4D 0C mov     ecx, [ebp+arg_4]
.text:080483E9 31 C0            xor       eax, eax
.text:080483EB 8B 5D 08 mov     ebx, [ebp+arg_0]
.text:080483EE 39 CA            cmp       edx, ecx
.text:080483F0 7D 16            jge       short loc_8048408
.text:080483F2 8D B4 26 00+     lea       esi, [esi+0]
.text:080483F9 8D BC 27 00+     lea       edi, [edi+0]
.text:08048400
.text:08048400     loc_8048400:
.text:08048400 02 04 1A add     al, [edx+ebx]
.text:08048403 42               inc       edx
.text:08048404 39 CA            cmp       edx, ecx
.text:08048406 7C F8            jl        short loc_8048400
.text:08048408
.text:08048408     loc_8048408:
.text:08048408 5B               pop       ebx
.text:08048409 0F B6 C0 movzx   eax, al
.text:0804840C F7 D8            neg       eax
.text:0804840E 5D               pop       ebp
.text:0804840F C3               retn
.text:0804840F          CRC     endp
```

Результат трансляции crc() компилятором GCC 3.4.2 **(4)**

```
.text:080484C0          CRC     proc near
.text:080484C0
.text:080484C0          arg_0   = dword ptr 8
.text:080484C0          arg_4   = dword ptr 0Ch
.text:080484C0
.text:080484C0 55               push      ebp
.text:080484C1 89 E5            mov       ebp, esp
.text:080484C3 8B 4D 0C mov     ecx, [ebp + arg_4]
.text:080484C6 53               push      ebx
.text:080484C7 31 C0            xor       eax, eax
.text:080484C9 8B 5D 08 mov     ebx, [ebp+arg_0]
.text:080484CC 31 D2            xor       edx, edx
.text:080484CE EB 04            jmp       loc_80484D4
.text:080484D0
.text:080484D0     loc_80484D0:
.text:080484D0 02 04 13 add     al, [ebx+edx]
.text:080484D3 42               inc       edx
.text:080484D4
.text:080484D4     loc_80484D4:
.text:080484D4 39 CA            cmp       edx, ecx
.text:080484D6 7C F8            jl        short loc_80484D0
.text:080484D8 0F B6 C0 movzx   eax, al
.text:080484DB F7 D8            neg       eax
.text:080484DD 5B               pop       ebx
.text:080484DE C9               leave
.text:080484DF C3               retn
```

образом (оптимизированный вариант crc() с развернутым циклом):

```
if ((a=n)>3)
  // обрабатываем первые n - (n % 4)
итераций
  for (a = 0; a < n — 3; a += 4)
  {
    crc_1 += p[a+0];
    crc_2 += p[a+2];
    crc_3 += p[a+3];
    crc_4 += p[a+4];
  }

// обрабатываем оставшийся «хвост»
for (a = n — x % 4; a < x; a++)
crc += p[a];

// складываем все воедино
crc += crc_1 + crc_2 + crc_3 + crc_4;
```

При сравнении со своим ассемблерным аналогом (так же развернутым), программа покажет практически идентичный результат по скорости и будет вполне сопоставима по размеру, поскольку львиная доля кода придется на развернутое тело цикла, на фоне которого меркнут мелкие накладные расходы. Но! Это справедливо только для циклов с большим количеством итераций, берущих данные из медленной оперативной памяти, а то и считывающих их с диска. Тут на отдельных машинных командах можно не экономить, главное — выбрать правильную стратегию разворота, а она для каждого из процессоров разная.

Часто вызываемые циклы с небольшим количеством итераций по-прежнему эффективнее писать на ассемблере. Если, конечно, мы вообще заботимся о производительности...

Кстати, обрати внимание, что в развернутом цикле ячейки памяти складываются в различные переменные, а не суммируются в одну! Если же неправильно развернуть цикл, выигрыш в производительности окажется намного меньшим:

```
// выполняем первые n - (n %4) итераций
for(a = 0; a < n — 3; a += 4)
{
  crc += p[a+0] + p[a+1] + p[a+2] + +p[a+3];
}
```

Почему?! Да потому что образуется паразитная зависимость по данным. Процессор не может выполнять «+ p[a+1]», пока не завершится сложение crc с p[a+0], и вычислительный конвейер вынужден простаивать в ожидании!

→ **напоследок.** И все же есть области, в которых ассемблер необходим, можно даже сказать, неизбежен. Это, в первую очередь, высокопроизводительные математические и графические библиотеки, использующие векторные инструкции типа MMX или SSE **C**

ИГРАЙ, ПОКА МОЛОДОЙ - ЧИТАЙ «PC ИГРЫ»!

СВЕЖИЙ НОМЕР УЖЕ В ПРОДАЖЕ

- DARK MESSIAH OF MIGHT AND MAGIC
- GAMES CONVENTION
- КИБЕРЖЕНЩИНА ТВОЕЙ МЕЧТЫ
- TOM CLANCY'S RAINBOW SIX VEGAS
- ROME: TOTAL WAR - ALEXANDER

ДВА двухслойных DVD

общий объем 17GB!

и многое-многое другое...

жонглирование ядром

КЕРНЕЛ–КОДИНГ

ДАЖЕ ЕСЛИ ТЫ НЕ СОБИРАЕШЬСЯ ПИСАТЬ ДРАЙВЕРА, ЭТА СТАТЬЯ ПРИГОДИТСЯ ДЛЯ ПОНИМАНИЯ НЕКОТОРЫХ ВНУТРЕННИХ ОСОБЕННОСТЕЙ ОС WINDOWS. НЕСМОТРЯ НА ТО, ЧТО MICROSOFT ЗАБОТИТСЯ О ПРОГРАММЕРАХ И ПРЕДОСТАВЛЯЕТ ДОСТАТОЧНО ПОДРОБНУЮ ИНФОРМАЦИЮ ОБ API, АРХИТЕКТУРА ОС ОСТАЕТСЯ НАИМЕНЕЕ ОТКРЫТОЙ, А ИНФОРМАЦИЯ — ОБРЫВОЧНОЙ

Михаил Фленов ака Horrific
www.vr-online.ru

→ **набор инструментов.** Для создания драйвера под Windows необходим специальный набор инструментов, который называется DDK (Driver Development Kit). Помимо этого, желательно установить специализированную версию Windows, которая содержит отладочную информацию. Это поможет в отладке драйвера, что не такое уж и простое занятие.

Для каждой версии ОС существует свой набор DDK, и распространяется он отдельно от компилятора. Это значит, что если даже у тебя установлена полная версия Visual Studio, DDK придется ставить отдельно. Причем просто так скачать его с сайта Microsoft нельзя (на момент написания этих строк DDK не доступен, хотя пару лет назад был в свободном доступе), потому что он распространяется вместе с платной подпиской MSDN. Но если у тебя есть лишняя денежка, то DDK можно заказать здесь: www.microsoft.com/whdc/devtools/ddk/default.mspx.

На самом деле, DDK дают на халяву, деньги же нужны за пересылку компакт-диска. Существуют и неофициальные версии и залежи наборов для разработчиков драйверов — ищи в поисковиках. За пять минут реально найти полную версию для Windows XP. Ссылки давать бессмысленно, так как на момент выхода номера они могут быть уже мертвы…

→ **язык программирования.** Стандартом при написании драйверов является язык C. Да, именно он. Конечно, можно ухитриться написать на C++, но я не рекомендовал бы играть с объектами. Можно написать драйвер даже на Delphi, но только старой версии (до Delphi 5), и для сборки проекта в драйвер все равно понадобиться DDK, потому что стандартные компиляторы не умеют собирать необходимый бинарник. В новых версиях создается не совместимый с утилитой BUILD объектный файл.

→ **властелин колец.** Теперь перейдем непосредственно к архитектуре. Как только процессоры от Intel стали 32-разрядными (начиная с процессора 386), их возможности значительно расширились. Процессор научился работать в четырех режимах, нумеруемых от 0 до 4. Эти режимы называют кольцами: от RING0 до RING3. В окнах используется только два кольца и два конца, в смысле, поддерживаются только RING0 и RING3.

Уровень RING0 самый привилегированный. На нем доступны абсолютно все возможности процессора, прямой доступ к железу, памяти и много всего вкусного, но здесь может работать только ядро и драйвера ОС. Пользовательские процессы работают на 3-м уровне с большими ограничениями. Именно поэтому хакеры так мечтают очутиться на нулевом кольце :).

Дабы изолировать драйвера от пользовательских программ, Окна выделили для них разные участки памяти. Таким образом, пользовательский процесс никогда не сможет вмешаться в работу драйвера или ядра.

При описании архитектуры в разных источниках даются разные схемы. Скорее всего это связано с тем, что эта часть ОС не так открыта, как Windows API, и при описании некоторых компонентов мы можем только догадываться об их существовании именно в данном месте и именно с такими связями. Но ясно, что режим ядра изолирован от пользовательских процессов, хотя мы можем обращаться к ним через библиотеку NTDLL.DLL или вызывая функции напрямую.

Прямой вызов не является хорошим решением, но и нельзя его отнести к запрещенным способам. Отрицательный момент кроется в том, что MS может поменять функцию в ядре или предоставить новую версию, тогда при работе через NTDLL перекомпиляция программы не понадобится. В реальности такие случаи происходят очень редко, и старые функции не исчезают, а продолжают поддерживаться для обеспечения обратной совместимости.

➜ **режим драйвера.** Драйвера можно разделить на две большие группы — пользовательские (user-mode driver) и ядерные (kernel mode driver). Есть еще куча различных классификаций, но в них нет смысла. В принципе, какая разница, в какую группу засунуть драйвер. Главное, чтобы он выполнял поставленную перед ним задачу. Если ее можно решить в пользовательском режиме, то используй его. Дело в том, что возможности драйвера и предоставляемые привилегии отличаются от уровня IRQL (о нем чуть ниже).

Драйвер может быть монолитным и многоуровневым. В первом случае все функции берет на себя один единственный драйвер. Он намного проще в отладке, но не всегда эффективен, потому что может оказаться слишком большим. Намного лучше (и поэтому чаще можно встретить) многоуровневые драйвера. Их можно сравнить с классами в C++. Каждый уровень выполняет небольшую задачу, но делает это хорошо. После выполнения необходимых действий управление передается драйверу следующего уровня, где обработка продолжается. Отлаживать многоуровневые драйвера немного сложнее, но зато один раз отшлифованный уровень будет работать великолепно, потому что он выполняет небольшую задачу.

Некоторые боятся множественности уровней, не понимая преимуществ. Рассмотрим на примере сетевой карты. Для получения данных из сети можно создать два драйвера. Первый будет разбирать пользовательский запрос, а второй — читать данные из сетевой карты. Чтобы написать снифер или сетевой экран, нужно написать драйвер, который будет работать между двумя существующими. Таким образом, драйверу снифера не нужно работать непосредственно с железом и не нужно разбирать

запросы пользователей, — мы только реализуем собственные функции в нужном месте. Помимо этого, драйвера можно еще разделить на:

1. ГРАФИЧЕСКИЕ;
2. МУЛЬТИМЕДИА;
3. СЕТЕВЫЕ ДРАЙВЕРА;
4. ДРАЙВЕРА ВИРТУАЛЬНЫХ УСТРОЙСТВ.

В принципе, по названию уже понятно, для чего они нужны.

→ **формат.** На самом деле, драйвер имеет стандартный PE (Portable Executable) формат, как и у любого другого приложения. Отличие состоит в том, что расширение желательно установить .sys, и внутри файла программист должен реализовать определенные функции, через которые будет происходить общение с операционной системой. Еще могут быть пользовательские функции, которые будут решать поставленные разработчиком задачи.

Каждая программа должна иметь точку входа. Программисты C++ знакомы с такой точкой хорошо и привыкли, что ее имя WinMain. У драйвера такую точку называют DriverEntry, и она имеет следующий вид:

```
NTSTATUS STDCALL DriverEntry (
  IN PDRIVER_OBJECT DriverObject,
  IN PUNICODE_STRING RegistryPath)
```

Эта процедура получает в качестве параметра два значения:

1. Pdriver_object — указатель на созданный драйвер.
2. Punicode_string — строка, содержащая раздел реестра, где прописан драйвер.

Конечно же, имя функции DriverEntry не является обязательным, и ты можешь ее назвать по-другому, но лучше следовать этому правилу, потому что читабельность кода никто не отменял. А вот количество и типы параметров менять вообще не желательно.

Вот так вот может выглядеть простейший драйвер на C:

```
#include <ntos.h>

NTSTATUS STDCALL DriverEntry (
  IN PDRIVER_OBJECT DriverObject,
  IN PUNICODE_STRING RegistryPath)
{
  return STATUS_UNSUCCESSFUL;
}
```

→ **загрузка драйвера.** Драйверы, как и сервисы, прописаны в реестре HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services. Здесь для каждого драйвера прописаны его параметры, и ты можешь прочитать их. Здесь же желательно сохранять параметры работы драйвера. Для хранения параметров необходимо использовать ветку реестра HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Имя\Parameters, где «Имя» — имя драйвера.

В качестве результата функция возвращает тип данных NTSTATUS. Если результат равен STATUS_SUCCESS, то драйвер считается загруженным верно и может обрабатывать ввод/вывод информации. Иначе драйвер не загружается в память и не может использоваться.

При выгрузке драйвера вызывается функция Dispatch, которая выглядит следующим образом:

```
NTSTATUS STDCALL Dispatch(
  IN PDEVICE_OBJECT DriverObject,
  IN PIRP Irp)
```

Если драйвер может быть выгружен во время работы системы, то необходимо реализовать еще и функцию Unload, которая должна выглядеть следующим образом:

```
VOID Unload (
  IN DRIVER_OBJECT DeriverObject
);
```

Многоуровневые драйвера должны реализовывать функцию IoCompletion, в которой необходимо освобождать структуру Irp:

```
NTSTATUS IoCompletion (
  IN PDEVICE_OBJECT DeriverObject,
  IN PIRP Irp,
  IN VOID Contex
);
```

→ **ввод/вывод.** Драйвера ввода/вывода должны создавать логическое, виртуальное или физическое устройство, с которым будет происходить обмен ввода/вывода. Такое устройство создается с помощью функции IoCreateDevice. Описывать эту функцию не будем, потому что она содержит аж 7 параметров и имеет кучу особенностей. Если ты будешь писать драйвера устройств, то обратись к MSDN. Для удаления устройства используется функция IoDeleteDevice.

Создав устройство, можно создавать символическую ссылку на него с помощью функции IoCreateSymbolicLink. Если ты создашь ссылку, к примеру, с именем «xakep», драйвер будет виден в системе /device/xakep.

→ **приоритеты прерываний.** Исполняемый код имеет определенный уровень прерывания IRQL (interrupt request levels), по которому определяется, что позволено делать, а что нет. Это не уровень потока, это именно уровень прерывания. Всего таких прерываний 32. Прерывание с нулевым номером обладает низшим приоритетом, а №31, соответственно, наивысшим. Наивысшим уровнем обла-

Лучший сайт для системщика

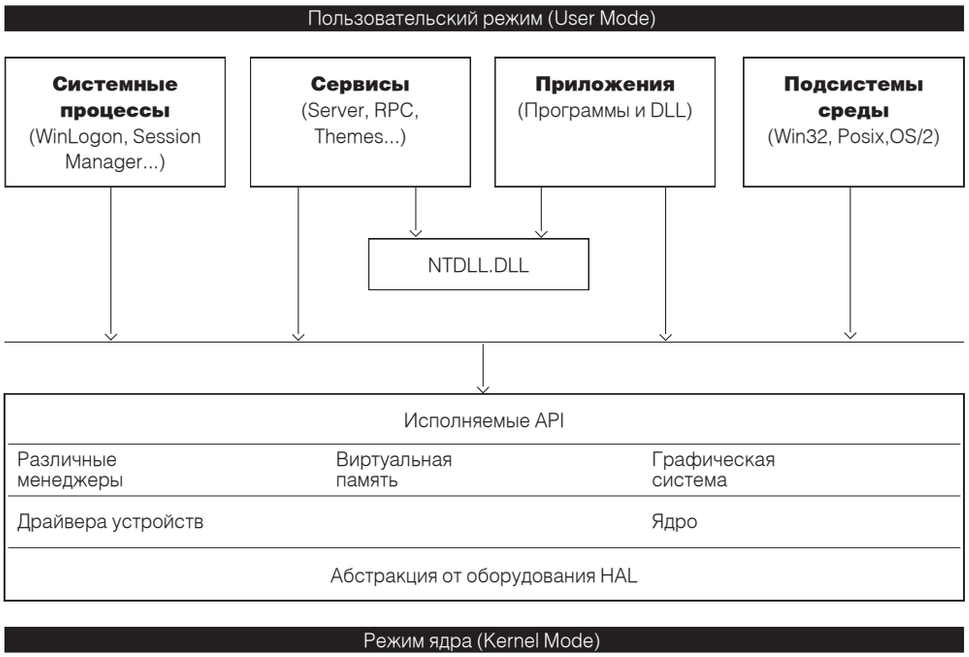
SPECIAL ОБЗОР

MEDIUM

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ В WINDOWS

СПб.: БХВ-Петербург, 2006 / Побегайло А.П. / 1056 страниц

Подробно рассматриваются вопросы системного программирования с использованием интерфейса Win32 API. Описываются: управление потоками и процессами, включая их диспетчеризацию, синхронизация потоков, передача данных между процессами с использованием анонимных и именованных каналов, а также почтовых ящиков, структурная обработка исключений, управление виртуальной памятью, управление файлами и каталогами, асинхронная работа данных, создание динамически подключаемых библиотек, разработка сервисов… Плюс весьма актуальное управление безопасностью Windows. Книга рассчитана на начинающих системных программистов, но в силу объема и полноты информации послужит отличным справочным пособием для более опытных коллег.



Наиболее простая схема архитектуры Windows

дают прерывания устройств, эти IRQL соответствуют аппаратным прерываниям. Два низших прерывания реализованы программно.

Нулевой уровень зарезервирован для потока, который работает, когда системе нечего делать. Если верить DDK, то это значение указывать нельзя. Мы не пробовали и не в курсе, какой будет результат, если попытаться :). Уровни с 1 до 15 являются динамическими, потому что по мере необходимости ОС может понижать или повышать эти значения, чтобы драйвер выполнялся как можно быстрее и не блокировал работу системы. Диапазон от 16 до 31 — фиксированные прерывания, и в их уровни ОС не вмешивается. Прерывания от 16 до 31 имеют одно очень важное свойство — их работа не может быть прервана другим процессом, если его приоритет прерывания ниже. В связи с этим, такой код должен быть максимально компактным и выполняться очень быстро, дабы не монополизировать процессор. В диапазоне от 1 до 15 приоритет прерывания может изменяться, поэтому в определенный момент код с большим приоритетом может быть прерван кодом с низшим значением только потому, что ОС решила поиграть с этими приоритетами.

В зависимости от уровня, драйвер может реализовывать еще несколько функций, например, функцию InterruptService. Она обязательна для всех драйверов устройств, которые генерируют прерывания. Специфичных для различных уровней и устройств функций много, и описать их нереально. Прежде чем писать драйвер, обязательно изучи DDK.

➜ **отладка.** Вообще, отладка драйверов — достаточно сложное и утомительное занятие. Дело в том, что если ты допустишь ошибку в пользовательском приложении, то такую задачу достаточно просто снять, перекомпилировать и запустить по новой. С драйверами такие шутки проходят редко, потому что если в коде допущена ошибка, то велика вероятность увидеть синий/черный экран смерти. После этого компьютер спасет только перезагрузка, вследствие которой найти проблемную строку кода не всегда возможно.

При отладке рекомендую следующее:

[1] СОХРАНЯЙ ВСЕ ДО НАЧАЛА ТЕСТИРОВАНИЯ. К ЭТОМУ ПРАВИЛУ ТЫ БЫСТРО ПРИВЫКНЕШЬ ПОСЛЕ ПАРЫ СИСТЕМНЫХ СБОЕВ И ПОТЕРИ НЕСКОЛЬКИХ ЧАСОВ РАБОТЫ.

[2] ИСПОЛЬЗУЙ СПЕЦИАЛЬНУЮ ВЕРСИЮ ОКОН С ОТЛАДОЧНОЙ ИНФОРМАЦИЕЙ, КОТОРАЯ ПОМОЖЕТ В ПОИСКЕ ОШИБОК ПОСЛЕ СБОЯ.

[3] ТЕСТИРУЙ КАК МОЖНО ЧАЩЕ И КАК МОЖНО БОЛЬШЕ. ПОСЛЕ КАЖДОГО (ПУСТЬ И МАЛЕНЬКОГО) ИЗМЕНЕНИЯ НЕОБХОДИМА ТЩАТЕЛЬНАЯ ПРОВЕРКА РАБОТЫ, ИНАЧЕ ПОТОМ ИСПРАВЛЯТЬ ОШИБКУ БУДЕТ НАМНОГО СЛОЖНЕЕ.

[4] ПРЕЖДЕ ЧЕМ ТЕСТИРОВАТЬ, ЕЩЕ РАЗ ПРОСМОТРИ КОД ДРАЙВЕРА.

[5] И ЕЩЕ РАЗ ПРОСМОТРИ КОД ДРАЙВЕРА, ЭТО ЛИШНИМ НЕ БЫВАЕТ :).

Лучше всего отлаживать драйвера в виртуальной машине, но это не есть «обязательно». Если драйвер глюкнул, то виртуалка не спасет.

Встроенный в IDE отладчик тут тоже не подойдет. Необходимо что-то более серьезное, например, SoftICE или Kernel Debugger из DDK. Второй вариант не особо удобен, потому что требует наличия двух компьютеров, но если ты используешь виртуальную машину, то все решаемо. SiftICE обладает широкими возможностями, большинство предпочитают именно его. Но ты волен выбирать другой отладчик, главное — чтобы он тебе был удобен и позволял отлаживать драйвера.

➜ **проще некуда.** А можно сделать разработку драйвера проще? Можно! Например, на www.jungo.com ты можешь найти DDK, при использовании которой тебе не нужно знать внутренностей ОС. Все достаточно просто: jungo уже написала универсальный драйвер, который берет на себя все сложные функции по работе непосредственно с ОС. Остается только создать надстройку, которая будет работать через драйвер.

К преимуществам пакета jungo можно отнести то, что разработка значительно упрощается: тебе не нужно знать DDK или внутренностей ОС. Но есть и недостаток — ограниченность возможностей. Данный пакет позволяет делать далеко не все. Но для простых задач и быстрого решения проблемы пакет незаменим.

Jungo — не единственный пакет, существуют другие подобные пакеты, например, от compuserve, который обладает большими возможностями. С его помощью, благодаря хорошему мастеру, можно создавать не только драйвера, работающие через универсальный драйвер, но и самостоятельные.

➜ **compiling complete.** Любой системный программист, и тем более хакер, должен четко представлять себе архитектуру Windows. Невозможно написать полноценные программы безопасности без драйверов. Например, полноценный сетевой экран или снифер невозможен на пользовательском уровне. Тут просто необходимо обращаться к более низкому уровню и писать драйвер. Можно использовать уже готовые разработки, но это несолидно для тебя и неприемлемо для коммерческого продукта. Удачной компиляции! C

шедевры кернел-кодинга

[1] **www.wasm.ru**
многие годы этот сайт остается лучшим для системщика и программиста на ассемблере. Здесь же есть информация и по программированию драйверов.

[2] **www.msdn.com**
msdn и справка из ddk. Без знания английского можно писать только простые программы, а хорошей информации по драйверам на русском очень мало. Поэтому свежачок можно найти только в файлах справки ddk и msdn.

[3] **www.jungo.com**
неплохой движок, упрощающий кернел-кодинг.

[4] **www.sysinternals.com**
здесь ты найдешь отличный отладчик для кернел-кодинга.

[5] **www.void.ru**
классика российского хакинга. Очень рекомендуется прочитать статью о прямом вводе/выводе.

[6] **www.nullsoft.com**
у этой компании есть движок, упрощающий кернел-кодинг. Сейчас компания перешла под крыло compuserve, но движок от этого хуже не стал.

знаменитые трюкачи

ПОПУЛЯРНЫЕ АЛГОРИТМЫ

ПРОГРАММИРОВАНИЕ ОКРУЖАЕТ ТЕБЯ СО ВСЕХ СТОРОН. НА РАБОТЕ ТЫ КОДИШЬ С VISUAL STUDIO .NET НОВЫЙ САЙТ ДЛЯ ТВОЕЙ КОНТОРЫ. ДОМА С УПОЕНИЕМ РАЗБИРАЕШЬСЯ В ПРЕМУДРОСТЯХ С ПОД *NIX И В СОТЫЙ РАЗ ПЕРЕКОМПИЛИРУЕШЬ МНОГОСТРАДАЛЬНОЕ ЯДРО...

Андрей «Orc» Серегин
andrey.seregin@stp-group.ru

А глубокой ночью, отходя ко сну, ты вдруг вспоминаешь первые уроки информатики, первые программы на алгоритмическом псевдокоде... Но некоторые алгоритмы, хочешь ты или нет, применяются в повседневном программировании и по сей день.

➜ **сортировка.** Современные методы применяются в основном для сортировки и упорядочивания массивов данных, чаще всего цифровых. Для начала рассмотрим самый простой и самый медлительный алгоритм сортировки — метод пузырька. Он так называется, потому что при такой сортировке самый «легкий» (наименьший) элемент массива постепенно поднимается «наверх» (к началу массива). То есть, просматривая числовой ряд, ищем такую последовательность рядом стоящих чисел, где a > b, и меняем a и b местами. После этого начинаем просматривать массив сначала, уменьшив количество просматриваемых элементов массива на 1. И повторяем это безобразие до тех пор, пока в просмотре не будет участвовать только первое и второе число из массива. Вот процедура, которая осуществляет сортировку массива таким способом:

```
procedure BubbleMethod(var Arr: array of
double; const N: integer);
var
  A:   integer;
  B:   integer;
  Tmp: double;
begin
  A := 0;
  while A <= N - 1 do
  begin
    B := 0;
    while B <= N - 2 - A do
    begin
      if Arr[B] > Arr[B + 1] then
      begin
        Tmp := Arr[B];
```

```
        Arr[B] := Arr[B + 1];
        Arr[B + 1] := Tmp;
      end;
      Inc(B);
    end;
    Inc(I);
  end;
end;
```

Arr — это и входной, и выходной массив, N — число элементов массива. Более быстрый метод сортировки — метод простых вставок (или метод Шелла — не что иное, как модификация метода простых вставок). Есть уже некая упорядоченная последовательность чисел в массиве, где мы располагаем новые сортируемые элементы. В коде это выглядит так:

```
procedure InsertMethod(var Arr: array of
integer; N: integer);
var
  A:   integer;
  B:   integer;
  C:   integer;
  Tmp: double;
begin
  N := N - 1;
  A := 1;
  repeat
    B := 0;
    repeat
      if Arr[A] <= Arr[B] then
      begin
        C   := A;
        Tmp := Arr[A];
        repeat
          Arr[C] := Arr[C - 1];
          C := C - 1;
        until not (C > B);
        Arr[B] := Tmp;
        B := A;
      end
      else
        Inc(B);
    until not (B < A);
    Inc(A);
  until not (A <= N);
end;
```

Теперь давай посмотрим, какой метод мы можем применить, если нам важна скорость сортировки. Предыдущие методы нам не подходят, так как они хоть и «классика жанра», но быстротой не отличаются. Самый скоростной алгоритм сортировки, известный на сегодняшний день, — это метод бинарных деревьев aka метод Уильяма-Флойда (это не имя и фамилия, а два разных человека). Суть алгоритма в том, что у каждого из элементов массива есть два элемента-потомка, а сам массив считается отсортированным, когда предок больше потомка. Хоть метод бинарных деревьев и очень быстр, в реализации он достаточно сложный (смотри листинг 1).

Метод Уильяма-Флойда

```
procedure TreeMethod(var Arr: array of double;
N: integer);
var
  A:   integer;
  B:   integer;
  C:   integer;
  D:   integer;
  Tmp: double;
begin
  if N = 1 then
    Exit;
  A := 2;
  repeat
    D := A;
    while D <> 1 do
    begin
      C := D div 2;
      if Arr[C - 1] >= Arr[D - 1] then
      begin
        D := 1;
      end
      else
      begin
        Tmp := Arr[C - 1];
        Arr[C - 1] := Arr[D - 1];
        Arr[D - 1] := Tmp;
        D   := C;
      end;
    end;
    A := A + 1;
  until not (A <= N);
  A := N - 1;
  repeat
    Tmp    := Arr[A];
    Arr[A] := Arr[0];
    Arr[0] := Tmp;
    D      := 1;
    while D <> 0 do
    begin
      C := 2 * D;
      if C > A then
      begin
        D := 0;
      end
      else
      begin
        if C < A then
        begin
          if Arr[C] > Arr[C - 1] then
          begin
            C := C + 1;
          end;
        end;
        if Arr[D - 1] >= Arr[C - 1] then
        begin
          D := 0;
        end
        else
        begin
          Tmp := Arr[C - 1];
          Arr[C - 1] := Arr[D - 1];
          Arr[D - 1] := Tmp;
          D  := C;
        end;
      end;
    end;
    Dec(A);
  until not (A >= 1);
end;
```

(1)

Следующий алгоритм — сортировка методом слияний aka метод фон Неймана. Метод этот заключается в том, что массив делится на упорядоченные группы. Сначала каждая из групп состоит из одного элемента, а затем, соединяя и укрупняя группы их соседями, получаем искомый отсортированный массив. Кстати, обрати внимание на то, что в исходнике для лучшего понимания метода будем использовать второй массив, который имеет тот же размер, что и сортируемый (смотри листинг 2).

Метод фон Неймана — самый сложный в реализации, но оптимальный. В повседневном программировании ты, конечно, сам волен выбирать, каким из описанных алгоритмов воспользоваться.

➔ **поиск.** Конечно, ты не переплюнешь таких монстров поиска, как Яндекс или Гугл (хотя, кто знает). Но то, что ты прочтешь далее, позволит тебе написать, к примеру, собственный поисковый движок по сайту.

За время развития программирования возможности поиска шагнули далеко вперед, но принципы и алгоритмы остаются теми же, что и 10 лет назад. Самый простой — метод брутфорса, то есть последовательного перебора. Берем первую букву в искомой строке, ищем ее в оригинале. Если находим — смотрим следующую букву. И так далее. Такой алгоритм проще пареной репы, запрограммировать его тоже очень просто (S — строка, P — искомая подстрока, а сама функция возвращает индекс, с которого в строке S начинается подстрока P):

```
function BFSearch(S, P: string): intger;
var
  A, B: integer;
begin
  Result := 0;
  if Length(P) > Length(S) then
    Exit;
  for A := 1 to Length(S)-Length(P)+1 do
    for B := 1 to Length(P) do
      if P[B] <> S[A + B - 1] then
        Break
      else if B = Length(P) then
      begin
        Result := A;
        Exit;
      end;
end;
```

Метод фон Неймана

```
procedure MergeMethod(var Arr: array of double; N: integer);
var
  Q:     boolean;
  A:     integer;
  A1:    integer;
  A2:    integer;
  N1:    integer;
  N2:    integer;
  MergeLen: integer;
  Tmp:   double;
  TmpArr: array of double;
begin
  SetLength(TmpArr, N - 1 + 1);
  MergeLen := 1;
  Q := True;
  while MergeLen < n do
  begin
    if Q then
    begin
      A := 0;
      while A + MergeLen <= n do
      begin
        A1 := A + 1;
        A2 := A + MergeLen + 1;
        n1 := A + MergeLen;
        n2 := A + 2 * MergeLen;
        if n2 > n then
        begin
          n2 := n;
        end;
        while (A1 <= n1) or (A2 <= n2) do
        begin
          if A1 > n1 then
          begin
            while A2 <= n2 do
            begin
              A  := A + 1;
              TmpArr[A - 1] := Arr[A2 - 1];
              A2 := A2 + 1;
            end;
          end
          else
          begin
            if A2 > n2 then
            begin
              while A1 <= n1 do
              begin
                A  := A + 1;
                TmpArr[A - 1] := Arr[A1 - 1];
                A1 := A1 + 1;
              end;
            end
            else
            begin
              if Arr[A1 - 1] > Arr[A2 - 1] then
              begin
                A  := A + 1;
                TmpArr[A - 1] := Arr[A2 - 1];
                A2 := A2 + 1;
              end
              else
              begin
                A  := A + 1;
                TmpArr[A - 1] := Arr[A1 - 1];
                A1 := A1 + 1;
              end;
            end;
          end;
        end;
      end;
      A := A + 1;
      while A <= n do
      begin
        TmpArr[A - 1] := Arr[A - 1];
        A := A + 1;
      end;
    end
    else
    begin
      A := 0;
      while A + MergeLen <= n do
      begin
        A1 := A + 1;
        A2 := A + MergeLen + 1;
        n1 := A + MergeLen;
        n2 := A + 2 * MergeLen;
        if n2 > n then
        begin
          n2 := n;
        end;
        while (A1 <= n1) or (A2 <= n2) do
        begin
          if A1 > n1 then
          begin
            while A2 <= n2 do
            begin
              A  := A + 1;
              Arr[A - 1] := TmpArr[A2 - 1];
              A2 := A2 + 1;
            end;
          end
          else
          begin
            if A2 > n2 then
            begin
              while A1 <= n1 do
              begin
                A  := A + 1;
                Arr[A - 1] := TmpArr[A1 - 1];
                A1 := A1 + 1;
              end;
            end
            else
            begin
              if TmpArr[A1 - 1] > TmpArr[A2 - 1] then
              begin
                A  := A + 1;
                Arr[A - 1] := TmpArr[A2 - 1];
                A2 := A2 + 1;
              end
              else
```

```
            begin                          (2)
              A := A + 1;
              Arr[A-1] := TmpArr[A1-1];
              A1:= A1 + 1;
            end;
          end;
        end;
      end;
    end;
    A := A + 1;
    while A <= n do
    begin
      Arr[A - 1] := TmpArr[A - 1];
      A := A + 1;
    end;
  end;
  MergeLen := 2 * MergeLen;
  Q := not Q;
end;
if not Q then
begin
  A := 1;
  repeat
    Arr[A - 1] := TmpArr[A - 1];
    A := A + 1;
  until not (A <= n);
end;
end;
```

А вот на алгоритме поиска по методу Бойера-Мура-Хорспула (Boyer-Moor-Horspool Pattern Search, чаще всего просто используют аббревиатуру BMH) стоит остановиться чуть более поподробно. Действие алгоритма начинается с построения «таблицы смещений» для искомой подстроки — каждому символу в подстроке приравнивается число, которое представляет собой смещение символа алфавита относительно последнего символа в подстроке. Следующим шагом совмещаем начало строки и подстроки и смотрим, совпал ли последний символ в подстроке с символом из строки, полученном при таком наложении. Если это условие не выполняется — смещаем подстроку на количество символов из таблицы смещений, а если выполняется — сравниваем предпоследние символы. И так далее. Самая простая реализация такого алгоритма может выглядеть так:

```
function BMHSearch(StartPos: integer; S,
P: string): integer;
type
  BMHTable = array[0..255] of integer;
var
  Pos, lp, i: integer;
  BMHT: BMHTable;
begin
  for i := 0 to 255 do
    BMHT[i] := Length(P);
  for i := Length(P) downto 1 do
    if BMHT[byte(P[i])] = Length(P) then
      BMHT[byte(P[i])] := Length(P) - i;
  lp := Length(P);
  Pos := StartPos + lp - 1;
  while Pos <= Length(S) do
    if P[lp] <> S[Pos] then
      Pos := Pos + BMHT[byte(S[Pos])]
    else if lp = 1 then
    begin
      Result := Pos;
      Exit;
    end
    else
      for i := lp - 1 downto 1 do
        if P[i] <> S[Pos - lp + i] then
        begin
          Inc(Pos);
          Break;
        end
        else if i = 1 then
        begin
          Result := Pos - lp + 1;
          Exit;
        end;
  Result := 0;
end;
```

Но, пожалуй, самый известный и популярный алгоритм — это поиск методом Кнута-Морриса-Пратта. Идея заключается в том, что мы предварительно обрабатываем искомую подстроку — создаем для нее префикс-функцию, то есть ищем в начале и в конце подстроки наибольшую последовательность повторяющихся символов. Например, для строки 12345qwerty12345 префикс-функция должна возвратить 12345:

```
procedure KMPPrefix;
var
```

blowfish

ИЗВЕСТНЫЙ АЛГОРИТМ ШИФРОВАНИЯ ДАННЫХ. СОСТОИТ ИЗ ДВУХ ЧАСТЕЙ: РАЗВЕРТЫВАНИЕ КЛЮЧА И ШИФРОВАНИЕ ДАННЫХ. РАЗВЕРТЫВАНИЕ КЛЮЧА ПРЕОБРАЗУЕТ КЛЮЧ ДЛИНОЙ ДО 448 БИТОВ В НЕСКОЛЬКО МАССИВОВ-ПОДКЛЮЧЕЙ. ШИФРОВАНИЕ ДАННЫХ СОСТОИТ ИЗ ПРОСТОЙ ФУНКЦИИ, ПОСЛЕДОВАТЕЛЬНО ВЫПОЛНЯЕМОЙ 16 РАЗ.

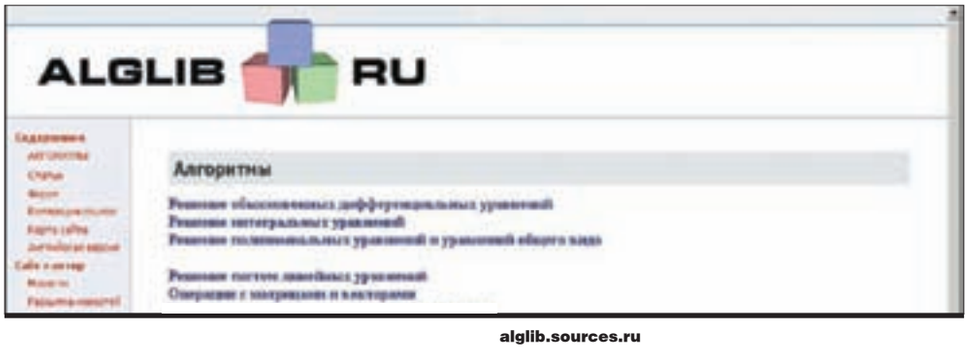
КАЖДЫЙ ЭТАП СОСТОИТ ИЗ ЗАВИСИМОЙ ОТ КЛЮЧА ПЕРЕСТАНОВКИ И ЗАВИСИМОЙ ОТ КЛЮЧА И ДАННЫХ ПОДСТАНОВКИ. ИСПОЛЬЗУЮТСЯ ТОЛЬКО СЛОЖЕНИЯ И XOR 32-БИТОВЫХ СЛОВ. ДОПОЛНИТЕЛЬНЫМИ ОПЕРАЦИЯМИ НА КАЖДОМ ЭТАПЕ ЯВЛЯЮТСЯ ЧЕТЫРЕ ИЗВЛЕЧЕНИЯ ДАННЫХ ИЗ ИНДЕКСИРОВАННОГО МАССИВА.

```
  S, P:    array of double;
  A, B, C: integer;
begin
  P[1] := 0;
  B    := 0;
  for A := 2 to C do
  begin
    while (B>0) and (S[B + 1] <> S[A]) do
      B := P[B];
    if S[B + 1] = S[A] then
      B := B + 1;
    P[A] := B;
  end;
end;
```

Поиск с использованием нашей функции :

```
var
  n: integer;
  T: array of double;
begin
KMPPrefix;
k := 0;
for i := 1 to n do
begin
  while (k > 0) and (S[k + 1] <> T[i]) do
    k := P[k];
  if S[k + 1] = T[i] then
    k := k + 1;
  if k = m then
  begin
    k := P[k];
    Exit;
  end;
end;
end
```

«Повседневных» алгоритмов, конечно, значительно больше, но уместить их все в рамки одной статьи, увы, нереально C



alglib.sources.ru

медитация

ПРАВИЛА СОСТАВЛЕНИЯ КОММЕНТАРИЕВ

КОММЕНТАРИИ — МЕРА КАЧЕСТВА ПРОГРАММНОГО КОДА. С ИХ ПОМОЩЬЮ УМЕНЬШАЕТСЯ СТОИМОСТЬ ВЛАДЕНИЯ КОДОМ

Антон Палагин aka Tony
tony@eykontech.com

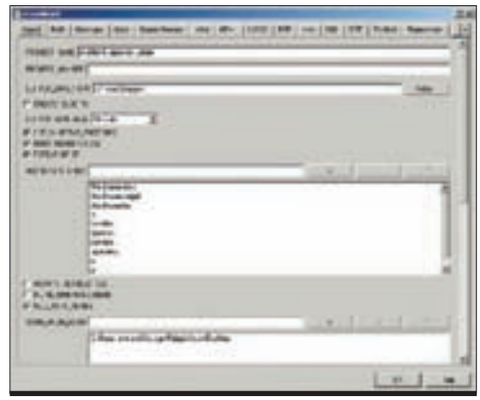
Существует два вида комментариев. Первые используются для процедуры автоматизированного документирования исходного кода и применяются в основном при разработке программных интерфейсов. Эти комментарии оформляются с помощью специальных форматов: doxygen, javadoc (без мелкософта тут, сам понимаешь, дело не обошлось, и у них тоже имеется собственный формат комментариев с трехэтажным названием XML Documentation Comments). Второй вид — это комментарии, сопровождающие исходный код, объясняющие его работу, описывающие данные и раскрывающие непонятные и сомнительные моменты.

➜ **автоматизированное документирование —** одна из самых важных сторон разработки программного кода. С помощью документации ты объясняешь своему клиенту, как надо пользоваться плодами твоих трудов. Идея автоматизированного документирования заключается в том, что в исходном коде, рядом с определением или реализацией используемых клиентом сущностей, пишутся комментарии в специальном формате. Далее запускается специальная утилита, которая парсит исходный код, вытаскивая комментарии и объединяя их в единый структурированный документ. Таким образом, с плеч программиста снимается вся черновая работа по созданию программной документации. Кроме того, такой способ документирования позволяет более строго контролировать версии программного кода и документации, поскольку при изменении программного интерфейса программист должен менять расположенные рядом комментарии.

➜ **комментарии исходного кода.** В отличие от комментариев для автоматизированного документирования, эти комментарии используются в любом программном коде. Стоимость владения исходным кодом уменьшается прямо пропорционально количеству и качеству комментариев. Это связано с тем, что хорошо структурированный и комментированный код легче поддерживается и сопровождается. То есть программистам просто требуется меньше времени для того, чтобы разобраться с ним. Для анализа качества исходного кода можно использовать специализированные инструменты (например, together), которые подсчитывают различные метрики кода, описывающие его качество.

➜ **используй одну нотацию имен.** Существует достаточно большое количество нотаций имен

ОПТИМАЛЬНЫМ СЧИТАЕТСЯ СООТНОШЕНИЕ СТРОК КОДА К СТРОКАМ КОММЕНТАРИЕВ — 1/3 И 1/4



Настройка сценария генерации

программных сущностей, а также стилей кодирования. Выбери себе одну из таких нотаций и стиль кодирования, и следуй им. Если ты работаешь в команде, то вся команда также должна придерживаться единого стиля.

➔ **будь внимательней при выборе кодировки и языка.** В последнее время ситуация с поддержкой кириллицы в средах разработки стала заметно лучше, однако на встраиваемых платформах до сих пор наблюдаются проблемы с русским языком. И твой файл с комментариями в кодировке CP-1251, написанными в MSVC, ни за что не откроется в том же Eclipse под QNX. Так что не исключено, что придется писать комментарии на английском языке.

➔ **форматируй и структурируй код.** Обращай внимание на форматирование своего кода и его структуру. Выравнивай фигурные скобки по одной колонке и строки по отступам. Обязательно делай так, чтобы внутри фигурных скобок код был выровнен по левому краю и с отступом от скобок. Используй для этого табуляцию. Объединяй несколько строк кода в группы по 3-5 строчек, которые выполняют однородное по своему смыслу действие. Группы отделяй друг от друга пустыми строками.

➔ **настрой табуляцию.** В настройках своего редактора поставь галочку, которая предписывает ему вместо символа табуляции вставлять определенное число пробелов (обычно 4 пробела). Это позволит твоему коду не расползаться, как толпа тараканов, по экрану, если клиенту приспичило открыть твой код в каком-нибудь «кривом» редакторе.

➔ **комментируй «длинные» скобки.** Если у тебя есть громадные условия и циклы, которые до полного счастья вложены друг в друга, то разобраться в последовательности закрывающих фигурных скобок сложнее, чем в даосизме с помощью Корана. Для того чтобы не сойти с ума, просто напиши строчный комментарий после «длинной» закрывающей фигурной скобки с оператором, их предваряющим.

➔ **комментируй объявляемые переменные.** Обязательно комментируй все объявляемые тобой переменные, указывая, что каждая из них делает и какие данные хранит. Если требования по используемой памяти тебе позволяют, старайся использовать каждую переменную по своему назначению. Не стоит хранить в счетчике цикла длину строки текста.

➔ **не старайся комментировать каждую строку.** Если ты объединяешь строки кода в группы по смыслу, то старайся писать каждой группе строчку, которая описывает, что эта группа делает. Не имеет смысла комментировать каждую строку: старайся все сводить к макроуровню комментариев. Оптимальным считается соотношение строк кода к строкам комментариев — 1/3–1/5.

➔ **используй строчные комментарии.** Старайся использовать строчные комментарии (предваряющиеся символами //) вместо многострочных (предваряющихся символами /*). В том случае, если ты захочешь закомментировать большой блок кода, существующие комментарии тебе не помешают.

➔ **не путай божий дар с яичницей.** Никакие комментарии не помогут, если ты используешь трехэтажные программные конструкции, которые так любят маньяки STL. Также не стоит злоупотреблять длинными строками (длиннее 80-100 символов) и длинными именами (равно как и короткими названиями). Переменная с именем the_sign_of_specified_mathematical_operation может довести до колик в животе не одну сотню «читателей» кода. Старайся писать более простой код, так, чтобы комментарии его дополняли, а не дублировали. Код сам по себе является самым совершенным и лаконичным комментарием, но только в том случае, если он логичен, последователен и прост.

➔ **используй весь спектр тегов автоматизированной документации.** Существует множество тегов, которые позволяют выделять специальным образом предупреждения, исключения, заметки и т.д. Также, с помощью тегов можно объединять функции и члены классов в именованные группы. В различных системах автоматизированной документации используются разные теги.

➔ **создавай титульную страницу.** Для автоматизированной документации существует возможность задать титульную страницу, на которой ты сможешь расположить базовую информацию по своему интерфейсу (некое интеграционное введение, которое описывает процесс использования твоего кода). Как подключить, какие дополнительные библиотеки требуются и так далее.

➔ **используй graphwiz.** Если ты хочешь, чтобы в твоей документации были красивые графики и UML-диаграммы, скачай и установи утилиту graphwiz. Далее тебе необходимо на закладке Dot (doxygen) поставить нужные галочки и указать путь к этой утилите. После чего на выходе ты получишь нужные диаграммы.

➔ **используй тег brief.** В doxygen этот тег позволит тебе задавать краткое описание твоего метода, которое будет отображаться, когда методы класса показываются в одном большом списке.

➔ **используй тег code.** С помощью этого тега можно задавать примеры кода в твоей документации. Обрати внимание на отступы и форматирование кода примеров. Закрывается пример с помощью тега endcode.

➔ **объединяй документацию.** Если ты разрабатываешь несколько библиотек, то их документацию ты сможешь хитрым образом интегрировать при помощи механизма ссылок. Также существует возможность встраивать в документацию отдельные документы и изображения. Если немного поковыряться с тегами и настройками, то на выходе можно получить громадный талмуд документации со всеми нужными клиенту данными.

➔ **рассмотри возможность использования LATEX.** Этот формат позволяет использовать в тексте формулы. Если они тебе строго необходимы для документирования, то без LATEX не обойтись. Внутри тега \f можно задавать формулу с использованием так называемого формата LATEX. Этот тег будет проигнорирован, если в качестве выходного формата используется не LATEX. В общем, латекс велик и вообще рулит.

➔ **используй различные схемы определения комментариев.** Если ты активно используешь Rational Rose для дизайна архитектуры и постоянно делаешь forward и reverse engineering, то следует обратить внимание на то, что не все комментарии будут ей верно интерпретированы. Например, она проигнорирует символы «/*!», которые маркируют начало комментария для doxygen. Но если твои комментарии начинаются с последовательности «//!», то их роза будет добавлять в генерируемые файлы при forward engineering'e и вставлять в твои сущности модели при reverse engineering'e. Таким образом, тебе не придется постоянно восстанавливать комментарии, и ты сможешь активно модифицировать свою модель и код **C**

www.stack.nl/~dimitri/doxygen/
http://java.sun.com/j2se/javadoc/
http://msdn2.microsoft.com/en-us/library/b2s063f7.aspx
www.graphviz.org
генераторы документации.

http://ru.wikipedia.org/wiki/
генератор документации глазами википедии.

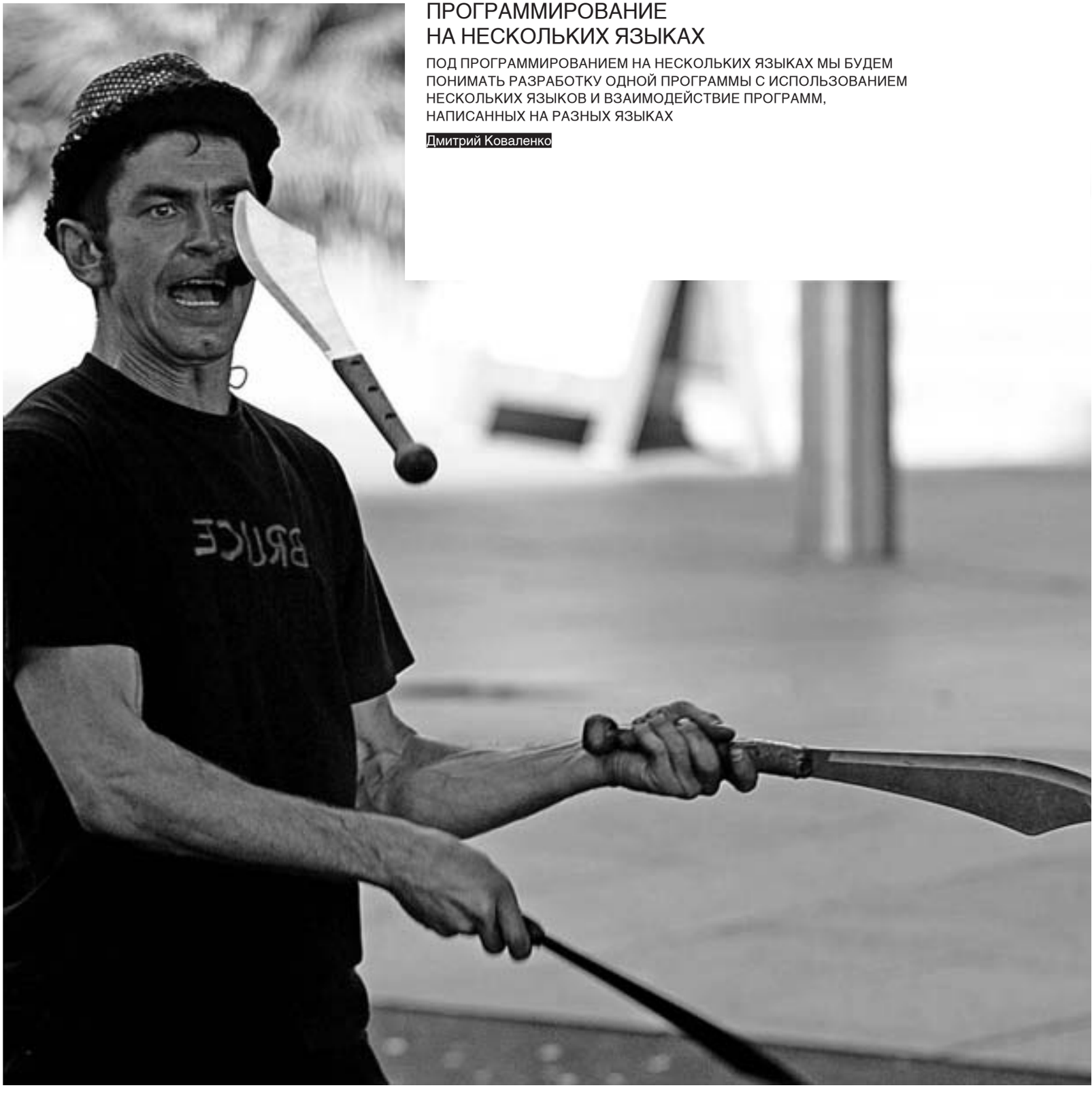
www.interface.ru/borland/bt2006.htm
borland together 2006.

стрельба с обеих рук

ПРОГРАММИРОВАНИЕ НА НЕСКОЛЬКИХ ЯЗЫКАХ

ПОД ПРОГРАММИРОВАНИЕМ НА НЕСКОЛЬКИХ ЯЗЫКАХ МЫ БУДЕМ ПОНИМАТЬ РАЗРАБОТКУ ОДНОЙ ПРОГРАММЫ С ИСПОЛЬЗОВАНИЕМ НЕСКОЛЬКИХ ЯЗЫКОВ И ВЗАИМОДЕЙСТВИЕ ПРОГРАММ, НАПИСАННЫХ НА РАЗНЫХ ЯЗЫКАХ

Дмитрий Коваленко



В программах на Visual C++ и Delphi можно делать ассемблерные вставки. Это называется inline-ассемблером. Обычно inline-ассемблер используется в двух случаях:

[1] Если нужно оптимизировать критичные по скорости и небольшие по объему участки программы. Грамотно написанный ассемблерный код всегда быстрее кода, который генерируется компилятором C++ или Delphi.

[2] Нужен прямой доступ к памяти и портам. Чаще всего используется в драйверах, так как из третьего кольца защиты с портами не очень-то поработаешь. Поскольку информации об inline-ассемблерах в интернете довольно мало, мы остановимся на этом подробнее.

➔ **Inline-ассемблер в Visual C++.** Ассемблерные вставки в исходниках Visual C++ оформляются с помощью блока __asm и выглядят примерно так:

```
// обычный код на C++
printf("Hello!\n");
// а тут ассемблерная вставка
__asm {
 mov eax, 2
 mov edx, 7
 add eax, edx
}
// а потом снова обычный код на C++
printf("Bye!");
```

или, что то же самое, так:

```
// ассемблерная вставка
__asm mov eax, 2
__asm mov edx, 7
__asm add eax, edx
```

а можно вообще в одну строчку:

```
// ассемблерная вставка
__asm mov eax, 2 __asm mov edx, 7 __asm
add eax, edx
```

В inline-ассемблере Visual C++ можно использовать все инструкции вплоть до Pentium 4 и AMD Athlon. Поддерживается также MMX. Поддержки Itanium и x64 пока нет :(.

По синтаксису inline-ассемблер Visual C++ частично совпадает с MASM. Например, как и в MASM, можно строить выражения с операндами и использовать offset с глобальными переменными (смотри листинг 1). Как и в MASM, можно использовать глобальные метки и принудительно определять короткие переходы (смотри листинг 2). Однако определить локальную метку с помощью @@ в inline-ассемблере Visual C++ не получится — замена lab: на @@lab: в предыдущем примере вызовет ошибку компиляции. Есть и другие отличия от MASM. Например, в inline-ассемблере Visual C++ нет никаких средств для объявления переменных, поэтому про привычные DB, DW, DD, DQ, DT, DF, DUP и THIS можно забыть. Зато можно использовать переменные, объявленные в программе на C++ (смотри листинг 3).

В inline-ассемблере также нельзя объявлять структуры — директивы STRUC, RECORD, WIDTH и MASK недопустимы. Вместо этого можно использовать структуры, объявленные в программе на C++ (смотри листинг 4). Кроме того, в inline-ассемблере можно использовать комментарии и HEX-числа в стиле C++.

В принципе, все отличия достаточно подробно описаны в MSDN 2006 (идет в комплекте с Visual Studio 2006). Обычно в Visual C++ inline-ассемблер используется для написания функций. Как правило, функции, написанные на inline-ассемблере, объявляют с использованием директивы _stdcall. В этом случае параметры передаются в функцию на стеке в обратном порядке, а результат работы возвращается в eax.

для примера рассмотрим функцию, которая находит длину ASCIIZ-строки:

```
int _stdcall get_str_length(char *input str)
{
 // чаще всего функция целиком состоит
из одной
```

Inline-ассемблер Visual C++ частично совпадает с MASM **(1)**

```
#include <stdio.h>

char format[] = "%s %s\n";

int main()
{
        __asm{
              mov  eax, offset format ; заносим в eax смещение строки format
              mov  bl, byte ptr [eax+2] ; теперь в bl третий байт из строки format
        }
}
```

Использование глобальных меток и принудительное определение коротких переходов **(2)**

```
__asm{
        mov ecx, 10   ; заносим в ecx число 10
        jmp short lab ; принудительный короткий переход на lab
        xor ecx, ecx  ; обнуление ecx (никогда не произойдет из-за предыдущего
                      ; короткого перехода)
        lab: dec ecx  ; декремент ecx
        cmp ecx, 0    ; сравниваем с 0
        jne lab       ; если не равно 0, переходим на метку lab
}
```

Использование переменных, объявленных на C++ **(3)**

```
#include <stdio.h>

char format[] = "%s %s\n";

int main()
{
        char i = 1;
        int j;

        __asm{
              mov eax, offset format ; заносим в eax смещение строки format
              mov bl, byte ptr [eax] ; заносим в bl первый символ строки
              mov i, bl              ; копируем bl в переменную i
              mov j, 1               ; заносим в переменную j единицу
              inc j                  ; инкрементируем j
        }
}
```

Иногда стоит заглядывать в MSDN

```
 // большой ассемблерной вставки
 __asm{
   ; можно свободно обращаться
к переменным,
   ; переданным в функцию
   mov edi, inputstr
   mov esi, edi
   mov ecx, -1
   xor al, al
   cld
   repne scasb
   sub edi, esi
 ; результат работы функции следует
 ; возвращать в eax
   mov eax, edi
   dec eax
 }
}
```

использовать эту функцию можно так:

```
int main()
{
 char str_1[]="Hello, world!";
 int i;
 i = get_str_length(str_1);
 printf("String: %s\nLength: %d", str_1, i);
}
```

При написании кода на inline-ассемблере Visual C++ следует помнить некоторые моменты. Во-первых, значения регистров не передаются между ассемблерными вставками. Например, если в ассемблерной вставке ты установил eax в 1, то в следующей ассемблерной вставке eax не обязательно будет равно 1 (смотри листинг 5).

Во-вторых, нужно быть осторожным с регистрами и стеком. В середине ассемблерных вставок нельзя менять регистры ds, ss, sp, bp и флаги. Если эти регистры все-таки меняются, перед выходом из ассемблерной вставки их нужно обязательно восстановить. Что касается стека, то тут нужно соблюдать правило, которое гласит: если в ассемблерной вставке нечто ложится на стек, то в той же ассемблерной вставке это «нечто» должно со стека сниматься.

рассмотрим, например, такой код:

```
#include <stdio.h>

// наша функция на ассемблере
void _stdcall Test()
{
 __asm{
  ; кладем на стек eax
  push eax

  ; перед тем как ассемблерная вставка
кончится, нужно
  ; снять eax со стека (например,
с помощью pop),
  ; но мы забыли это сделать ;)
  }
}

int main()
{
  // вызываем функцию... и любуемся
сообщением об ошибке :)
  Test();
}
```

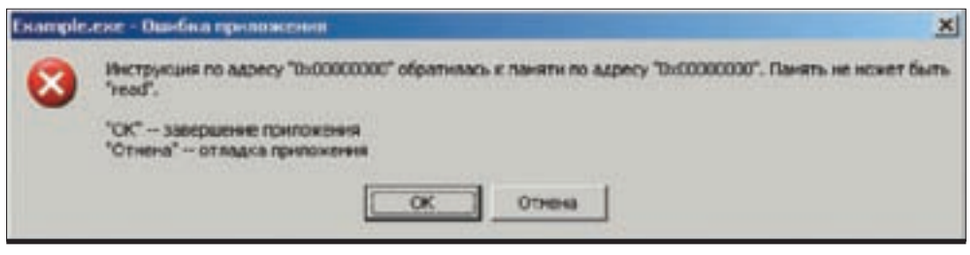
И, наконец, если ты пишешь не драйвер, а обычное Win32-приложение, в ассемблерной вставке не должно быть привилегированных инструкций. Вот, пожалуй, и все про inline-ассемблер Visual C++. Тем, кто хочет узнать больше, советуем почитать MSDN 2006 — там есть вся необходимая информация.

➔ **Inline-ассемблер в Delphi** во многом похож на inline-ассемблер Visual C++. Поэтому, чтобы не повторяться, мы рассмотрим некоторые моменты вобщем, без особых подробностей. Ассемблерные вставки в Delphi размещаются между asm и end, например:

```
{обычный код на Pascal}
Writeln('Hello!');
{вставка на ассемблере}
asm
 mov al,1
 mov bx,2
end;
{и снова обычный код на Pascal}
Writeln('Bye!');
```

Inline-ассемблер Delphi поддерживает все инструкции вплоть до Pentium 4 и AMD Athlon. Также можно использовать инструкции AMD 3DNow! для AMD K6 и AMD Enhanced 3DNow! для AMD Athlon. К сожалению, поддержки 64-разрядного кода нет, так что про Itanium и x64 можно забыть.

По синтаксису inline-ассемблер Delphi в основном похож на MASM. Например, поддержива-



К чему приводит изменение регистров edi, esi, esp, ebp и ebx

ются выражения MASM и локальные метки. Также разрешено использование offset для глобальных переменных, объявленных в программе (смотри листинг 6). Как и в MASM, можно использовать глобальные метки, но они должны быть объявлены в секции label (смотри листинг 7).

Есть и отличия от MASM. К примеру, комментарии в ассемблерных вставках должны быть обязательно в стиле Delphi, в операторах безусловного перехода нельзя использовать short и т.п. Все эти отличия описаны в документации, которая идет с Borland Developer Studio (файл Reference.pdf).

Как и в inline-ассемблере Visual C++, в inline-ассемблере Delphi нельзя определять переменные и структуры. Однако можно использовать константы, переменные и записи, уже определенные в программе (смотри листинг 8). При написании кода на inline-ассемблере Delphi нужно соблюдать те же предосторожности, что и в inline-ассемблере Visual C++. Пожалуй, единственная разница заключается в том, что нельзя изменять регистры edi, esi, esp, ebp и ebx. А вот регистры eax, ecx и edx можно изменять свободно.

Обычно на inline-ассемблере Delphi пишутся функции. При написании функций следует учитывать некоторые особенности. Во-первых, все параметры, занимающие в памяти больше 4 байтов, передаются в функцию так, как если бы перед ними стояли директивы var (даже если эти директивы не указаны). Во-вторых, результат функции должен возвращаться в eax.

типичная функция выглядит примерно так:

```
{умножение X*Y}
function LongMul(X, Y: Integer):Longint;
asm
mov eax,X
imul Y
end;
```

➜ **OBJ-модули.** Мы не будем подробно останавливаться на OBJ-модулях, их упоминание здесь — скорее дань традиции. В славные времена MS DOS OBJ-модули были чуть ли не единственным способом использовать одновременно несколько языков программирования.

Если кто не в курсе, OBJ-модуль — это файл с расширением .obj. В OBJ-модуле в специальном формате содержится машинный код, обычно — набор каких-то полезных функций. Функции из OBJ-модулей можно вызывать из программ на Visual C++ и Delphi. Создавать OBJ-модули можно с помощью компиляторов тех же Visual C++ и Delphi. Таким образом, OBJ-модуль, написанный на Visual C++, теоретически может использоваться в программах на Delphi и наоборот.

Но это только теоретически: на практике форматы OBJ-модулей, генерируемых Visual C++ и Delphi, несовместимы между собой. Кроме того, в Visual C++ и Delphi по умолчанию приняты разные соглашения о передаче параметров в функции. Это значит, что компиляторы Visual C++ и Delphi по-разному вызывают функции из OBJ-модулей и генерируют для этого принципиально разный машинный код. Отсюда возникает масса всяких нюансов, описание которых занимает не одну страницу и не две (если кого-то интересуют подробности, то можно посмотреть книгу В. Юрова «Assembler. Учебник»).

Короче, во времена MS DOS OBJ-модули были очень даже отличной штукой, но сейчас это далеко не самый простой способ программирования на нескольких языках.

➜ **межпрограммное взаимодействие.** Выше мы обсуждали, как «подружить» в одной программе куски кода, написанные на разных языках программирования. Сейчас мы поговорим о том, как «подружить» между собой программы, написанные на разных языках. Под «подружить» имеется ввиду «научить разные программы обмениваться между собой данными».

Начнем с того, что в Windows у каждого процесса есть свое адресное пространство, недоступное другим процессам. С одной стороны, это хорошо — каждый процесс работает в своем адресном пространстве и не мешает другим процессам. С другой стороны, это плохо, потому что если одна программа хочет передать другой какие-то данные, она не может сделать это напрямую через память. Одним из самых простых решений в этой ситуации является использование memory-mapped файлов.

Memory-mapped файл — это, по сути, именованный участок оперативной памяти, к которому может получить доступ любой работающий процесс. Приведем две программы, на примере которых рассмотрим основные методы работы с memory-mapped файлами. Первая программа — MemoryMapped1 — будет создавать memory-mapped файл и записывать в него строчки, введенные с клавиатуры. Вторая программа — MemoryMapped2, будет считывать эти строчки из memory-mapped файла и выводить их на монитор. Обе эти программы будут консольными.

Начнем с первой программы. Сначала напишем ее на Visual C++. Откроем Visual Studio и создадим новое консольное приложение MemoryMapped1.

SPECIAL МНЕНИЕ

КРИС КАСПЕРСКИ
САМЫЙ ИЗВЕСТНЫЙ В КОМПЬЮТЕРНОМ СООБЩЕСТВЕ ГРЫЗУН. ЭТОТ ЧЕЛОВЕКООБРАЗНЫЙ ХАКЕР ТОЧИТ ВСЕ — НАЧИНАЯ ОТ БЕЗОБИДНЫХ ПРОГРАММ И ЗАКАНЧИВАЯ КРЕПКИМ КОМПЬЮТЕРНЫМ ЖЕЛЕЗОМ. НЕ ПОПАДАЙСЯ ЕМУ ПОД ХВОСТ!

ВАВИЛОНСКАЯ БАШНЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

«Зачем так много языков, неужели нельзя было обойтись одним, а теперь вот сиди и учи!» Но на самом деле их не так уж и много, да и те исповедуют схожие концепции и парадигмы. Инновация, появившаяся в одном из них, со временем проникает во все остальные. Языки, выпадающие из этой общей группы, просто не выживают (форт, лисп). Язык определяет сознание, а сознание, в свою очередь, формирует язык. Один. Конец XX века прошел под знаменем ООП, распространившим свое влияние практически на все языки и даже просочившимся в ассемблер! Новые языки возникают всякий раз, когда их творец сталкивается с задачей, для реализации которой не существует адекватных инструментов. Тогда он берет наиболее близкие ему парадигмы, добавляет несколько свежих идей и… если язык получается удачным, им начинают пользоваться во всем мире. Это эволюционный путь языкового развития (B → C → C ++). Так же существует специализированные языки (sh, php, sql), заточенные для решения узконаправленных задач.

Использование структур, объявленных на C++

```
#include <stdio.h>

struct my_str1{
int mem1;
char both;
};

struct my_str2{
int mem2;
char both;
};

int main()
{
          my_str1 str1;
          my_str2 str2;

          __asm{
                 lea eax, str1 ; заносим в eax адрес str1
                 ; поскольку both есть и в str1 и в str2, для доступа к
                 ; str1.both нужно явно указывать str1
                 mov [eax]str1.both, 1
                 ; поскольку mem1 есть только в str1, для доступа к str1.mem
                 ; указывать str1 не обязательно
                 mov [eax].mem1, 1
          }
}
```

Eax не обязательно будет равно 1

```
#include <stdio.h>

int main()
{
          int i;

          __asm{
                 ; установим eax в 1
                 mov eax,1
          }

          // выведем значение eax
          printf("EAX=1\n");

          __asm{
                 ; присвоим значение eax переменной i
                 mov i, eax
          }

          // выведем значение eax и убедимся, что оно
          // не совпадает с предыдущим
          printf("EAX=%d", i);
```

(4) Сразу же отключим в настройках компилятора использование UNICODE. Пропишем в MemoryMapped1.cpp заголовочные файлы windows.h (чтобы можно было без проблем использовать Windows API) и stdio.h:

```
// заголовочные файлы
#include <windows.h>
#include <stdio.h>
```

затем впишем в main() следующий код:

```
int main()
{
 // создаем memory-mapped файл
 HANDLE hFile =
CreateFileMapping(INVALID_HANDLE_VALUE,
NULL,\
 PAGE_READWRITE, 0, 1024,
LPTSTR("DigitalPoem"));
```

Приведенный выше код вызывает API Create FileMapping, которая создает memory-mapped файл. Коротко остановимся на параметрах, передаваемых в CreateFileMapping. Первый параметр — хендл ранее открытого файла на диске. Никакого файла на диске мы с вами не открывали, поэтому мы передаем здесь INVALID_HANDLE_VALUE. Второй параметр — указатель на структуру SECURITY_ATTRIBUTES, в которой мы **(5)** можем установить права доступа к нашему memory-mapped файлу. Ничего особого мы устанавливать не собираемся, поэтому второй параметр у нас 0. Третий параметр — защита (protection) memory-mapped файла. Мы хотим читать и писать memory-mapped файл, поэтому перейдем к константе PAGE_READWRITE. Четвертый и пятый параметр — это двойные слова. Вместе они определяют одно 64-разрядное число — размер memory-mapped файла в байтах. Нам хватит одного килобайта, поэтому четвертый параметр (который определяет старшее двойное слово в значении размера) у нас равен 0, а пятый параметр (самое младшее двойное слово в значении размера) равен 1024. И, наконец, пятый параметр «DigitalPoem» — это уникальное имя нашего memory-mapped файла.

Если API CreateFileMapping завершилась успешно, в hFile будет находиться хендл созданного файла. В случае неудачи в hFile будет 0. Чтобы отследить это, вставляем в нашу программу проверку:

```
// ошибка?
if(hFile==0)
{
 // да, ошибка — выведем сообщение
и закончим работу
 printf("Error! CreateFileMapping
returns NULL.\n");
 return 0;
}
```

После того, как memory-mapped файл создан, его надо отобразить в адресное пространство приложения. Для этого используется API MapViewOfFile. Код следующий:

```
// отобразим memory-mapped файл с полным доступом
LPVOID pFileContent = MapViewOfFile(hFile,\
FILE_MAP_ALL_ACCESS, 0, 0, 0);
```

Опять коротко остановимся на параметрах, которые мы передаем в MapViewOfFile. Первый параметр — хендл memory-mapped файла. Вторым параметром у нас передается FILE_MAP_ALL_ACCESS. Это значит, что мы хотим получить полный доступ к отображенному файлу. Третий и четвертый параметры определят 64-разрядное смещение отображаемого участка относительно начала файла. Поскольку мы хотим отображать файл с начала (с нулевого смещения), у нас нули. Ну, и, наконец, пятый параметр — число типа DWORD, которое определяет, сколько байт нужно отобразить. Мы передаем 0 — это значит, что нам нужно все.

Если MapViewOfFile завершилась успешно, в pFileContent будет смещение отображаемого файла. В случае неудачи там будет NULL. Отслеживаешь, и если MapViewOfFile вернула NULL, то закрываешь хендл memory-mapped файла с помощью API CloseHandle и выходишь:

```
// memory-mapped файл отображен?
if(pFileContent==NULL)
{
 // нет, открыть memory-mapped файл не удалось
 printf("Error! MapViewOfFile returns NULL.");
 // закрываем хендл memory-mapped файла и выходим
 CloseHandle(hFile);
 return 0;
};
```

Дальше организуем цикл, который читает строки с клавиатуры. В качестве буфера для этих строк используется наш memory-mapped файл.

цикл выглядит следующим образом:

```
// инициализируем счетчик сообщений
int i = 1;

// будем передавать через memory-mapped файл сообщения
// до тех пор, пока пользователь не введет «stop»
while(strcmp((char*)pFileContent, "stop")!=0)
{
 // ожидаем ввода
 printf("Message #%d: ", i);
 scanf("%1023s\0", (char*)pFileContent, 1024);

 // увеличиваем счетчик сообщений
 i++;
};
```

Когда программа завершает работу, прекращаем отображение memory-mapped файла с помощью API UnmapViewOfFile и закрываем его хендл:

```
// закрываем отображение и хендл memory-mapped файла
UnmapViewOfFile(pFileContent);
CloseHandle(hFile);

// выходим
return 0;
}
```

На Delphi MemoryMapped1 реализуется точно так же. Чтобы не повторяться, прокомментируем принципиальные моменты.

Открываешь Borland Delphi и создаешь новое консольное приложение. Сохраняешь его под именем MemoryMapped1. Затем прописываешь в начале файла MemoryMapped1.dpr:

```
{$APPTYPE CONSOLE}
// использовать Windows API
uses Windows;
```

затем прописываешь необходимые переменные:

```
var
// хендл memory-mapped файла
hFile: THandle;

// указатель на отображение memory-mapped файла в памяти
pFileContent: pointer;

// счетчик количества отображений
i: integer = 1;

// строка, в которую будет осуществляться ввод
s: string;
```

SPECIAL ОБЗОР

MEDIUM

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА C#:

Учебное пособие — М.: БИНОМ, Лаборатория знаний, 2006 / Биллиг В.А. / 483 страницы

Язык C# изучается с самых азов — встроенных типов данных, способов организации данных, выражений и операторов языка, управляющих структур, процедур и функций. Особое внимание уделяется наследованию и универсальным классам. Рассматривается среда разработки Visual Studio .Net и классы библиотеки FCL каркаса Framework .Net. Обсуждаются вопросы корректности программных систем, их устойчивости, повторного использования и расширяемости. Масса наглядных примеров, что называется, «инклудид».

HARD

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Язык Си: учеб. пособие — СПб.: БХВ-Петербург, 2006 / Демидович Е.М. / 440 страниц

По секрету скажем, что книга — не что иное, как утечка информации с курса лекций по дисциплине «Основы алгоритмизации и программирования», читаемого в Белорусском государственном университете информатики и радиоэлектроники на факультете компьютерных систем и сетей. Основные приемы программирования в общем случае не зависят от языка программирования, но конкретно в этой книге в качестве базового используется язык Си. Книга просто кишит примерами. Для некоторых задач рассматривается несколько вариантов и способов программирования, что гораздо нагляднее, нежели лобовое навязывание чего-то одного. Почти все программы пособия компилировались и выполнялись в среде BorlandC 3.1 и Microsoft Visual C++ 6.0.

Inline-ассемблер Delphi в основном похож на MASM

```
program Example;

{$APPTYPE CONSOLE}

var
        myVar: integer;

begin
        asm
              mov myVar, -1
              mov eax, offset myVar {offset с глобальной переменной}
              mov bl, byte ptr [eax+2]{выражение MASM}
        @@1: {локальная ссылка}
              dec eax
              cmp bl, 0
              jl @@1
        end;
end.
```

Как и в MASM, можно использовать глобальные метки

```
program Example2;

{$APPTYPE CONSOLE}

label lab1; {объявляем глобальную метку}

begin
        asm
              jmp lab1 {безусловный переход на глобальную метку}
              mov eax, 1
        lab1:
        end;
end.
```

Можно использовать константы, переменные и записи, уже определенные в программе

```
program Example3;

{$APPTYPE CONSOLE}

label lab1;

var

a: record
        a_mem: integer;
        a_mem2: char;
end;

i: integer;

begin
        asm
              {работаем с переменной i}
              mov eax, 1
              mov i, eax

              {работаем с записью a}
              mov eax, offset a
              {при обращении к a_mem запись a указывать обязательно
              и именно в таком формате}
              mov [eax+a].a_mem, 1
        end;
end.
```

(6) **затем создаешь memory-mapped файл и проверяешь, успешно ли завершилась CreateFileMapping:**

```
begin
// создаем memory-mapped файл
hFile:= CreateFileMapping(INVALID_HAN-
DLE_VALUE, nil,
PAGE_READWRITE, 0, 1024, 'DigitalPoem');

// memory-mapped файл открыт?
if(hFile=0) then
begin
 // нет, открыть memory-mapped файл
не удалось
 writeln('Error! MapViewOfFile returns
nil.');
 exit;
end;
```

отображаешь memory-mapped файл в память и проверяешь, получилось ли:

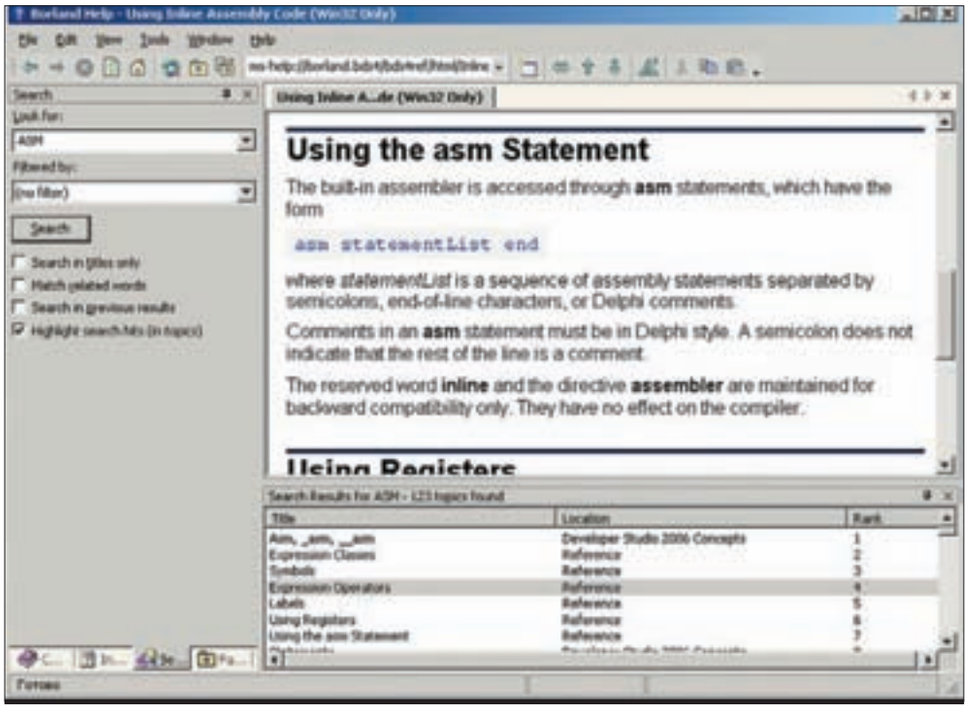
(7)

```
// отобразим memory-mapped файл с полным
доступом
pFileContent:= MapViewOfFile(hFile,
FILE_MAP_ALL_ACCESS, 0, 0, 0);

// memory-mapped файл отображен?
if pFileContent=nil then
begin
 // нет, открыть memory-mapped файл
не удалось
 writeln('Error! MapViewOfFile returns
nil.');

 // закрываем хендл memory-mapped файла
и выходим
 CloseHandle(hFile);
 exit;
end;
```

(8)

организуешь цикл передачи в memory-mapped файл введенных с клавиатуры строк:

```
// будем передавать через memory-mapped
файл сообщения
// до тех пор, пока пользователь
не введет «stop»
while s<>'stop' do
begin
 // читаем строку с клавиатуры
 write('Message #', i, ': ');
 readln(s);

 // записываем строку в memory-mapped
файл
 CopyMemory(pFileContent, pchar(s),
length(s));

 // увеличиваем счетчик сообщений
 inc(i);

end;
```



Справка Borland Help

Если не снять eax со стека, получишь ошибку

и после всего:

```
// закрываем отображение
и хендл memory-mapped файла
UnmapViewOfFile(pFileContent);
CloseHandle(hFile);
end.
```

Теперь можно начинать писать программу MemoryMapped2, которая будет считывать строчки из memory-mapped файла и выводить их в консоль. Сначала рассмотрим соответствующий код на Visual C++.

Снова открываешь Visual Studio и создаешь консольное приложение MemoryMapped2. Выделяешь исходный код MemoryMapped1 и копируешь в файл MemoryMapped2.cpp. Затем целиком удаляешь цикл ввода строк, начинающийся со строчки while(strcmp((char*)pFileContent, "stop")!=0), и вместо него вписываешь цикл чтения строк из memory-mapped файла:

```
// считываем из memory-mapped файла
сообщения, пока
// не считаем «stop»
while(strcmp((char*)pFileContent,
"stop")!=0)
{
// в memory-mapped файле есть строка?
if(strcmp((char*)pFileContent, "")!=0)
{
 // да — выводим сообщение
 printf("Incoming message
#%d: %s\n",\
 i, (char*)pFileContent);

 // очищаем (заполняем нулями) буфер
 memset(pFileContent, '\0', 1024);

 // увеличиваем счетчик сообщений
 i++;
 }
};
```

Вот и все. Как видишь, единственное отличие MemoryMapped2 от MemoryMapped1 в том, что чтение из memory-mapped файла, а не запись в него. Аналогично, чтобы получить MemoryMapped2 на Delphi, в исходном коде MemoryMapped1.pas достаточно заменить цикл записи строк, который начинается с while s<>'stop' do, на цикл чтения:

```
// считываем из memory-mapped файла
сообщения, пока
// не считаем «stop»
while s<>'stop' do
begin

 // в memory-mapped файле есть строка?
 if (byte(pFileContent^)<>0) then
 begin
  // да — выводим сообщение
  s:=string(pchar(pFileContent));
  writeln('Incoming message ',i,': ',s);

  // очищаем (заполняем нулями) буфер
  FillMemory(pFileContent, 1024, 0);

  // увеличиваем счетчик сообщений
  inc(i);
 end;
end;
```

Конечно, это не единственный способ «подружить» между собой две программы. Есть еще DDE, именованные каналы, сокеты, COM, DCOM... Но это намного сложнее и требует больше кода C

www.osp.ru/text/302/178361/
эмпирическое сравнение семи языков программирования (Лутц Прехельт)

http://schools.keldysh.ru/sch444/museum/LANR/evol.htm
эволюция языков программирования

dynamic link libraries

DLL — ОДИН ИЗ САМЫХ ПРОСТЫХ И ПРАВИЛЬНЫХ СПОСОБОВ ПРОГРАММИРОВАНИЯ НА НЕСКОЛЬКИХ ЯЗЫКАХ. ДЛЯ ОЗНАКОМЛЕНИЯ РЕКОМЕНДУЕТСЯ ПОЧИТАТЬ ДВЕ ОЧЕНЬ ХОРОШИЕ СТАТЬИ:

[1] А.Н. ВАЛЬВАЧЕВ, К.А. СУРКОВ, Д.А. СУРКОВ, Ю.М. ЧЕТЫРЬКО. «ДИНАМИЧЕСКИ ЗАГРУЖАЕМЫЕ БИБЛИОТЕКИ» WWW.RSDN.RU/ARTICLE/DELPHI/DELPHI_7_05.XML

[2] А. УВАРОВ. «РАБОТА С БИБЛИОТЕКАМИ ДИНАМИЧЕСКОЙ КОМПОНОВКИ (DLL)» WWW.REALCODING.NET/ARTICLE/VIEW/2713

В ЭТИХ СТАТЬЯХ ИЗЛОЖЕНО ПРАКТИЧЕСКИ ВСЕ НЕОБХОДИМОЕ, ЧТОБЫ НАЧАТЬ СОЗДАВАТЬ И ИСПОЛЬЗОВАТЬ DLL НА VISUAL C++ И DELPHI. ЕСЛИ ЭТОГО ПОКАЖЕТСЯ МАЛО — GOOGLE В ПОМОЩЬ

как сделать из слона муху

ОБРАБОТКА БОЛЬШИХ ОБЪЕМОВ ДАННЫХ В НЕБОЛЬШОМ АДРЕСНОМ ПРОСТРАНСТВЕ

СОВРЕМЕННЫЙ КОМПЬЮТЕР ОБЛАДАЕТ КАК МИНИМУМ 512 МЕТРАМИ ПАМЯТИ (ПЛЮС ФАЙЛ ПОДКАЧКИ), ПОЭТОМУ ПРОГРАММИСТЫ НЕ ОСОБО ЗАБОТЯТСЯ ОБ ЭКОНОМИИ ОПЕРАТИВКИ. МЫ ВЫДЕЛЯЕМ ПОД СОБСТВЕННЫЕ НУЖДЫ ТОННЫ ЯЧЕЕК И СТРАНИЦ, НЕ ОБРАЩАЯ ВНИМАНИЯ НА ВОЗМОЖНЫЕ ПРОБЛЕМЫ. А ВЕДЬ ПАМЯТЬ — НЕ РЕЗИНОВАЯ И МОЖЕТ КОГДА-НИБУДЬ ЗАКОНЧИТЬСЯ. ЕСЛИ КАЖДЫЙ БУДЕТ ТАК БЕЗДАРНО РАСХОДОВАТЬ СОДЕРЖИМОЕ МИКРОСХЕМ, ТО ДЛЯ НОРМАЛЬНОЙ РАБОТЫ НЕ ХВАТИТ И ГИГА ОПЕРАТИВКИ

Михаил Фленов aka Horrific
www.vr-online.ru

→ **стоит ли экономить.** Даже при наличии 512 Мегабайт расходовать память, не думая о последствиях, глупо. Дело в том, что Windows XP в домашней редакции уже съедает от этого объема 128 метров, а профессиональная редакция отнимает и все 256. Всякие примочки и побрякушки в районе часов, антивирусы и сетевые экраны могут отнять еще 64 метра. Это не дело, и кроме того, расходовать предоставленные ресурсы разумно необходимо всегда!

→ **массивы.** Самый банальный и бессмысленный расход памяти происходит из-за использования массивов фиксированной длины. Допустим, что тебе нужно хранить в памяти заранее неизвестное количество чисел. Как поступить в этом случае? Можно зарезервировать максимально возможный буфер и никаких проблем. Да, проблем никаких, а вот утечка памяти произойдет достаточно серьезная.

Допустим, что ты выделил массив из 1024 чисел. Если каждое твое число типа Integer, то из-под ног утекает 4096 байт памяти. А если тип данных Int64? А если это не число, а структура размером в 100 байт? Это уже сто килобайт памяти. Десять таких массивов и драгоценный мегабайт уходит в небытие. Простейший вариант решения проблемы — в массиве хранить не сами структуры, а указатели. В этом случае размер массива значительно сокращается, но он все равно не оптимален. Мы резервируем 1024 элемента, а реально в программе может использоваться только один.

Идеальное решение — использование динамических массивов. Не стоит бояться динамики, она безобидна по сравнению с Фредди Крюгером :). Можно выделить два основных типа динамических массивов:

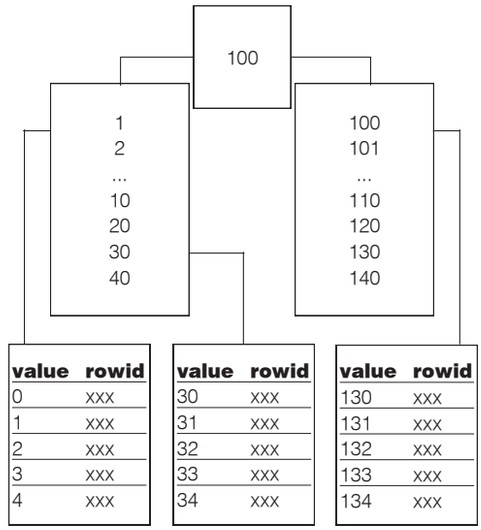
[1] Просто выделяем память для хранения элементов массива, а по мере необходимости память расширяется. Данный вариант самый простой и самый компактный, но очень неудобный, когда необходимо изменять последовательность элементов (например, для сортировки) или удалять элементы из середины массива.

[2] Можно написать класс, который будет управлять динамическим массивом. Простейший вариант такого класса набросан в листинге. Данный код упрощен до минимума. На компакт-диске ты найдешь более полный вариант с примером использования. Да, такое хранение данных требует некоторой избы-

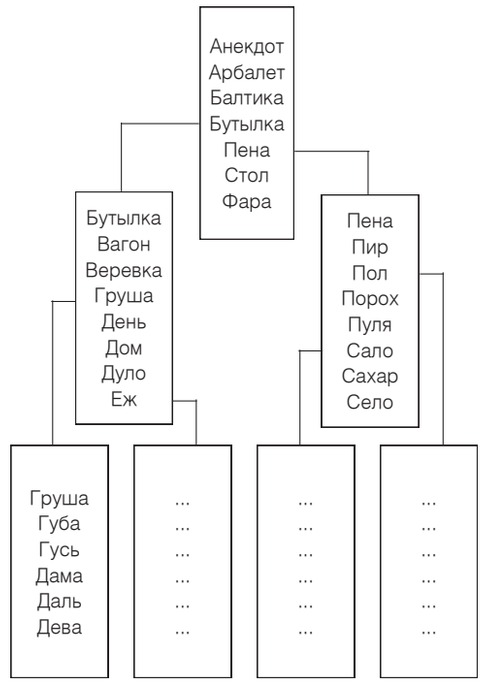
точности для каждого элемента, но зато массив полностью динамичен и занимает в памяти ровно столько, сколько необходимо. Ничего лишнего резервировать не нужно. А главное — размерность массива ограничена размером типа данных, который используется для хранения количества элементов.

Как вариант, для хранения динамического массива указателей можно использовать класс TList (в данном случае совет относится к программистам Delphi).

На самом деле, алгоритмов создания списков очень много, но мы выделили эти два, как наиболее эффективные и достаточно простые в реализации. В зависимости от задачи можно придумать и другой алгоритм: все зависит от того, как хранятся данные, нужно ли их сортировать, есть ли необходимость удалять из середины, куда добавляются новые элементы (в конец или могут вставляться в середину списка) и т.д.



B-Tree деревья в Oracle (на кончиках веток находятся значения индекса и ROWID)



Пример древовидного индекса

➔ **по чуть-чуть.** Допустим, что ты пишешь программу воспроизведения/записи звука или видео. На первый взгляд, для воспроизведения mp3-файла необходимо столько же памяти, сколько данные занимают в файле. Достаточно только загрузить все в память и отправить данные звуковой карте. Ошибочка вышла. Не факт, что звуковая карта сможет понять сжатые данные. Придется производить декомпрессию самостоятельно и направлять звуковой карте несжатые данные. Если разложить по полочкам mp3-файл, то для хранения всех данных может понадобиться 60, а то и все 100 мегабайт памяти. Такого расточительства тебя ни один пользователь не простит.

Еще один пример — простой ZIP-архиватор. Что если необходимо сжать файл размером в 500 метров? Загрузить его в память, а потом заархивировать его там будет достаточно проблематично.

Во всех этих случаях вполне логичным будет выделить два небольших участка памяти. В первый необходимо загружать данные из файла для последующей манипуляции (компрессии/декомпрессии), а во второй будет помещаться результат работы. Тут нужно быть аккуратным, потому что размеры сжатых и несжатых данных разные и легко выйти за границу буфера. Не стоит пытаться запихнуть в оперативку все сразу, потому что это бессмысленно. Всегда загружай данные небольшими порциями.

➔ **циклическое использование памяти.** При работе со звуком Microsoft предоставляет нам очень удобные механизмы работы с памятью, которые мы сейчас и рассмотрим.

Итак, MMSystem получает при воспроизведении и возвращает при записи звуковые данные через специализированные буферы. Так как данные загружаются порциями, для предотвращения задержек в воспроизведении/записи музыки используется очередь из буферов. Для нормальной работы достаточно двух блоков памяти, но можно сделать и больше.

Как пример, выделяем память для хранения двух буферов данных. Размер памяти можно выбрать любой, но он должен быть не слишком маленьким, дабы избежать слишком частой смены буферов, и не слишком большим, чтобы сэкономить память для других целей. Точный размер подбирается в зависимости от качества звука, потому что от этого зависит и размер необходимых данных. Из личного опыта — выделяй столько, чтобы хватило на воспроизведение/запись не менее 5 и не более 10 секунд звука.

Теперь нужно использовать буфер 1, а буфер 2 помещаем в очередь. Так как MMSystem работает в отдельном потоке, ждем, когда буфер 1 освободится. Когда это произойдет, система переключится на следующий буфер в очереди (буфер 2), и пока она будет с ним работать, мы освобождаем буфер 1 или заполняем его новыми данными и снова помещаем в очередь. Таким образом, с помощью цикла можно по частям обработать данные любого размера с минимальными расходами памяти.

Два буфера в данном случае минимально необходимы, но желательно использовать 3 или 4. Дело в том, что в момент освобождения буфера система может быть занята и не успеет его очистить, подготовить и вернуть обратно в очередь. В этом случае может произойти заикание в воспроизведении или даже сбой, если идет запись.

ДОНАЛЬД КНУТ

«Если вы считаете себя действительно хорошим программистом, прочтите книгу «Искусство программирования» Д.Кнута. Вам определенно следует прислать мне резюме, если вы прочли эту книгу до конца» — Билл Гейтс.

Дональд Е. Кнут — автор всемирно известной серии книг, посвященной основным алгоритмам и методам вычислительной математики, создатель настольных издательских систем TEX и METAFONT, предназначенных для верстки физико-математических книг. Его перу принадлежит 19 книг и более 160 статей. Дональд Кнут является почетным профессором Стэндфордского университета в области программирования и вычислительной математики.

Профессор Кнут удостоен многочисленных премий и наград, среди которых можно отметить ACM Turing Award, Medal of Science президента Картера, AMS Steele Prize за серию научно-популярных статей. В ноябре 1996 года Дональд Кнут был удостоен престижной награды Kyoto Prize в области передовых технологий.

➔ **слабое место.** Когда данных очень много, то даже для очень мощного процессора просматривать их все очень сложно и проблематично. Дело в том, что процессор — не самое главное в современном компьютере. Слабым местом по-прежнему остается жесткий диск. Это механика и, не смотря на то, что производители пытаются из нее выжать максимум, она остается очень слабой.

Можно попытаться оптимизировать процесс поиска данных через дефрагментацию, дабы головка жесткого диска не прыгала по блину как угорелая, а читала их последовательно, но затраты в соотношении с результатом слишком большие. Можно читать данные большими кусками, а обработку перенести в отдельный поток, чтобы максимально распараллелить задачи, но это тоже не совсем то.

Если данных действительно очень много, а памяти очень мало, в бой вступают мозги и алгоритмы. Мы должны реализовать алгоритм так, чтобы не было необходимости просматривать все данные. И в этом нам помогут сортировка и индексация.

➔ **сортировка.** Когда данные упорядочены, с ними проще работать и искать необходимую информацию. Допустим, что есть список всех жителей города, а тебе нужно найти в нем номер телефона Иванова. С такой фамилией в целом городе может быть не один человек, а сотня. Просматривать весь список достаточно проблематично, а если он будет упорядочен по фамилиям, то достаточно будет просмотреть все записи на букву «И» и готово. Реальная экономия времени и памяти.

➔ **индексация.** Поддерживать списки в упорядоченном виде достаточно проблематично, особенно если записи очень часто изменяются, добавляются, удаляются, и при этом каждая из них может иметь произвольный или очень большой размер. В этом случае записи хранят на диске в произвольном виде, а для быстрого поиска используют индексы. Так поступает большинство баз данных, чтобы сэкономить память и процессорное время.

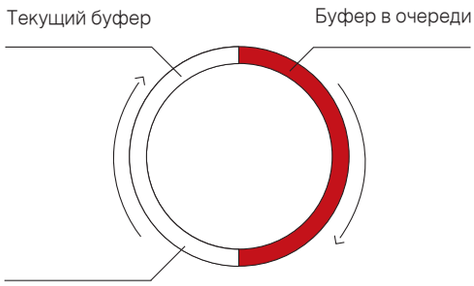
Индексы могут состоять из двух колонок: поле, по которому происходит упорядочивание данных, и указатель на место, где находятся данные этой записи. Сами данные хранятся в произвольном порядке, а вот индексная табличка сортируется. Так как она очень маленькая и состоит только из двух полей, такую табличку достаточно легко поддерживать в упорядоченном виде, а для ее хранения в памяти нужно намного меньше места, чем для хранения всего списка.

Теперь, если нам снова нужно найти номер телефона по фамилии, мы бежим по индексной таблице и ищем необходимую фамилию. Когда она найдена, переходим по указателю из индексной таблицы и получаем данные человека, в том числе его номер телефона и адрес.

Индексные таблички хороши, но только в меру. Для телефонного справочника может понадобиться три индекса: для фамилии, телефона и адреса. Благодаря трем индексам мы можем легко упорядочить данные по любому из этих парамет-

Пример динамического массива

```
type
 PMassivItem = ^TMassivItem;
 TMassivItem = record
 nextItem:PMassivItem; //следующий элемент массива
 prevItem:PMassivItem; //предыдущий элемент массива
 Name:String; // пример данных, которые нужно хранить
 // еще здесь могут быть поля элемента массива
 end;

 TMassiv = class
 public
 FirstItem:PMassivItem; // первый элемент списка
 LastItem:PMassivItem; // последний элемент списка
 itemsNumber:Integer; // количество элементов
 constructor Create(str:String); // конструктор
 procedure AddItem(str:String); // добавление элемента
 procedure DeleteItem(index:Integer); // удаление
 end;

procedure TMassiv.AddItem(str:String);
var
 newItem:PMassivItem;
begin
 newItem:= new(PMassivItem);
 newItem.Name:=str;
 newItem.nextItem:=nil;
 newItem.prevItem:=LastItem;
 LastItem.nextItem:=newItem;
 LastItem:=newItem;
 Inc(itemsNumber);
end;

constructor TMassiv.Create(str:String);
begin
 itemsNumber:=1;
 FirstItem:=new(PMassivItem);
 FirstItem.Name:=str;
 FirstItem.nextItem:=nil;
 FirstItem.prevItem:=nil;
 LastItem:=FirstItem;
end;

procedure TMassiv.DeleteItem(index: Integer);
var
 i:Integer;
 currentItem:PMassivItem;
begin
 // поиск удаляемого элемента
 currentItem:=FirstItem;
 for i:=0 to index-1 do
 currentItem:=currentItem.nextItem;

 // наводим новые связи и освобождаем память
 if currentItem.prevItem<>nil then
 currentItem.prevItem.nextItem:=currentItem.nextItem;
 if currentItem.nextItem<>nil then
 currentItem.nextItem.prevItem:=currentItem.prevItem;
 Dispose(currentItem);//очистка памяти
 Dec(itemsNumber); //уменьшаем счетчик
end;
```



Циклическое использование памяти

ров, но содержать в упорядоченном виде придется уже три таблички, а это лишние сложности.

➔ **деревья.** Индексы — очень мощное средство для экономии памяти и упрощения сортировки. Но для поиска данных все равно приходится сканировать всю таблицу, пока мы не найдем необходимую запись. Если искомый текст начинается на букву «А», то мы найдем его быстро, ведь он будет находиться в самом начале. Но если искомая строка начинается на «Я», то нам придется сканировать все данные до конца, и экономия скорости будет небольшой. Да, мы все еще экономим память, так как загружаем в оперативку только индексную таблицу, а не все данные, но скорость невысокая. А если построить индекс в виде дерева, то сэкономим и место, и время.

Индексы в виде деревьев очень часто используются в базах данных, в том числе и в MS SQL Server. Так что понимание пригодится не только программистам (для реализации в собственных проектах), но и администраторам, — для лучшего понимания внутренностей баз данных.

Индекс не зря называют деревом, потому что все данные в нем располагаются именно в таком виде. Рассмотрим, как выглядит дерево в памяти, и ты сразу увидишь все преимущества и недостатки.

➔ **прогулка по дереву.** Итак, дерево состоит из блоков равного размера. Размер блока должен быть достаточен для хранения хотя бы нескольких записей, но он и не должен быть слишком большим. Некоторые специалисты рекомендуют делать его равным 8 Кило (любимый размерчик). Неизвестно, откуда взялась эта рекомендация, но блок получается не большой и не маленький, в самый раз. Внутри блока все строки желательно упорядочить, дабы упросить поиск. Так как блоки у нас небольшие, то поддерживать все записи в упорядоченном состоянии достаточно просто.

Попробуем найти слово «Гусь». Начинаем поиск с первого блока и последовательно просматриваем все строки. Находим слово «Бутылка», которое меньше, чем искомый «Гусь», но следующее слово «Пена» будет уже больше. Переходим в следующий блок, на который указывает найденная «Бутылочка». Ищем здесь и видим, что ближайшее к искомому — слово «Груша». Переходим по ссылке на следующий блок, где как раз и есть нужный нам «Гусь».

Ощутил преимущества древовидного индекса? Если нет, то вот они:

[1] Для поиска даже самой последней записи в наборе данных не нужно просматривать абсолютно все индексные блоки. Вполне возможно, что достаточно будет просмотреть только три или четыре.

[2] На поиск любых данных затрачивается примерно одинаковое время.

[3] Для эффективной работы вполне достаточно столько памяти, сколько занимает один индексный блок. Все блоки держать в оперативке нет смысла, но если есть лишняя память, то можно кэшировать блоки.

Недостаток один, но очень серьезный — поддерживать такой индекс не так уж и просто.

Вот так при минимуме затрат можно быстро искать и обрабатывать большие объемы информации. В данном случае мы затронули разновидность лиственных под названием B-Tree (Balanced Tree или сбалансированные деревья). Бывают и другие разновидности, но это уже совершенно другая история.

➔ **кластер или нет.** Существует две разновидности деревьев — кластерные и некластерные. В первом случае на кончиках веток индексного дерева находятся сами данные. Это значит, что, прогулявшись по всем блокам, мы находим непосредственно искомые данные. Получается, что в данном случае данные упорядочены с помощью такого индекса физически. Конечно же, на один список (таблицу) можно создать только один кластерный индекс, потому что упорядочить физически по двум разным параметрам один и тот же набор данных просто нереально.

В некластерном дереве на кончиках веток находятся только ссылки на данные, а сами данные могут находиться где угодно на диске и в любом порядке. Таких индексов можно создавать сколько угодно.

➔ **итого.** Это только основные алгоритмы экономии памяти, плюс мы немного затронули оптимизацию. Удачи! **C**

SPECIAL ОБЗОР

MEDIUM

ПРОГРАММИРОВАНИЕ WIN32 API В DELPHI

СПб.: БХВ-Петербург, 2005 / Кузан Д.Я. / 368 страниц
Разумная цена: 185 рублей

Применение различных интерфейсов прикладного программирования Windows (Win32 API) при разработке приложений с использованием Borland Delphi. Основы работы с API, практическое применение API при создании приложений для работы с электронной почтой (MAPI), со средствами коммуникаций (TAPI), мультимедиа (MMCI), графическим интерфейсом и т.п. Причем в книге нет столь популярного «визуального» программирования типа «возьмите компонент такой-то из палитры компонентов такой-то и положите его на форму такую-то». Все сугубо про взаимодействие Delphi с различными API. Но это скорее вводная книга в мир API, так как в ней ты не найдешь подробного и полного описания каждого интерфейса прикладного программирования. И она «заставит» тебя начать изучать различные API.

HARD

ПРОГРАММИРОВАНИЕ НА C++ ГЛАЗАМИ ХАКЕРА

СПб.: БХВ-Петербург, 2006 / Фленов М.Е. / 336 страниц
Разумная цена: 160 рублей

Множество нестандартных приемов программирования, примеры использования недокументированных функций и возможностей языка C++. Причем автор подошел к проблеме шутя, показывая все на примере маленьких смешных программ, с помощью которых можно легко разыграть друзей. По ходу книги ты узнаешь, как оптимизировать размер и скорость выполнения программ. Большая часть примеров — программирование в интернете, к примеру, создание сканера портов или троянского коня. Приведены алгоритмы написания утилит и их подробный анализ. Так что в итоге ты научишься не только писать свой троян, но и некую защиту от троянов, зная их принцип работы.

чудеса легкости

РЕФАКТОРИНГ — НЕОБХОДИМОСТЬ ИЛИ МОДА?

ЧТО ТАКОЕ РЕФАКТОРИНГ? Я ВИДЕЛ ДОСТАТОЧНО МНОГО ОПРЕДЕЛЕНИЙ ЭТОГО ПОНЯТИЯ, НО ВСЕ ОНИ СВОДЯТСЯ К УЛУЧШЕНИЮ СУЩЕСТВУЮЩЕГО КОДА. ЕСЛИ ТЫ ДУМАЕШЬ, ЧТО ПИШЕШЬ ИДЕАЛЬНЫЙ КОД, КОТОРЫЙ НУЖНО УЛУЧШАТЬ ТОЛЬКО В ТЕХ СЛУЧАЯХ, КОГДА ОН НЕ РАБОТАЕТ, ТО СИЛЬНО ЗАБЛУЖДАЕШЬСЯ. УЛУЧШЕНИЯ НУЖНЫ ДАЖЕ ТОГДА, КОГДА КОД РАБОТАЕТ ВПОЛНЕ КОРРЕКТНО. ДЛЯ ЧЕГО, КОГДА И КАК НУЖНО УЛУЧШАТЬ — УЗНАЕШЬ В ЭТОЙ СТАТЬЕ

Фленов Михаил
http://www.vr-online.ru

➔ **для чего нужен рефакторинг.** Что можно улучшить в коде, который и так уже работает и выполняет возложенные на него функции? Если программу не планируется улучшать и добавлять новые возможности, то можно больше ничего не трогать. Лучше даже удалить исходники, дабы не пытаться разбирать бардак или использовать его в будущем. Но если программа нужна не на один день, то рефакторинг необходим.

Ты хотя бы иногда убираешь на своем рабочем столе? Хоть иногда очищаешь компьютер от мусора и неиспользуемых программ? Ну и, наконец, убираешь в квартире, чтобы не ходить по ковру из бумажек и окурков? Последний пример, может быть, и не очень удачный, потому что убираться в квартире большинство из нас не любит (сбрасывая это занятие на маму/жену/подругу), но ходить по мусору и жить в бардаке и пыли уж точно неприятно.

То же самое касается и кода. Если он написан хорошо и легко читается, то его приятно сопровождать, добавлять новые функции или оптимизировать. Такой код можно даже использовать в других проектах. Но если код написан ужасно, то намного приятнее переписать все с нуля. Из-за плохого кода лет пять назад я забросил один из любимых проектов и переписал заново. Этот проект я начинал еще в 1996 году и ни разу не задумывался о рефакторинге. Тогда даже понятия такого не знал (возможно, его и не было). Получив этот неприятный опыт, я теперь на всех этапах написания кода задумываюсь о его улучшении и при первом же появлении мусора улучшаю и очищаю код.

Важность рефакторинга подчеркивает и то, что во всех последних версиях сред разработки (Delphi 2005/2006, JBuilder 9 и выше, Visual Studio 2005) появились различные мастера и функции для улучшения кода. Эти функции не могут охватить все сферы рефакторинга, да и без знания основ их использовать проблематично.

В этой статье я постараюсь рассказать о ряде принципов улучшения кода. Основой этих принципов является мой личный многолетний опыт, и кто-то может с ним не согласиться, а кто-то найдет для себя что-то новое.

➔ **комментарии.** Большинство из нас во время написания кода никогда не задумывается о комментариях и не любит писать их. Да, хорошо написанный и оформленный код должен читаться без комментариев, но все же, небольшие пояснения никогда не помешают. После написания метода я стараюсь выделить пару минут своего драгоценного времени на комментарии, которые в последствие позволят мгновенно понять, что я натворил.

К комментариям применимо одно простое правило — они должны быть короткими и понятными. Если среда разработки поддерживает комментарии TODO, то не стесняйся использовать их, они реально помогают.

➔ **читабельность.** Многие из программистов абсолютно не обращают внимание на имена переменных. Когда программа состоит из 1000 строк кода, то понять назначение переменной не сложно. Но когда исходный код исчисляется 5-ю тысячами строк и более, начинаются серьезные проблемы. Особенно через годик после его написания. Спросите программиста, что может храниться в переменной Temp, Str или Param? Первое, что приходит в голову — там хранятся отходы жизнедеятельности человека, которые мы сбрасываем в туалет.

Если по имени переменной нельзя понять, для чего она и что хранит, то на чтение кода приходится тратить лишнее время. Необходимо сначала найти место, где объявляется переменная, а потом определить, что в нее записывается. А то иногда без бутылки пива с кодом разобраться просто невозможно.

А что если переменная называется iFileLength? Вот тут уже легко понять, что это целочисленная переменная Integer или int (в зависимости от языка) и она содержит длину файла. Чтение такого кода и сопровождение значительно упрощается.

О переменных можно говорить очень много, но я кратко дам несколько основных правил, которых желательно придерживаться:

1 Имя переменной должно содержать какой-либо префикс, который укажет нам на тип данных.

2 Имя должно быть осмысленное, и без лишних пояснений должно быть понятно, что за ним скрывается.

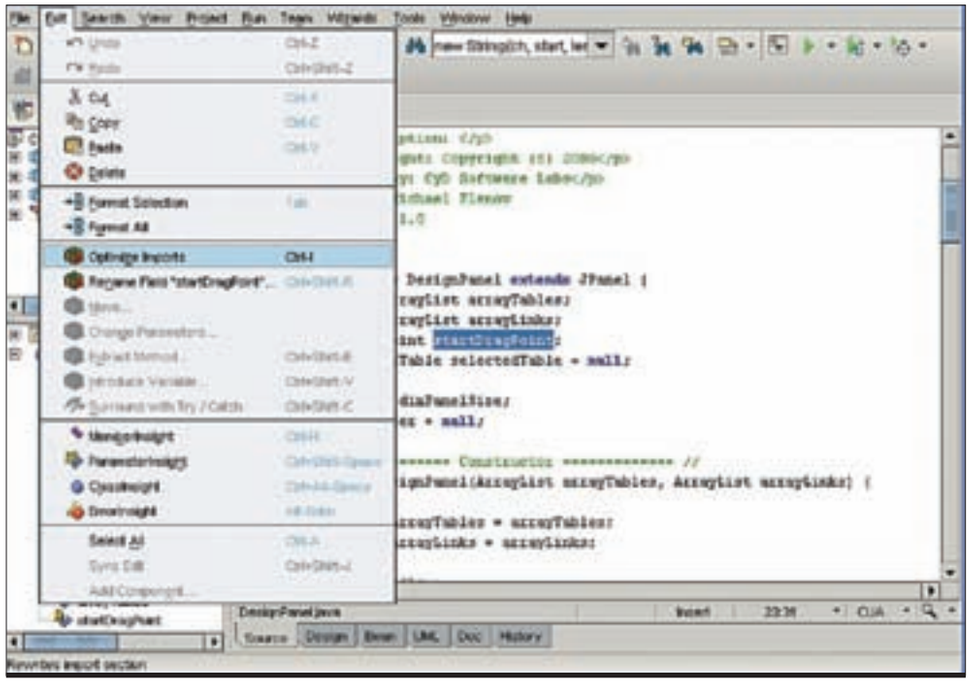
3 Одна переменная никогда не должна выполнять сразу два действия в одном и том же блоке кода. Например, внутри одной процедуры переменная IfileLength не должна сначала содержать длину файла, а потом использоваться в качестве счетчика в цикле. В последствии очень просто забыть этот нюанс и неправильно использовать значение переменной, что приведет к проблемам безопасности.

Для улучшения переменных нужно просто дать всем им понятные имена и убедиться, что каждая из них используется только для одной цели.

Да, лишняя переменная отнимает место в памяти, но если она имеет простой тип, то это не смертельный размер и минимальная плата за простую, но очень хорошую защиту от неправильного использования переменных. Прикинь сейчас, какие ошибки ты чаще всего исправляешь? Лично я чаще исправляю собственную невнимательность, и большинство программеров, я думаю, тоже. Если следовать описанным выше правилам, вероятность ошибок из-за невнимательности уменьшается.

➔ **методы.** Я думаю, не стоит и говорить о том, что имя метода должно быть понятным. Понятными должны быть все имена. Давайте поговорим о нюансах, которые относятся именно к методам/процедурам/функциям.

Каждый метод должен выполнять одну и только одну функцию. Лучше всего, если эта функция будет максимально узкой. Например, если мы создаем метод загрузки файла, то метод должен только загружать файл, а анализ и другие возможности необходимо реализовывать в других методах. Но при этом нельзя отделять такие функции, как проверка корректности и безопасности. Выносить их в отдельный метод очень сложно,



В JBuilder функции рефакторинга спрятаны в контекстном меню и меню Edit

иногда невозможно, а в большинстве случаев — глупо. Реализация и проверка корректности неразделимы, поэтому в методе загрузки файла нужно не забыть проверить наличие файла, корректность его открытия, доступность данных, размер буфера для чтения и т.д.

Как же мы любим объединять весь возможный код в одном методе! А ведь это грозит нам следующими проблемами:

— МЕТОД ОЧЕНЬ СЛОЖНО ЧИТАТЬ, ДАЖЕ ПРИ НАЛИЧИИ БОЛЬШОГО ЧИСЛА ПОДРОБНЫХ КОММЕНТАРИЕВ.

— КОД СЛОЖНЕЕ ИСПОЛЬЗОВАТЬ ПОВТОРНО. КОГДА МЕТОД ВЫПОЛНЯЕТ УЗКУЮ ЗАДАЧУ, ТО ЕГО МОЖНО ИСПОЛЬЗОВАТЬ В ДРУГОМ МЕСТЕ ПРОГРАММЫ, ГДЕ НЕОБХОДИМЫ ТЕ ЖЕ РАСЧЕТЫ. ЕСЛИ МЕТОД РЕШАЕТ НЕСКОЛЬКО ЗАДАЧ, ТО ВЕРОЯТНОСТЬ НЕОБХОДИМОСТИ ВЫПОЛНЕНИЯ ВСЕГО ТОГО ЖЕ В ДРУГОМ МЕСТЕ — НАМНОГО НИЖЕ.

— УСЛОЖНЯЕТСЯ ОТЛАДКА МЕТОДА, А В БОЛЬШИХ ПРОЕКТАХ И ОТЛАДКА ТЕСТИРОВАНИЯ ОТНИМАЕТ ДОСТАТОЧНО МНОГО ВРЕМЕНИ И СИЛ.

Проблем у больших и универсальных методов очень много, но эти пункты я бы выделил в качестве основных. Если методы выполняют узкие задачи, то мы избегаем всех этих непростых моментов.

Проблем у больших и универсальных методов очень много, но эти пункты я бы выделил в качестве основных. Если методы выполняют узкие задачи, то мы избегаем всех этих непростых моментов.

У данного совета могут быть противники, потому что избыточное количество вызовов методов — это снижение скорости. Да, вызов каждой процедуры требует лишних расходов, особенно если она получает много параметров. А если процедура за время выполнения программы будет вызываться сотни, а то и тысячи раз, то это уже серьезный удар по производительности. Да, лишние методы не нужны, но когда приходится выбирать между качеством кода и оптимизацией, я всегда выбираю первое и всем советую действовать так же.

➔ **улучшение методов.** В случае с методами банальным переименованием уже не обойтись. Тут уже приходится вмешиваться в код более радикально. Если функция получилась очень большой или в глаза бросается выполнение нескольких задач, необходимо разбить ее на несколько более маленьких функций.

В Delphi 2005 появилась достаточно интеллектуальная функция Refactor/Extract Method. В JBuilder та же функция спрятана в меню Edit/Extract Method. Чтобы воспользоваться ею, необходимо выделить отрывок кода, который нужно переместить в отдельный метод и выбрать указанный пункт меню. Перед нами открывается диалоговое окно, в котором достаточно ввести имя нового метода. Все остальное среда разработки сделает сама, а именно:

[1] БУДЕТ СОЗДАН И КОРРЕКТНО ОБЪЯВЛЕН НОВЫЙ МЕТОД.

[2] ВМЕСТО КОДА, КОТОРЫЙ МЫ ВЫДЕЛИЛИ, БУДЕТ ВСТАВЛЕН ВЫЗОВ ВНОВЬ СОЗДАННОГО МЕТОДА.

Улучшение может потребоваться и в тех случаях, когда схожий код встречается в нескольких методах. И пусть они решают узкую задачу, но повторение кода не есть хорошо. Намного лучше выделить отдельный метод: это упростит сопровождение программы.

➔ **размер метода.** Многие авторитетные программисты считают, что методы должны быть максимально короткими, и если код не помещается на экран, то его необходимо разбить на несколько методов. Я бы не злоупотреблял этим правилом, потому что большое количество методов — тоже минус. Если метод решает одну и только одну задачу, то не стоит его делить на несколько, даже если он занимает два экрана. Просто купи монитор побольше :). Это, конечно же, шутка, — монитор побольше не нужен. Достаточно один раз отладить задачу и забыть про нее.

➔ **видимость методов.** Видимость методов также относится к рефакторингу. Ни один лишний метод не должен быть виден другим классам для прямого вызова. Дабы не следить за видимостью, я всегда создаю все методы закрытыми, и только если какой-то из них понадобилось вызвать извне, я делаю его открытым.

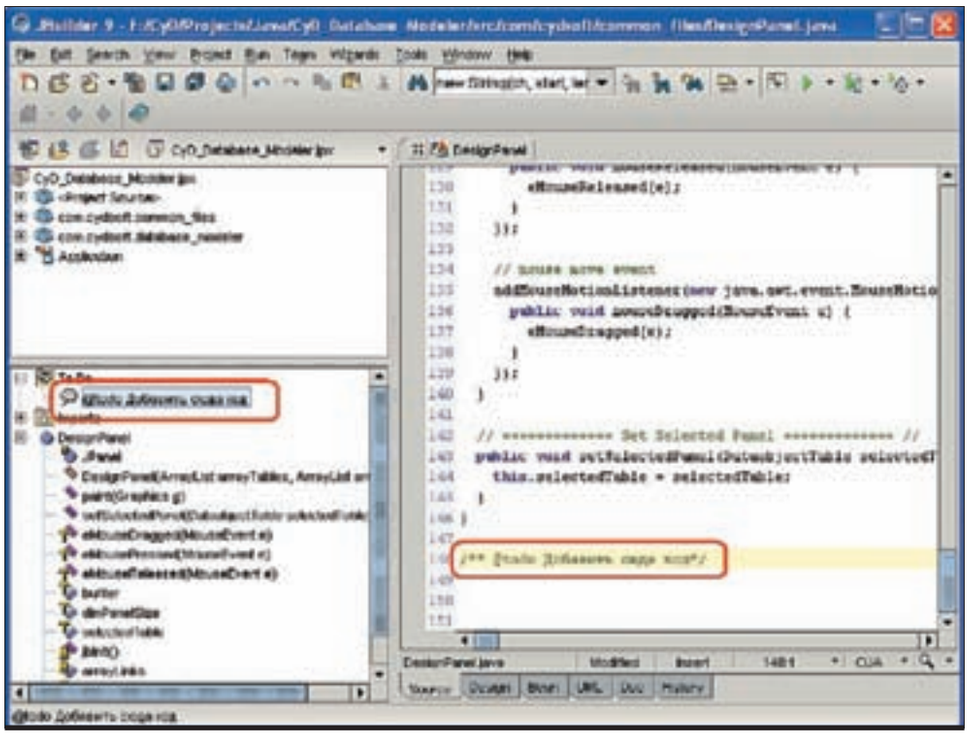
Когда ты заканчиваешь работу над программой, просмотри все классы и видимость их методов. Возможно, какой-либо метод ты сделал открытым, а потом нашел другой способ решения задачи. Не помешает закрыть этот метод. Да, на производительности программы это не скажется, но код станет лучше и опрятнее.

➔ **читабельность метода.** Язык C++ великолепен и позволяет нам одну и ту же операцию записать несколькими способами. Программисты на других языках так же могут записывать определенные операции в одну строку. Если строка кода становится нечитабельной, то следует задуматься, а не разбить ли ее на две строки?

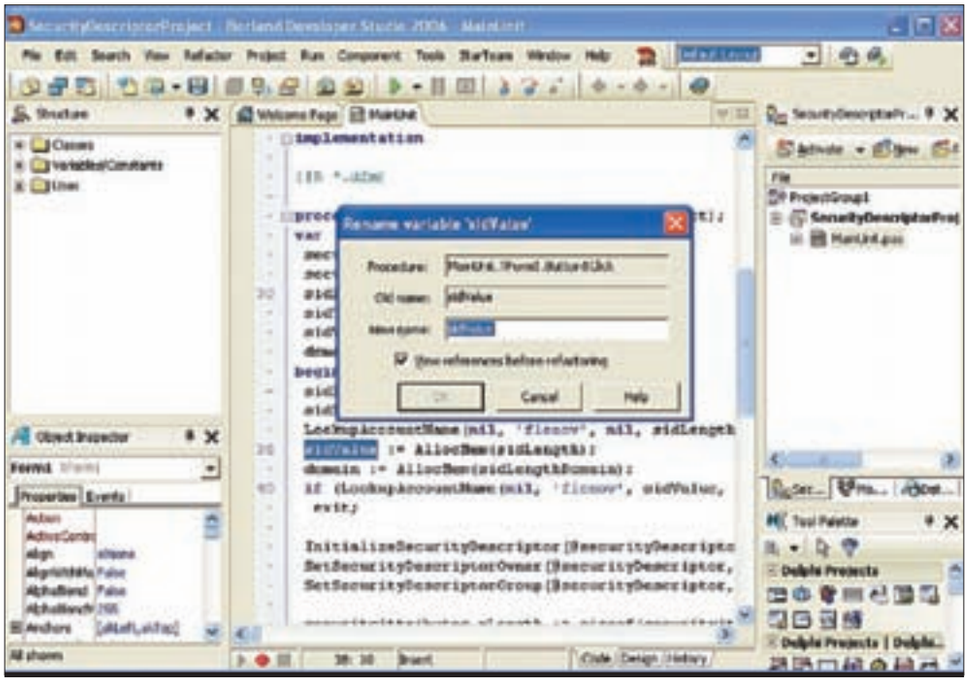
Длинные строки кода очень тяжелы для восприятия. Лучше всего, если строка будет помещаться на экран полностью. Если с вертикальной прокруткой для просмотра метода целиком можно смириться, то горизонтальная неудобна, и с ней необходимо бороться.

➔ **классы.** Помимо хорошего именования классов, они также, как и методы, должны решать узкую задачу. Не стоит создавать один класс, который будет решать задачу дома, сарая и гаража одновременно. Лучше выстроить иерархию из нескольких классов (возможно, с одним базовым, который будет содержать общие методы и свойства).

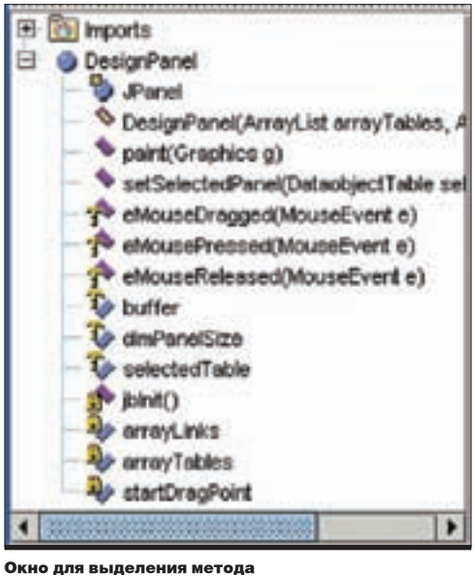
В JBuilder и Delphi 2006 есть очень удобная возможность выделения методов в отдельный



Вот такая удобная поддержка todo-комментариев в JBuilder



Окно переименования переменных в Delphi 2006



Окно для выделения метода

класс или интерфейс. Эти возможности спрятаны в меню Refactor/Extract Interface и Refactor/Extract superclass. Допустим, ты написал класс, который характеризует дом. Затем в программе понадобилось создать гараж. Так как некоторые характеристики гаража будут такими же, как и для дома, можно выделить их в суперкласс, а затем наследовать от него гараж, сарай и другие строения со схожими параметрами.

➔ **улучшение классов.** Что еще можно сделать, чтобы классы стали лучше? Тут уже нужен более глубокий анализ на уровне методов. Необходимо выяснить, действительно ли методы относятся к задаче, решаемой классом. Если нет, то метод необходимо вынести за пределы класса. Куда, — тут я ничего хорошего посоветовать не могу, потому что все зависит от программы.

Если метод принимает очень много параметров, то это не есть хорошо. Посмотри: возможно, некоторые параметры можно хранить в качестве членов класса. Да, в этом случае понадобятся методы для установки значения данного члена (ничего пошлого, так называют свойства классов), но если данный параметр нужен в нескольких методах, то эти затраты оправданы на все 100%.

➔ **форматирование кода.** Оформление кода также можно отнести к рефакторингу. Тут у программистов Visual Studio немного больше преимуществ. Чего стоит меню Edit/Advanced. Тут есть все необходимое, чтобы сделать код более удобным для чтения. В Delphi тоже есть большинство этих функций, но они по умолчанию спрятаны. И если ты знаешь горячие клавиши и вынесешь соответствующие кнопки на панель, тебе будет гораздо удобнее работать.

О том, как оформлять код, написано уже много. Лично мое мнение — он должен быть оформлен так, чтобы тебе было удобно его читать. Если для понимания отдельного метода или отдельной строки приходится напрягать мозг, то форматирование неудачное и следует задуматься о его улучшении.

Современные среды разработки автоматически форматируют код. Visual Studio и JBuilder делают это уже давно, а теперь и в Delphi появились подобные возможности. Но настройки JBuilder пошли дальше. Здесь можно выбрать, где и как должны располагаться скобки { и }. Существуют и отдельные мастера для разных сред разработки, которые автоматически отформатируют код.

➔ **структура кода.** Обычная сортировка методов также может повысить читабельность. Как сортировать методы? Это зависит от многих факторов, но желательно, чтобы порядок методов соответствовал заранее определенному правилу. Какие могут быть порядки?

1 По алфавиту. Банальный, но очень удобный порядок, позволяющий быстро находить место, где должен располагаться нужный метод. Рекомендую использовать его, когда модуль/класс очень большой и содержит очень много методов.

2 По типу. Сначала можно расположить методы, устанавливающие значения свойств, затем производящие расчеты, а потом — выводящие информацию на экран.

Возможно, ты предпочтешь другую сортировку, главное, чтобы она была удобной и позволяла быстро найти нужный метод. В современных средах разработки есть специальные окна, которые упрощают поиск, но при наличии сортировки поиск будет еще проще.

➔ **рефакторинг и безопасность.** Если посмотреть на рефакторинг, то он не вносит в код ничего сверхъестественного. Безопасность остается на том же уровне, а производительность может даже упасть. Но это только на первый взгляд. На самом деле, безопасность растет, пусть косвенно, но очень качественно:

> 1 ВО ВРЕМЯ РЕФАКТОРИНГА ТЫ ЛУЧШЕ ПОНИМАЕШЬ КОД ПРОГРАММЫ И В ЭТОТ МОМЕНТ МОЖЕШЬ НАЙТИ НЕДОЧЕТЫ В ЛОГИКЕ.
>
> 2 СДЕЛАЙ КОД БОЛЕЕ ПОНЯТНЫМ, И ЕГО ЛЕГЧЕ БУДЕТ ОТЛАЖИВАТЬ И НАХОДИТЬ НЕДОЧЕТЫ.
>
> 3 ЕСЛИ МЕТОД ИЛИ КЛАСС ВЫПОЛНЯЕТ УЗКУЮ ЗАДАЧУ, ТО НАМНОГО СЛОЖНЕЕ СОВЕРШИТЬ ОШИБКУ.

Рефакторинг еще и решает одну простую, но очень важную проблему безопасности — защищает от возможных проблем в будущем. Через некоторое время, когда ты вернешься к коду для его улучшения или изменения возможностей, многое забывается, и если код плохой, то очень легко допустить ошибку и навредить безопасности. Ну как можно вспомнить через год, что определенная переменная внутри функции используется тремя разными способами и хранит различные данные. Чем проще код для чтения и проще использование методов/параметров, тем менее вероятна ошибка.

➔ **refactoring complete.** За отведенное мне место мы смогли рассмотреть только базовые понятия рефакторинга, но если следовать им, то ты сможешь реально повысить качество кода. Попробуй и уже скоро ты увидишь, что рефакторинг реально помогает. На банальное улучшение никогда не хватает времени, но если его выкроить хотя бы чуточку, то код станет намного лучше и с ним будет приятно работать в будущем **C**

акробатика для программиста

МОЩЬ И БЕСПОМОЩНОСТЬ АВТОМАТИЧЕСКОЙ ОПТИМИЗАЦИИ

К КОНЦУ 90-Х ГОДОВ КОМПИЛЯТОРЫ ПО СВОЕЙ ЭФФЕКТИВНОСТИ ВПЛОТНУЮ ПРИБЛИЗИЛИСЬ К АССЕМБЛЕРУ, ОДНАКО, ВСЕ ЕЩЕ СУЩЕСТВУЕТ МНОЖЕСТВО КОНСТРУКЦИЙ, НЕПОДДАЮЩИХСЯ АВТОМАТИЧЕСКОЙ ОПТИМИЗАЦИИ, НО ЛЕГКО ТРАНСФОРМИРУЕМЫХ ВРУЧНУЮ. ПОКАЖЕМ, КАК НАДО И КАК НЕ НАДО ОПТИМИЗИРОВАТЬ ПРОГРАММЫ НА ПРИМЕРЕ MICROSOFT VISUAL C++, INTEL C++, BORLAND BUILDER, GCC И HEWLETT-PACKARD C++

Крис Касперски ака мыщъх
no e-mail

С точки зрения прикладного программиста, компилятор — это черный ящик, заглатывающий исходный текст и выплевывающий двоичный файл. Какие процессы протекают в его «пищеварительном тракте» — неизвестно. Разработчики компиляторов крайне поверхностно описывают механизмы оптимизации в прилагаемой документации, так и не давая ответа на вопрос: что именно оптимизирует компилятор, а что — нет.

Как следствие — одни разработчики пишут ужасно кривой код, надеясь, что все огрехи исправит компилятор (ведь он же «оптимизирующий!»). Другие же, наоборот, пытаются помочь компилятору, оптимизируя программу вручную и производя кучу глупых и ненужных действий, например, заменяя a = b/4 на a = b >> 2, хотя любой компилятор сделает это и сам. А вот поместить в регистр переменную, переданную по ссылке, он уже не решается (почему — см. «удаление лишних обращений к памяти»), то же самое относится и к выносу инвариантных функций из тела цикла.

Для достижения наивысшей эффективности необходимо помочь компилятору, придерживаясь определенных правил программирования (кстати говоря, не описанных в штатной документации). Если ты недоволен быстродействием откомпилированной программы, не спеши переписывать ее на ассемблере. Попробуй сначала оптимизировать код путем реконструкции исходного текста: в большинстве случаев получишь тот же самый результат, но потратишь меньше времени и сохранишь переносимость.

➜ **что не надо оптимизировать.** Начнем с того, что не надо оптимизировать, позволяя транслятору сделать это за нас (нехай делает). В частности, практически все оптимизирующие компиляторы умеют вычислять константы на стадии трансляции. В различных русскоязычных источниках этот прием оптимизации называется как «сверткой», так и «размножением» констант, что соответствует английским терминам «constant folding/propagation». Еще один английский термин из той же кучи — «constant elimination» (буквально — «изгнание констант»). Все это синонимы, и описывают они один и тот же механизм вычисления константных выражений (как целочисленных, так и вещественных), в результате чего a = 2 * 2 превращается в a = 4, а x = 4*y/2 — в x = 2*y.

Побочным эффектом оптимизации становится потеря переполнения (если таковое имело место быть). С точки зрения математика, выражения foo = bar/4*4 и foo = bar полностью эквивалентны, но если переменные foo и bar целые, то неоптимизированный вариант обнуляет два младших бита bar! Некоторые программисты умышленно используют этот прием, вместо того чтобы воспользоваться «foo = bar & (~3)», а потом ругаются на «глючный» оптимизатор!

За исключением Intel C++, все рассматриваемые компиляторы поддерживают «улучшенную свертку констант» («advanced constant fold-

ing propagation»), заменяя все константные переменные их непосредственным значением, в результате чего выражение a = 2; b = 2 * a; c = b a; превращается в c = 2, а переменные a и b (если они нигде более не используются) уничтожаются.

В операторах ветвления («if», «?» и «switch») константные условия встречаются редко и обычно являются следствием чрезмерного увлечения #define, вот, например, как здесь:

неоптимизированный вариант

```
#define MAX_SIZE                    1024
#define REQ_SIZE                     512
#define HDR_SIZE                       6
…
int a = REQ_SIZE + HDR_SIZE;
if (a <= MAX_SIZE) foo(a); else return
ERR_SIZE;
```

За исключением Intel C++, все рассматриваемые компиляторы выполняют константную подстановку, оптимизируя код, избавляясь от ветвления и ликвидируя «мертвый код», который никогда не выполняется:

```
foo(528);
```

Выполнять эту оптимизацию вручную совершенно необязательно, поскольку оптимизированный листинг менее нагляден и совершенно негибок. По правилам этикета программирования, все константы должны быть вынесены в #define, что значительно уменьшает число ошибок.

➔ **удаление копий переменных.** Для повышения читабельности листинга программисты обычно загоняют каждую сущность в «свою» переменную, не обращая внимания на образующуюся избыточность: многие переменные либо полностью дублируются, либо связаны друг с другом несложным математическим соотношением, и для экономии памяти их можно сократить алгебраическим путем.

В англоязычной литературе данный прием называется «размножением копий» («copy propagation»), что на первый взгляд не совсем логично, но если задуматься, то все проясняется: да, мы сокращаем переменные, размножая копии хранящихся в них значений, что наглядно продемонстрировано в следующем примере:

переменные a и b — лишние

```
main(int n, char** v)
{
  int a, b;
  …
  a = n+1;
  b = 1-a;        // избавляется от
переменной a: (1 - (n + 1));
  return a-b;     // избавляется от
переменной b: ((n + 1) - (1 - (n + 1)));
}
```

После оптимизации переменные a и b исчезают, а return возвращает значение выражения (2*n+1):

```
main(int n, char** v)
{
  return 2*n+1;
}
```

➔ **устранение хвостовой рекурсии.** Хвостовой рекурсией («tail recursion») называется такой тип рекурсии, при котором вызов рекурсивной функции следует непосредственно за оператором return. Классическим примером тому является алгоритм вычисления факториала:

```
int fact(int n, int result)
{
  if(n == 0)
  {
    return result;
  }
    else
  {
    return fact(n – 1, result * n);
  }
}
```

Вызов функции — достаточно «дорогостоящая» (в плане процессорных тактов) операция, и, за исключением Intel C++, все рассматриваемые компиляторы трансформируют рекурсивный вызов в цикл:

```
for(i=0; i<n; i++) result *= n;
```

Естественно, оптимизированный код менее нагляден, поэтому выполнять такое преобразование вручную — совершенно необязательно.

➔ **что надо оптимизировать.** Теперь поговорим о том, с чем оптимизирующие компиляторы не справляются и начинают буксовать, резко снижая эффективность целевого кода. Помочь им выбраться из болота — наша задача! Чип и Дейл уже спешат! Ну а мыщъх вращает хвостом. Руководит, значит.

Начнем с функций. Из всех рассматриваемых компиляторов только Intel C++ поддерживает глобальную оптимизацию, а остальные — транслируют функции по отдельности, задействуя «сквозную» оптимизацию только на встраиваемых (inline) функциях. Отсюда следует, что чем выше степень дробления программы на функции (и чем меньше средний размер одной функции), тем ниже качество оптимизации, не говоря уже о накладных расходах на передачу аргументов, открытие кадра стека и т. д.

На мелких функциях, состоящих всего из нескольких строк, оптимизатору просто негде «развернуться», а задействовать агрессивный режим подстановки, «вживляющий» все мелкие функции в тело программы, нежелательно, поскольку это приводит к чрезмерному «разбуханию» программного кода.

Оптимальная стратегия выглядит так: выключаем режим автоматического встраивания и стремимся программировать так, чтобы средний размер каждой функции составлял не менее 100-200 строк.

➔ **удаление неиспользуемых функций.** Большинство компиляторов не удаляют неиспользуемые функции из исходного текста, поскольку используют технологию раздельной компиляции, транслируя исходные тексты в объектные модули, собираемые линкером в исполняемый файл, динамическую библиотеку или драйвер.

Функция, реализованная в одном объектном файле, может вызываться из любых других, но информацией о других модулях компилятор не обладает и потому удалять «неиспользуемые» (с его точки зрения) функции не имеет права.

Теоретически, неиспользуемые функции должен удалять линкер, но популярные форматы объектных файлов к этому не располагают и в грубом приближении представляют собой набор секций (.text, .data и т. д.), каждая из которых, с точки зрения линкера, представляет монолитный блок, внутрь которого линкер не лезет, а просто объединяет блоки тем или иным образом. Вот потому-то и не рекомендуется держать весь проект в одном файле (особенно если это библиотека). Помещай в файл только «родственные» функции, всегда используемые в паре и по отдельности не имеющие никакого смысла.

Посмотрим, как устроена стандартная библиотека языка Си. Большинство функций реализовано в «своем» собственном файле, компилируемом в obj, содержащим только эту функцию и ничего сверх нее! Множество таких obj объединяются библиотекарем в один lib-файл, откуда линкер свободно достает любую необходимую функцию, не таща ничего остального! Собственно говоря, именно для этого библиотеки и придумали. Программисты, помещающие реализации всех функций своей библиотеки в один-единственный файл, совершают большую ошибку!

Из всех рассматриваемых компиляторов, только Intel C++ умеет отслеживать неиспользуемые функции, предотвращая их включение в obj (для этого ему необходимо указать ключ — ipo, активирующий режим глобальной оптимизации).

➔ **вынос инвариантных функций из циклов.** Инвариантными называются функции, результат работы которых не зависит от параметров цикла, и потому их достаточно вычислить всего один раз. Компиляторы, к сожалению, так не поступают, поскольку транслируют все функции по отдельности и не могут знать, какими побочными эффектами обладает та или иная функция (исключение составляют встраиваемые функции, непосредственно вживляемые в код программы).

рассмотрим типичный пример:

```
for(a=0;a<strlen(s);a++) b+=s[a];
```

Если только компилятор не заинлайнит функцию strlen, она будет вычисляться на каждой итерации

цикла, что приведет к значительному снижению производительности. Но если вынести инвариант за пределы цикла, все будет OK:

```
t = strlen(s);
for(a=0;a<t;a++) b+=s[a];
```

➔ **нормализация циклов.** Нормализованным называется цикл, начинающийся с нуля и в каждой итерации увеличивающий свое значение на единицу. В книгах по программированию можно встретить утверждение, что нормализованный цикл компилируется в более компактный и быстродействующий код, однако, это только теоретическая схема, и многие процессорные архитектуры (включая x86) предпочитают иметь дело с циклом, стремящимся к нулю.

рассмотрим типичный цикл:

```
for (a = from; a < to; i+=(-step))
{
  // тело цикла
}
```

алгоритм нормализации выглядит так:

```
for (NCL = 0; i < (to — from + step)/
step — 1; 1)
{
  i = step*NLC + from;
  // тело цикла
}
i = step * _max((to — from + step)/
step, 0) + from;
```

Наибольшую отдачу нормализация дает на циклах с заранее известным количеством итераций, то есть когда выражение (to from + step)/step представляет собой константу, вычисляемую еще на стадии трансляции.

Формально, все рассматриваемые компиляторы поддерживают нормализацию циклов, но не всегда задействуют этот механизм оптимизации, поэтому в наиболее ответственных ситуациях циклы лучше всего нормализовать вручную.

➔ **разворот циклов.** Процессоры с конвейерной архитектурой (к которым относится и x86) плохо справляются с ветвлениями (а циклы как раз и представляют одну из разновидностей ветвлений), резко снижая свою производительность. Образно их можно сравнить с гоночной машиной, ползущей по петляющей дороге. И у машины, и у процессора максимальная скорость достигается только на участках, свободных от ветвлений.

Компактные циклы вида for(a=0;a<n;a++) *dst++= *src++; исполняются крайне медленно и должны быть развернуты (unrolled). Под «разворотом» в общем случае понимается многократное дублирование цикла, которое в классическом случае реализуется так:

```
for(i=1; i<n;i+)
  k += (n % i);
```

цикл, развернутый на 4 итерации (меньший размер, большая скорость)

```
for(i=1; i<n;i+=4)
{
  k += (n % i) +\
  (n % i+1) + \
  (n % i+2) + \
  (n % i+3);
}

// выполняем оставшиеся итерации
for(i=4; i<n;i++) k += (n % i);
```

За исключением Microsoft Visual C++, все остальные рассматриваемые компиляторы умеют разворачивать циклы и самостоятельно, но... делают это настолько неумело, что вместо ожидаемого увеличения производительности сплошь и рядом наблюдается ее падение, поэтому автоматический разворот лучше сразу запретить и оптимизировать программу вручную, подбирая подходящую степень разворота опытным путем (вместе с профилировщиком).

Тут ведь как — чем сильнее разворот, тем больше места занимает код, и появляется риск, что в кэш первого уровня он может вообще не влезть, вызывая обвальное падение производительности! (Подробнее о влиянии степени разворота на быстродействие можно прочитать в моей «технике оптимизации», электронная копия которой, как обычно, лежит на моем мыщъхином ftp://nezumi.org.ru).

➔ **программная конвейеризация.** Классический разворот цикла порождает зависимость по данным. Несмотря на то, что загрузка обрабатываемых ячеек (см. предыдущий листинг) происходит параллельно, следующая операция сложения начинается только после завершения предыдущей, а все остальное время процессор ждет.

Чтобы избавиться от зависимости по данным, необходимо развернуть не только цикл, но и «расщепить» переменную, используемую для суммирования. Такая техника оптимизации называется программной конвейеризацией («software pipelining»), и из всех рассматриваемых компиляторов ее поддерживает только GCC, да и то лишь частично. В то же самое время, она элементарно реализуется «руками»:

```
// обрабатываем первые XXL - (XXL % 4)
итерации
for(i=0; i<XXL;i+=4)
{

  sum_1 += a[i+0];
  sum_2 += a[i+2];
  sum_3 += a[i+3];
  sum_4 += a[i+4];
}

// обрабатываем оставшийся «хвост»
for(i-=XXL; i<XXL;i++)
  sum += a[i];

// складываем все воедино
sum += sum_1 + sum_2 + sum_3 + sum_4;
```

➔ **авто-параллелизм.** Многопроцессорные машины, двуядерные процессоры и процессоры с поддержкой Hyper-Threading эффективны лишь при обработке многопоточных приложений, поскольку всякий поток в каждый момент времени может исполняться только на одном процессоре. Программы, состоящие всего из одного потока, на многопроцессорной машине исполняются с той же скоростью, что и на однопроцессорной (или даже чуть-чуть медленнее, за счет дополнительных накладных расходов).

Для оптимизации под многопроцессорные машины следует разбивать циклы с большим количеством итераций на N циклов меньшего разме-

SPECIAL ОБЗОР

MEDIUM

КЛАССИКА ПРОГРАММИРОВАНИЯ: АЛГОРИТМЫ, ЯЗЫКИ, АВТОМАТЫ, КОМПИЛЯТОРЫ. ПРАКТИЧЕСКИЙ ПОДХОД

СПб.: Наука и Техника, 2006
Мозговой М.В. / 320 страниц
Разумная цена: 197 р.

Что такое алгоритм? Почему одну задачу решить просто, другую сложно, а третью вообще никак не удается? Как создать машину, решающую задачи? Над этими вопросами ученые начали размышлять, когда еще компьютеров не было впомине. И анализ этих вопросов во многом предопределил дальнейшее развитие теории вычислений. Изучение алгоритмов и моделей будет весьма полезно тем, кто занимается практическим программированием. С одной стороны, это отличная возможность расширить кругозор и углубить понимание основных принципов и проблем компьютерной науки. С другой стороны, это реальный способ пополнить собственный инструментарий для ежедневного применения. Практика создания своего компилятора, интерпретация промежуточного кода, системы Линденмайера, машины Тьюринга и многое другое.

ра, помещая каждый из них в свой поток, где N — количество процессоров, обычно равное двум. Такая техника оптимизации называется авто-параллелизмом («auto-parallelization») и наглядно демонстрируется следующим примером:

```
for (i=0; i<XXL; i++)
a[i] = a[i] + b[i] * c[i];
```

Поскольку зависимость по данным отсутствует, цикл можно разбить на два. Первый будет обрабатывать ячейки от 0 до XXL/2, а второй — от XXL/2 до XXL. Тогда на двухпроцессорной машине скорость выполнения цикла практически удвоится:

```
/* поток A */
for (i=0; i<XXL/2; i++)
  a[i] = a[i] + b[i] * c[i];

/* поток B */
for (i=XXL/2; i<XXL; i++)
  a[i] = a[i] + b[i] * c[i];
```

Intel C++ — единственный из всех рассматриваемых компиляторов, поддерживающий технику автопараллизации, активируемую ключом — parallel. Однако, качество оптимизации оставляет желать лучшего, и эту работу лучше осуществлять вручную.

➔ **упорядочивание обращений к памяти.** При обращении к одной-единственной ячейке памяти, в кэш первого уровня загружается целая строка, длина которой, в зависимости от типа процессора, варьируется от 32-х до 128-х или даже 256 байт, поэтому большие массивы выгоднее всего обрабатывать по строкам, а не по столбцам.

обработка массивов по столбцам (неоптимизированный вариант)

```
for(j=0;j<m;j++)
  for(i=0;i<n;i++)
    a[i][j] = b[i][j] + c[i][j];
```

Здесь три массива обрабатываются по столбцам, что крайне непроизводительно, и для достижения наивысшей эффективности циклы i и j следует поменять местами. Устоявшегося названия у данной методики оптимизации нет, и в каждом источнике она называется по-разному: «loop permutation/interchange/reversing», «rearranging array dimensions» и т. д. Как бы там ни было, оптимизированный вариант выглядит так:

```
for(i=0;i<n;i++)
  for(j=0;j<m;j++)
    a[i][j] = b[i][j] + c[i][j];
```

Все рассматриваемые компиляторы поддерживают данную стратегию оптимизации, однако их интеллектуальные способности очень ограничены, и со следующим примером справляется только Hewlett-Packard C++:

сложный случай обработки данных по столбцам

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    for (k=0; k<n; k++)
      a[j][i] = a[j][i] + b[k][i] *
c[j][k];
```

➔ **удаление лишних обращений к памяти.** Компиляторы стремятся размещать переменные в регистрах, избегая «дорогостоящих» операций обращения к памяти, однако, компилятор никогда не может быть уверен, адресуют ли две переменных различные области памяти или обращаются к одной и той же ячейке памяти.

```
f(int *a, int *b)
{
  int x;
  x = *a + *b;// сложение содержимого
двух ячеек
  *b = 0x69;      // изменение ячейки *b,
адрес которой не известен компилятору
  x += *a;     // нет гарантии, что запись
в ячейку *b не изменила ячейку *a
}
```

Компилятор не имеет права на размещение содержимого ячейки *a в регистровой переменной, поскольку, если ячейки *a и *b частично или полностью перекрываются, модификация ячейки *b приводит к неожиданному изменению ячейки *a! Бред, конечно, но ведь Стандарт этого не запрещает, а компилятор обязан следовать Стандарту, иначе его место — на свалке. То же самое относится и к следующему примеру:

лишние обращения к памяти, от которых можно избавиться вручную

```
f(char *x, int *dst, int n)
{
  int i;
  for (i = 0; i < n; i++) *dst += x[i];
}
```

Компилятор не имеет права выносить переменную dst за пределы цикла, в результате чего обращения к памяти происходят в каждой итерации. Чтобы повысить производительность, код должен быть переписан так:

```
f(char *x, int *dst, int n)
{
  int i,t =0;
  for (i=0;i<n;i++) t+=x[i];
  //сохранение суммы во временной
  переменной *dst+=t; запись конечного
  результата в память
}
```

➔ **регистровые ре-ассоциации.** На x86 платформе регистров общего назначения всего семь и их всегда не хватает, особенно в циклах. Чтобы втиснуть в регистры максимальное количество переменных (избежав тем самым обращения к медленной оперативной памяти), приходится прибегать ко всяким ухищрениям. В частности, совмещать счетчик цикла с указателем на обрабатываемые данные.

Код вида «for (i = 0; I < n; i++) n+=a[i];» легко оптимизировать, если переписать его так: «for (p= a; p < &a[n]; p++) n+=*p;». Насколько известно мыщъх'у, впервые эта техника использовалась в компиляторах фирмы Hewlett-Packard, где она фигурировала под термином «register reassociation». А вот остальные рассматриваемые нами компиляторы этого делать, увы, не умеют.

Вот еще один пример, демонстрирующий оптимизацию цикла с тройной вложенностью:

неоптимизированный кандидат на регистровую ре-ассоциацию

```
int a[10][20][30];
void example (void)
{
  int i, j, k;
  for (k = 0; k < 10; k++)
    for (j = 0; j < 10;j++)
      for (i = 0; i < 10; i++)
        a[i][j][k] = 1;
}
```

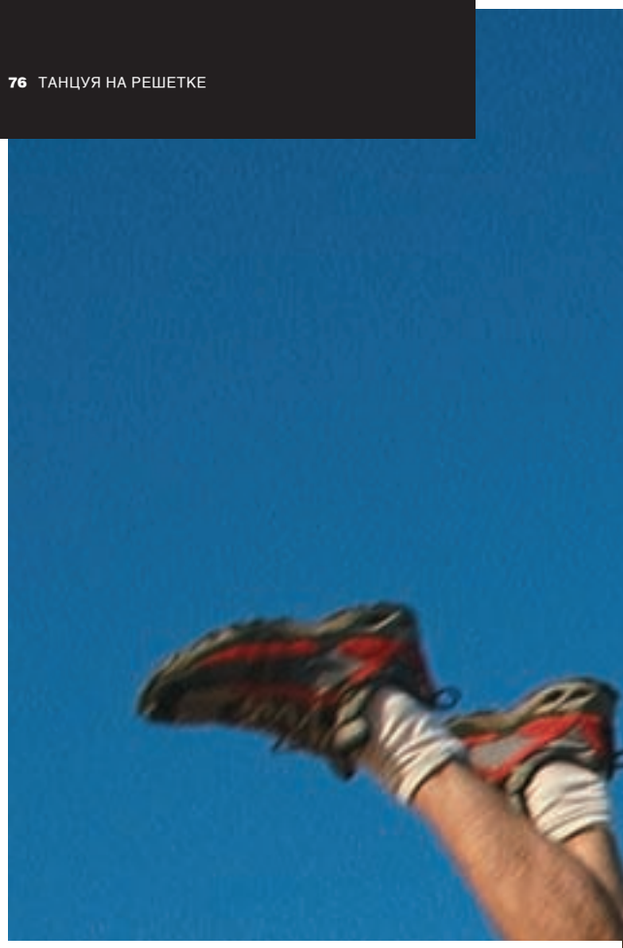
Для достижения наибольшей производительности код следует переписать так (разворот циклов опущен для наглядности):

оптимизированный вариант — счетчик цикла совмещен с указателем на массив

```
int a[10][20][30];
void example (void)
{
  int i, j, k;
  register int (*p)[20][30];
  for (k = 0; k < 10; k++)
    for (j = 0; j < 10; j++)
      for (p = (int (*)[20][30])
&a[0][j][k], i = 0; i < 10; i++)
        *(p++[0][0]) = 1;
}
```

➔ **шаг в будущее.** Собирать свою коллекцию «как надо и как не надо оптимизировать программы» мыщъх начал уже давно (здесь приведена лишь крошечная ее часть). Время шло, компиляторы совершенствовались, и все больше примеров перемещалось из первой категории во вторую. А затем... разработчики компиляторов поутихли, и со временен Microsoft Visual C++ 6.0 новых рывков что-то не наблюдается, поэтому у статьи есть все шансы сохранить свою актуальность в течение нескольких лет. А, возможно, и нет. Так или иначе — твори! И не забывай почитывать творения небезызвестного Кнута на ночь :) C

НОВЫЕ ФОКУСЫ



орочий кульбит

C# 3.0 + LINQ = ЛЮБОВЬ

В ЭТОЙ СТАТЬЕ Я ХОЧУ РАССКАЗАТЬ О ПОСЛЕДНИХ ДОСТИЖЕНИЯХ В ОБЛАСТИ СОЗДАНИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ. РЕЧЬ ПОЙДЕТ О ЯЗЫКЕ, КОТОРЫЙ ТОЛЬКО ГОТОВИТСЯ К ВЫХОДУ — C# ТРЕТЬЕЙ ВЕРСИИ. ЭТОТ ЯЗЫК БУДЕТ ВКЛЮЧЕН В СЛЕДУЮЩУЮ ВЕРСИЮ VISUAL STUDIO — ФЛАГМАНСКИЙ ПРОДУКТ КОМПАНИИ МАЙКРОСОФТ, КОТОРЫЙ ПОЛУЧИЛ КОДОВОЕ ИМЯ «ORCAS». LINQ — ЭТО НОВАЯ ЧАСТЬ ЯЗЫКА C# 3.0 (А ЗАОДНО И VISUAL BASIC 9.0)

Борис Вольфсон
boris@splendot.com http://splendot.com

→ **сначала был...** Возможно, ты будешь смеяться, но сначала был Турбо Паскаль. Да-да, именно этот синий монстр, который сейчас наводит страх на многих программистов. Этот компилятор разработал Андерс Хейлсберг, чье имя можно увидеть в окне «About» Турбо Паскаля. Но время шло — наступила пора Windows, а DOS отошел в мир иной. Общеизвестно, что фирма Борланд выпускает новую интегрированную среду разработки приложений Delphi.

Догадайся, кто возглавлял группу разработчиков Delphi? А откуда в C# появились свойства, система обработки событий и многое другое? Хотя стоп, я забегаю немного вперед, ведь следующий этап нашей истории — появление .NET — технологии Майкрософт, сердцем которой стал новый язык C#. Первая версия языка была очень похожа на Java версии 1.4, хотя и содержала ряд усовершенствований. В октябре 2003 года общественности стала доступна вторая версия языка, главной фишкой которой была работа с генериками (generics) — аналогом шаблонов в языке C++. Теперь на пороге третья версия, которую мы и рассмотрим подробно.

→ **новые фичи языка C# 3.0.** Для начала проверим, что за зверь этот третий си шарп. Самым революционным введением является, конечно же, LINQ (Language-Integrated Query). Фактически, это встроенный язык структурированных запросов, который можно использовать для контейнеров, XML-данных и баз данных. Другим нововведением являются лямбда-выражения, которые служат удобной заменой делегатам. Также в новой версии языка можно инициализировать свойства объекта при его создании, что позволяет немного сократить код:

```
var man = new Man { Name = "Адам"; }
```

Есть новая фишка для самых ленивых программистов, которые не любят писать тип локальных переменных — пишем просто var:

```
var s = "Привет"
```

Если серьезно, это нам очень пригодится при получении результатов LINQ-запросов. К тому же, необязательность указания типа отнюдь не нарушает строгую типизацию языка, так как компилятор сам определяет тип локальной переменной, — фактически мы просто устранили дублирование.

Отбираем претендентов в команду

```
string xml = "<college>" +
             "<student height='195'>John Smith</student>" +
             "<student height='195'>Merry Popins</student>" +
             "<student height='185'>Bill Gates</student>" +
             "<student height='175'>Vasy Pupkin</student>" +
             "<student height='170'>Ivan Petrov</student>" +
             "</college>";
XElement college = XElement.Parse(xml);

var query =
    from student in college.Elements("student")
    where (int)student.Attribute("height") > 180
    select student;

foreach(var result in query)
    Console.WriteLine("Студент {0} имеет рост {1} см",
              (string)result,
              (string)result.Attribute("height"));
```

Ваяем запрос к базе данных

```
SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName],
[t0].[ContactTitle], [t0].[Address], [t0].[City], [t0].[Region],
[t0].[PostalCode], [t0].[Country], [t0].[Phone], [t0].[Fax]
FROM [Customers] AS [t0]
WHERE [t0].[City] = @p0
```

Добавим бонус в базу данных

```
var q =
    from c in db.Customers
    where c.Region == "WA"
    select c;

Console.WriteLine("*** BEFORE ***");
ObjectDumper.Write(q);

Console.WriteLine();
Console.WriteLine("*** INSERT ***");
var newCustomer = new Customer { CustomerID = "MCSFT",
                                 CompanyName = "Microsoft",
                                 ContactName = "John Doe",
                                 ContactTitle = "Sales Manager",
                                 Address = "1 Microsoft Way",
                                 City = "Redmond",
                                 Region = "WA",
                                 PostalCode = "98052",
                                 Country = "USA",
                                 Phone = "(425) 555-1234",
                                 Fax = null
                                 };
db.Customers.Add(newCustomer);
db.SubmitChanges();

Console.WriteLine();
Console.WriteLine("*** AFTER ***");
ObjectDumper.Write(q);
```

(1) Дальше по списку идут типы без имени, позволяющие создавать объекты без указания типа:

```
var man = new { Name = "Адам"; }
```

И, наконец, еще одна новая фишка в ООП — возможность расширять уже существующие классы методами. Для этого служат, как ни странно, методы-расширения :). Подробные сведения можно найти в официальных документах на сайте microsoft.com — конкретные адреса даны на врезке, а мы переходим к главному блюду.

➔ **linq**. Есть такой замечательный язык SQL, который отличается от привычных языков тем, что он является декларативным, то есть на нем не надо описывать, как решить задачу, а достаточно описать, что должно получиться в результате. Хотим получить список книг ценой более двухсот рублей, упорядоченных по имени автора, — так и пишем:

(2)

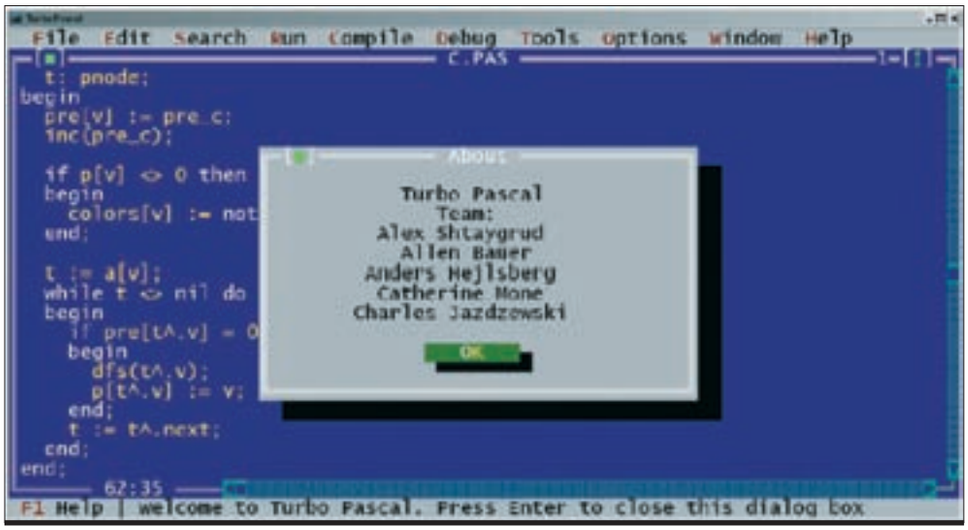
```
SELECT * FROM books
WHERE price > 200
ORDER BY author
```

На других языках нам бы пришлось описывать целый алгоритм пробега по массиву и отбора нужных книг, а в SQL — просто один запрос! Почему

(3) бы ни привнести в язык C# такую функциональность? Первыми такой подход (точнее, похожий подход) использовали авторы языка Cw (читается «Си-омега»), а затем он появился в C# в том виде, в котором мы можем видеть его сейчас. Для тех, кому интересен язык Си-омега, сообщаю, что его компилятор также бесплатен и общедоступен.

➔ **install.** Чтобы установить LINQ, нам понадобится Visual Studio 2005 и немного терпения. Скачать файл можно с сайта одной небезызвестной компании :). Русской версии, к сожалению, на сайте нет, поэтому при установке на русскоязычную Windows появится страшное окно, которое говорит, что нет пользователя или группы пользователей с данным именем — надо просто его создать в панели управления. На вопрос «апдейтить ли язык C# при установке» стоит ответить положительно, с бейсиком я не экспериментировал — так что желающие могут попробовать на свой страх и риск. Также рекомендую скачать «101 пример по LINQ» («100 LINQ Samples»). После установки очень советую изучить папку C:\Program Files\LINQ

Сотня примеров использования LINQ



Turbo Pascal — знаменитый «синий экран DOS»

Preview, то есть папку, куда ты установил LINQ. В ней ты найдешь необходимую документацию, примеры программ и утилиты. Я постараюсь сосредоточиться именно на написании программ — использование утилит остается на откуп читателям. Начнем с самого простого — с контейнеров...

➔ **контейнеры.** Напишем культовую программу — «Hello, world». Идею, как написать ее на LINQ, я самым бесстыдным образом украл у одного программиста в его блоге :).

```
string[] words = { "Hello", "Dog", "Cat",
"Foo", "world" };

var result =
   from w in words
   where w.Length == 5
   select w;

foreach (var s in result)
   Console.Write(s + " ");
```

Давай посмотрим, что делает этот код. Первая строчка абсолютно обычная — она просто создает массив из строк (контейнер), в котором мы будем вести поиск. Далее объявляется бестиповая переменная result и ей присваивается результат запроса. Запрос похож на обычный SQL, только написанный задом наперед :). Это не баг — это фича! Такой порядок нужен, чтобы работал механизм IntelliSense для вывода автоматических подсказок. Теперь препарируем сам запрос. Ключевое слово from указывает, к какому именно контейнеру будет происходить запрос и объявляет переменную w, которая представляет собой отдельный элемент контейнера и будет использована в запросе. Дальше следует оператор where (ограничивающий оператор) — он отбирает только нужные элементы, то есть слова, длина которых равна пяти символам. И последний оператор select указывает, что именно должно попасть в результат запроса. Нижние строчки, как можно было догадаться, просто печатают результат :). Итак, жмем на педаль (она же F5) и получаем долгожданные «Hello world», потому что в массиве words только эти строчки имеют длину пять символов (мы указали ее в операторе where).

➔ **глубокий анализ.** Теперь разберемся, что же мы все-таки написали. Сначала посмотрим под микроскопом на объявление переменной result и подумаем, настолько ли она бестиповая... Совсем нет — компилятор определяет ее тип по результату запроса. Теперь становится понятным, зачем нужны такие переменные, — ведь если мы изменим тип результата запроса, придется менять и тип переменной result.

Попробуем расписать запрос через методы и лямбда-выражение:

```
var result = words.Where( (w) => w.Length
== 5) ;
```

Проясняется и роль лямбда-выражений — они фактически используются как условия отбора в запросах.

Поиграемся с select'ом: создадим список чисел и их буквенных эквивалентов и выведем информацию об их четности:

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7,
2, 0 };
string[] strings = { "zero", "one", "two",
"three", "four", "five", "six", "seven",
"eight", "nine" };

var digitOddEvens =
   from n in numbers
   select new {Digit = strings[n], Even =
   (n % 2 == 0)};
foreach (var d in digitOddEvens)
   Console.WriteLine("The digit {0} is {1}.",
   d.Digit, d.Even ? "even" : "odd");
```

Тут все становится поинтересней, потому что в операторе выбора создается безымянный объект (и эта возможность языка пригодилась :)). Объект будет состоять из двух свойств: Digit содержит наименование цифры, а Even — определяет четность/нечетность числа.

➔ **XLinq.** С контейнерами разобрались. Теперь идем дальше и немного оглядываемся назад. Примемся за следующую часть — за XML — универсальный язык разметки. С помощью него можно хранить любые данные в текстовом формате. Для работы с XML используется часть LINQ, которую обычно называют XLinq.

Задачу поставим простую: есть база данных в формате XML некоего учебного заведения (пусть будет колледж), и нам надо отобрать претендентов в команду. Главный критерий отбора — это рост: он должен быть более 180 сантиметров, иначе будет сложно играть в баскетбол :).

Первым делом мы создали нашу XML-базу данных (смотри листинг 1). Для большей наглядности я привел ее прямо в коде, хотя, разумеется, она обычно лежит в отдельном файле. Дальше базу надо перевести во внутренний формат, чтобы над ней можно было совершать манипуляции. Такой процесс называют парсингом XML. Затем делаем обычный LINQ-запрос, где в качестве условия указываем значение атрибута height больше 180.

➔ **DLinq.** Все, что было прежде — цветочки, теперь начинаются ягодки. Действительно, истинная мощь новой концепции встроенного языка запросов раскрывается при работе с базами данных. Для работы с базой данных, как это ни банально звучит, необходим сервер баз данных. В нашем случае это будет Microsoft SQL Server. В поставку Visual Studio входит его бесплатная версия SQL Server Express — она тоже подойдет для наших экспериментов.

Теперь надо выбрать базу данных, с которой мы будем работать. Я возьму стандартную базу данных NorthWind, которая поставляется с LINQ. Проще всего посмотреть ее структуру и содержание прямо из самой Visual Studio, просто добавив новое соединение и выбрав файл C:\Program Files\LINQ Preview\Data\NORTHWND.MDF.

После приготовлений можно приступать к кодированию. Для начала напишем простейший запрос на выборку всех заказчиков, которые живут в Париже:

```
var q =
   from c in db.Customers
   where c.City == "Paris"
   select c;
ObjectDumper.Write(q);
```

Обрати внимание, что для печати используется специальный класс ObjectDumper, который на основе механизма отражения (reflection) печатает объект, переданный ему. Кроме того, он напечата-

ет еще и запрос, который был сгенерирован к базе данных (смотри листинг 2).

При сортировке данных используем специальный оператор orderby с указанием сортировать по возрастанию или по убыванию. Попробуем отсортировать продукты по цене:

```
var q =
  from p in db.Products
  orderby p.UnitPrice descending
  select p;
```

Думаю, принцип создания простых запросов на выборку понятен. Надо научиться добавлять данные. Рассмотрим один из стандартных примеров по этой теме. Сначала надо вывести всех клиентов со свойством Region, равным «WA». После этого надо вставить нового заказчика, для чего просто создаем объект, используя инициализацию свойств. Добавляем нового заказчика и подтверждаем транзакцию, в результате чего новый заказчик благополучно оказывается в базе данных, о чем свидетельствует вывод на консоль (смотри листинг 3).

Существует очень интересный механизм, который позволяет ускорить работу приложения и сократить его код — хранимые процедуры, или просто «хранимки». «Хранимка» — это откомпилированный код программы на SQL, который можно вызвать по имени и передать параметры. За счет чего достигается скорость, думаю, понятно, — откомпилированный код всегда исполняется быстрее интерпретируемого. Использовать «хранимки» стало довольно просто — достаточно обратиться к соответствующему методу:

```
int count =
db.CustomersCountByRegion("WA");
Console.WriteLine(count);
```

Код очень короткий и быстрый. Чтобы понять механизм работы, посмотрим текст самой хранимой процедуры. Она принимает один строковый параметр, который считается регионом, затем делает запрос на выборку с подсчетом.

```
ALTER PROCEDURE dbo.[Customers Count By
Region]
  (@param1 NVARCHAR(15))
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @count int
  SELECT @count = COUNT(*)FROM Customers
  WHERE Customers.Region = @Param1
  RETURN @count
END
```

Те, кто работает с базами данных сейчас, наверняка используют классы DataSet. Хочу отметить, что весь код — под третью версию языка C#, так как LINQ отлично работает и с ними. Необходимо привести DataSet к нужному интерфейсу, а сам запрос практически не отличается от обычных.

```
var numbers =
testDS.Tables["Numbers"].ToQueryable();

var lowNums = from n in numbers
  where n.Field<int>("number") < 5
  select n;

Console.WriteLine("Numbers < 5:");
foreach (var x in lowNums) {
  Console.WriteLine(x[0]);
}
```

➜ **напоследок.** Напишем код, который объединяет в себе все выше описанное. В качестве формата хранения данных выберем XML. Загрузим файл bib.xml, который есть в стандартной поставке LINQ. Он хранит информацию о книгах, из которой мы выдернем нужную нам. Выбирать будем книги, которые выпустило издательство Addison-Wesley после 1995 года. Результат в виде XML-файла выведем на консоль:

```
XDocument bib = XDocument.Load("bib.xml");

var result = new XElement("bib",
  from b in
  bib.Element("bib").Elements("book")
  where(string)b.Element("publisher")
  =="Addison-Wesley" &&
(int)b.Attribute("year") > 1995
  select new XElement("book",
  new XAttribute("year",b.Attribute(
  "year"). Value),
b.Element("title")));

Console.WriteLine(result);
```

Таким методом можно извлекать данные из базы и упаковывать их в массив или наоборот.

➜ **завершаем.** Главное значения LINQ — это облегчить программистам работу с данными. Теперь контейнеры, XML и базы данных обрабатываются единообразно, и мы можем получать данные из хранилища одного типа (или сразу нескольких) и класть в хранилище другого типа. Надеюсь, благодаря этому программы, которые работают с данными (то есть все программы :)), станут короче, а код у них чище! C

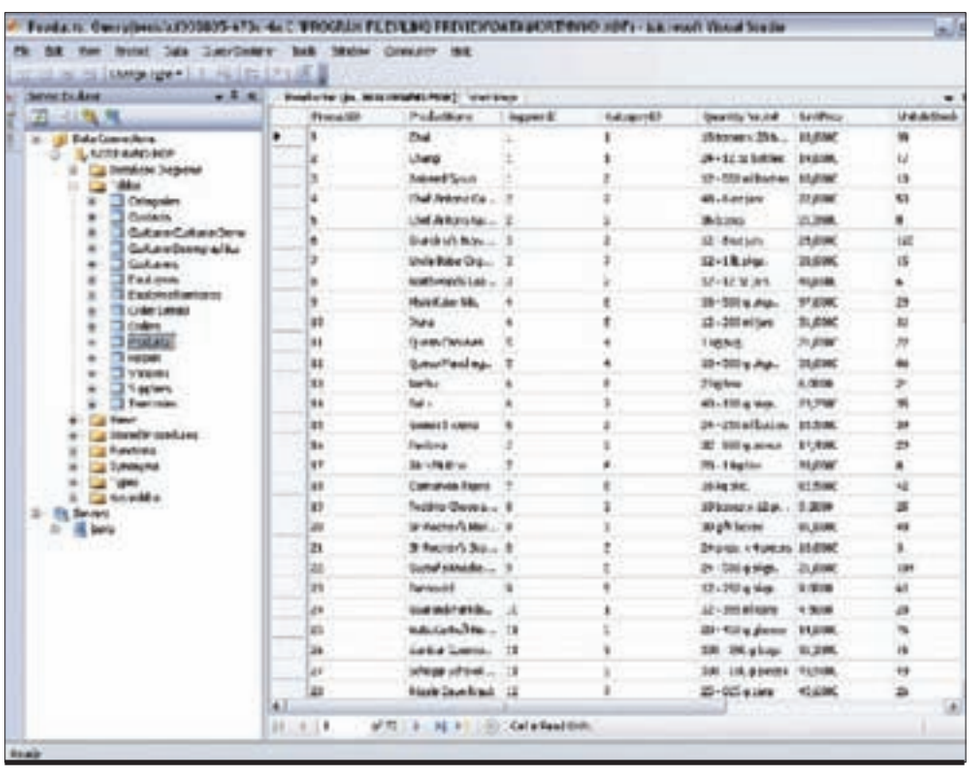
http://msdn.Microsoft.Com/data/ref/linq/
официальная страница linq на сайте Майкрософт. Здесь выкладывают официальные спецификации, примеры, компиляторы и прочие вкусности.

http://research.Microsoft.Com/comega/
официальная страница экспериментального языка Си-омега, у которого C# 3.0 очень много позаимствовал.

http://msdn.Microsoft.Com/vcsharp/future/linqsamples/
примеры, как использовать linq, очень наглядные и с нормальными объяснениями.

http://channel9.Msdn.Com/showpost.Aspx?Postid=114680
знаменитое выступление андерса хейлсберга о C# 3.0 в видео-формате.

http://en.wikipedia.org/wiki/C_Sharp
C sharp на вики.



Turbo Pascal — знаменитый «синий экран DOS»

Анонс

БЕЗОПАСНОСТЬ WINDOWS СЕГОДНЯ И ЗАВТРА

В СЛЕДУЮЩЕМ НОМЕРЕ МЫ РАЗБЕРЕМ АКТУАЛЬНЫЕ ВОПРОСЫ БЕЗОПАСНОСТИ WINDOWS:

— БЕЗОПАСНОСТЬ СЛУЖБ В .NET
— ПРОБЛЕМЫ .NET REMOTING
— BROWSERSHIELD
— ПОСЛЕДНИЕ БАГИ WINDOWS XP
— IPSEC ПОД WINDOWS
— АУТЕНТИФИКАЦИЯ И ШИФРОВАНИЕ В ЭЛЕКТРОННОЙ ПОЧТЕ
— PKI В XP
— ЗАЩИТА БЕСПРОВОДНЫХ И ПРОВОДНЫХ XP-КЛИЕНТОВ

ЭКСКЛЮЗИВ: КРИС КАСПЕРСКИ ТОЧИТ ВИСТУ — 20 СТРАНИЦ ПРО X-ИССЛЕДОВАНИЕ WINDOWS VISTA

шоу дельфинов

DELPHI 2006 — НОВАЯ РЕАЛЬНОСТЬ

ПОСЛЕ ВЫХОДА DELPHI 7 КОРПОРАЦИЮ BORLAND ОЖИДАЛИ СЕРЬЕЗНЫЕ ПРОБЛЕМЫ, ПОТОМУ ЧТО DELPHI 8 И 2005 ПРОЛЕТЕЛИ, КАК ФАНЕРА НАД ПАРИЖЕМ. ПЕРВУЮ ЖДАЛ ПРОВАЛ ИЗ-ЗА ТОГО, ЧТО .NET ЕЩЕ НЕ ПОЛУЧИЛА ДОСТАТОЧНУЮ ПОПУЛЯРНОСТЬ, А ВТОРАЯ ВЕРСИЯ НАКРЫЛАСЬ МЕДНЫМ ТАЗОМ ИЗ-ЗА ГЛЮЧНОСТИ. НО С ПОЯВЛЕНИЕМ DELPHI 2006 ВСЕ ВОЗВРАЩАЕТСЯ НА КРУГИ СВОЯ. ЭТО ШЕДЕВР, КОТОРЫЙ ПОСТЕПЕННО СТАНОВИТСЯ БЕСТСЕЛЛЕРОМ ДАЖЕ В США, ГДЕ ВЛАСТВУЮТ VISUAL C++ И JAVA

Фленов Михаил ака Horrific
http://www.vr-onlne.ru

Некоторые считают, что Delphi 2006 — всего лишь исправленная 2005, но это не так. Если посмотреть на размер Reviewer Guide, который занимает аж 70 страниц, то понимаешь, что перед нами совершенно новый продукт, и об этом говорит абсолютно все. Рассмотреть все новые возможности мы не сможем, но по основным нововведениям пробежимся.

➔ **варианты запуска.** Первое, что бросается в глаза после установки Delphi 2006 — это то, что можно запустить среду разработки в одном из трех вариантов:

[1] DELPHI DEVELOPER STUDIO — ВКЛЮЧАЕТ В СЕБЯ ВСЕ ВОЗМОЖНОСТИ, НО ЗАГРУЖАТЬСЯ БУДЕТ ОЧЕНЬ ДОЛГО.

[2] DELPHI FOR MICROSOFT WIN32 — ЗАГРУЖАЕТСЯ БЫСТРЕЕ, НО ПОЗВОЛЯЕТ СОЗДАВАТЬ ПРОЕКТЫ ТОЛЬКО ДЛЯ ПЛАТФОРМЫ WIN 32.

[3] DELPHI FOR THE MICROSOFT .NET — НЕТРУДНО ДОГАДАТЬСЯ, ЧТО ГРУЗИТЬСЯ ЭТОТ ВАРИАНТ БУДЕТ ДОСТАТОЧНО БЫСТРО, НО РАБОТАТЬ МОЖНО БУДЕТ ТОЛЬКО С ПРИЛОЖЕНИЯМИ ДЛЯ .NET.

Скорость загрузки Developer Studio — больная тема. Обширные возможности привели к тому, что полный вариант запускается очень долго, и возможность загрузить только необходимую версию просто неоценима.

➔ **первый запуск.** После запуска среды разработки в редакторе кода появляется закладка Welcome Page с загруженной HTML-страницей. На ней находятся ссылки на разделы документации, в которых можно ознакомиться с новыми возможностями и средой разработки Delphi, что может быть удобным для новичков.

Через месяц работы (а может быть и сразу) на этой страничке ты будешь использовать только кнопки New, Open Project, Open File и Help, которые располагаются на самом верху, и список последних открытых проектов (Recent Projects).

Список типов создаваемых проектов немного расширился и зависит от редакции. В наиболее полной редакции можно создавать проекты Delphi для Win32, Delphi для .NET, C# и даже C++ проекты.

➔ **визуальный дизайнер.** Не изменялся уже достаточно долгое время. А действительно, что там можно сделать такого нового, когда среда разработки является самой визуальной?! Оказывается, можно улучшить позиционирование. Что-то подобное есть уже в Visual Studio, но Borland пошла дальше и сделала все намного круче.

Когда мы двигаем компонент по форме или изменяем его размер, то в определенный момент на экране могут появиться тоненькие полоски, соединяющие два или более компонентов. Это указывает на то, что компоненты находятся на одной линии (горизонтальной или вертикальной). Если линия синего цвета, то компоненты имеют общую нижнюю или верхнюю границу. Если линия красного цвета, то на одной оси находятся центральные точки компонентов.

Таким образом, построение качественных интерфейсов становится более простой задачей, и я надеюсь, что теперь они будут более аккуратными. Почему-то программисты Delphi очень часто бросают компоненты на форму и даже не пытаются их выравнивать, чтобы форма выглядела хоть немного эстетичнее.

➔ **редактор кода.** Как же иногда достает писать эти «begin» и «end;» ;(. Благодаря Delphi 2006 вто-

рое больше писать не придется. Просто пишешь «begin», нажимаешь Enter, и «end» появляется автоматически. Первое время меня это даже раздражало, потому что по привычке начинаешь писать слово «end», которое среда разработки уже поставила, но со временем я привык и очень доволен.

Для переключения между визуальной формой и редактором кода используется клавиша <F12>. Редактор кода в Delphi также претерпел значительные изменения. Первое, что бросается в глаза — возможность сворачивать участки кода. Слева можно видеть полосу с такими же квадратиками, как и в дереве TreeView. Щелкая по этим крестикам, можно сворачивать и разворачивать код процедур или объявления методов.

Если ты уже отработал какую-то процедуру, оптимизировал и отладил ее работу, то можно закрыть код, чтобы он не занимал лишнее место на экране. Вместо тела процедуры в редакторе будет отображаться только ее имя и параметры.

После закрытия среды разработки все закладки теперь сохраняются, что очень удобно. Меня раздражала необходимость расставлять закладки при каждом открытии проекта в тех местах, над которыми я сейчас работаю. Напоминаю, что закладки устанавливаются клавишами <Ctrl+Shift+Цифра>.

Ах да, есть еще одна очень интересная новая фишка — напротив измененных строк теперь появляются желтые полоски. Таким образом, сразу видно, какие строки в модуле мы сегодня редактировали или добавляли. Если строка изменялась, но уже была сохранена, то полоса изменяет свой цвет на зеленый. Это ,конечно, мелочь, но очень приятная и удобная.

➔ **менеджер памяти.** Теперь поговорим о невидимом изменении, которое получат все твои программы, скомпилированные в Delphi 2006 — это новый менеджер памяти FastMM. Он повышает производительность Win32-приложений. Просто перекомпилируй проект в Delphi 2006 и все! Новый менеджер намного лучше и быстрее освобождает нужную память.

В приложениях .NET такой фишки нет, потому что тут менеджер памяти самой платформы .NET и Borland Delphi повлиять на него практически не может.

➔ **совместная работа.** Работать над большим проектом в одиночку очень сложно, а если проект очень большой, то просто невозможно. А как же работать над одним проектом большому количеству программистов? Просто разделить задачи или модули не получается, потому что задачи очень часто пересекаются, поэтому необходимо использовать специализированные системы. В Delphi 2006 уже встроена StarTeam — система совместной работы над проектом.

Конечно же, наличие StarTeam не означает, что ты обязан использовать именно его. Например, я привык использовать более простую Microsoft Visual Source Safe и без проблем делаю это. За это я и люблю Borland: она не заставляет использовать строго определенные технологии, а предоставляет возможность выбора.

➔ **рефакторинг.** В новой версии расширены возможности рефакторинга кода. Если кто-то уже работал с версией 2005, то может спросить — куда уже дальше расширять? Что может быть проще? Оказывается, есть куда расширять и есть что упрощать.

В этом же номере ты можешь более подробно узнать о рефакторинге, и все, что там говориться, уже реализовано в Delphi 2006. Только попробуй и ты ощутишь все преимущества новых возможностей.

➔ **моделирование.** Ну, наконец-то и в Delphi моделирование перестало быть просто информативным: теперь оно позволяет реально воздействовать на исходный код проекта. На мой взгляд, это наиболее мощное нововведение, которое реально упростит жизнь. Теперь визуально можно управлять методами и свойствами, что очень удобно, особенно, когда проект очень большой. Выбери меню View/Model view и ты увидишь панель управления моделями (диаграммами).

Теперь дважды щелкаем по имени модуля, диаграмму которого хочешь увидеть. В основном окне появится новая закладка, на которой можно увидеть свой класс и связанные с ним классы.

Теперь посмотрим, как можно управлять этой моделью. Допустим, что необходимо добавить новый член класса — (переменную) типа Integer. Щелкни правой кнопкой в разделе Fields и выбери пункт Add field или просто нажми <Ctrl+W>. На диаграмме в разделе Fields появится новая строка, в которой можно ввести имя свойства и его тип. Чтобы добавить метод, щелкаем правой кнопкой в разделе Methods и выбираем пункт меню Add Function или просто давим кнопки <Ctrl+M>.

Добавлять к классу визуально можно почти все. Щелкни правой кнопкой в шапке модели класса (где написано имя класса) и выбери раздел Add. Как видишь, здесь есть практически все, что только может понадобиться в повседневной жизни.

С помощью UML теперь можно создавать классы с нуля. Давай на простейшем примере ощутим всю мощь моделирования. На панели инструментов выбери кнопку «класс» и создай на диаграмме две модели класса. Одну назовем Class1, а другую TTest.

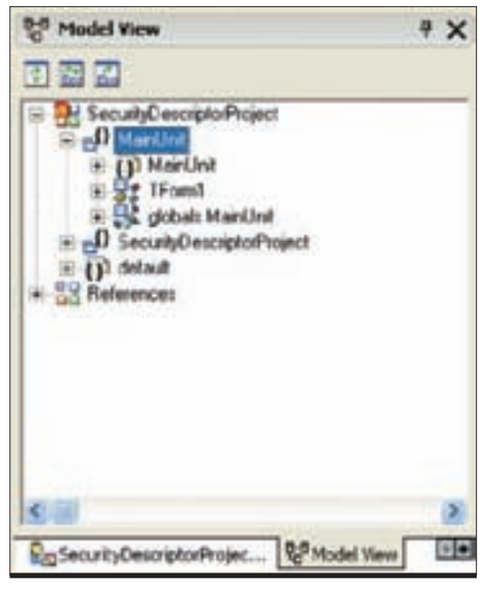
Если сейчас заглянуть в исходный код модуля, диаграмму которого мы мучаем, то можно будет увидеть заготовки этих классов. Но не это было нашей целью, смотрим дальше. Выбираем кнопку Association, щелкаем сначала на Class1, а потом на TTest. В результате в классе Class1 будет создана переменная Field1 типа TTest и наведена связь между классами.

Уже неплохо. Напоследок создадим одну процедуру в классе Class1 с именем TestProc и теперь посмотрим на исходный код:

```
TTest = class

end;

Class1 = class
 public
  procedure TestProc;
 var
 Field1 : TTest;
end;

procedure Class1.TestProc;
begin
end;
```

Мы просто визуально описали то, что нам будет нужно, а Delphi автоматически создал весь необходимый код, и нам не пришлось писать ни строчки! Все получилось очень быстро и красиво. Вот оно — будущее, которое наступает уже сейчас! Вот оно — UML-моделирование и все его преимущества! Мы можем визуально строить модель классов, наводить связи между классами, а потом только перескочить в исходный код модуля и увидеть все, что мы создали, в виде реального кода. Теперь действительно приходиться кодить только логику программы. Моделирование классов и визуального интерфейса происходит визуально.

Моделирование UML необходимо не только для визуального создания классов, методов и свойств, но и для документирования проектов, и даже контроля качества. Если у тебя есть большой проект, то открой его и посмотри на диаграмму. Если между классами слишком много связей, они беспорядочны или пересекаются, то классы спроектированы неверно. Тут следует задуматься о рефакторинге классов **C**



Панель управления моделями

дублер каскадера

АЛЬТЕРНАТИВА XML

ПРОБЛЕМА ОПИСАНИЯ ДАННЫХ СТОИТ ПЕРЕД СОЗДАТЕЛЕМ ПРАКТИЧЕСКИ ЛЮБОГО ПРИЛОЖЕНИЯ. ВНЕ ЗАВИСИМОСТИ ОТ ТОГО, КАКОВА ПРИРОДА ДАННЫХ. ВНЕ ЗАВИСИМОСТИ ОТ ТОГО, ДОЛЖНО ЛИ ПРИЛОЖЕНИЕ ИХ СОЗДАВАТЬ И ИЗМЕНЯТЬ ИЛИ ДАННЫЕ ИСПОЛЬЗУЮТСЯ ТОЛЬКО ДЛЯ ЧТЕНИЯ, ПЕРЕДАЮТСЯ ДАННЫЕ ПО СЕТИ ИЛИ СУЩЕСТВУЮТ НА ЖЕСТКОМ ДИСКЕ, ОНИ ДОЛЖНЫ ХРАНИТЬСЯ В ДОСТАТОЧНО УДОБНОМ ФОРМАТЕ

Александр Гладыш, ведущий программист компании «Step creative group»
www.stepgames.ru

Удобство формата определяется тем, насколько легко можно реализовать загрузку и сохранение данных, насколько большие накладные расходы по объему и скорости чтения влечет за собой его использование и насколько выбранный формат удобен для отладки. То есть насколько он читабелен, и насколько адекватные сообщения об ошибках в данных способен выдать загружающий их код.

Для удобства использования немаловажен объем имеющихся наработок по данному формату — наличие сторонних и собственных библиотек, средств перевода «живых» объектов логики приложения в сохраненные данные и обратно (сериализации, от английского serialization) и прочее. Такие средства могут предоставлять разную степень интеграции с логикой программы — от API для создания, сохранения, загрузки и модификации «голых» данных до инструментария тесной интеграции, позволяющего сериализовать объекты логики без написания промежуточного кода работы с форматом данных.

В отличие от данных, для хранения которых существуют устоявшиеся форматы (например, видео — AVI, WMV, музыка — MP3, OGG, документы — DOC, PDF), для данных, специфичных для логики конкретного приложения, в общем случае разработчик должен создать формат хранения самостоятельно (хотя некоторые средства разработки и языки программирования упрощают этот процесс вплоть до полной автоматизации). При этом зачастую одному приложению требуется несколько различных форматов хранения для разных наборов данных.

Как и всегда, разработка формата хранения данных с нуля чревата многими проблемами. Поскольку сериализация объектов логики обычно не предъявляет экстремальных требований к эффективности по скорости и объемам данных, лучше использовать одно из обобщенных решений — создать конкретный формат при помощи некоего готового «метаформата» и связанного с ним набора технологий.

➜ **существующие решения.** Самое распространенное на данный момент «обобщенное» решение — использование языка XML. Вокруг этого языка существует огромная развитая как в плане технологий, так и в плане инструментария инфраструктура. Широчайшая распространенность и связанные с этим бонусы — главное преимущество XML.

Однако у XML существует ряд недостатков. Основной из них — избыточность, увеличивающая объем, занимаемый данными, замедляющая процесс чтения и записи данных и усложняющая их ручное редактирование. Эта избыточность усугубляется тем, что XML — сугубо текстовый формат (впрочем, имеется несколько нестандартизованных вариантов Binary XML).

Помимо XML, в динамических языках (Python, Ruby и т.д.) получила распространение технология YAML (www.yaml.org). В языках программирования, ориентированных на веб-разработку, нередко используется JSON (JavaScript Ob-ject Notation, www.json.org). Обширный список альтернатив XML можно изучить по адресу www.pault.com/pault/pxml/xmlalternatives.html.

Мы же рассмотрим еще один подход к хранению данных, основанный на применении языка Lua.

➔ **начнем с примера.** В качестве примера будем использовать описание минимального пользовательского интерфейса главного меню игры. Мы рассматриваем случай, когда визуальное отображение интерфейса, вид и координаты его элементов жестко заданы имеющимися графическими ресурсами (для получения этих данных используется специальный инструментарий). Наши данные описывают лишь логическую структуру интерфейса.

Нужно отметить, что хотя пример основан на реальных данных, при его реализации не ставилась цель продемонстрировать создание полнофункциональной системы описания пользовательского интерфейса. Основная цель примера — иллюстрация описываемого подхода к разработке форматов хранения данных. Полный код примера вместе со всеми необходимыми для сборки материалами можно найти на диске к журналу.

Прототип пользовательского интерфейса, который мы хотим описать, схематически представлен на рисунке. Для каждого состояния всех элементов интерфейса (включая «задники» панелей) художник создает уникальное изображение.

Схему можно описать при помощи XML примерно таким образом:

```
<gui_layout name="gamegui">
  <panel name="mainmenu" modal="true"
hidden="false">
    <button name="newgame"/>
    <button name="settings"/>
    <button name="exit"/>
  </panel>
  <panel name="settings" modal="true"
hidden="true">
    <checkbox name="mute_all"/>
    <checkbox name="mute_sfx"/>
    <checkbox name="mute_speech"/>
    <checkbox name="mute_music"/>
    <button name="ok"/>
    <button name="cancel"/>
  </panel>
</gui_layout>
```

Нужно обратить внимание на то, что здесь отсутствует описание логики работы интерфейса — дана только его структура. В зависимости от реализации можно выбрать разные средства задания логики — вплоть до встраивания кода обработчиков событий на Lua непосредственно в данные на XML.

Преимущества читабельности налицо — объем текста меньше в два раза при том же объеме полезной информации. К тому же, приведенный выше текст — валидный код на Lua. Не просто описание данных — а именно код, набор вызовов функций.

➔ **как добиться, чтобы это работало.** Начнем с того, что функции в Lua — значения первого класса (first class values). Это, в частности, значит, что функции можно присваивать переменным, передавать функциям в качестве параметров и использовать в качестве возвращаемых значений функций.

От своего предка, языка описания данных Sol, Lua унаследовал некоторые особенности синтаксиса, позволяющие этому коду выполняться. Здесь используется то, что Lua позволяет опускать скобки при передаче функции в качестве единственного параметра строкового литерала или конструктора таблицы. Таким образом, с точки зрения языка, в коде, приведенном выше, вокруг каждой пары кавычек и каждой пары фигурных скобок стоят «неявные» круглые скобки.

Конструкция foo "string" { key = "value" } эквивалентна конструкции foo("string")({ key = "value" }). Сначала вызывается функция foo, которой передается параметр «string». Функция foo возвращает другую (анонимную) функцию, которая вызывается с параметром-таблицей { key = "value" }. Например, реализованная следующим образом foo в этом случае выведет на экран «string value»:

```
foo = function(str)
  return function(tab) print(str,
tab["key"]) end
end
```

Эти полезные особенности Lua предоставляют нам достаточно широкие возможности для реализации «самозагружающихся» описаний данных. Благодаря тому, что Lua позволяет компилировать программы в байт-код для виртуальной машины, мы можем «бесплатно» получить бинарное представление данных с еще меньшей избыточностью, к тому же снимающее затраты на парсинг и компиляцию текста (впрочем, в неэкстремальных случаях затраты на компиляцию итак достаточно невелики). Кроме того, текущая реализация языка Lua легко справляется с большими объемами данных — «пережевать» файл в несколько мегабайт для нее не проблема (конечно, при адекватной реализации пресловутой системы «самозагрузки»).

Если при выполнении кода (смотри листинг 1) в области действия будут находиться реализованные должным образом функции gui_layout, panel, button и checkbox, получим набор вызовов, показанный на рисунке. Переменные modal и hidden могут содержать значения любого типа, главное, чтобы они тоже были видимы.

Фактически, можно сказать, что мы видим здесь данные, записанные при помощи языка описания данных (data description language, DDL). Перечисленные имена переменных — ключевые слова (keywords) в создаваемом нами языке описания пользовательского интерфейса. Нужно заметить, что наши данные носят ярко выраженный иерархический характер.

Поскольку описание данных — валидный код на Lua, мы имеем возможность, например, «разбавить» декларативное описание интерфейса кодом описания его функциональности — скажем, описать обработчики событий по клику на кнопку:

```
panel "mainmenu"
{
modal;
  button "newgame" { action =
StartNewGame; };
  button "settings" { action = function()
showpanel("settings") end; };
  button "exit" { action =
confirm(ExitGame); };
};
```

Конечно, если код обработчика пишется вручную, такой подход следует применять с осторожностью, всячески ограничивая API, доступный таким функциям, и проводя их максимальную проверку на корректность и безопасность. Помимо этого, наличие функций непосредственно в описании данных несколько затрудняет вариант реализации DDL с промежуточной кодогенерацией — такие обработчики придется как-то сохранять вместе со всеми данными. Например, их можно сохранять в виде байт-кода при помощи системной функции string.dump — накладываемое ограничение на отсутствие у таких функций ссылок на внешние переменные (upvalues) в данном случае не слишком существенно.

Схема (смотри рисунок) может быть представлена следующим образом:

```
gui_layout “gamegui”
{
      panel “mainmenu”
{
      modal;
            button ”newgame”;
            button ”settings”;
            button “exit”;
      };
      panel “settings”
{
modal;
hidden;
            checkbox ”mute_all”;
            checkbox ”mute_sfx”;
            checkbox ”mute_speech”;
            checkbox ”mute_music”;
            button ”ok”;
            button ”cancel”;
      };
};
```

Все параметризованные функции — ключевые слова, кроме корневого, можно реализовать по следующей схеме:

```
-- Вместо keyword_name нужно подставить имя ключевого слова
keyword_name = name_data(“keyword_name”) (function(name, data)
      process_child_keywords(data)
      data.type_ = keyword_name
      data.name_ = name
      return function(dest)
            if not dest.children_ then dest.children_ = { } end
            table.insert(dest.children_, data)
      end
end)
```

На выходе для данных из листинга 1 будет сгенерированна примерно такая таблица (для экономии места описание панели «settings» пропущено):

```
{
      type_ = “gui_layout”;
      name_ = “gamegui”;
      children_ = {
            [1] = {
                type_ = “panel”;
                name_ = “mainmenu”;
                modal = true;
                visible = true;
            children_ = {
                [1] = { type_ = “button”; name_ = ”newgame”; };
                [2] = { type_ = “button”; name_ = ”settings”; };
                [3] = { type_ = “button”; name_ = ”exit”; };
                };
            };
            [2] = { type_ = “panel”; name_ = “settings”; … };
      };
};
```

(1) Как же получить загруженные данные? Можно реализовать функции-ключевые слова нашего DDL — таким образом, чтобы они изменяли некую глобальную сущность. Однако это повышает связанность кода и уменьшает его гибкость.

Язык Lua позволяет явным образом возвращать значение (или даже несколько значений) при загрузке файла. Для этого достаточно в его конце написать выражение, использующее ключевое слово return так же, как если бы это была обычная функция. Удобно реализовывать корневую функцию иерархии (в нашем случае gui_layout) так, чтобы она возвращала собранные в процессе выполнения остального кода данные (либо сгенерированную функцию, выполняющую некие действия на основе этих данных).

Тогда, если дописать в начало листинга 1 ключевое слово return, можно будет добраться до загруженных данных, используя примерно такую конструкцию:

```
local data =
assert(loadfile(“myguilayout.lua”))()
```

(2) В переменную data будет помещено значение, возвращенное корневой функцией gui_layout. Для большей изящности ключевое слово return можно «подклеивать» автоматически перед загрузкой данных. В этом случае вместо load-file удобнее использовать loadstring.

Очевидно, помимо реализации необходимого набора ключевых слов, мы должны ограничить пользователя в написании кода на Lua в файлах с данными только этим набором. Без дополнительного разбора текста описания данных невозможно запретить использование в нем произвольного кода, но это и не обязательно. Главное — не дать такому коду доступ к окружающему миру, как говорится, выполнять его в «песочнице» (sandbox). Тогда «лишний» код может быть как минимум бесполезен (но не вреден), но как максимум его можно использовать во благо — например, для генерации данных на лету. Строго говоря, еще нужно учитывать возможность подвесить загрузку данных, написав в файле с данными бесконечный цикл. Если требуется стопроцентная гарантия от вредоносного кода, необходимо контролировать процесс загрузки, например, при помощи установки хуков на исполнение кода в функции debug.sethook, либо при помощи установки «вотчдога» в отдельном потоке.

(3)

К счастью, язык Lua предоставляет эффективное средство для создания такой «песочницы» — возможность явного задания таблицы глобального окружения (environment table) для функций при помощи стандартной функции setfenv. При загрузке данных в глобальное окружение необходимо поместить только функции-ключевые слова нашего DDL.

Функция loadfile возвращает функцию, содержащую весь код загружаемого файла. Глобальное окружение используется по умолчанию для всего

этого кода. Заменив это окружение до вызова функции, можно заменить окружение, которое будет назначено коду в файле. Можно написать следующую функцию для загрузки файлов с данными:

```
load_ddl_file = function(name,
ddl_keywords)
 local loader = assert(loadfile(name))
 setfenv(loader, ddl_keywords)
 return loader()
end
```

Использовать эту функцию можно аналогично loadfile:

```
local data =
assert(load_ddl_file("myguilayout.lua",
ddl_keywords))
```

Подразумевается, что реализация ключевых слов языка находится в таблице ddl_keywords. Для большего контроля можно запретить запись в эту таблицу и чтение из нее данных по отсутствующим ключам, переопределив в ее метатаблице методы __index и __newindex.

Учитывая все вышесказанное, можно перейти к обсуждению путей реализации самих функций — ключевых слов для нашего примера.

➔ **реализация.** Большинство ключевых слов требует двух вызовов для получения всех данных. В первом вызове обычно передается имя, во втором — таблица с данными. Чтобы облегчить реализацию, введем вспомогательную функцию — генератор «сборщиков данных». Функция-сборщик собирает переданные ей данные, одновременно проверяя соответствие их типов заданным. После того, как нужное количество аргументов собрано, они передаются пользовательской функции (sink):

```
pipeline = function(name, ...)
assert(type(name) == "string")
local types = {...}
local nargs = #types
return function(sink)
local args = { }
local n = nargs
local collector
collector = function(a)
local i = 1 + nargs - n
local argt = types[i]
if argt ~= "any" and type(a) ~= argt then
error(string.format(
"%s: Bad argument %d, %s expected"
name, i, types[i]
))
end
args[i] = a
if n > 1 then
n = n - 1
return collector
else
n = nargs
return sink(unpack(args))
end
end
return collector
end
end
```

Далее зададим функции для генерации «сборщиков» аргументов заданных фиксированных типов (произвольный тип аргумента можно указать, используя в качестве имени типа слово «any»). Например, для ключевых слов, требующих аргументы по схеме «имя-данные» подойдет такая функция:

```
name_data = function(name) return
pipeline(name, "string", "table") end
```

Наиболее прямолинейный способ реализации ключевых слов — просто собирать все данные в одну большую иерархическую таблицу, занося «имена» и «типы» элементов интерфейса в служебные поля таблиц данных этих элементов. Фактически, это прямое, без каких-либо дополнительных действий, преобразование «исполнимого кода», описывающего данные в сами данные.

Ключевые слова без аргументов в этом и в остальных рассматриваемых случаях — функции, задающие значение определенного ключа в таблице данных. Такие функции играют роль «синтаксического сахара», повышающего наглядность. Например (des — таблица данных для изменения):

```
hidden = function(dest) dest.visible =
false end
```

Все ключевые слова без аргументов попадают в порядке упоминания в списочную часть таблицы данных ключевого слова-родителя. Значения, возвращенные «дочерними» ключевыми словами, также попадают в списочную часть. Для единообразия эти значения тоже можно сделать функциями, которые при вызове будут добавлять данные об элементе-потомке в специальный список, хранящийся в таблице данных элемента-родителя.

Перед анализом данных родительского элемента следует сделать обход списочной части для сбора данных о потомках:

```
process_child_keywords = function(data)
for key, child in pairs(arg_table(data))
do
if type(child) == "function" then
                data[key] = child(data)
end
end
end
```

В реализации унарных ключевых слов очевидно, что нужно опустить упоминание name_data, поскольку единственный аргумент передается за один вызов функции.

У корневого ключевого слова (gui_layout) не может быть предков, поэтому оно должно возвращать готовую таблицу с данными. В остальном оно реализуется по той же схеме:

```
gui_layout= name_data("gui_layout")
(function(name, data)
 process_child_keywords(data)
 data.type_ = gui_layout
 data.name_ = name
 return data
end)
```

Приведенная выше реализация функции pipeline не позволяет эффективно обнаружить один из видов ошибок в данных (когда пользователь указывает недостаточное число аргументов для ключевого слова). Например, если для панели указано только имя:

```
panel "bad_panel";
```

Как видно из его реализации, сборщик аргументов collector возвращает самого себя до тех пор, пока не наберет нужного числа аргументов. Поэтому в такой ситуации в списочной части таблицы данных ключевого слова-предка оказывается не результат работы функции — реализации ключевого слова-потомка sink, — а сам сборщик.

Такого типа ошибку можно обнаружить, создав сборщика из функции таблицей с заданным метаметодом __call (реализацией этого метаметода будет нынешнее тело функции-сборщика) и заданным (мета)полем, обозначающим, что это — сборщик.

В process_child_keywords при обнаружении таблицы вместо функции необходимо проверять значение этого поля. Если обнаружен сборщик, нужно выдавать ошибку о недостаточном количестве аргументов. Дополнительную отладочную информацию о природе сборщика (его имя, типы, число аргументов) можно хранить в его же (мета)таблице.

Тем же способом можно реализовать задание значений аргументов по умолчанию — вместо выдачи ошибки нужно просто вызывать сборщик с этим значением (или значениями) до тех пор, пока он не наберет достаточное количество аргументов.

Если нужна поддержка значений параметров по умолчанию, их установку целесообразно вставить после обработки ключевых слов-потомков. Например, панели должны быть видимы по умолчанию, значит, для ключевого слова panel после вызова process_child_keywords нужно добавить строчку:

```
if data.visible ==
nil then data.visible = true end
```

➔ **«самозагружающиеся» данные.** Уже в описанном виде с данными можно эффективно работать. Однако, это не всегда удобно — если сохраненные данные соответствовали каким-то «живым» объектам логики приложения, придется писать отдельный код по созданию и настройке этих объектов на основе получившихся таблиц. Было бы удобнее создавать объекты сразу же. Это вполне возможно. Один из путей — встроить код создания объектов логики в функции-модификаторы, возвращаемые дочерними ключами, и изменить логику обхода потомков. Если потомку нужна информация о предке, нужно делать обход потомков не сразу при вызове функции-ключевого слова, как сделано выше, — а начиная с корневого элемента. При этом нужно дополнительно передавать модификаторам данные о предке.

Пример случая, когда для создания нашего элемента пользовательского интерфейса нужна информация о родительском окне:

```
panel = name_data("panel")
(function(name, data)
return function(dest, parent_wnd)
local wnd = Window.Create(name,
parent_wnd)
process_child_keywords(data, wnd)
end
end)
```

Подразумевается, что функция process_child_keywords была модифицирована, чтобы передавать вместе с данными предка еще один параметр — его окно. Нужно модифицировать и реализацию ключевых слов без параметров. При таком подходе они должны уметь изменять свойства самого окна — изменять данные уже поздно. Например, для hidden:

```
hidden = function(data, wnd)
wnd:SetVisible(false) end
```

Если параметр, изменяемый ключевым словом без аргумента, должен быть доступен на этапе до того, как возможна обработка всех потомков (например, если в приведенном примере нужно передать модальность окна в Window.Create), необходимо реализовать тем или иным способом обход потомков в два прохода. Самый простой способ — вложить функцию-модификатор параметризованного ключевого слова в еще одну такую же и вызывать process_child_keywords дважды (такую реализацию можно абстрагировать подобно name_data):

```
panel = name_data("panel")
(function(name, data)
return function(dest)
return function(dest2, parent_wnd)
process_child_keywords(data) —
Data modifiers
local wnd = Window.Create(name,
parent_wnd, data.modal)
process_child_keywords(data, wnd)
-- Child windows
end
end
end)
```

➔ **кодогенерация.** Видно, что с увеличением сложности структуры данных усложняется и реализация ключевых слов, и сопутствующего им кода. Растут накладные расходы по производительности загрузки данных. В последнем примере требуется, как минимум, пять вложенных вызовов функций, прежде чем дело доходит до непосредственного создания объекта. В действительности это не очень страшно — виртуальная машина Lua обладает очень высокой производительностью и, к тому же, поддерживает «хвостовые» вызовы функций (tail function calls) (в частности, хвостовую рекурсию tail recursion), что позволяет эффективно оптимизировать вызовы вида return functioncall и снимает ограничение на количество вложенных вызовов функций такого рода.

Тем не менее, может возникнуть необходимость оптимизации процесса загрузки данных и создания из них «живых» объектов логики приложения. Наиболее эффективный способ сделать это — убрать всякую загрузку данных и оставить только код создания и настройки объектов с зашитыми намертво параметрами, узкоспециализированный под конкретный рассматриваемый случай и набор данных. Уничтожить всю архитектурную прослойку в коде и оставить только нужную в данный момент функциональность.

Существует способ добиться такого эффекта автоматически и относительно бесплатно на основе имеющихся данных в описываемом формате. Впрочем, такой способ эффективен только в том случае, когда данные статичны относительно основного приложения, то есть число загрузок данных значительно превышает число их изменений. Рассматриваемый в статье пример с пользовательским интерфейсом игры — как раз такой случай. При его создании требуется гибкость и легкость внесения изменений, но эти свойства совершенно не нужны в готовой игре (однако, с точки зрения производительности загрузки данных абсолютно никаких показаний к проведению такой оптимизации нет).

Убрать (или существенно снизить) накладные расходы на гибкость позволяет использование входных данных не для непосредственного создания объектов логики, но для генерации узкоспециализированного кода, создающего эти объекты. При этом одна из возможных стратегий состоит в том, что на этапе активного изменения данных этот код генерируется, компилируется и исполняется налету (благо компиляция в Lua — весьма быстрый процесс), а после стабилизации в дистрибутиве приложения остается и используется только сгенерированный вариант.

Простейший подход к такой кодогенерации — непосредственная замена кода создания объектов в реализации ключевых слов на формирование текста этого кода. При этом все условные переходы переносятся из выполняемого кода в генерирующий, а также (по мере сил) производятся другие оптимизации на основе имеющейся на момент генерации кода информации об объектах и обо всей системе в целом. Если присутствуют какие-либо «декоративные» архитектурные прослойки между генерируемым кодом и конечным системным кодом, их тоже можно попытаться ликвидировать.

Вероятно, наиболее эффективный с точки зрения возможностей оптимизации подход — построение и многопроходный анализ дерева данных, аналогичного дереву, показанному в листинге 3.

LUA

Расширяемый скриптовый язык расширений (extensible ex-tension language) с динамической типизацией. Код в статье написан на наиболее свежей версии языка — 5.1, но должен работать и на предыдущей 5.0. Руководство по языку и первое издание книги одного из авторов языка — Роберто Иерусалимского (Roberto Ierusalimschy) «Programming in Lua» — можно найти в электронном виде на официальном сайте www.lua.org.

Выгода от применения описываемого в статье подхода удваивается, когда в приложении есть потребность во встроенном скриптовом языке — подключение Lua к коду на C, C++ (а также любом языке, имеющем интерфейс взаимодействия с C — существует, например, интерфейс Lua-Python) достаточно легко осуществимо. При этом и для хранения данных, и для задания конечной логики приложения используется один и тот же язык, что упрощает процесс освоения системы.

Наиболее ярко это проявляется в компьютерных играх. Язык Lua вообще получил наибольшее распространение именно в этой области во многом благодаря своей скорости, мощности, гибкости и легкости подключения к коду приложения. Однако, помимо компьютерных игр, Lua широко используется и в других областях — от средства для профессиональной работы с фотографиями (до 40% кода Adobe Lightroom написано на Lua, labs.adobe.com/technologies/lightroom/) до системы обработки данных о геноме человека, использующей Lua для хранения огромных объемов данных (www.cbrc.jp/~ueno/slides/lua05u3.pdf). Постоянно растущий список проектов, использующих Lua, можно найти на официальном сайте языка — www.lua.org/uses.html.

Самый простой способ генерации текста (а значит и кода) в Lua существует благодаря развитой встроенной поддержке собственного языка регулярных выражений. Вот реализация функции smart_str, позволяющей заполнять строки-шаблоны конкретными значениями (в варианте, не требующем написания скобок):

```
smart_str = function(str)
return function(context)
return string.gsub(str, "$%((.-)%)",
function(capture)
local c = context[capture]
if c == nil then error
("Context '" .. capture .. "' not found")
end
return tostring(c)
end
)
end end
```

Здесь стандартная функция string.gsub используется для замены всех вхождений $(имя_переменной) на значение этой переменной в переданной вторым параметром таблице-контексте. Пример использования, печатающий текущую дату:

```
local str = smart_str "Today is $(date)"
{ date = os.date() }
print(str)
```

Подробно возможности встроенного языка регулярных выражений описаны в руководстве по языку Lua.

Иногда бывает необходим больший, чем может дать приведенная реализация smart_str, контроль над кодогенерацией. Например, нужно использовать циклы или условные переходы. Можно пойти по пути усложнения синтаксиса, поддерживаемого этой функцией. Однако есть другой способ, родственный «наивному» подходу генерации строк, когда код просто «подклеивается» к строке с результатом. Так как строки в Lua неизменяемы, код, подобный следующему, весьма неэффективен (он засоряет память промежуточными значениями result):

```
local result = 'return {'
for i = 1, 1000 do
  result = result .. math.random() .. ';'
end
result = result .. '}'
```

Значительно более эффективно во время генерации сохранять строки в таблице, а после генерации склеить их при помощи стандартной функции table.concat. После некоторой доработки получаем второй способ кодогенерации:

```
local strings = { }
local i = 1
local _ = function(val) strings[i] =
tostring(val); i = i + 1 end

_ 'return {'
for i = 1, 1000 do
  _ (math.random()) _ ';'
end
_ '}'

local result = table.concat(strings)
```

В не слишком запущенных случаях этот способ позволяет писать достаточно наглядный код, одинаково хорошо показывающий как шаблон генерируемого кода, так и логику генерации. Но в тяжелых случаях со сложной логикой генерации помочь может только реализация «объектной модели» кода Lua (code document ob-ject model).

Рассмотрим, как можно организовать генерацию кода для нашего случая с пользовательским интерфейсом. Пойдем по простейшему пути. Запускаем обход от самого нижнего элемента подобно листингу 2. Каждый элемент возвращает строку, которую нужно «подклеить» к общему коду. Функцию process_child_keywords заменяем на функцию concat_child_keywords, конкатенирующую (сцепляющую) строки, полученные от всех дочерних ключевых слов.

```
concat_child_keywords = function(data)
  local strings = { }; local i = 1
  for _, child in ipairs(data) do
    if type(child) == "function" then
      strings[i] = child(data); i = i + 1
    end
  end
  return table.concat(strings)
end

hidden = function(dest) dest.visible =
false; return "" end

panel = name_data("panel")
(function(name, data)
local create_children =
concat_child_keywords(data)
if data.visible == nil then data.visible =
true end
if data.modal == nil then data.modal =
false end
return smart_str (
'do local wnd=Window.Create($(name)
,parent_wnd,$(modal));' ..
'wnd:SetVisible($(visible))' ..
'push_parent_wnd(parent_wnd);' ..
'parent_wnd=wnd;' ..
'$(create_children);' ..
'parent_wnd=pop_parent_wnd();end;\n'
) {
name = '"' .. name .. '"';
modal = data.modal;
visible = data.visible;
create_children = create_children;
}
end
end)
```

Иногда требуется умение загружать и снова сохранять данные, не имея описания их конкретного формата. Как и для XML, для нашего случая такое тоже возможно. Данные достаточно легко переводятся в вид, показанный в листинге 3, если обобщить код листинга 2, используя метаметод __call таблицы глобального окружения для получения имени конкретного ключевого слова.

Сохранение загруженных таким образом данных в тот же формат реализуется как достаточно тривиальная процедура кодогенерации с рекурсивным обходом всего дерева элементов. Может быть удобным создание специальной объектной модели для такого дерева. Это должно облегчить написание кода, который будет модифицировать дерево.

➔ **заключение.** Предлагаемый подход — не панацея и не полная замена технологий, основанных на XML. Его главный и определяющий недостаток — малая, относительно XML, распространенность применяемых технологий, что влечет за собой недостаток информации по методикам работы, недостаток проработанных библиотек для использования Lua в таком ключе и прочее.

Но если в твоем проекте еще не реализована полноценная поддержка технологий XML, применение Lua для хранения данных вполне способно дать определенную выгоду. Потенциальная мощность описанного подхода сравнима с XML.

Преимущества такого подхода — наличие «бесплатного» бинарного представления, меньшая избыточность, возможность реализации «самоподнимающихся» данных и так далее. Все это делает его достойным рассмотрения, особенно если ты уже используешь Lua в своем проекте, либо у тебя есть необходимость во встраивании в проект произвольного скриптового языка.

С другой стороны, малый, по сравнению с полнофункциональными библиотеками работы с XML, объем и высокая скорость работы виртуальной машины языка Lua может оправдать ее встраивание исключительно с целью использования Lua для хранения данных C

www.yaml.org
технология yaml.

www.json.org
json (javascript ob-ject notation).

www.pault.com/pault/pxml/xmlalternatives.html
список альтернатив xml.

www.lua.org
официальный сайт lua.

www.lua.org/uses.html
список проектов, использующих lua.

http://en.wikipedia.org/wiki/XML
Extensible Markup Language From Wikipedia, the free encyclopedia.

мобильные представления

SYMBIAN TIPS'N'TRICKS

РЕШИЛ ЗАНЯТЬСЯ ПРОГРАММИРОВАНИЕМ ПОД СМАРТФОНЫ НА SYMBIAN? ВЕРНО ЧУЕШЬ, КУДА ВЕТЕР ДУЕТ, ПРИЯТЕЛЬ: КОДИНГ ПОД МОБИЛЬНЫЕ УСТРОЙСТВА — ОЧЕНЬ ПЕРСПЕКТИВНОЕ НАПРАВЛЕНИЕ. СПЕЦОВ В ЭТОЙ ОБЛАСТИ МАЛО, А СПРОС НА НИХ ПОВЫШАЕТСЯ С КАЖДЫМ ДНЕМ. КРОМЕ ТОГО, ПРОЦЕСС СОЗДАНИЯ ПРОГРАММ ДЛЯ МОБИЛ — КРАЙНЕ УВЛЕКАТЕЛЬНЫЙ, НО НАЧИНАЮЩИХ ЖДЕТ МАССА ПОДВОДНЫХ КАМНЕЙ И ПРЕПЯТСТВИЙ. О РЕШЕНИИ НЕКОТОРЫХ ТИПОВЫХ ПРОБЛЕМ И ЗАДАЧ Я И ХОЧУ РАССКАЗАТЬ ТЕБЕ В ЭТОЙ СТАТЬЕ

Дмитрий Тарасов aka dem@n
pink2000-0@mail.ru

➜ **выбор среды разработки и SDK.** На данный момент наиболее популярные среды разработки под Symbian — это CodeWarrior от Metrowerks и C++ BuilderX Mobile Edition от Borland. Ты можешь использовать их, но я бы посоветовал обратить внимание на надстройку над Visual Studio.NET под названием Carbide.VS. Плюс ее в том, что она бесплатна (сама надстройка, естественно) и лишь требует халявной регистрации по истечении 14 дней использования. Кроме того, в процессе работы с этой средой ты избавляешься от разного рода глюков вроде неправильно прописанных путей в package-файле. И потом, ты наверняка используешь студию для той или иной работы, поэтому работа в привычной среде — тоже явный плюс. Правда, стоит иметь ввиду, что среда, по непонятным причинам, в процессе сборки проекта иногда исключает необходимые библиотеки из файла описания проекта, поэтому рекомендую в настройках Carbide снять галочки с опций «update .mmp file» и «update .pkg file».

Что касается выбора SDK, то инфы по этому поводу предостаточно в Сети. Единственное, на что я хочу обратить внимание, так это на то, что большая часть софта для Series 60 всех версий до S60 3rd Edition не будет работать на новых мобилах (например, новая N-серия от Nokia), работающих на S60 3rd Edition, поэтому понадобится отдельный SDK для последней версии.

➜ **особенности процесса сборки проекта.** Есть две конфигурации сборки проекта — для запуска и отладки в эмуляторе и для создания установочного (SIS) файла. Когда мы разрабатываем и тестируем программу, надо периодически собирать проект для обеих конфигураций, поскольку это позволяет сразу обнаружить дурацкие ошибки вроде непрописанных библиотек или модулей в файле конфигурации проекта. В случае с Visual Studio, например, бывает, что после добавления модуля к проекту стандартными средствами («add item») проект в конфигурации для эмулятора нормально компилируется, а в конфигурации для создания установочного файла возвращает ошибку. Связано это с тем, что при добавлении нового файла в проект его нужно явно прописывать в файле конфигурации проекта. Например, если ты реализовал какой-либо функционал в файле megasource.cpp, не забудь добавить в файл описания проекта (mmp) следующую строку:

```
SOURCE    megasource.cpp
```

➜ **почему программа закрывается при запуске.** Так получилось, что в Symbian C++ реализован собственный механизм обработки исключений, который адаптирован под выполнение кода на мобильной платформе. Не буду грузить тебя деталями, ибо для этого есть SDK Help, а скажу лишь, что в большинстве случаев ты будешь создавать объекты, помещая их в так называемый CleanupStack. То есть создание объектов обычно выглядит примерно так:

```
CXaObject *xaObj = new CXaObject;
CleanupStack::PushL(xaObj);
```

Часто в коде одной функции создается несколько объектов, и после того, как они уже были использованы и больше не нужны, необходимо освободить ресурсы и удалить объекты из CleanupStack, что делается следующим образом:

```
CleanupStack::PopAndDestroy(xaObj);
```

Если после выполнения функции созданные локальные объекты не будут удалены, наша программа аварийно закроется при выполнении. Работа с освобождением ресурсов — очень важный момент при кодинге под Symbian, поэтому рекомендую тщательно изучить соответствующую документацию.

➜ **что есть что в Application Framework.** У начинающих кодеров под Symbian при созерцании взятого из SDK простейшего примера приложения типа HelloWorld возникает вопрос: «А зачем тут столько файлов и классов?». Действительно, каркас приложения с графическим интерфейсом (консольные приложения под series 60 также можно создавать, но они не представляют никакого интереса) состоит как минимум из четырех классов.

Класс application служит для запуска приложения и для создания так называемого документа приложения.

Класс document представляет собой движок приложения и создает объект класса интерфейса пользователя.

Класс AppUi служит для обработки действий пользователя и команд и представляет наибольший интерес, поскольку именно в этом классе определяют такие любопытные виртуальные функции базового класса «base» как:

— HADLEKEYEVENT(), ВЫЗЫВАЕМУЮ ПРИ НАЖАТИИ НА КЛАВИШУ СМАРТФОНА;

— HANDLESWITCHONEVENTL(), ОБРАБАТЫВАЮЩУЮ СОБЫТИЯ ПРИ СТАРТЕ СМАРТФОНА;

— HANDLECOMMAND(), ВЫЗЫВАЕМУЮ ДЛЯ ОБРАБОТКИ КОМАНД ПОЛЬЗОВАТЕЛЯ.

Класс Container требуется для отображения данных на экране смартфона. При разработке своего собственного приложения целесообразно использовать данный шаблон, который идет с любым SDK, и наращивать функциональность указанных классов.

➜ **как подружить ПО с родной речью.** Вполне возможно, что ты захочешь использовать в своем приложении надписи и диалоги на русском языке

(хотя ориентироваться на отечественного потребителя в большинстве случаев бессмысленно). Все надписи в пунктах меню, диалогах и строки хранятся в файле ресурсов. Для того чтобы вместо абракадабры на экране мобилы отображалась нормальная кириллица, достаточно сохранить этот файл в кодировке utf-8. Разумеется, для этого подойдет наш любимый Блокнот :).

➔ **как заставить программу автоматически запускаться при старте телефона.** Как ты, наверное, знаешь, в Symbian нет такого понятия как реестр. Автозагрузки, соответственно, тоже нет. Поэтому для автозапуска приложения обычно приходится извращаться, используя рекогнайзеры — программы, либо части кода проекта, которые служат для связывания определенных типов приложений (текстовые файлы, графика, и т.д.) с определенными программами, служащими для их открытия. Следовательно, если написать рекогнайзер, ставящий файл нашей программы в соответствие приложению, создающему процесс при загрузке телефона (apprun.exe, к примеру), то можно заставить этот процесс запускать нашу прогу :). Кодинг рекогнайзеров — тема довольно обширная, поэтому я не буду углубляться сейчас в детали. Я предлагаю воспользоваться уже готовой разработкой от парней с http://newlc.com. Продукт называется ezboot и представляет собой sis-файл, который легко интегрируется с нашим проектом. Прога эта бесплатна для некоммерческого использования, что очень приятно. На том же сайте есть и исходник для особо интересующихся. Для того чтобы заставить запускаться прогу при старте телефона, потребуется:

[1] Пойти на http://newlc.com/ezboot.html, скачать вариант ezboot для target-платформы и поместить его в директорию с sis-файлом нашей проги.

[2] Интегрировать ezboot.sis с установочным файлом проекта. Для этого добавим в конец pkg-файла код:

```
@"ezboot.sis",(0x101fd000)
```

[3] создать в каталоге с pkg-файлом текстовый файл appname.boot, содержащий строку:

```
boot:\system\apps\ appname\ appname.app
```

[4] прописать в pkg-файле:

```
" appname.boot" –"!:\system\programs\
ezboot\boot\appname.boot"
```

Таким образом ты дашь линкеру знать, что при установке файл AppName.boot должен быть скопирован в указанную директорию.

Все готово, теперь мы сможем наблюдать, как наша прога стартует через небольшое время после запуска смартфона.

➔ **как убрать программу из меню.** Зачастую нет необходимости (меньше знаешь — крепче спишь, как известно), чтобы программа была видна в главном меню. В этом случае достаточно модифицировать файл информации о приложении, имеющий имя вида YoutAppaif.rss, следующим образом:

```
#include <aiftool.rh>
RESOURCE AIF_DATA
  {
  app_uid=0x0871aba4;
  ...
  hidden = KAppIsHidden; // прячем иконку
из меню
  }
```

После установки такая прога не будет появляться в меню, но при запуске будет видна в task-manager (аналог менеджера запущенных приложений в Windows).

➔ **как убрать программу из Task-manager.** Естественно, программисты, пишущие злые трояны для мобил, не хотят, чтобы они были видны в менеджере приложений. Чтобы прога никогда там не появилась, нужно переопределить виртуальную функцию UpdateTaskNameL() в классе документа приложения. Выглядеть это должно вот таким образом:

```
void
CXaSMSDocument::UpdateTaskNameL(CApaWind
owGroupName* aWgName)
{
CAknDocument::UpdateTaskNameL(aWgName);
//вызываем стандартную функцию
UpdateTaskNameL
aWgName->SetHidden(ETrue);
aWgName->SetSystem(ETrue);
// прячем иконку
}
```

➔ **как не дать приложению получить фокус.** Ну а если хочется, чтобы даже при прямом запуске выполняемого файла через файловый менеджер пользователь не видел нашу прогу, понадобится модифицировать метод ConstructL класса AppUi следующим образом:

```
void CXaSMSAppUi::ConstructL()
  {
  BaseConstructL();//базовый метод
CEikonEnv::Static()-
>RootWin().EnableReceiptOfFocus(EFalse);
CEikonEnv::Static()-
>RootWin().SetOrdinalPosition(-1000,
ECoeWinPriorityNeverAtFront);
// заставляем приложение терять фокус
  ...
  }
```

➔ **это конец.** Как я уже говорил, кодинг смартфонов — направление перспективное. Помимо этого, процесс этот очень занятный и имеет некий дух исследования. А где взять инфу по кодингу под Symbian? Вот он, самый главный вопрос :). На русском инфы практически нет. Единственная книга по программированию смартфонов на Symbian на русском языке вызывает спазмы желудка. Поэтому придется читать много документации на английском, взятой с официальных сайтов Nokia и Symbian, а также SDK HELP. Пара ссылочек, посмотреть которые обязательно, прилагаются **C**

http://forum.nokia.com
наиболее полное собрание ресурсов по кодингу под symban.

http://newlc.com
отличный ресурс с массой нетривиального материала.

SPECIAL МНЕНИЕ

ДРОЗДОВ АНДРЕЙ AKA SULVERUS
OFFBIT SECURITY TEAM.

Сегодня одной из самых прибыльных отраслей программирования является программирование под мобильные устройства, такие как мобильные телефоны, смартфоны, КПК и тому подобные. А с чего все начиналось? А с далекого 1995 года, когда вышла платформа Java, которая за счет своей кроссплатформенности очень быстро осела в мобильных устройствах. На Яве делались и делаются всевозможные игры и программы. В 2002-2003 годах появились такие средства разработки, как eMbedded Visual C++ SDk для карманных компьютеров и Ms Smartphone SDK для смартфонов. В этот период подобные системы очень раскрутились, была даже известная афера — отправка СМС Богу, на которой было наварено около миллиона долларов! В наше время появилась среда разработки приложений Visual Studio .NET 2005. Теперь можно писать и отлаживать все приложения под мобильные устройства на ПК на любом удобном программисту языке. Возможно, в скором времени разработка ПО под КПК и смартфоны догонит обычное ПО, поэтому вливайся, пока есть ниша.

танцуя на решетке

ОБЗОР НЕСТАНДАРТНЫХ ВОЗМОЖНОСТЕЙ C#

ЧЕЛОВЕК РАЗУМНЫЙ, В ЛИЦЕ КОРПОРАЦИИ МАЙКРОСОФТ, СО ВРЕМЕНЕМ ПОРОДИЛ ПЛАТФОРМУ .NET, ДЛЯ КОТОРЫЙ БЫЛ СОЗДАН СПЕЦИАЛЬНЫЙ ЯЗЫК C#. ГОСПОДА ИЗ МАЙКРОСОФТ СДЕЛАЛИ КОД «УПРАВЛЯЕМЫМ», ПРИДУМАЛИ РЕФЛЕКСИЮ ТИПОВ, МЕТОД ИНДЕКСАТОРА, ДЕЛЕГАТЫ И ДОБАВИЛИ ЕЩЕ МНОГО НОВЫХ ВОЗМОЖНОСТЕЙ. В ДАННОЙ СТАТЬЕ МЫ УЗНАЕМ ВСЮ ПОДНОГОТНУЮ ЯЗЫКА C#

Андрей Дроздов aka Sulverus, Offbit security team
sulverus@mail.ru

Для начала я расскажу о ряде преимуществ языка C#, благодаря которым, на данный момент времени, он является вершиной эволюции программирования. C# вобрал в себя все лучшее: от C/C++ он взял возможность перегрузки операторов для типов, созданных программистом, и интеграцию COM, от Java — автоматическое управление памятью, от VB — использование свойств-классов, от RUBY — рефлексию типов.

➔ **создаем жука для опытов.** Для наших опытов нам нужно будет написать класс, над которым мы будем издеваться. Назовем его bug. В этой статье я бы хотел рассмотреть технологии организации двухстороннего взаимодействия объектов в приложении — это делегаты и события. Так же мы узнаем о том, как можно перегружать операторы в C#, для большей наглядности я буду приводить «примеры из жизни», то есть показывать, для чего тот или иной алгоритм нужен программисту или хакеру. Начнем мы с простого, а именно — с перегрузки операторов. Я приведу несложный пример перегрузки оператора сложения. Напишем функцию, аналогичную функции strcpy в C++:

```
string result = to + from;
```

Это элементарно. Желающие могут реализовать функцию, аналогичную lstrcpy :). Теперь перейдем к более серьезной перегрузке операторов, например, к перегрузке операторов равенства. Если мы хотим сравнить не просто какие-то два числа, а, предположим, два структурных типа данных, мы будем использовать метод Object.Equals().

перегрузка операторов равенства

```
public override bool Equals(object obj)
{
  if (((bug)obj).bugA == this.bugA &&
((bug)obj).bugB == this.bugB)
//перегружаем операторы
    return true; //если равны
  else
    return false; //если не равны
}
```

Разберем написанный код: слово override указывает на то, что мы будем использовать перегрузку; в роли параметров для функции будут служить 2 объекта; также мы заранее объявили 2 переменные типа int с именами bugA и bugB. Далее мы замещаем метод Equals(): в случае, если переменные равны, то возвращается значение true, в обратном случае передается значение false (вполне естественно, ведь мы используем логический тип данных bool). Так же для удачного замещения необходимо выполнить замещение GetHashCode(). Для этого напишем еще одну функцию, возвращающую это замещение: объявляем тип public override int и говорим ему возвращать HashCode: return this.ToString().GetHashCode(). Чтобы усвоить вышесказанное, перейдем к примеру, из которого видно, зачем нужно столько непонятного кода.

➔ **перебираем стог сена.** Для наглядности мы напишем брутфорс, который будет подбирать число, равное некому числу n. Вначале загадаем некое число n, равное 68491. Теперь надо, чтобы программа угадала его. Объявляем функцию public static void brute(long diap). Мы будем передавать функции параметры диапазона чисел, в котором она будет искать наше число. Для реализации перебора мы будем использовать написанный нами метод для перегрузки операторов равенства и цикл for.

код брутфорса

```
public static void brute(long diap)
{
  n = 68491; //задаем некое число
  NewMessage("w8. Bruteforce working…");
  //вызываем событие
   for (int i = 0; i < diap; i++)
//мутим цикл
  {
    d = i;
    string brute =
Convert.ToString(bug.Equals(n, d));
//сравниваем и возвращаем результат
в строку
    if (brute == "True")
    {
      BruteComplete(d.ToString());
//вызываем событие при удачном переборе
      break;
    }
  }
  if (ok == false)
  {
    NewMessage("Brete Failed:(");
  }
}
```

Как видно из этого кода, мы сравниваем число n с текущим числом и возвращаем значение в строковый тип данных brute методом System.Convert.ToString(). Если числа равны, то цикл прекращается. Поясню возможно сложившиеся у читателя непонятки относительно методов NewMessage() и BruteComplete(): это не методы, а вызовы событий, которые, в свою очередь, работают с делегатами, а делегаты вызывают методы.

➔ **делегация из Африки.** Программисты создают массу классов, методов, типов и прочих кусков кода. Все они, так или иначе, взаимосвязаны друг с другом: одни объекты порождают другие, пересылают им какие-то параметры и т.д. Однако все это работает только в одну сторону. А если порожденный объект захочет отослать что-то объекту, который его породил? Раньше в Win32/C/C++ было понятие обратного вызова, но у этого способа был один существенный недостаток: создавался просто указатель на функцию, то есть ссылка на адрес в оперативной памяти, и, следовательно, такой способ приводил к массе ошибок, переполнению буфера и т.д. В языке C++ нет решения подобной проблемы, а в языке C# для ее решения есть спе-

циальный класс (System.MulticastDelegate). Благодаря работе делегата исключаются ошибки переполнения буфера и срыва стека, поскольку при создании делегата указывается не только ссылка на функцию, но и набор передаваемых ей параметров. Объявить делегат очень просто: public delegate void BlahBlah(string Blah). Заметим, что делегат может иметь ссылку на несколько функций: для этого нужно использовать метод MulticastDelegate.Combine(). Единственное, что должно быть общего у этих методов — список параметров. Для оптимизации кода при совмещении нескольких функций в одном делегате можно использовать оператор сложения. Таким же образом можно объединять два делегата в один:

```
public static delegate play(string
file_position);
public static delegate pause(string
file_position);
MulticastDelegate Play_and_Stop =
play+pause;
```

➔ **события давно минувших дней.** Вернемся к описанию кода нашего брутфорса: в нем я упоминал о событиях. События были и в C++, но теперь они строятся на основе делегатов. То есть мы «вешаем» события на нужный нам делегат, и они передают методу нужные параметры. В нашем коде есть два события: NewMessage() и BruteComplete(). Первое событие находит делегат метода msg(), который вызывает метод, созданный для того, чтобы выводить в консоль сообщение; второе наступает в случае, если перебор был удачным. Если просмотреть событие дизассемблером (не обычным, а NET'овским), то мы увидим, что оно состоит из двух методов: add_[Имя События]() и remove_[Имя События]. Для того чтобы параметры не ушли «в никуда», мы должны создать приемники событий, которые будут ожидать их наступления. Что должен делать приемник? Он должен добавить принимающий метод в таблицу указателей делегата. Для этого нужно использовать перегруженный оператор «+=», а для удаления «-=». Добавим два приемника для делегатов каждой функции:

```
bug.NewMessage +=
new bug.msg_sender(bug.msg);
bug.BruteComplete +=
new bug.bOK(bug.BruteOk);
```

Теперь вместо того, чтобы писать полный адрес функции, можно просто писать NewMessage(string msg_text);. Для нашего брутфорса нам понадобится написать метод, который будет включать и выключать приемники событий:

переключатель событий

```
public static void EventSwitch(bool
current_switch)
{
 if (current_switch == true)
 {
   bug.NewMessage +=
new bug.msg_sender(bug.msg);
//включаем приемники
   bug.BruteComplete +=
new bug.bOK(bug.BruteOk);
 }
 else if (current_switch == false)
 {
   bug.NewMessage -=
new bug.msg_sender(bug.msg);
//выключаем приемники
   bug.BruteComplete -=
new bug.bOK(bug.BruteOk);
 }
}
```

В этом коде нет ничего сложного: для того чтобы включить или выключить приемники, необходимо передать функции значение логической переменной.

➔ **о жизненных потоках.** Рассказывая об особенностях программирования на C#, нельзя не упомянуть о потоках. Для работы с потоками существует пространство имен System.Threading. Для примера мы можем написать программу, которая будет считать числа, а для закрепления пройденного материала мы будем делать это не просто с помощью цикла for, но и используя перегрузку операторов квадратных скобок («[]»). Прежде чем создавать поток, нам нужно написать функцию для работы с числами. В данном случае мы будем использовать метод индексации (это и есть перегрузка операторов квадратных скобок).

➔ **дальше — больше.** Давай на время вспомним, как работают процессы в Win2k. Раньше приложение могло состоять из нескольких процессов, которые могли порождать новые потоки. В платформе .NET все немного иначе, соответственно и в Windows Vist'e тоже. Теперь есть понятие «домен приложения», которое является своеобразным контейнером для потоков в .NET-платформе, то есть домен приложения полностью огораживает поток и его ресурсы от других потоков и доменов приложений. В результате разные домены приложений не могут совместно использовать одни и те же ресурсы, даже если они являются глобальными. Непонимание этого факта приводит к довольно серьезным ошибкам. Для работы с доменами приложений есть класс AppDomain, так давай же поковыряем его! Чтобы было проще это понять, напишем класс BindingCodeDomain, в котором будет 2 функции: первая получает строку и дописывает туда данные, вторая регистрирует первую, как новый метод в созданном нами делегате msg_sender в классе bug.

➔ **объектно сориентируемся.** Теперь я бы хотел внести ясность в тему работы с объектами. Для того чтобы создать объект, нужно использовать директиву new. Для примера напишем два класса, которые будут имитировать жуков, бегающих наперегонки, и заодно будем закреплять пройденное.

Для начала придумаем концепцию: создаем метод, который делает объект класса BugSpeedy и предает ему определенные параметры, после чего создаем два процесса, и жуки бегут наперегонки. Для создания объектов у нас будет использоваться конструктор объектов, которому нужно будет передавать параметры. Также конструктор будет создавать поток для жука. Мы создадим двух жуков (Петю и Васю), заранее подготовив в классе BugSpeedy метод Run для бега. В методе Run будет просто реализован цикл for и набор простых математических операций, чтобы рассчитать скорость жука. Теперь, благодаря таким нехитрым заморочкам, мы оптимизировали код и вся игра в жуков будет занимать 4 строки:

битва жуков

```
 bug.BugSpeedy Vasya =
new bug.BugSpeedy("Vasya", 100, 25);
//создаем жука Васю
 Thread VasyaTrack = bug.BugTrack;
//Вася на беговой дорожке
 bug.BugSpeedy Petya =
new bug.BugSpeedy("Petya", 110, 60);
//создаем жука Петю
 Thread PetyaTrack = bug.BugTrack;
//Петя на беговой дорожке
```

Также нужно поставить условие, проверяющее, кто из жуков пришел первым. Вуаля, теперь можно наслаждаться тараканьими бегами, запустив процесс методом Start() :). На таком простом примере хорошо видна объектно-ориентированная модель программирования, — надеюсь, что он поможет тебе лучше ее понять. С жуками закончим.

➔ **конец — делу венец.** Я думаю, что у тебя в голове уже сложился вопрос: зачем нужно столько объектно-ориентированных наворотов и извращений? В больших и сложных компьютерных системах, использующих большой ресурс памяти, такой подход необходим по двум причинам. Во-первых, использование подобных надстроек сильно оптимизирует код, а значит, уменьшает время работы программы и увеличивает быстродействие сервера. И во-вторых, в современном программировании есть задачи, которые без подобных техник не решить, например, работа с базами знаний, нейронными сетями, искусственными интеллектами и т.д. Конечно, я не успел рассказать обо всех особенностях объектно-ориентированного программирования на C# (это тема отдельной книги), но с какой-то частью этого вопроса ознакомил. Благодаря делегатам у программистов резко сократится количество ошибок, а их программы станут на порядок безопаснее. Если у тебя есть какие-нибудь вопросы или интересные идеи — пиши на почту sulverus@mail.ru, я с радостью помогу, отвечу и поучаствую. Напоследок замечу, что у языка C# самая лучшая на сегодняшний день концепция ООП, поэтому, если ты решил заняться каким-либо серьезным проектом, используй C# **C**

SPECIAL ИНТЕРВЬЮ

Михаил Фленов — широко известный в узких кругах программер, автор и книгописатель.

Интервью брал Александр Лозовский.

НЕСМОТРЯ НА ТО, ЧТО БОЛЬШИНСТВО НАШИХ ЧИТАТЕЛЕЙ ТЕБЯ ЗНАЮТ, ВСЕ РАВНО, ПРЕДСТАВЬСЯ.

МИХАИЛ ФЛЕНОВ: Фленов Михаил, а в девичестве Horrific.

ДАВАЙ НАЧНЕМ СО СПИСКА ТВОИХ СВЕРШЕНИЙ. СКОЛЬКО ПРОГРАММ ТЫ УЖЕ УСПЕЛ НАПИСАТЬ?

МИХАИЛ ФЛЕНОВ: 37. В эту цифру входят все программы с www.cydsoft.com, а также проекты, написанные для предприятий, где я работал, а абсолютно все программы упомнить не смогу. Когда учился в институте, написал еще пару программ (одна для бухгалтерии, и еще одна — для тестирования студентов), но они были небольшие. Если учитывать все проекты, в которых я участвовал, и прибавить примеры для книг, то цифра будет состоять из четырех разрядов.

ОТЛИЧНО, НО Я ВОТ ЗАМЕЧАЮ, ЧТО ТАМ ПОРЯДОЧНО АНАЛОГОВ ВСЕМИРНО ИЗВЕСТНЫХ ПРОГРАММ. НАПРИМЕР, НЕТРУДНО ДОГАДАТЬСЯ, ЗАЧЕМ НУЖНА ПРОГРАММА CYD GIF STUDIO PRO. КАК ЖЕ ОНА ВЫДЕРЖИВАЕТ КОНКУРЕНЦИЮ?

МИХАИЛ ФЛЕНОВ: А никак. Она не выдерживает конкуренции, потому что всем приходится заниматься самому. Нужно писать код, тестировать, продвигать программу, поддерживать базу пользователей, которых накопилось за все это время большое количество и многое другое. При этом программы за последние три года практически не обновлялись. А ту, которую назвал ты, я заморозил почти четыре года назад и только недавно сделал обновление. Сейчас я усиленно занялся обновлением и в ближайшее время планирую начать продвижение, что является самым важным моментом.

МНОГИЕ ИЗ НАШИХ ЧИТАТЕЛЕЙ ЗАХОТЯТ ЗАРАБАТЫВАТЬ ДЕНЬГИ СВОИМ ПРОГРАММЕРСКИМ ТРУДОМ. ПО-ТВОЕМУ, НУЖНО ЛИ ОТКРЫВАТЬ СВОЮ ШАРОВАРНУЮ КОНТОРУ?

МИХАИЛ ФЛЕНОВ: Контору открывать очень рискованно. Давай посмотрим на простом примере — WinZIP. Эта программа не идеальна и обладает не лучшими возможностями. Есть архиваторы намного круче, которые поддерживают больше форматов и стоят дешевле, но WinZIP — бестселлер. Почему? Программы продаются не благодаря возможностям и коду, а благодаря ухищрениям маркетологов. Если ты умеешь втулить пользователю кал обезьяны, то сможешь продать любую программу. Если не можешь продать за копейки золото, то не стоит даже пытаться. Отдача будет минимальна, даже у самого лучшего продукта.

Помню, как на заре интернета один парень на спор поднял абсолютно пустую страницу со счетчиком чуть ли не на первое место в рейтинге rambler.ru. Вот такой человек продаст любую программу, и он может открывать свою контору.

А КАК ДЕЛА С ПРОДАЖАМИ? ОХОТНО ЛИ БУРЖУИНЫ ПОКУПАЮТ ТВОИ ПРОДУКТЫ?

МИХАИЛ ФЛЕНОВ: Покупают, и для меня одного этого достаточно. Если сравнить продажи и трафик, который идет с сайта, то все не так безнадежно, даже несмотря на тотальную старость всех программ.

КАКИЕ ЖЕ ПРОГРАММЫ ПОЛЬЗУЮТСЯ НАИБОЛЬШИМ СПРОСОМ (ПРО РУСИЧЕЙ НЕ СПРАШИВАЮ, ПОСКОЛЬКУ ИХ СПРОС НЕ НЕСЕТ ЗА СОБОЙ ДОХОДА)?

МИХАИЛ ФЛЕНОВ: На данный момент наибольший спрос идет на те программы, которые я продвигаю, а их три. Уж извини, но секрет фирмы открывать не буду. Все остальные висят на сайте в качестве довеска и приносят небольшой доход.

НЕ ПОРА ЛИ ПЕРЕХОДИТЬ НА МОБИЛЬНЫЕ ПЛАТФОРМЫ? УЖ КУДА АКТУАЛЬНЕЕ, ПО-МОЕМУ.

МИХАИЛ ФЛЕНОВ: У меня своя корова, и я ее дою. В другие сферы прыгать не хочу и никому не советую дергаться из стороны в сторону. Если взял один курс, то нужно его максимально четко придерживаться. Только такая корпорация, как Microsoft, может работать на всех рынках сразу, а все остальные работают на узких рынках. Моя основная специализация — простые программы для WEB-дизайна, а все остальное вторично.

Но я хочу и даже мечтаю выпустить собственную небольшую игру. Тут проблема в том, что я моделирую в 3D-редакторах как курица лапой, а рисую еще хуже. Когда бог раздавал эти таланты, я видимо разглядывал красоток. И все же, возможно, когда-нибудь из под моей клавиатуры выйдет полноценная игра. Но она будет под другим лейблом.

ЕСЛИ УЖ ГОВОРИТЬ О ДЕНЬГАХ, ТО, МОЖЕТ БЫТЬ, ТЕЛЕРАБОТА НА НЕКОЕГО ТОЛСТОГО БУРЖУИНА ЛУЧШЕ? ПОЧЕМУ ТЫ НЕ ВЫБРАЛ ДЛЯ СЕБЯ ТАКОЙ ПУТЬ?

МИХАИЛ ФЛЕНОВ: Телеработа была популярна и приносила доход только в течение первого года с момента появления. Тогда я мечтал зарабатывать собственными силами и в это дело не включился. Сейчас проще попасть на обман, чем найти реального работодателя, который будет платить за телеработу за бугром. Все буржуины переключились на оффшор, как более качественное и достаточно дешевое решение собственных проблем.

С другой стороны, я занимаюсь телеработой с момента возникновения журнала X. Но ведь при этом я не сижу в редакции «Геймленд» и даже нахожусь в другом городе. Да и книги пишу удаленно, сидя на диване за чашкой кофе. Так что телеработа для меня стала вторым домом.

А ЕЩЕ ЛУЧШЕ — ВООБЩЕ ЗА ГРАНИЦУ УЕХАТЬ. НЕ БЫЛО ТАКОЙ МЫСЛИ?

МИХАИЛ ФЛЕНОВ: Раньше таких мыслей не было, потому что был патриотом своей страны (да и сейчас это чувство еще остается). Много раз предлагали, но я отказывался. Когда же появились дети, я взглянул на жизнь с другой стороны. Сейчас мне кажется, что мы живем, мягко говоря, в полном ужасе. В последнее время я действительно стал подумывать о том, чтобы двинуть в теплые страны на нормальные заработки. Но все еще немного побаиваюсь такого кардинального изменения собственной жизни. Пока мне нравится в Питере, и я полюбил этот город, хотя только-только сюда переехал.

Недавно хотел уехать, и меня уже готовы были взять две очень серьезные фирмы, одна из которых в Германии, а другая — в США. Но мне отказали по банальной причине — плохое знание английского и абсолютное незнание немецкого. После этого я временно прекратил попытки перебраться за границу. Я хорошо читаю и без проблем понимаю английскую речь. Но вот меня понимают очень плохо, потому что мое произношение и излишние нервы на интервью делают свое дело. Когда я говорю на английском, то забываю все слова и сильно нервничаю. И тут даже не помогает мой опыт и знания в программировании, потому что в каждом объявлении на работу в США написано обязательное требование — коммуникабельность и общительность. Если с тобой тяжело общаться, то скоро уволят, каким бы специалистом ты не был. Поэтому сейчас я усиленно подгоняю свой английский. Вдруг все же окончательно решусь переезжать за бугор.

ТЕПЕРЬ О НАУКЕ. ИЗВЕСТНО, ЧТО У ТЕБЯ НЕ ПРОГРАММЕРСКОЕ ОБРАЗОВАНИЕ. КАК ЖЕ ТЫ САМООБУЧАЛСЯ?

МИХАИЛ ФЛЕНОВ: С литературой действительно было сложно. Читал все подряд — все эхи в ФИДО, мануалы и справочники. Английский я не знал, потому что в школе учил французский, но вооружился толстым словарем и начал переводить. Так выучил английский и научился программировать одновременно. В общем, читал все, что попадалось под руку.

А КОГДА ТЫ РЕШИЛ ПОДЕЛИТЬСЯ ЗНАНИЯМИ? ПРЕДШЕСТВЕННИК НЫНЕШНЕГО VR-ONLINE.RU ПОЯВИЛСЯ НАМНОГО РАНЬШЕ ПЕРВЫХ СТАТЕЙ В X И ТЕМ БОЛЕЕ КНИГ? КАК ЖЕ ОН НАЗЫВАЛСЯ-ТО... X-C-R.COM?

МИХАИЛ ФЛЕНОВ: Я сам уже не помню первое название :), потому что первый домен регистрировал не я, а он достался на халяву от одного из читателей. Насколько я помню, этот домен или накардили, или спионерили :).

Как начал делиться? Просто хотел помочь другим, ведь в те времена литературы практически не было. Я не собирался зарабатывать на этом деньги и сейчас не собираюсь, поэтому не особо продвигаю свой сайт. Он чисто для общения с читателями, друзьями и для обмена опытом между программистами. Мы помогаем всем, кто нуждается в помощи.

ЭТО САЙТ, А КАК ТЫ ОКАЗАЛСЯ В X? ПОМНИШЬ ЛИ ТЫ СВОЮ ПЕРВУЮ СТАТЬЮ?

МИХАИЛ ФЛЕНОВ: Однажды, выходя с работы, я увидел самый первый номер X и купил. Мне понравился журнал, а на первой странице был призыв записаться в ряды авторов. Я написал в редакцию, и через день мне ответил SINtez. С тех пор я был в команде и получал всю рассылку. Хорошие были времена, и тогда действительно ощущался дух команды. Да, было много разгильдяйства, но и душок был приятный. Сейчас в редакции работа более серьезная и профессиональная, и душок изменился. Первая статья была про обман шароварных каталогов. Для меня это была больная тема, ведь приходилось часто работать с ними.

ЧТО НАСЧЕТ КНИГ? КАК ТЫ ДОШЕЛ ДО ЭТОЙ МЫСЛИ? ВООБЩЕ, КНИГИ-ТО, НЕБОСЬ, ПОСЛОЖНЕЕ ПИСАТЬ?

МИХАИЛ ФЛЕНОВ: Скажу так — книги писать интересно. А как я начал? Однажды я понял, что мои статьи на сайте сложно читать, потому что информация разрознена. Я начал собирать ее в одну большую книгу, и получилось 10 глав «Библии Delphi», которая стала доступна всем бесплатно. Но ее увидели в издательстве «Символ-Плюс» и предложили издать. Я доделал книгу и сдал, но, по непонятным причинам, издательство отказалось. Я выложил всю книгу в интернет бесплатно, как и планировал в самом начале, и в этот момент со мной связались из БХВ. Я написал для них книгу «Программирование в Delphi глазами хакера», а в последствии доработал и издал «Библию Delphi».

ЧТО ПОСОВЕТУЕШЬ ЧИТАТЕЛЮ НАПОСЛЕДОК? СТОИТ ЛИ ЗАКАНЧИВАТЬ ПРОГРАММЕРСКИЙ ВУЗ, СТОИТ ЛИ ПРОГРАММИТЬ, ИЛИ МОЖЕТ БЫТЬ ЛУЧШЕ АДМИНИНГОМ ПРОБАВЛЯТЬСЯ?

МИХАИЛ ФЛЕНОВ: Учиться всегда стоит, но только если действительно хочется. Если душа лежит к программированию, к созиданию, то учеба не пройдет даром, особенно высшее образование. Программисты нужны, особенно в мегаполисах, и будут нужны всегда. В вузе программистов обучают математическим наукам, которые пригодятся, даже если после этого стать админом, телефонистом или дворником :) C

SPECIAL ОПРОС

АЛЕКСЕЙ ПЕТРОВ

В IT 20 лет. Эксперт в области защиты данных, эксперт по компьютерным преступлениям, эксперт по сетевым коммуникациям и телефонии. Сертификаты от Novell/ 3com/Bay/Siemens/Cisco/ ISACA. Консультант по вопросам ИТ-безопасности в Secproof Oy *(www.secproof.com)*. Свободный консультант Arhont.com, iPRO.lv.

КРИС КАСПЕРСКИ

Известен еще как мыщъх. Компьютеры грызет еще с тех времен, когда Правец-8Д считался крутой машиной, а дисковод с монитором были верхом мечтаний. Освоил кучу языков и операционных систем, из которых реально использует W2K, а любит FreeBSD 4.5. Живет в норе, окруженной по периметру компьютерами и стеллажами с литературой.

АНАТОЛИЙ СКОБЛОВ

Последние 17 лет — системный программист, аналитик. Работает дома на себя или на заказчиков. Из известного — ядро outpost personal firewall, модем russian courier. Сфера профессиональных интересов — безопасность, телефония, интернет и так далее.

АЛЕКСАНДР ЛОЗОВСКИЙ

Если в двух словах — этому человеку СПЕЦ обязан своей жизнью. Практически каждый номер Александр делает ему искусственное дыхание в условиях, приближенных к реальным военным действиям :). Кроме того, — редактор «Кодинг» Хакер'а.

ИВАН (SKYWRITER) КАСАТЕНКО

Возбуждающий идеи на редколлегии. Редактор диска к журналу хакерСПЕЦ. В прошлом и настоящем успешный программист. Горячий любитель .net.

СЛУЧИТСЯ ЛИ КОГДА-НИБУДЬ, ЧТО ОДИН ИЗ ЯЗЫКОВ БУКВАЛЬНО «ЗАВОЮЕТ МИР»?

АЛЕКСЕЙ ПЕТРОВ: Очень маловероятно. Утверждение подобного рода можно сравнить с попыткой заявить что «когда-нибудь проявится universal-swiss-knife, в котором будет все, и он вытеснит все ножи и инструменты с рынка». Каждый язык программирования для решения конкретной комплексной задачи имеет свои плюсы и минусы. Каждый язык программирования имеет свою направленность и специализацию. Языки бывают узкопрофильные и широкопрофильные, сложные и простые, компилируемые и интерпретируемые, близкие к аппаратной части (низкоуровневые) и аппаратно-независимые слоеные-кроссплатформенные пироги (языки высокого уровня).

Причем высокоуровневые языки от аппаратной реализации компьютера помимо множества плюсов имеют и минусы. В частности, они не позволяют создавать простые и точные инструкции к используемому оборудованию. Программы, написанные на языках высокого уровня, проще для понимания программиста, но гораздо менее эффективны, чем их аналоги, создаваемые при помощи низкоуровневых языков. Одним из следствий этого стало добавление поддержки того или иного языка низкого уровня (язык ассемблера) в большинство современных профессиональных высокоуровневых языков программирования.

ДМИТРИЙ КОВАЛЕНКО
Системный программист, разработчик в infopulse ukraine. Хобби — теоретическая вирусология, в частности, математическое моделирование полиморфных алгоритмов в вирусах.
www.vr-online.ru

МИХАИЛ ФЛЕНОВ
Внештатный автор X почти с самого рождения журнала, создатель сайта *www.vr-online.ru*, автор 11 книг на русском и 4 на английском языке.

Изначально глупо писать узкоспециализированные драйвера устройства на далеком от железа и его основ кроссплатформенном языке, таком как, скажем, Java (хотя в жизни могут найтись всякие извращенцы). В реальности вполне возможны ситуации, когда на языке «А» задачу придется кодить в 3-4, а то и в 10 раз дольше, но удастся решить рациональнее по ресурсам, а на языке «Б» ее проще будет описать, но работать будет долго и ресурсоемко. А в жизни решить задачу надо экономически выгодно, посему будет взят кодер, который знает только язык «В», на котором эта задача и будет реализована — долго, сложно, ресурсоемко, неэффективно — но зато дешево и сердито...

КРИС КАСПЕРСКИ: По данным лингвистов, недавно собравшихся на одной шумной конференции, 70% ныне используемых языков через несколько десятков лет скорее всего перейдут в разряд мертвых. И будут как греческий или латынь. Кстати, если вспоминать историю, то и греческий, и латынь оказали колоссальное влияние на большинство современных языков.

То же самое наблюдается и с языками программирования. Термин «завоевание» — очень точный. Фирмы-создатели компиляторов/фреймворков и прочих технологий вкладывают в них нехилые деньги и двигают на рынок, зачастую действуя вопреки интересам пользователей. Новых идей ни у кого нет, поэтому все языки становятся более и более похожими друг на друга. Иные концепции просто отметаются. Взять тот же forth или lisp. И где они сейчас?! Вокруг засилье Си++-подобных языков с одинаковыми парадигмами объективного программирования, планомерно эволюционирующего в метапрограммирование, которое выросло из шаблонов, а шаблоны выросли из препроцессоров.

Так что ничего радикально нового на рынке, по сути, и нет. Меняется только синтаксис и оверхид. Как говорится, усложнять легко — упрощать трудно. Чтобы изобрести простой и гибкий язык, каким является тот же Си — это же талант нужен. А добавлять в Си++ новые фичи может любой индус. Только кто этим языком будет пользоваться?! Говорят, что среднестатистический пользователь MS office использует 2% возможностей. А сколько возможностей использует средневзятый приплюснутый программист, если все 100% фич пока что не поддерживает ни один компилятор?!

На самом деле, война умов уже давно закончена и сейчас идет брожение. С другой стороны, существует такая классная штука, как язык Пролог, но под него не существует эффективных компиляторов. А не существует их потому, что не существует адекватных процессоров, но они могут появиться в любой момент! Мы не знаем технологий, которые еще не открыты, поэтому не можем сказать, каким будет мир через десять лет. Но навряд ли один язык победит остальные, поскольку даже если он появится, тут же кто-то придумает другой язык, более удобный для решения определенного круга задач, который может как расширяться, так и сужаться...

АНАТОЛИЙ СКОБЛОВ: Возможно, это будет арабский язык. Языки программирования — исключено.

АЛЕКСАНДР ЛОЗОВСКИЙ: Я думаю, произойдет следующее. Прямо скажем, друзья, произойдет апокалипсис. Энтропия вселенной достигнет критического уровня, в результате люди станут злыми, брат пойдет войной на брата, сын — на отца. С неба посыплются гигантские камни, машины восстанут против людей, разверзнутся хляби небесные, а реки станут красными от крови. Но пока что это нам не грозит. C#, может быть, и имеет шанс «завоевать» мир, но это не значит, что кроме .NET'а ничего никому не будет нужно.

ИВАН КАСАТЕНКО: Я так не думаю. Давай проведем параллель с реальным миром — на протяжении многих веков существует целое множество языков: кто-то говорит на одном, кто-то на другом. И не факт, что самый популярный из них статистически является самым удобным (взять, хотя бы, китайский). Но даже несмотря на популярность, ни один из них так и не стал общепринятым стандартом, хотя и были даже искусственные попытки его ввести.

Причин тому несколько. Каждый язык формировался согласно национальным особенностям и необходимостям, а потому наиболее удобен конкретной нации. Человеку свойственна привычка, он не хочет менять того, что было веками. С языками программирования абсолютно та же ситуация. Есть самые распространенные: C, C++, C# (стараниями Брата), но никто не отменяет существования языков совершенно другой природы — процедурных, декларативных и т.п. Каждый из них (Nemerle, Haskell, Prolog, даже COBOL) хорош по-своему, у каждого есть сторонники, а значит — каждый обречен на выживание.

ДМИТРИЙ КОВАЛЕНКО: Думаю, такого не случится. Уже сейчас в программировании столько всяких идеологий, архитектур и платформ! Ни один язык не может полностью «накрыть» все это разнообразие. И в будущем вряд ли что-то существенно поменяется.

ЧТО ЛУЧШЕ: СИ ИЛИ СИ++?

АЛЕКСЕЙ ПЕТРОВ: C++ родился из C и очень похож на родителя. Изначально C++ был препроцессором, переводящим его конструкции в код C, который уже дальше передавался компилятору. C++ включает в себя C, и даже практически полностью с ним совместим, что позволяет некоторым писать на C++ как на C, считая его просто расширением (хотя рано или поздно либо уходят в полный C++, либо спускаются обратно на C). Это также позволяет использовать в C++ старые наработки на C. Но сравнивать C++ и C — сопоставимо с извечным вопросом, что лучше: курица, гусь или яйцо. C++ и C, продолжая развиваться и влияя друг на друга, давно представляют собой хороший конгломерат здорового симбиоза. Заговоздкой языка C стало то, что он оказался слишком низкоуровневым для задач. Задача красиво решалась, но тонула и пряталась от понимания в технической реализации. Предназначением C++ было

сделать написание программ более простым и приятным, немножко подняв планку уровня С и расширив его возможности. Новые веяния требовали от С средства работы с абстрактными типами данных, объектов, что и было реализовано введением в С++ механизма классов, позволяющих определять и использовать новые типы данных на основе существующих.

Кстати, в качестве базового языка С (для С++) был выбран не случайно, потому что он:

1. МНОГОЦЕЛЕВОЙ, ЛАКОНИЧНЫЙ И ОТНОСИТЕЛЬНО НИЗКОГО УРОВНЯ.
2. ОТВЕЧАЕТ БОЛЬШИНСТВУ ТРЕБОВАНИЙ СИСТЕМНОГО ПРОГРАММИРОВАНИЯ.
3. ИДЕТ ВЕЗДЕ И НА ВСЕМ.
4. В ТОМ ЧИСЛЕ ПРИГОДЕН ДЛЯ ПРОГРАММИРОВАНИЯ НА UNIX.

КРИС КАСПЕРСКИ: Это то же самое, что сравнивать километры с литрами, хотя в каком-то смысле автомобилисты так и поступают (расход топлива). Но все-таки это разные языки, и далеко не во всех задачах оправдано использование Си++. И уж тем более не факт, что время, вложенное в его изучение, окупится ускоренной разработкой программ. Но это уже священные войны начинаются...
Си — низкоуровневый язык, далеко не все приемлют его парадигму. Си++ — нечто очень большое и сложное, плюсов у него всего два (да и те достались в наследство от Си), а вот минусов...
ИВАН КАСАТЕНКО: Лично мне более предпочтительным кажется Си. Не знаю уж, связано ли это с моей работой или просто исторически сложившаяся симпатия. Си нравится своей простотой и читабельностью хорошо написанного кода. Гораздо сложнее (мне лично) читать код на Си++. Так что если уж выбирать классы, проектирование с использованием шаблонов и т.п., то это должен быть язык поудобнее. Мне в этом плане симпатизирует Java и C#. Так что — либо Си, либо Java/C#.
ДМИТРИЙ КОВАЛЕНКО: Если вопрос только в языках, то Си++ лучше, поскольку Си является подмножеством Си++, а значит, Си++ обладает всеми возможностями Си. Если же вопрос в том, какой подход лучше — процедурный (как в Си) или объектно-ориентированный (как в Си++), то смотря какие задачи надо решать. Для больших проектов, которые делает много людей, лучше подходит объектно-ориентированный подход. В небольших проектах вполне оправдан процедурный подход.
МИХАИЛ ФЛЕНОВ: Идеальных языков не бывает, и все зависит от задачи. Если необходимо написать офисную программу с большими возможностями, то использовать Си проблематично, а разработка отнимет очень много времени. Поэтому выбор должен пасть на Си++. Если необходима маленькая и быстрая утилита, то С++ будет излишним. И в данном случае выиграет старичок Си. Желательно знать несколько разных языков и при решении определенной задачи выбирать тот, который лучше подходит в данный момент.

КАКИЕ ПЛЮСЫ И МИНУСЫ У UNIX WAY?

КРИС КАСПЕРСКИ: Сложный вопрс, в двух словах об этом и не скажешь. Если же говорить предельно кратко, то плюсы такие:

- ВОЗМОЖНОСТЬ ПОСТРОЕНИЯ СЛОЖНЫХ СИСТЕМ ИЗ ПРОСТЫХ «КИРПИЧИКОВ», КАЖДЫЙ ИЗ КОТОРЫХ МОЖЕТ БЫТЬ ЗАМЕНЕН ДРУГИМ ИЛИ МЕЖДУ ДВУМЯ КИРПИЧИКАМИ ВСТАВЛЕН ТРЕТИЙ.
- МИНИМУМ ПОВТОРНОГО ИСПОЛЬЗОВАНИЯ КОДА, ОТКРЫТЫЕ ПРОТОКОЛЫ, ЧЕТКОЕ РАЗДЕЛЕНИЕ НА УРОВНИ.

Что касается минусов, то они такие:

- ПО МЕРЕ РОСТА СИСТЕМЫ РАБОТАТЬ С НЕЙ СТАНОВИТСЯ ВСЕ ТРУДНЕЕ И ТРУДНЕЕ, ПОСКОЛЬКУ ВМЕСТО МОНОЛИТНОГО БЛОКА У НАС ИМЕЕТСЯ МНОЖЕСТВО МАЛЕНЬКИХ БЛОКОВ, ЧАСТО ОТ НЕЗАВИСИМЫХ ПОСТАВЩИКОВ, БЕЗ ЧЕТКИХ СПЕЦИФИКАЦИЙ. И ЭТО УЖЕ НЕ ПРОГРАММА ПОЛУЧАЕТСЯ, А КОНСТРУКТОР, С КОТОРЫМ БОЛЬШЕ ТРАХАЕШЬСЯ, ЧЕМ РАБОТАЕШЬ.

АНАТОЛИЙ СКОБЛОВ: Минус — менее дружелюбная среда для пользователей (по сравнению с Windows), в первую очередь из-за того, что пользователи обычно с *nix не знакомы. Плюс — наличие бесплатных *nix'ов с открытыми исходниками, с которыми можно делать все, что угодно. Если, конечно, это требуется. Все остальное — лишь религиозные споры или частности.
АЛЕКСАНДР ЛОЗОВСКИЙ: По этому вопросу лучше обратиться к статье Криса Касперски «Так ли открыты открытые исходники» (www.xakep.ru/magazine/xs/060/076/1.asp) и статье Константина Клягина «Свободу софту» (www.xakep.ru/magazine/xs/053/032/1.asp). И труды волосатого Ричарда Столлмана будут полезны.
ИВАН КАСАТЕНКО: Плюс — свобода. Все мы любим свободу, равенство, братство. Соответственно, двигатель тут, в основном, — энтузиазм. А программисты-энтузиасты, взрощенные на ниве этой са-

мой свободы, способны горы свернуть. Минус — практическая нежизнеспособность крупных проектов. Из моего опыта не припомню ни одного жизнеспособного крупного ГНУтого проекта. Кроме, пожалуй, ядра Linux. Все остальное, прямо скажем, нещадно глючит. В качестве оправдания приводят обычно бесплатность. В общем, для крупных проектов тут не хватает главного — денег и (в большей степени) ответственности. Денег, которые позволят нанять грамотных специалистов, способных управлять командой, организовывать проект и так далее. Впрочем, деньги часто инвестируют и в «свободные» проекты. А вот второй компонент — ответственность — в большей степени все-таки свойственна коммерческим компаниям.

КАКИЕ ОСНОВНЫЕ ТЕНДЕНЦИИ СЕЙЧАС В ПРОГРАММИРОВАНИИ?

КРИС КАСПЕРСКИ: 1 АУТСОРТИНГ В СЛАБОРАЗВИТЫЕ СТРАНЫ.
2 ПРОЕКТИРУЮТ НЕ ИНЖЕНЕРЫ, А МАРКЕТОЛОГИ.
3 ТТХ НЕ ИМЕЮТ ЗНАЧЕНИЯ, ГЛАВНОЕ — ЦВЕТ.
4 ВАВИЛОНСКАЯ БАШНЯ ВСЕ ВЫШЕ И ВЫШЕ, ЗАЧЕМ — НЕПОНЯТНО, НО ВЫШЕ.
5 ДУМАТЬ НЕ НАДО, НАДО КОДИТЬ.

ДМИТРИЙ КОВАЛЕНКО: Виртуализация. Сейчас столько расплодилось всяких кроссплатформенных виртуальных машин, интерпретаторов, сред, поддерживающих скриптовые языки, data-driven технологий и прочей дряни, что разработчики почти не пишут живого машинного кода.

МИХАИЛ ФЛЕНОВ: Все движется в сторону компонентности и визуальности. Еще лет 8 назад я написал статью, в которой описывал историю языков программирования. Эта статья еще вошла в книгу «Библия Delphi». Там я говорил, что в ближайшее время победит компонентность, она станет основной технологией, что мы и увидели в последние годы (технологии Java, .NET и язык программирования Delphi — яркие представители компонентного программирования). Но вот что будет дальше, я пока сказать не могу. Следующего яркого рывка пока не вижу.

Достигли ли мы предела? Не знаю и не уверен. Когда в 93 году программировал на объектах, то думал, что это предел совершенства, — но нет, появились компоненты, которые удобнее и проще. Возможно, что на первый план выйдут web-программы, и мы уже не будем запускать на своем компьютере приложения для решения каких-либо задач. Если нужна будет офисная программа, то просто заходим на определенный сайт и работаем с нужной программой через браузер. Офисные web-программы уже есть у MS и Google. Но для того, чтобы они завоевали мир, необходима тотальная халява и высокая скорость интернета.

КАК НАПИСАТЬ БЕЗОПАСНЫЙ КОД?

АЛЕКСЕЙ ПЕТРОВ: 1 ПРОЧИТАТЬ И ИЗУЧИТЬ FAQ ПО SECURITY КОНКРЕТНОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ, А ТАКЖЕ ПО БЕЗОПАСНОСТИ СРЕДЫ, В КОТОРОЙ ПРОГРАММЕ ПРИДЕТСЯ РАБОТАТЬ, И ПО БЕЗОПАСНОСТИ СОПУТСТВУЮЩИХ ПРИЛОЖЕНИЙ.

2 УДЕЛЯТЬ БОЛЬШЕ ВНИМАНИЯ ВСЕМ ВНЕШНИМ ПАРАМЕТРАМ, ПРОПУСКАЯ ИХ ЧЕРЕЗ ФУНКЦИЮ-СТЕРИЛИЗАТОР.

3 ПРОДУМАТЬ ВСЮ СТРУКТУРУ ПРОГРАММЫ И ПРОРАБОТАТЬ ОПАСНЫЕ МЕСТА.

Как правило, программисты, которые досконально знают возможности и нюансы конкретного языка и сопутствующих приложений (скажем, SQL/WEB), просто предугадывают возможные опасности и ошибки. И заранее при кодировании либо решают их, либо обходят «тонкий лед».

КРИС КАСПЕРСКИ: Во-первых, исключить всю избыточность, убрать ненужный функционал. Во-вторых, писать вдумчиво, а не на скорую руку. Но, в принципе, ответа на этот вопрос никто не знает, хоть ежегодно и выходит много книг по этой теме и появляются инструменты, призванные обезопасить программиста от себя самого, но... ошибки все равно идут косяками, программы глючат, хакеры радуются, а все почему?!

Потому что уровень культуры программирования неуклонно падает: программированию нельзя научиться по наитию и уж тем более нельзя допускать к серьезным проектам новичков, которые окончили курсы, написали несколько программ и все. Кто не умеет писать опасный код, тот никогда не сможет написать безопасный, поскольку компилятор для него — черный ящик, и он не знает, как хакеры атакуют программы.

АНАТОЛИЙ СКОБЛОВ: Любой код — безопасный, пока им пользуется небольшая группа людей. Верно и обратное — как ни старайся, ошибки будут всегда. И чем популярней твой продукт, тем больше найдется дыр. А конкретные техники написания «более безопасного» кода общеизвестны.

АЛЕКСАНДР ЛОЗОВСКИЙ: Читать книжки, искать в интернете, общаться с умными людьми, читать наш журнал :), а потом — взять, сесть и написать!

МИХАИЛ ФЛЕНОВ: Невозможно. Везде есть ошибки, поэтому тестирование, внимательность, тестирование, внимательность, тестирование, внимательность и жесткое соблюдение основных правил могут только снизить вероятность и количество ошибок. Программы пишут люди, которым свойственно ошибаться C

SPECIAL ОБЗОР

Несомненно, у каждого возникают свои идеи по тому или иному поводу. У кого-то они более глобальные, у кого-то — более эффективные.
Как поведать о своих идеях и продемонстрировать свои достижения в кодинге?
Сегодня я расскажу о самых популярных и известных олимпиадах по спортивному программированию, а так же рассмотрю online-проекты, которые постоянно устраивают конкурсы с кругленькими суммами в качестве приза.
Читай и выбирай, что подойдет для тебя.

Юрий Наумов
(crazy_script@mail.ru)

ACM/ICPC

Крупнейшее командное соревнование по программированию — International Collegiate Programming Contest, проводящееся каждый год, начиная с 1977. Организатором этого мирового турнира является влиятельная организация Association for Computer Machinery. Совместными усилиями, на пару с известной всем компанией IBM, выступающей в последние годы в роли спонсора соревнования, ACM привлекла в этом году 5606 команд из 84 стран мира. Это самое массовое участие в ICPC за всю историю турнира. Мало того, количество участников неуклонно растет. Например, в 2004 году за звание лучших кодеров планеты боролись 3150 команд. Столь высокая популярность ICPC скорее объясняется своей красочностью и оригинальностью, что выделяет ее среди аналогичных мероприятий. Задачи на турнире на уровень сложнее обычных олимпиадных, и времени на их выполнение сравнительно немного.
А с учетом того, что, по правилам, на команду положен всего один комп, как ни крути, без понимания и слаженности далеко не уйдешь. Соревнование проходило в командной игре (3 человека в команде и 1 запасной) на 4-х видах оружия: Pascal, C, C++, Java.

В нынешнем году финал проходил 12 апреля в американском Сан-Антонио. Удостоенным участвовать в заключительной части чемпионата пришлось сначала состязаться в так называемых «отборочных играх» на районных олимпиадах, а некоторым — еще и в университетских.
Из тысяч команд в финальном турнире остались лишь 83 сильнейшие. Конечно же, не обошлось в этой компании и без наших студентов, которые, в принципе, и выиграли в 30-ой международной олимпиаде по программированию ACM/ICPC. Студенты Саратовского Государственного Университета быстрее всех справились с 5-ю задачами из 10 предложенных компьютером. Столь же выдающихся успехов удалось достичь лишь соотечественникам из Алтайского ГТУ, остальные команды больше 4-х не решили. В итоге, в десятке самых шустрых кодерских коллективов финишировали 4 команды из России: студенты из Питера и Москвы заняли соответственно 6 и 8 места.

Столь низкий процент решения задач организаторы турнира объясняют тем, что задачи сложные, а времени мало. Причем направления задач очень разнообразны: от задач с «физмат-уклоном» про подсчет соединения часового механизма часовой и минутной стрелки до более творческих, про разработку системы взаимного соединения разных узлов корпоративной сети. Да еще необходимо разработать и найти наиболее экономичный способ решения.

архивы задач с ACM ICPC

http://online-judge.uva.es/
университет Вальядолида

http://acm.timus.ru/
Уральский университет

http://acm.sgu.ru/
Саратовский университет

http://acm.pku.edu.cn/
Пекинский университет

http://www.livejournal.com/community/ru_acm/
сообщество русских участников чемпионата

Imagine Cup

Тебя никогда не посещала мысль создать свой инновационный проект в области технологий? Попытаться донести его до общественности, продемонстрировать свои успехи и объяснить его особенности? Все это ты, в принципе, можешь осуществить на конкурсе Imagine Cup, который Microsoft каждый год организует для студентов. Но все-таки нужно немного придерживаться темы проекта, оглашаемой перед каждым турниром организатором. Проводится он каждое лето, вот уже четвертый год. Конкурс организован в поддержку молодых разработчиков, готовых реализовывать свои проекты в области информационных технологий. Ведущие отечественные разработчики, такие как Diasoft, Digital Design и Лаборатория Касперского поддерживают конкурс и входят в состав жюри.

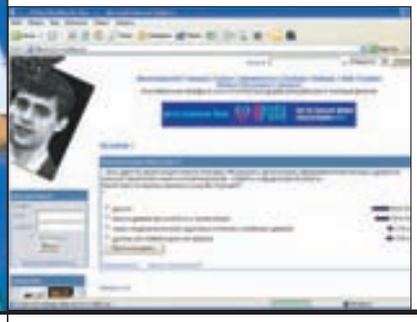
Правда, если захочешь опробовать свои силы, хочу тебя огорчить: четвертый Imagine Cup завершился 11 августа. 65 тысяч человек успели посетить «кубок» программирования, проводимый в столице Индии — Дели. Участники конкурса представляли свои достижения в категориях информационных технологий, разработки алгоритмов и приложений, проектировании интерфейсов. Соревнования проходили в два этапа: сначала региональный, затем международный. Причем в финальной части турнира приняли участие 42 команды. Победителю в состязании по программированию обещали 25 тысяч зеленых бумажек за лучшую программу. И, как ни странно, не соврали: итальянские студенты с программой о сборе и отправке информации сорвали банк. Наши в этом году остались ни с чем, зато в прошлый раз постарались: Team Inspiration представила лучший программный проект, а команда Fibra — лучшее офисное приложение. Естественно, уровень подготовки участников на таких международных мероприятиях довольно высок, но это не повод оставить свои возможно даже гениальные идеи при себе. Поэтому, если есть возможность, а главное, стремление — собирайся в Сеул следующим летом. В любом случае, получишь неоценимый опыт :).

JavaKonkurs

Этот конкурс, проводимый Sun Microsystems, призван скорее привлечь внимание к вопросам программирования на Java. Хотя для java-кодеров это отличный шанс показать себя в деле. Да и призом турнира является кругленькая сумма от организатора мероприятия. За первое место дадут 2000 зеленых, за второе и третье — 1500 и 1000 соответственно. К участию принимаются оригинальные работы, созданные на платформе Java (ME, SE, EE). Оценивать все это будет строгое жюри, в которое войдут настоящие специалисты. Так же будут учитываться мнения любого желающего. Торопись: итоги первого тура будут подведены 15 октября. Времени мало!

VR-ONLINE

Прекрасный проект для начинающих и не только кодеров. Если ты не в курсе, это проект Horrific'a, известного тебе по рубрике «Кодинг» и вообще по всему X. Он постоянно организует конкурсы в самых различных областях IT. Раньше сам проект представлял собой некий электронный журнал. Сейчас на сайте можно найти и журналы, но главное — это постоянно пополняющийся архив статей. Поэтому VR нередко проводит конкурс лучших статей на тему информационных технологий. Для кодеров здесь проводятся конкурсы по разработке ПО. Участвовать может любой человек, независимо от возраста, цвета кожи и семейного положения. На данный момент стартовал новый конкурс по программированию. А если в ходе разработки возникнут проблемы, всегда можно обратиться на форум — там всегда ответят и помогут.



Google Code Jam

Ну а если до следующего лета ждать лениво, да и проводить бесценные летние дни в Корее не особо охота, могу предложить альтернативный вариант от многоуважаемого Гугла — онлайн-соревнования в кодерском мастерстве.

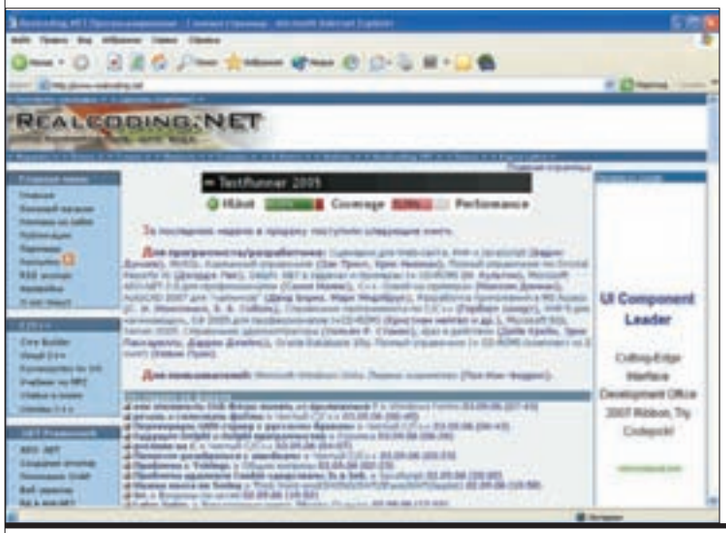
Этапы соревнования напоминают футбольную систему. Только на первом этапе тебе вообще ехать никуда не надо — онлайн-оформление на участие в «гугловской олимпиаде» началось 14 августа и продлится до 5 сентября (облом, журнал выходит в начале октября :(. — прим. Dr.). Затем сразу начинается квалификационный отбор лучших программеров в своем регионе в финал. Европе отведено 50 мест в финале Code Jam, а билет на самолет и проживание оплачивает собственно организатор — сам Google.

Финалисту на выбор предоставят два варианта: 2 задачи на час, либо 3 задачи на 1.15. Окончательные победители будут определяться с помощью гибкой системы оценки. Турнир проходит в три этапа: кодинг, просмотр решений соперников (и поиск ошибок в них) и тестирование. После этого выставляется оценка. Баллы варьируются от 150 до 1200 за одно решение.

В предыдущем году турнир собрал в общем 14,5 тысяч кодеров из 32 стран мира. Соревновались на Java, C++, C# и VB.NET. Победитель получил $25000. Первая десятка по $10000, вторая по $5000 и так далее. Добрый Гугл не оставит в обиде даже аутсайдеров финала — по $750 на пиво обеспечено :). Подробное расписание этапов и форму для регистрации ищи на сайте олимпиады.

RealCoding

Этот ресурс, думаю, тоже многим известен не понаслышке. Real'ный ресурс, содержательность которого описывать не имеет смысла. Удобный, с большим архивом статей на самые разнообразные темы программирования, с отличным форумом, где всегда удастся найти помощь. Ну а если помощь не нужна, всегда можно поучаствовать в небольших конкурсах, устраиваемых как админами, так и рядовыми форумчанами. В основном, это задачи по математике и логике. Но также есть, например, и викторины по истории компьютера и программированию. Награды соответствующие, в основном — шестизначный уин или повышение статуса.



TopCoder

Международный чемпионат для всех желающих. Здесь не имеет значения, студент ты или нет. TopCoder Open для состязания в категориях спортивного программирования и разработчиков ПО собрал 4500 участников. Это мероприятие можно назвать неофициальным чемпионатом мира, и уровень подготовки участников на высоте. В этом году в Лас-Вегасе список «топовых кодеров» возглавил наш российский программист.

IOI

IO Informatics — самая глобальная международная олимпиада. 18-ая IOI прошла в августе в мексиканском городе Мерида, где соревновались представители из 85 стран мира. Ввиду технических неполадок каждый тур задерживали, зато после первого дня соревнования шел разгрузочный день — посещение местного пляжа. Это очень кстати, учитывая тот факт, что мозгу все же нужен отдых. На других соревнованиях таких прелестей не предусматривается. Все в основном проходит в сжатые сроки с экономией времени. После второго дня усиленной мозговой деятельности организаторы предоставили участникам возможность побывать в живописных местах за городом. Там же и было сделано групповое фото IOI2006. И лишь на следующий день состоялось награждение.

Тем не менее, отличные условия проживания и прекрасная культурная программа переплеталась с явными техническими недоработками и несбалансированным уровнем сложности задач, который был либо низким, либо очень высоким. В общем, обо всем, что происходило на олимпиаде, можно узнать на официальном сайте ioinformatics.org.

TBT

Не такое популярное мероприятие, как IC или GCJ, но зато оно доступно практически любому желающему. Дело в том, что уровень подготовки может быть совершенно разный. Задания имеют 10 категорий сложности, от самых простых до олимпийской сложности. Если навыки программирования у тебя не особо высокие, можно без проблем подобрать уровень под себя. Здесь не имеет значения, кто ты и откуда. В этих соревнованиях участвуют все.

В течение года Test-the-Best.by проводит 2 чемпионата, 5 соревнований из серии BrainStrike (для высокого уровня), а также кубок TBT — крупнейшее игровое событие. Из них только лишь финалы соревнований проводятся offline в разных городах.

SPECIAL МНЕНИЕ

АЛЕКСАНДР ЛОЗОВСКИЙ

Выпускающий редактор СПЕЦ'а. Редактор рубрики «Coding» ХАКЕР'а

Можно сказать, что свою карьеру в X и СПЕЦе я начал именно с участия в программерском конкурсе. В те далекие времена (что-то около 1999 года) я программил на Паскале и Дельфи, интересовался всяким недобрым софтом, и однажды набрел на ресурс x-c-r.com (так тогда назывался сайт Михаила Фленова aka Horrific'a). Там я нашел конкурс статей и решил черкнуть туда матерьяльчик. Статья эта ему понравилась, и с тех пор, после небольшого периода подработки на сайте Хакера, я перешел в «Кодинг» этого журнала, а потом — и в СПЕЦ. Это я к тому, что никогда не стоит стесняться участвовать в конкурсах и писать статьи — всегда есть определенный шанс, что тебя кто-то возьмет на заметку. В хорошем смысле этого слова.

g6prog.narod.ru

Достичь успеха в каком-либо соревновании без тренировок просто невозможно, будь то футбол, покер или кодинг. Отличный ресурс судента МГУ, цель которого — как раз помочь в подготовке к глобальным состязаниям — g6prog.narod.ru. На сайте подробно разобраны задачи с самых разномасштабных олимпиад: от школьных до международных. Все это сопровождается очень даже неплохой собственной «библиотекой» ресурса с топовыми книгами кодерского мастерства, а так же статьями автора. Задач пока не очень много (чуть более сотни), но с постоянными обновлениями архив может вырасти довольно быстро. Ну а если уж ждать совсем нет времени — советую глянуть в раздел с неплохой подборкой ссылок по олимпиадным задачам.

SPECIAL FAQ

На вопросы отвечает друг и защитник мышей Крис Касперски ака мыщъх

Q КАКИЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ СЛЕДУЕТ ИЗУЧАТЬ В ПЕРВУЮ ОЧЕРЕДЬ?

A Важен не сам язык, а мысли, которые этим языком выражают. Практически любую алгоритмическую задачу можно решить на любом языке (за какое время и с какой эффективностью — это уже другой вопрос), поэтому неважно, с какого языка начинать, — все равно за время его изучения он успеет устареть. И следует выбирать тот язык, для изучения которого есть хорошие учебники по программированию, а так же знакомые специалисты, способные проконсультировать и помочь, если вдруг что-то пойдет не так, ведь язык — это не только способ записи алгоритма, но еще и средство общения! В этом качестве языку Си нет равных, и de facto он стал международным стандартом типа английского. Знать его нужно не только затем, чтобы на нем программировать, но и чтобы понимать листинги, приведенные в книгах, посвященных сетевым протоколам или устройству осей. Visual Basic — de facto стандарт в области макроязыков на платформе Windows, и без его знания невозможно эффективно работать с Microsoft Visual Studio. Но Basic совершенно чужд миру UNIX, где ведущую роль играют Perl, AWK и дру-

гие скриптовые языки. Если человек может решить свою задачу на каком-либо языке, переход на другой язык для него не будет проблемой.

СТОИТ ЛИ ХВАТАТЬСЯ ЗА НОВЫЕ ТЕХНОЛОГИИ ТИПА .NET ИЛИ ПОСОВЕТУЕШЬ ДЕРЖАТЬСЯ СТАРЫХ?

Отечественная система образования до безобразия консервативна и крайне неохотно реагирует на новые технологические веяния и прорывы, поэтому в 99% случаев выпускник ВУЗа к реальной работе не готов и ему следует еще учиться и учиться. А все потому, что у нас традиционно учили фундаментальным основам, а вот американцы, напротив, делают упор на конкретное практическое применение. Девушки, окончившие 2-недельные курсы по Visual Basic'у, составляют определенную конкуренцию специалистам, знающим Си++, потому что они в курсе того, какие есть библиотеки и как ими пользоваться, но создать их самостоятельно не в состоянии. Все, что они могут — это сложить готовые компоненты воедино (а другого зачастую и не требуется). Алгоритмически-ориентированный программист готов запрограммировать что угодно, но… он совершенно не в курсе, какие существуют библиотеки, и не умеет с ними работать. Поэтому при решении типовых задач наиболее конкурентоспособным оказывается программист, идущий в ногу с прогрессом и осваивающий новые библиотеки и framework'и по мере их появления, а вот при решении нетипичных задач программисты, знающие фундаментальные основы, получают огромное преимущество. Некоторые ухитряются совмещать оба качества, но это удается лишь немногим.

НАСКОЛЬКО ВАЖНО ЗНАТЬ АССЕМБЛЕР?

Несмотря на то, что Ассемблер сдает свои позиции, профессиональному программисту знать его необходимо, хотя бы затем, чтобы разбирать аварийные дампы и правильно интерпретировать сообщения о критических ошибках. Не говоря уже о том, что создание эффективного кода без знания архитектуры процессора (и всего компьютера в целом) — невозможно. Знание Ассемблера позволяет заглянуть внутрь откомпилированной программы и понять, почему она ведет себя не так, как этого хочется тебе. Операционная система перестает быть черным ящиком, а отсутствие исходных текстов Windows уже не становится преградой в археологических раскопках ее недр. Аргументы в пользу Ассемблера можно перечислять бесконечно, и как бы современные языки ни абстрагировались от железа, в жизни каждого программиста периодически возникает жгучая необходимость написать пару ассемблерных строк или понять, что означают уже написанные.

КАК УСТРОИТЬСЯ НА ХОРОШУЮ РАБОТУ В РОССИИ И ЗА РУБЕЖОМ?

Из двух работ лучшей будет та, которая больше нравится тебе. Отсюда следует, что работу нужно искать в соответствии со своими предпочтениями, при этом не пытаясь навязывать эти предпочтения другим. Если на такой-то фирме используется преимущественно DELPHI, глупо пытаться втиснуться туда, зная один лишь Си или Ассемблер. Знания чего бы то ни было при трудоустройстве вообще вторичны. Первично умение себя подать. Матчасть здесь отдыхает, a body language очень рулит. Основная ошибка начинающих — перечисление в резюме огромного перечня языков, сред программирования, операционных систем и библиотек, с которыми они как бы умеют работать. А работодателю на фиг не нужен программист-универсал, по чуть-чуть нахватавшийся всего. Ему нужен как раз человек, знающий свою узкую предметную область, но знающий ее глубоко. К тому же, не так важно, сколько программ ты написал, — важнее, сколько из них ты поддерживаешь в настоящий момент.

ПОЧЕМУ БОЛЬШИНСТВО ПРОГРАММИСТСКИХ ПРОЕКТОВ ПРОВАЛИВАЮТСЯ?

В основном это происходит из-за чрезмерного оптимизма и незнания рыночной ситуации. Никогда не стоит исходить из положительных предпосылок. Следует сразу подготовить себя к тому, что проект не будет закончен в намеченный срок, достигнуть стабильной работы программы не удастся, пользователи не будут платить, продукт не будет востребован, и он не будет превосходить имеющиеся на рынке аналоги :). Сумеешь ли ты вести бизнес в таких условиях?! Да, сумеешь, если не станешь сразу замахиваться на большое, а начнешь с малого. Сумеешь, если засунешь идею создать массовый продукт глубоко под хвост и сконцентрируешься на удовлетворении нужд небольшой группы пользователей, игнорируемой компаниями-гигантами. Сумеешь, если вместо одного большого шага будешь делать сто маленьких шажков.

РЕФРАКТОРИНГ — БУЗВОРД ИЛИ СЕРЕБРЯНАЯ ПУЛЯ?

Существует мнение, что если код не поддерживает автоматическое тестирование, то это отстой, и что перманентный рефракторинг — залог успеха. На самом же деле, сложность тестирующих модулей зачастую в разы превосходит сложность тестируемого ими кода, к тому же тестирующие модули нужно тоже как-то тестировать, а это уже бесконечная рекурсия получается. Существует определенная граница сложности, ниже которой автоматическое тестирование кода уже экономически неоправданно. А системы, взаимодействующие с внешней средой, могут быть протестированы в автоматическом режиме только если мы создадим программный эмулятор этой самой среды, который, в свою очередь, тоже будет нуждаться в тестировании. Как разорвать этот замкнутый круг? Разбивать код на множество мелких модулей, каждый из которых будет настолько прост, насколько это вообще возможно. Тестирование модулей при этом значительно упрощается, но резко возрастает количество связей между модулями и, как следствие, появляются ошибки в их взаимодействии. Более того, модульная структура сама по себе гораздо более сложная, чем монолитная. Так что выводы неутешительны. Серебряных пуль нет, качество программного обеспечения по-прежнему остается низким, и радикальных прорывов в этой области не наблюдается. Это не значит, что рефракторинг бесполезен, это значит, что он должен применяться с умом.

КАК ЗАЩИТИТЬ СВОЙ КОД ОТ ХАКЕРОВ?

Хакеры ломают все, что трассируется (а что не трассируется, то дизассемблируется), и надеяться, что выбранная система остановит их, может только идеалист. Хакеры существуют — это факт. И развивать свой бизнес, игнорируя кряки, все равно что вести строительство в Сибири из расчета, что лето никогда не закончится и температура всегда будет держаться выше нуля. Если твоя программа — продукт, ты — «покойник». Если же твоя программа — услуга, хакеры идут лесом, поскольку «взломать» услугу нельзя. Следовательно, нужно развивать онлайновые службы, организовывать обучающие семинары, ориентироваться на комплексные решения (где сама программа — всего лишь часть большой бизнес-машины, то есть сама по себе лишена смысла). Допустим, ты написал такую простейшую штуку как каталогизатор аудиодисков. А теперь прикрути к нему онлайновую службу, позволяющую пользователям обмениваться своими базами данных (если один человек «вбил» описание дисков в базу, зачем тысячам других делать то же самое), затем начинай привлекать лейблы (или просто магазины, торгующие дисками), предоставив им возможность информировать пользователей о новинках и вести мониторинг реальной популярности своей продукции. В этом случае основным источником прибыли окажутся именно лейблы, а прибыль будет тем выше, чем больше у тебя пользователей. Для этого программа должна распространяться бесплатно. Но даже если она платная, то взломать ее все равно очень и очень сложно, поскольку основную ценность составит онлайновая база данных, доступ к которой контролировать гораздо легче… C

SPECIAL ОБЗОР

MEDIUM

Delphi. Разработка баз данных

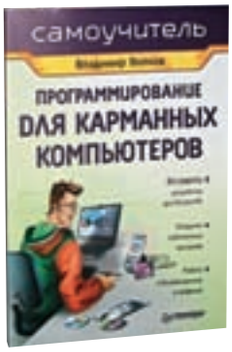
СПб.: Питер, 2005 / Сорокин А.В. / 477 страниц
Разумная цена: 186 рублей

Практически любая современная организация нуждается в базе данных, чтобы хранить там свои «пожитки». Так что тема весьма актуальна для программистов. Одна беда — ощущается недостаток литературы для начинающих по данной теме. Задача этой книги — максимально просто объяснить, как разрабатывать распределенные СУБД, излагая максимум сопутствующего материала. Delphi предоставляют разработчику поистине великолепный набор простых в использовании инструментов, позволяющих быстро создавать сложные проекты с приятным и удобным пользовательским интерфейсом. Причем в Delphi просто работать с любым современным сервером баз данных, для которого есть соответствующий драйвер.

EASY

Программирование для карманных компьютеров. Самоучитель

СПб.: Питер, 2006 / Волков В.Б. / 304 страницы
Разумная цена: 186 рублей



Для тех, кто начинает программировать для Pocket PC. Выполнив приведенные в книге упражнения, ты сможешь быстро создавать достаточно сложные приложения для своего наладонника. Понятно, что ты не узнаешь о программировании для Pocket PC все, но очень быстро получишь достаточно большой и разнообразный материал, который и будет отправной точкой. А дальше будешь самостоятельно набираться опыта. Огромное количество примеров позволит эффективно совершенствовать навыки программирования и обогащать свои знания. Причем книжка рассчитана на начинающих программистов. Рынок программ для карманных компьютеров растет семимильными шагами, так что, кто знает, может быть, ты займешь свое место в этом мире.

Как мы отбираем книги в обзор? Берем список имеющихся на складе книг (несколько тысяч наименований). Из них делаем выборку по теме номера. Лучшее попадает в журнал.
Если тебя заинтересовали описанные книги, можешь заказать их по разумным ценам в букинистическом интернет-магазине «OS-книга» (*www.osbook.ru*), либо по адресу oskniga@mail.ru

HARD

Программирование искусственного интеллекта в приложениях

М.: ДМК Пресс, 2006 / Джонс М.Т. / 312 страниц
Разумная цена: 211 рублей

Ранние разработки искусственного интеллекта копировали поведение человека (сильные ИИ), но со временем от них отказались в пользу более практичных алгоритмов и технологий ИИ, встраиваемых в программное обеспечение (слабые ИИ), дабы сделать ПО более умным и полезным. То есть в пользу программ, гибко подстраивающихся под требования и привычки пользователя. В книге рассмотрены некоторые алгоритмы ИИ и их принцип работы. Среди них: нейронные сети, генетические алгоритмы, системы, основанные на продукционных правилах, нечеткая логика, алгоритмы муравья и умные агенты. Для каждого алгоритма есть наглядные примеры приложений. Правда, только некоторые из приложений полезны на практике, остальные же относятся скорее к теоретическим изысканиям, но это не делает их менее интересными.

EASY

Как пасти котов. Наставления для программистов, руководящих другими программистами

СПб.: Питер, 2006 / Рейнвотер Дж. / 256 страниц
Разумная цена: 265 рублей

О лидерстве и руководстве — как первое совмещать со вторым. По сути, это сборник трудных случаев управления IT-проектами. Вне зависимости от возраста, пола и социального статуса, книга поможет укрепить позиции в роли лидера в команде программистов. Унаследовав навыки менеджера, ты должен будешь перегрузить типичные параметры мышления новыми типами и значениями. То есть, грубо говоря, испытать полиморфизм собственного характера. За счет этого ты инкапсулируешь в своем программистском мозгу совершенно новый вид искусства — искусство лидерства и руководства. Своевременность разработок и качество программных продуктов теперь в твоих руках :).

EASY

Программирование: ступени успешной карьеры

СПб.: БХВ-Петербург, 2006 / Кузнецов М.В. / 320 страниц
Разумная цена: 160 рублей

Книга о том, что необходимо для успешной карьеры в области информационных технологий. То есть она не о кодировании, а о тех вопросах, на которые многие начинающие (и не только) программисты не обращают должного внимания: как устроиться на работу, как вести переговоры с клиентом, как грамотно работать в команде и т.п. По опыту, большинство программистских карьер рушится не из-за того, что программист не умел писать код, а по той причине, что авторы гениальных программ как раз не уделяли внимания «второстепенным» вопросам. А при нынешней конкуренции на рынке программных продуктов не обращать внимания на подобные «мелочи» — смерти подобно.

HARD

Искусство программирования на Ассемблере, 3-е изд.

СПб.: ООО «ДиаСофтЮП», Питер, 2006 / Голубь Н.Г. / 820 страниц
Разумная цена: 434 рубля

Описание элементов языка Ассемблера процессоров Intel x86: системы счисления, внутреннее представление данных и команд, основы 16- и 32-разрядного программирования, программирование процессора, ввод-вывод информации в DOS и Windows, использование макросредств, потоковых мультимедийных MMX- и XMM-команд. Подробно, шаг за шагом, на многочисленных примерах законченных программ рассмотрены идеи и принципы организации вычислений на Ассемблере, с использованием аналогии со стороны алгоритмических языков Pascal или C/C++. Книга рассчитана на начинающих программистов и тех, кто хочет глубже разобраться в особенностях организации и функционирования ЭВМ.

ПОДПИСКА В РЕДАКЦИИ

С 1 ОКТЯБРЯ ПО 31 ДЕКАБРЯ ПРОВОДИТСЯ
СПЕЦИАЛЬНАЯ АКЦИЯ ДЛЯ ЧИТАТЕЛЕЙ ЖУРНАЛА

СПЕЦ

ГОДОВАЯ ПОДПИСКА ПО ЦЕНЕ 11 НОМЕРОВ!

2040 ~~руб.~~ ⇨ **1870 руб.**

ПЛЮС ПОДАРОК

ОДИН ЖУРНАЛ
ДРУГОЙ ТЕМАТИКИ

ОФОРМИВ ГОДОВУЮ ПОДПИСКУ В РЕДАКЦИИ, ВЫ МОЖЕТЕ БЕСПЛАТНО ПОЛУЧИТЬ ОДИН СВЕЖИЙ НОМЕР ЛЮБОГО ЖУРНАЛА, ИЗДАВАЕМОГО КОМПАНИЕЙ «ГЕЙМ ЛЭНД»:

- ЯНВАРСКИЙ НОМЕР – ПОДПИСАВШИСЬ ДО 30 НОЯБРЯ,
- ФЕВРАЛЬСКИЙ НОМЕР – ПОДПИСАВШИСЬ ДО 31 ДЕКАБРЯ.

ВПИШИТЕ В КУПОН НАЗВАНИЕ ВЫБРАННОГО ВАМИ ЖУРНАЛА, ЧТОБЫ ЗАКАЗАТЬ ПОДАРОЧНЫЙ НОМЕР.

И ЭТО НЕ ВСЕ!

31 ДЕКАБРЯ СРЕДИ ЧИТАТЕЛЕЙ,
ОФОРМИВШИХ ПОДПИСКУ НА ВЕСЬ 2007 ГОД,
БУДЕТ РАЗЫГРАНО 200 MP3-ПЛЕЕРОВ QUMO X

ВЫГОДА ■ ГАРАНТИЯ ■ СЕРВИС

КАК ОФОРМИТЬ ЗАКАЗ

1. Разборчиво заполните подписной купон и квитанцию, вырезав их из журнала, сделав ксерокопию или распечатав с сайта www.xakep.ru.
2. Оплатите подписку через Сбербанк .
3. Вышлите в редакцию копию подписных документов — купона и квитанции — любым из нижеперечисленных способов:

- по электронной почте **subscribe@glc.ru;**
- по факсу **8 (495) 780-88-24;**
- по адресу **119992, Москва, ул. Тимура Фрунзе, д. 11, стр. 44-45, ООО «Гейм Лэнд», отдел подписки.**

ВНИМАНИЕ!

Подписка оформляется в день обработки купона и квитанции в редакции:

- в течение пяти рабочих дней после отправки подписных документов в редакцию по факсу или электронной почте;
- в течение 20 рабочих дней после отправки подписных документов по почтовому адресу редакции.

Рекомендуем использовать факс или электронную почту, в последнем случае предварительно отсканировав или сфотографировав документы.
Подписка оформляется с номера, выходящего через один календарный месяц после оплаты. Например, если вы производите оплату в ноябре, то журнал будете получать с января.

Подписка на журнал «Хакер Спец» **на 6 месяцев стоит 1020 руб.**
Подарочные журналы при этом не высылаются

ПО ВСЕМ ВОПРОСАМ, СВЯЗАННЫМ С ПОДПИСКОЙ, ЗВОНИТЕ ПО БЕСПЛАТНЫМ ТЕЛЕФОНАМ **8(495)780-88-29** (ДЛЯ МОСКВИЧЕЙ) И **8(800)200-3-999** (ДЛЯ ЖИТЕЛЕЙ ДРУГИХ РЕГИОНОВ РОССИИ, АБОНЕНТОВ СЕТЕЙ МТС, БИЛАЙН И МЕГАФОН). ВОПРОСЫ О ПОДПИСКЕ МОЖНО ТАКЖЕ НАПРАВЛЯТЬ ПО АДРЕСУ **INFO@GLC.RU** ИЛИ ПРОЯСНИТЬ НА САЙТЕ **WWW.XAKEP.RU**

ПОДПИСНОЙ КУПОН

ПРОШУ ОФОРМИТЬ ПОДПИСКУ
НА ЖУРНАЛ «ХАКЕР СПЕЦ»

☐ на 6 месяцев
☐ на 12 месяцев
начиная с _____ 200__г.

Прошу выслать бесплатный номер журнала
_____

☐ Доставлять журнал по почте на домашний адрес
☐ Доставлять журнал курьером на адрес офиса (по г. Москве)
Подробнее о курьерской доставке читайте ниже*

(отметьте квадрат выбранного варианта подписки)

Ф.И.О. _____

дата рожд. ☐☐. ☐☐. ☐☐ г.
(день. месяц. год)

АДРЕС ДОСТАВКИ:

индекс _____
область/край _____
город _____
улица _____
дом _____ корпус _____
квартира/офис _____
телефон (код) _____
e-mail _____
сумма оплаты _____

* Курьерская доставка осуществляется только по Москве на адрес офиса. Для оформления доставки курьером укажите адрес и название фирмы в подписном купоне.

Извещение

ИНН 7729410015 ООО «Гейм Лэнд»
ЗАО ММБ
р/с № 40702810700010298407
к/с № 30101810300000000545
БИК 044525545 КПП - 772901001
Плательщик
Адрес (с индексом)

Назначение платежа	Сумма
Оплата журнала «СПЕЦ» с _____ (месяц) 200__г.	

Ф.И.О.
Подпись плательщика

Кассир

Квитанция

ИНН 7729410015 ООО «Гейм Лэнд»
ЗАО ММБ
р/с № 40702810700010298407
к/с № 30101810300000000545
БИК 044525545 КПП - 772901001
Плательщик
Адрес (с индексом)

Назначение платежа	Сумма
Оплата журнала «СПЕЦ» с _____ (месяц) 200__г.	

Ф.И.О.
Подпись плательщика

Кассир

hard

материнское сердце

МАТЕРИНСКИЕ ПЛАТЫ ПОД SOCKET 754

ЕВГЕНИЙ ПОПОВ

тестовый стенд:

ПРОЦЕССОР: **AMD Sempron 64 3000+**
ПАМЯТЬ: **2x512 Мб, Corsair Value Select**
ВИДЕОПЛАТА: **ASUS EN6600GT, 256 Мб**
ВИНЧЕСТЕР: **Seagate Barracuda 7200.8 ST3400832AS, 400 GB, 7200 RPM, SATA**
ОПТИЧЕСКИЙ ПРИВОД: **SONY CRX300E, CD-RW/DVD**
БЛОК ПИТАНИЯ: **1350 Вт**

Ни для кого не секрет, что наиболее востребованными платформами под процессоры AMD на сегодняшний день являются материнские платы с разъемами Socket 754 и Socket 939. Если первая выигрывает в основном с помощью ценового показателя, то вторая — продвинутыми возможностями. В данном обзоре мы постараемся выяснить, насколько оправданным является решение на основе 754-ого сокета и будет ли оно в скором времени совсем вытеснено привлекательным 939-м. Ну и на сладкое нам предстоит чудесное препарирование нескольких интересных моделей на основе все того же разъема 754.

➔ **задача выбора.** На сегодняшний день платформы под 754-контактные процессоры позиционируются как продукты Low-End сектора, и самые пессимистичные критики пророчат им скорую отставку, а также перманентный уход в страну забвения. Напротив, платформы из разряда Socket-939 жестоко раскручиваются маркетологами компании AMD и, по заверениям представителей, будут еще долго и продуктивно существовать. Но не стоит бросаться в омут и вешать ярлыки типа «754 — для бедных». Не так уж он и плох, на самом деле. Даже процессоры, в принципе, в том и другом случае используются похожие, но распаянные под разные сокеты. Все ограничения обозначены маркетинговыми играми. Другой вопрос в различных контроллерах памяти. У процессоров на Socket 754 контроллер памяти одноканальный, а у 939-камней, соответственно, двухканальный — именно на обеспечение данного факта и идут дополнительные ножки в разъеме.

Недостаток одноканальности более-менее подкованный пользователь может восполнить с помощью разгона, если уж так хочется, чтобы попугаев в 3DMark было «столько же, сколько у соседа». Если говорить о частоте шины «Hyper Transport», то для 754-платформ она равна 800 МГц против 1000 МГц на платах под Socket 939. На практике это отличие не играет большой роли, и разница в производительности не столь велика. Между тем, в случае Socket 754 может легко повториться схожая история, какая была с процессорами Barton и Thorton. Тогда процессоры с неработоспособной частью кэша выходили под маркировкой Thorton и, между прочим, довольно успешно продавались, благодаря соблазнительной цене. Так что на платформу 754 компания AMD, например, потенциально может переводить процессоры с одним неработоспособным каналом из двух, или неполной по частоте шиной HyperTransport.

Платы на основе сокета 754 по карману самому скромному в финансовом отношении пользователю, а уж для офисных закупок это настоящая золотая жила. Плюс ко всему только недавно появилась на рынке бюджетная линейка камешков AMD Sempron и сразу же очень хорошо показала себя в тестах. Так что рано еще хоронить — поживет S754, будь уверен.

➔ **методика тестирования.** Ну, раз мы возвели плаформу 754 в статус Ленина, то бишь живее всех живых, значит наступает время художественного экспериментирования. На операционном столе расположились самые достойные представители данного направления, и вот какой набор истязаний мы для них приготовили. В первую очередь, с помощью синтетических бенчмарков мы замеряли общую производительность, после чего в ход шел набор бытовых программ, а именно Lame, WinRar и Adobe Photoshop. Не забыли мы и о кодировании видео в XVID и DIVX. Количество FPS замерялось в игровых приложениях F.E.A.R и Half Life 2. Качество аудио оценивалось с помощью RMAA 5.5. Оценки выставлялись не только по принципу «кто сильнее — тот и прав». Во внимание принимались возможности BIOS'a — разгон и настройка, доступность к элементам платы, уровень охлаждения и комплектация.

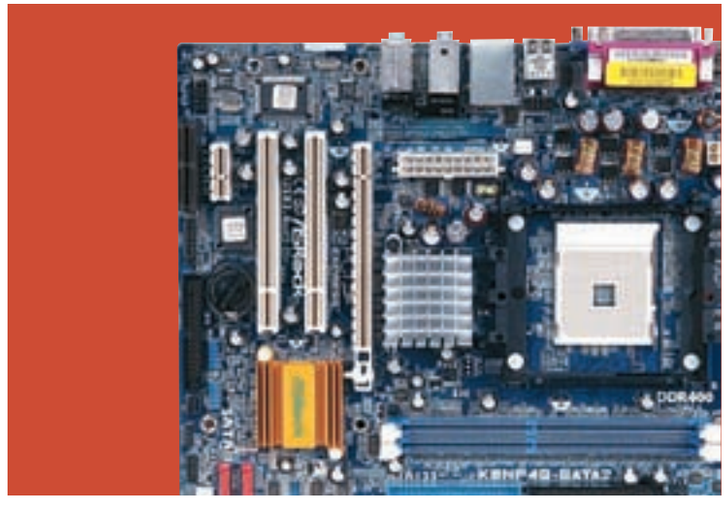
ASROCK K8NF4G-SATA2

($60) 7 звезд

ЧИПСЕТ: **NVIDIA GeForce 6100 (NB) + NVIDIA nForce 410 MCP (SB)**
ПАМЯТЬ: **2xDDR DIMM 400/333/266 (2 Гб max)**
СЛОТЫ РАСШИР.: **1xPCI-Ex16, 1xPCI-Ex1, 2xPCI, 1xAMR**
SATA: **2xSATA II**
FIREWIRE: **нет**
LAN: **Realtek PHY RTL8201CL**
АУДИО: **Realtek ALC850 7.1, 7-канальный AC'97**

➔ **плюсы.** AsRock K8NF4G-SATA2 основан на двухчиповой системной логике, где за графику отвечает северный мост NVIDIA GeForce 6100, а все прочие контроллеры функционируют с помощью NVIDIA nForce 410 MCP. На данных мостах установлены два небольших алюминиевых радиатора. В принципе, вентилятора там и не требуется, поскольку обе части чипсета греются не столь сильно. Производители не забыли о прогрессе — имеется два порта для подключения SATA-устройств. В BIOS'e платы есть приличные средства для разгона и настройки — CPU Frequency Stepless Control, ASRock U-COP и B.F.G. (Boot Falure Guard). Имеется поддержка камней AMD Sempron.

➔ **минусы.** Размеры платы весьма компактны (244x203 мм), поэтому доступ к планкам памяти затруднен, да и комплектация скудна.



CHAINTECH VNF3-250
($63) 7 звезд

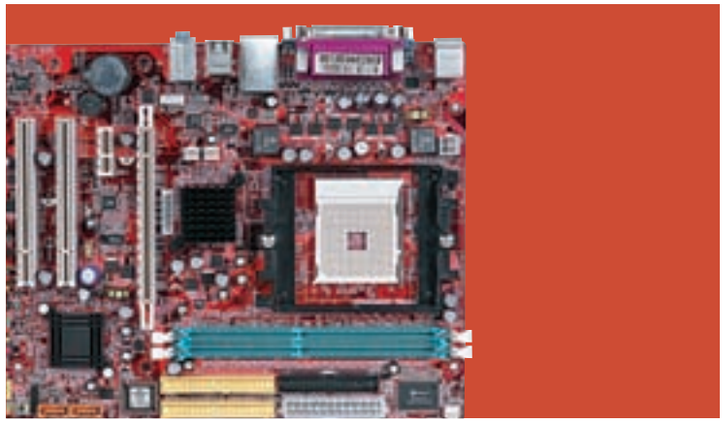
ЧИПСЕТ: **NVIDIA nForce3 250**
ПАМЯТЬ: **3xDDR DIMM 400/333/266 (2 Гб max)**
СЛОТЫ РАСШИР.: **1xAGP, 5xPCI, 1xCMR**
SATA: **2xSATA**
FIREWIRE: **нет**
LAN: **Realtek RTL8100C**
АУДИО: **Chaintech Multimedia Card 5.1, 5-канальный AC'97**

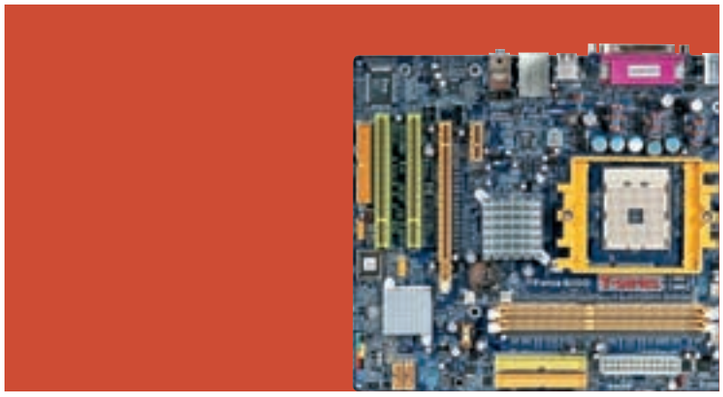
→ **плюсы.** Неплохое предложение от Chaintech. Построена плата на основе добротного чипсета третьего поколения от NVIDIA, и поддерживает процессоры AMD Sempron и SATA-диски. Аудиочип помещается на отдельной карте, которая подключается через специальный порт CMR. Наверняка в этом есть какой-то глубокий смысл. С помощью утилиты DigiDoc становится возможным контроль температурного режима, а также регулировка частот и напряжений на компонентах платы. Настоящее раздолье ожидает любителей PCI-устройств — на плате расположено целых пять портов PCI. Просто праздник какой-то!
→ **минусы.** К сожалению, о поддержке PCI Express речь не идет — пользователям придется довольствоваться портом AGP. Странно выглядят три DIMM-слота.

MSI K8NGM-V
($61) 7 звезд

ЧИПСЕТ: **NVIDIA GeForce 6100 (NB) + NVIDIA nForce 410 MCP (SB)**
ПАМЯТЬ: **2xDDR DIMM 400/333/266 (2 Гб max)**
СЛОТЫ РАСШИР.: **1xPCI-Ex16, 1xPCI-Ex1, 2xPCI**
SATA: **2xSATA II**
FIREWIRE:**нет**
LAN: **Realtek® 8201CL PHY**
АУДИО: **Realtek ALC655, 6-канальный AC'97**

→ **плюсы.** В MSI K8NGM-V использован чипсет с двухкомпонентной структурой. Пользователь может ограничиться встроенной графикой, однако при желании всегда можно задействовать свободный порт PCI ExpressX16. Само собой, имеется поддержка как Athlon 64 камней, так и Sempron. Южный же мост поддерживает построение массивов RAID 0 и RAID 1. Набор для настройки через BIOS более-менее стандартен — регулировка частот, таймингов памяти и напряжений.
→ **минусы.** При тестировании были сложности с установкой опции Cool'n'Quiet. Проблема решается только перепрошивкой BIOS. Немного расстраивает отсутствие FireWire и слабая комплектация.

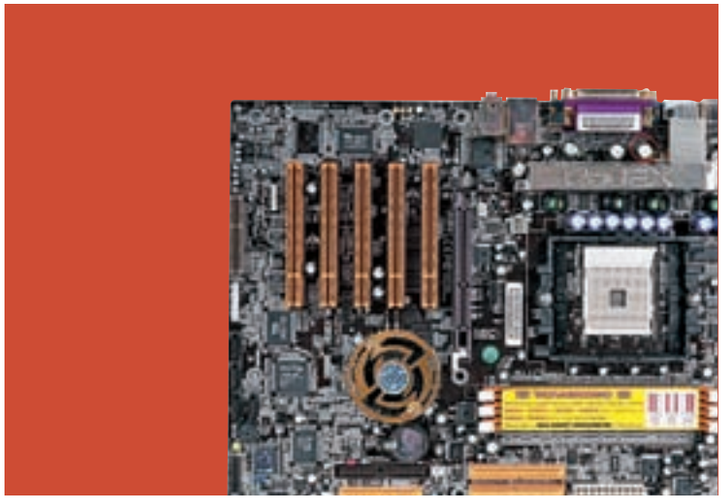




BIOSTAR TFORCE 6100
($72) 8 звезд

ЧИПСЕТ: **NVIDIA GeForce 6100 (NB) + NVIDIA nForce 410 MCP (SB)**
ПАМЯТЬ: **2xDDR DIMM 400/333/266 (2 Гб max)**
СЛОТЫ РАСШИР.: **1xPCI-Ex16, 1xPCI-Ex1, 2xPCI**
SATA: **2xSATA II**
FIREWIRE: **нет**
LAN: **Realtek PHY RTL8201CL**
АУДИО: **Realtek ALC655, 6-канальный AC'97**

→ **плюсы.**Яркие и кислотные цвета в дизайне платы. Охлаждением чипсета занимаются алюминиевые ребристые радиаторы. Плата не только подойдет на роль офисного решения (благодаря встроенному видео), но и удовлетворит потребности требовательных энтузиастов. Дело в том, что Biostar TForce 6100 обладает уникальным overclocking-движком, к которому имеется даже отдельный мануал.
→ **минусы.** Из минусов можно отметить только отсутствие в стандартной прошивке поддержки процессоров AMD Sempron. В остальном поведение платы нареканий не вызвало.



CHAINTECH ZNF3-250
($165) 7 звезд

ЧИПСЕТ: **NVIDIA nForce3 250**
ПАМЯТЬ: **3xDDR DIMM 400/333/266 (2 Гб max)**
СЛОТЫ РАСШИР.: **11xAGP, 5xPCI, 1xCMR**
SATA: **4xSATA**
FIREWIRE: **VIA VT6306**
LAN: **Realtek RTL8100C**
АУДИО: **Chaintech Multimedia Card 7.1, 7-канальный AC'97**

→ **плюсы.** Еще один вариант Chaintech VNF3-250, претерпевший только поверхностные изменения. Во-первых, комплектация — такого обилия от Chaintech мы не ожидали. Продумано все до мелочей: шнуры в оплетках, специальная отвертка, термопаста, панель SPDIF-Out, пакет программного обеспечения, а также фронтальная внешняя панель с кард-ридером под пятидюймовый отсек. Во-вторых, охлаждением занимаются два радиатора — один на чипсете и второй, уникальный, под названием RadeX — на блоке MOSFET. Звуковая плата теперь поддерживает семиканальность, а SATA-портов стало четыре. И, наконец, это первое устройство в обзоре с поддержкой FireWire.
→ **минусы.** В принципе, минусов у данной платы практически нет, но хотелось бы отметить неприятный звук, издаваемый вентилем на радиаторе RadeX.

GIGABYTE GA-K8NE
($65) 7 звезд

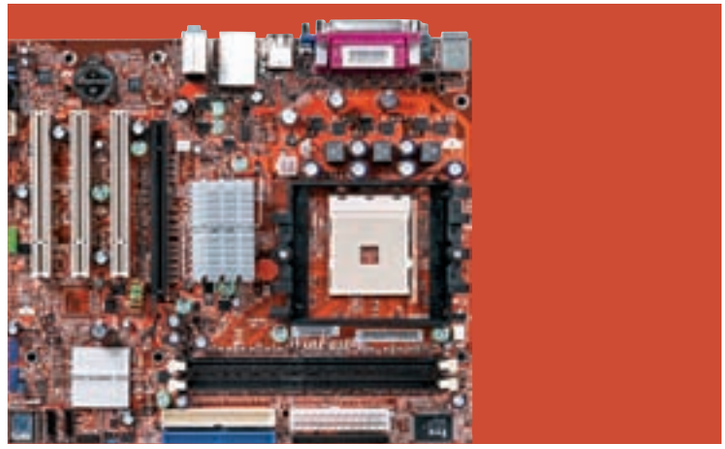
ЧИПСЕТ: **NVIDIA nForce4-4X**
ПАМЯТЬ: **3xDDR DIMM 400/333/266 (2 Гб max)**
СЛОТЫ РАСШИРЕНИЯ: **1xPCI-Ex16, 2xPCI-Ex1, 3xPCI**
SATA: **4xSATA**
FIREWIRE: **Нет**
LAN: **Marvell 88E1111**
АУДИО: **RealTek ALC850, 8-канальный AC'97**

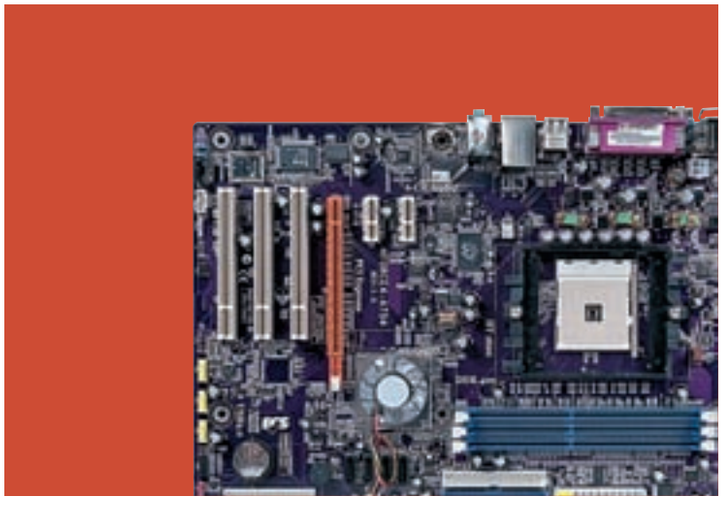
→ **плюсы.** На традиционно синем текстолите под широким радиатором расположился довольно мощный чипсет от NVIDIA. Плата работает с поддержкой процессоров как Athlon 64, так и Sempron. Интегрированный RAID-контроллер встроен в чипсет, и позволяет строить RAID массивы уровней RAID 0, 1, 0+1 из устройств Serial ATA и IDE. Несмотря на бюджетный уровень платы, на ней установлен неплохой восьмиканальный кодек AC 97 от RealTek. Из специальных технологий отметим NVIDIA Firewall и Gigabyte DualBIOS.
→ **минусы.** Поддержка FireWire отсутствует. В коробке только самое необходимое, нет даже дополнительной пары SATA-шнуров.

FOXCONN WINFAST 6100K8MB
($67) 7 звезд

ЧИПСЕТ: **Northbridge: NVIDIA GeForce 6100 + Southbridge: NVIDIA nForce 410 MCP**
ПАМЯТЬ: **2xDDR DIMM 400/333/266 (2 Гб max)**
СЛОТЫ РАСШИРЕНИЯ: **1xPCIEx16, 3xPCI**
SATA:**2xSATA II**
FIREWIRE: **нет**
LAN: **Realtek PHY RTL8201BL**
АУДИО: **Realtek ALC653 6.1, 6-канальный AC'97**

→ **плюсы.** Плата выполнена на текстолите оранжевого цвета размера 244 на 216 миллиметров. Стоит отметить отсутствие PCIEx1, но зато имеется три, на наш взгляд, более нужных PCI-порта. В BIOS ничего особенного замечено не было, кроме функции по слежению за температурой компьютерных компонентов PC Health Status. Комплектация довольно скупа, зато имеется плакат по установке на плату компонентов будущего ПК. К устройству прилагается диск с драйверами, два шлейфа IDE и FDD, и кабель для обеспечения устройств SATA-питанием.
→ **минусы.** Из явных минусов хочется отметить пластмассовый, а не металлический рычажок на процессорном сокете, выглядящий довольно хрупко.

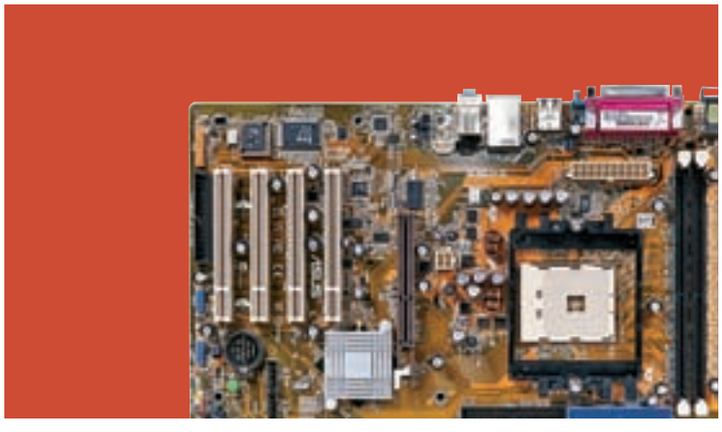




ECS NFORCE4-A754
($66) 7 звезд

ЧИПСЕТ: **NVIDIA nForce4-4X**
ПАМЯТЬ: **3xDDR DIMM 400/333/266 (2 Гб max)**
СЛОТЫ РАСШИР.: **1xPCI-Ex16, 2xPCI, 3xPCI**
SATA: **4xSATA**
FIREWIRE: **нет**
LAN: **Marvell 88E1111**
АУДИО: **Realtek ALC655, 6-канальный AC'97**

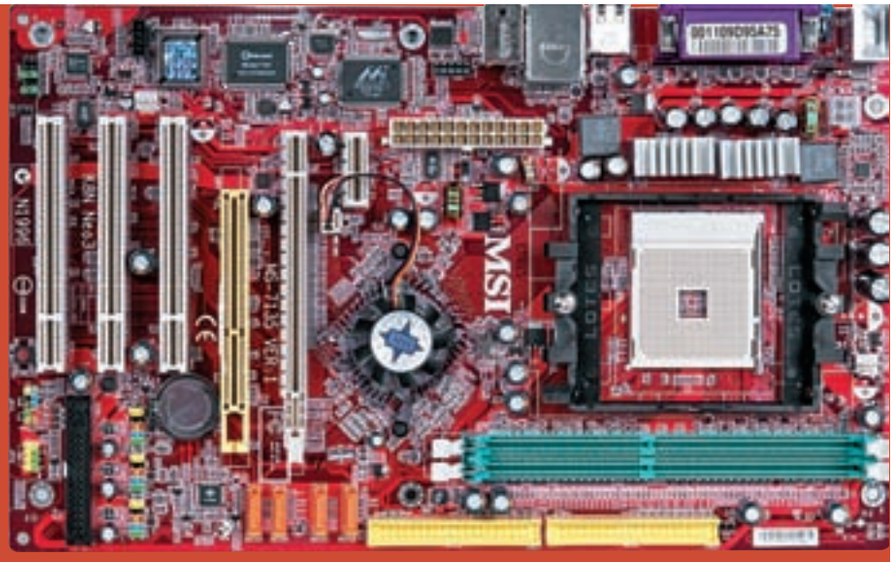
→ **плюсы.** Еще одна плата с тремя слотами DIMM. Поддерживает процессоры Socket 754 AMD Athlon 64 и Sempron, и оснащена слотом PCI Express x16, двумя слотами PCI Express x1, не считая трех обычных PCI. При производстве был использован чипсет NVIDIA nForce 4-4x с максимальной частотой шины HyperTransport 800 МГц. На плате имеется три внутренних разъема для подключения оставшихся шести USB-портов из десяти, которые поддерживает чипсет. Также присутствуют 24-контактный и ATX12V разъемы питания, набор коннекторов для вывода звука на переднюю панель, в том числе и SPDIF, и три разъема для подключения вентиляторов, один из которых занят чипсетным.
→ **минусы.** Скромная комплектация, отсутствуют дополнительные планки для USB и звука, невозможно регулировать скорости вращения вентиляторов, да и разгонный потенциал оставляет желать лучшего.



ASUS K8U-X
($55) 6 звезд

ЧИПСЕТ: **ULi M1689**
ПАМЯТЬ: **2xDDR DIMM 400/333/266 (2 Гб Макс)**
СЛОТЫ РАСШИРЕНИЯ: **1xAGP, 4xPCI**
SATA:**2xSATA II**
FIREWIRE: **нет**
LAN: **Realtek RTL8201CL**
АУДИО: **6-канальный ADI AD1888**

→ **плюсы.** Новая серия плат от ASUS рассчитана на бюджетный сектор рынка. Данный экземпляр работает с поддержкой как процессоров Sempron, так и камней Athlon 64. Чипсет ASUS K8U-X изготовлен компанией ULi, следовательно, о поддержке стандарта PCI Express можно забыть. Сеть обеспечивает интегрированный в чипсет контроллер, плюс имеется интерфейс Realtek RTL8201CL. Из особенных технологий: ASUS MyLogo, ASUS EZ Flash и ASUS CrashFree BIOS2.
→ **минусы.** Все-таки компании ASUS лучше удаются платы для среднего и верхнего ценовых сегментов. В бюджетном секторе ASUS K8U-X конкурировать практически не может из-за слабого аудиокодека и отсутствия PCI-E.



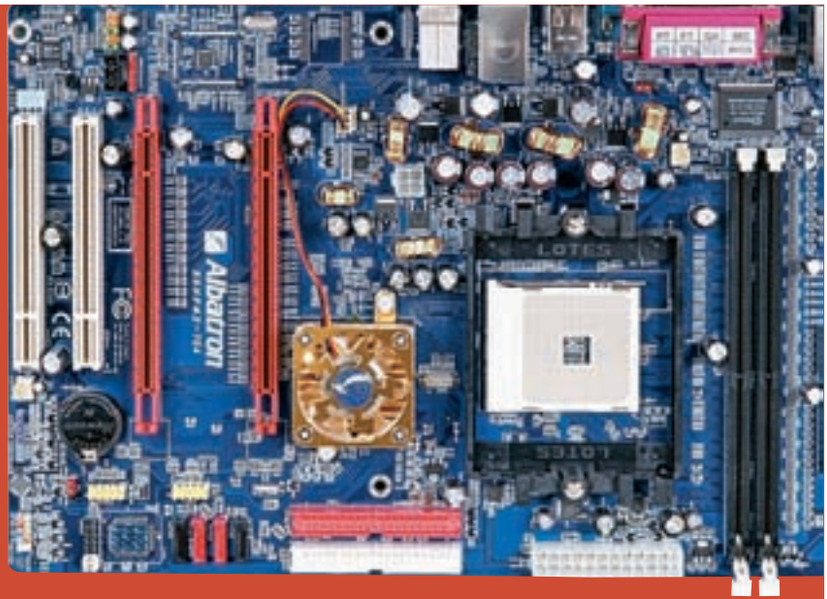
лучшая покупка

MSI K8N NEO3-F
($65) 8 звезд

ЧИПСЕТ: **NVIDIA nForce4-4X**
ПАМЯТЬ: **2xDDR DIMM 400/333/ (2 Гб max)**
СЛОТЫ РАСШИР.: **1xPCI-Ex16, 1xPCI-Ex1, 3xPCI, 1xAGR**
FIREWIRE: **2xSATA**
LAN: **Realtek RTL8201CL, Marvell PHY88E1111**
АУДИО: **Realtek ALC655, 6-канальный AC'97**

→ **плюсы.** MSI K8N Neo3-F — плата нестандартных размеров (300x185 мм). Обеспечивается поддержка технологии RAID 0, 1, 0+1. Сам чипсет расположен весьма необычно, под углом 45 градусов к граням платы. Его охлаждением занимается небольшой радиатор с вертушкой. Особенностью платы является наличие модифицированного AGP-порта, помимо присутствующего PCI ExpressX16. Так что проблема выбора интерфейса с видеокартой отпадает. Разгон и настройка параметров компонентов осуществляется через Cell Menu — уже традиционное для MSI. Для управления памятью имеется особенное подменю, просто набитое опциями.

→ **минусы.** Индекс Gold Edition на упаковке оказал должное влияние на слюнные железы, но в плане комплектации нас ждало разочарование. Из бонусов — только планка с USB-портами да шнуры в оплетке.



выбор редакции

ALBATRON K8NF4X-754
($69) 8 звезд

ЧИПСЕТ: **NVIDIA nForce4-4X**
ПАМЯТЬ: **2xDDR DIMM 400/333/266 (2 Гб max)**
СЛОТЫ РАСШИРЕНИЯ: **12xPCI-Ex16, 2xPCI**
SATA: **4xSATA**
FIREWIRE:**нет**
LAN: **Broadcom AC 131**
АУДИО: **Realtek ALC655, 6-канальный AC'97**

→ **плюсы.** Карта выделяется наличием двух PCI-Ex16 портов. В комплекте имеется бридж для соединения видеокарт, поддерживающих SLI. Помимо этого она поддерживает работу процессоров AMD Sempron. С охлаждением чипсета справляется золотистая комбинация из радиатора с вентилятором. Работает практически бесшумно. BIOS карты снабжен необходимыми для разгона функциями, среди которых — изменение напряжения на памяти, ядре и чипсете, защита от перегрева, автосброс при неудачном разгоне.

→ **минусы.** Плата была бы просто уникумом, если бы производитель включил поддержку FireWire.

→ **выводы.** При всех недочетах платформы 754 понятно, что спросом они пользовались и будут пользоваться. Примером тому могут служить наши подопытные агрегаты — при сравнительно низком ценовом пороге ты получаешь довольно шуструю лошадку со всеми недавними прелестями и достижениями хай-тека. Награду «Лучшая покупка» хотелось бы вручить плате MSI K8N Neo3 за поддержку AGP и PCI-E, высокое быстродействие и богатую функциональность. «Выбор Редакции» отдаем Albatron K8NF4X-754 за быстродействие, поддержку SLI и большой разгонный потенциал **C**

Soft

admining

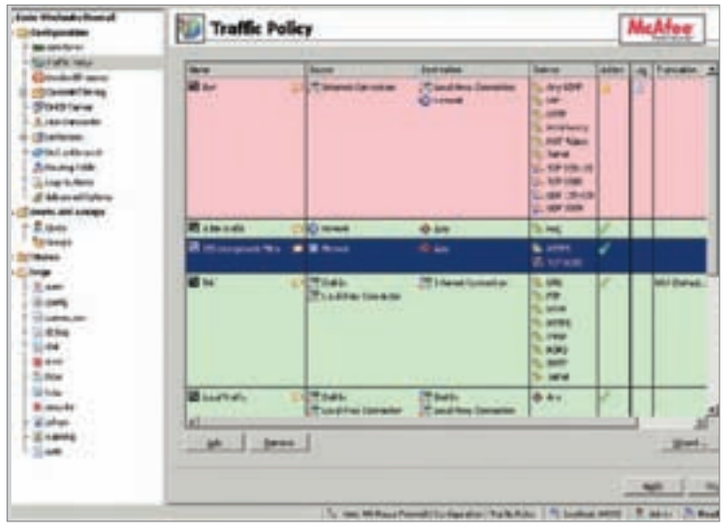
НАСТРОЙКА FIREWALL

АЛЕКСАНДР ПРИХОДЬКО
(SANPRIH@MAIL.RU)

Прежде чем начать устанавливать и настраивать «стены», тебе необходимо определиться, на базе какой программы ты это будешь делать. Уже много раз писалось и переписывалось в нашем журнале про способы защиты сети от внешнего вторжения. По большому счету, принцип работы всех «горящих стен» одинаков (я имею в виду программные продукты). При настройке «стены» создаются правила, в соответствии с правилами «стена» либо пропускает пакеты внутрь/наружу твоей сети, либо отбрасывает. Мы рассмотрим построение «стены» на примере программного продукта фирмы Kerio. Достаточно демократичный по цене продукт, обеспечивающий потребности небольшой сети. Если у твоей конторы нет денег на покупку сервера для установки файрвола, то хочу тебя обрадовать — на базе обыкновенного четвертого пня с 512 метрами памяти, ты сможешь построить файрвол, который легко справится с сеткой до 100 машин. Ну, операционку лучше поставить, конечно, серверную. Еще кое-что о приправах. Фирма Kerio написала софтину, которая интегрируется в Active Directory. Это говорит о том, что если установить файрвол на комп, который является членом домена, то при настройке он легко засосет базу данных пользователей. И при создании пользователя в домене, он автоматом получит доступ к инету (при правильной настройке). Однако, мы опять вступаем на тонкую стезю религии. Я, например, предпочитаю полностью контролировать все процессы в сети, и поэтому мы сейчас начнем устанавливать файрвол на комп, который не только не является членом домена, но и лежит в совершенно другой группе. При такой постановке вопроса нам придется всех пользователей заводить руками на файрволе. В принципе, нам все равно пришлось бы заводить пользователей отдельно на файрволе, так как продукт фирмы Kerio не дружит с кириллицей, а все пользователи в Active Directory заведены в русской раскладке. Начинаем. Установку операционки пропустим. Дадим имя компьютеру и рабочей группе, куда мы положим комп, одинаковое — FENCE. Так как компьютер затачиваем под файрвол, пристрелим все ненужные сервисы и произведем установку по минимуму. Ничего лишнего нам не нужно.

Чуть не забыли одну штуку. Файрвол имеет две сетевые карты: одна подключена к локальной сети и имеет адрес, который мы указываем всем клиентам, как адрес прокси-сервера, вторая сетевая карта подключена к провайдеру и имеет IP-адрес, выданный тебе провайдером с маской и адресом DNS-сервера. Дабы при настройке файрвола мы не путались в сетевых картах, мы переименуем сетевое подключение, идущее к провайдеру в «Internet Connection», а сетевое подключение, идущее в локальную сеть, оставим с именем по умолчанию.

Начинаем установку. Как обычно, к нам на помощь спешит визард. Долго жмем на кнопку «Next», пока не попадаем на экран выбора установки. Здесь, конечно, выбираем тип установки «Custom».



Далее, если не планируешь использовать через свой файрвол VPN-туннели, можешь сбросить эту галочку. Ну и если на досуге не будешь читать хелп на чешском языке, исключаешь и его при установке. «Next».

На странице с заведением админского пароля и логина пропиши сразу логин и пароль для администрирования, иначе потом это все сменить не удастся. Таковы реалии программы. На следующей странице тебе предлагается установить адрес, с которого ты сможешь удаленно рулить программой. Пока отложим это.

Теперь ждем некоторое количество минут, пока идет установка. Далее перегружаем комп и смотрим, что же получилось. После перезагрузки Kerio предложит запустить администраторскую консоль. Запускаем. Далее вводишь логин и пароль. И опять заботливые программисты сделали все, что бы мы сильно не перетрудились, морща свой мозг, — нам опять предлагают визард для настройки файрвола. Здесь остановимся на минутку. Как обычно есть два пути: пройтись по визарду и получить на выходе готовый работоспособный файрвол, в котором потом при необходимости можно подправить все правила или отказаться от услуг визарда и начать все делать ручками. Мы не ищем легких путей и сделаем все руками. Отменяем работу визарда. Теперь при запуске административной консоли нам необходимо либо зарегистрировать копию (только при наличии интернета), либо использовать демо-версию (при наличии интернета), или установить имеющийся лицензионный ключ. Устанавливаем ключ, он ведь есть у нас. С регистрацией закончили. Теперь смотрим, что мы имеем. А имеем мы одно правило (его нельзя ни удалить, ни изменить), которое полностью закрывает все порты.

Правило запретов

Теперь пришло время визарда. На странице «Traffic Policy» внизу находим кнопку «Wizard» и мощным рывком нажимаем ее. Первая страница рассказала нам о том, как визард настроит правила (читать можно только от скуки) → «Next». Вторая страница: выбираем тип подключения к интернету, если не диал-ап, то оставляем по умолчанию. На следующей страничке выбираем интерфейс, который подключен к Сети (мы его назвали «Internet Connection»). «Next». Теперь нам предлагаются порты, которые будут открыты. Предлагаю оставлять все по умолчанию, все равно будем переделывать. На пятой страничке правило, относящееся к VPN: если оное не предполагается использовать, то сбрасываем галочку. «Next». Вот пришли к входящим правилам. На шестой странице визард предлагает указать, какие сервисы, используемые в локальной сети, должны быть видны из интернета (твой веб-сервер, почтовый сервер, фтп-сервер и так далее). Если ничего такого нет, на странице ничего и не добавляем. «Next». На седьмой странице правило использования NAT. Это важная часть — оставляем отмеченным данное правило. На восьмой странице желанная кнопка «Finish». Мы получили готовый файрвол.

Если ты внимательно посмотришь на правила, которые создал визард, то увидишь, что, в принципе, все основное и необходимое для выхода из локальной сети в интернет есть. Однако, перед тем как начнем заводить пользователей на файрволе, необходимо сделать еще кое-какие настройки. При такой настройке пользователь не получит доступ в интернет, так как не имеет на это прав, но некоторые сервисы, запущенные на машине пользователя, смогут достучаться в интернет, так как существует правило, позволяющее протоколам (HTTP, например), выходить в интернет. Первое, что необходимо сделать, это исключить возможность выхода из локальной сети в Сеть чему бы то ни было. Ни один сервис, ни одно приложение не должны иметь такой возможности. Пока не трогаем правила, которые создал визард. Начинаем дописывать свои. Создадим два правила на уровне интерфейсов: одно разрешает сетевой карте локальной сети (Local Area Connection) полный доступ на файрвол, второе — запрещает локальной сети доступ на сетевую карту «Internet Connection».

На странице «Traffic Policy» нажимаем «ADD», и у нас появилось новое правило «New rule». Отредактируем его: двойной щелчок на правиле → даем ему имя «Proxy1» → меняем цвет (я делаю это для того, что бы отличать правила, созданные мной, от правил, созданных визардом) → в Description пишем: «Обмен между внутренней сетью и файрволом». Теперь редактируем источник: двойной щелчок на правиле Proxy1 в столбце Source. В открывшемся окне Edit Source нажимаем Add, и в выпавшем списке выбираем Network connection to interface. Ну и в списке интерфейсов выбираем Local Area Connection. Таким же образом редактируем столбец Destination. Только теперь выбираем Firewall host. Колонку Service не трогаем: в ней по умолчанию оставляем Any (все сервисы). А в колонке Action необходимо выбрать Permit. Расшифруем правило: мы разрешили обмен всеми возможными пакетами любых возможных сервисов между сетевой картой файрвола, смотрящей во внутреннюю сеть (Local Area Connection), и программным комплексом файрвола. Вторым правилом мы запретим обмен любыми данными между сетевой картой Local Area Connection и сетевой картой Internet Connection. Добавляем правило, даем имя Proxy2, меняем цвет, даем описание, источником выбираем Local Area Connection (Source), приемником выбираем Internet Connection (Destination), сервисы оставляем все (Any), а в колонке Action выбираем Deny. Таким образом, ни один сервис, протокол или пакет не сможет пройти, минуя файрвол, из внутренней сети во внешнюю, и наоборот. И еще один нюанс: файрвол читает правила сверху вниз. Если первое правило разрешает пакету или сервису действие, то к этому пакету или сервису прикладывается следующее правило и так до самого низа списка правил. И если ни одно правило не запретило активность пакету-сервису, то он выпускается наружу или, наоборот, проходит внутрь сети. Разместим наши вновь созданные правила после правила, созданного визардом Firewall Traffic. Для перемещения правил вверх или вниз справа от таблицы правил существуют стрелочки.

Не забываем нажимать кнопку Apply после всех правильных действий, иначе все труды будут утеряны при выходе из программы! Вот теперь пришло время разобраться с остальными правилами.

Запретим доступ из интернета в нашу локальную сеть. Создаем правило, назовем его Ban. Помещаем его на самый верх. Это правило должно отсекать все попытки проникнуть в нашу внутреннюю сеть. Делаем его красным. Все разрешающие правила — зеленые, запрещающие — красные. Остальные — любого другого цвета. Тебе проще будет разбираться. Настраиваем правило Ban. Источником выбираем Internet Connection, приемником (Destination) — Local Area Connection и Firewall Host. Сервисы запрещаем следующие: Any ICMP, DNS, HTTP, HTTP Proxy, KWF Admin, Telnet. И закрываем следующие порты: TCP с 135 по 139, TCP 3389, TCP 445, UDP 135-139, UDP 3389. В колонке Action ставим Deny. В колонке Log выбираем Log matching packets. Теперь в логах, в случае попыток проникновения в твою сеть, ты увидишь, кто и с какого адреса что-то хотел сотворить с твоей сетью.

Теперь отключим некоторые правила, созданные визардом, а некоторые подправим. Для отключения правила достаточно снять галочку возле имени и не забыть нажать Apply. Итак, отключаем следующие правила: ISS OrangeWeb Filter (спам-фильтр, который через некоторое время запросит денег), правило NAT (мы задали вместо него свои правила), правило Local Traffic (аналогично). Теперь подправим правило FireWall Traffic. Все нужные тебе сетевые сервисы добавляй в это правило. Мы добавим пока ICQ, IRC, Ping.

Вот, в первом приближении настроили. Теперь для примера создадим правило, которое будет резать рекламу в аське. Создаем правило, называем его «Баннеры», источник — Any, Destination — ar.atwola.com, Service — Any, Action — Deny, и писать лог, дабы проследить работоспособность правила.

Ну и напоследок создадим правило, разрешающее контроллеру домена обновлять зону DNS и синхронизировать время с серверами точного времени. Создаем правило, называем его по имени контроллера домена (дабы потом разобраться, что мы тут натворили), в источнике указываем IP-адрес контроллера домена, в назначении — Any, Service — DNS,NTP, Action — Permit, и в Translation выбираем Translate to IP address outgoing interface (typical setting).

Таким образом, мы рассмотрели случаи отрезания ненужных вещей (на примере правила «Баннеры») и создания специальных разрешений на примере правила XakDomain. Теперь, читая полезные форумы и наткнувшись где-нибудь на предупреждение о каком-либо вирусе, используй определенный порт. Тебе достаточно этот порт внести в правило BAN, и через твой файрвол вирь не пролезет. Аналогично ты поступаешь с навязчивой рекламой: вычисляешь адрес рекламы и прописываешь его в правило «Баннеры». Если тебе необходим какой-либо дополнительный сервис — прописываешь его в правило Firewall Traffic. Если необходимый тебе сервис отсутствует в таблице выбора, ты можешь создать его сам. Для этого раскрываешь закладку Definitions, далее Services, жмешь ADD, даешь имя сервису (например, MyService), выбираешь протокол, указываешь порты, заполняешь описание, нажимаешь OK и Apply. Теперь созданный тобой сервис станет доступен в правилах.

Проверяем: лезем в Traffic Policy, добавляем наш вновь созданный сервис в правило Firewall Traffic, двойной щелчок на Service и в списке находим наш сервис.

Мы рассмотрели примерное создание правил для защиты сети и обеспечения необходимой функциональности. В следующем модуле мы продолжим настройку нашего файрвола, разберемся с пользователями, квотами и контентным фильтром **C**

Созданные правила

Разрешенные сервисы

Soft

noname

НАИСВЕЖАЙШИЕ ПРОГРАММЫ ОТ NNM.RU
DOC@NNM.RU

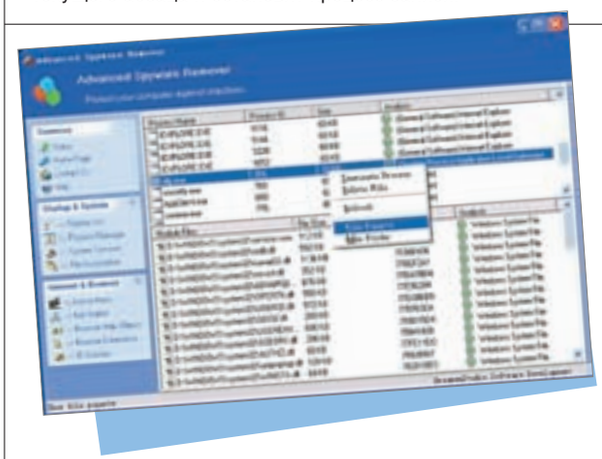
ABoo 0.6

ABoo — программа для преобразования текста в аудиокнигу в формате mp3.

Для создания аудикниги сделаем следующее:

1. Откроем файл с текстом книги.
2. Выберем каталог для сохранения аудиокниги.
3. Нажмем кнопку «пуск».

В выбранном каталоге будет создан подкаталог с названием текстового файла. В него будут записываться части аудиокниги — аудиофайлы с именами 0001, 0002 и так далее. Процесс записи аудиокниги можно остановить, нажав кнопку «Пауза». При этом программа закончит обработку текущего абзаца и остановит процесс записи.



Advanced Spyware Remover v.1.5

Обновилась Advanced Spyware Remover, простая в использовании утилита, предназначенная для защиты компьютера от шпионского ПО, рекламных программ, hijack'ов, вредоносного ПО, дозванивальщиков, кейлоггеров и т.д.

KlipFolio 3.1

Замечательная программа, представляющая собой службу информационных каналов. Канал — это поток последних новостей определенного сайта, сотрудничающего со службой KlipFolio (такой канал построен на основе RSS). Принципиально отличается от программ, получающих новости по RSS-каналам (просто убивает их), в первую очередь, методом отображения информации — она демонстрируется прямо на рабочем столе (в симпатичных окошках, внешний вид которых, конечно же, можно наладить под себя :).

IP Shifter v2.1

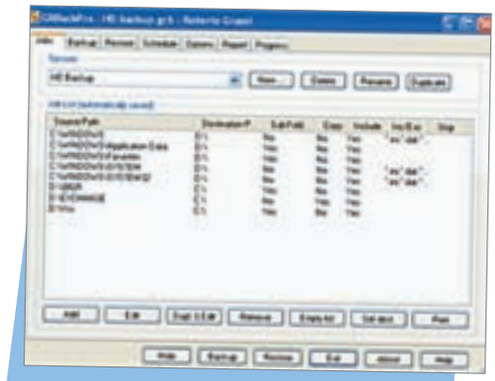
Программа пригодится владельцам ноутбуков. Например, ты работаешь дома и используешь доступ в интернет с другими настройками доступа. Тебе приходится каждый раз изменять параметры адреса IP, маску подсети, шлюз, доменное имя… Теперь все это сделает за тебя специальная программа :).

ApexDc++ 0.2.1

ApexDC++ — DC++ клиент. В новой версии нет особых изменений. Пока что авторы, можно сказать, занимаются ерундой, не уделяя внимания новым функциям и возможностям. Обещали много, а сделали пока что мало чего интересного. В новой версии несколько багфиксов и несколько незначительных изменений.

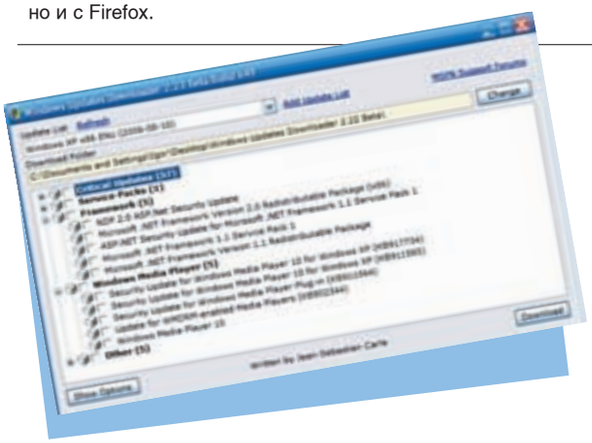
History Sweeper 2.71

History Sweeper — это программа для автоматической очистки временных файлов, которые в большом количестве остаются после серфинга. Таким образом, программа может выполнять несколько функций: обеспечивать конфиденциальность, экономить место на жестком диске, а также удалять разнообразные шпионские модули. History Sweeper автоматически очищает заданные папки, среди которых могут быть временные файлы, файлы History, Cookies, автозаполнение паролей и форм, корзина и прочие. При этом программа делает очистку «налету». History Sweeper поддерживает не только работу с Internet Explorer, но и с Firefox.



GRSoftware GRBackPro v6

GRBackPro — небольшая, но функциональная программа, разработанная для облегчения процесса создания резервных копий данных и информации под операционными системами Windows. GRBackPro позволяет работать сразу с несколькими сессиями, каждая из которых имеет свои уникальные параметры. В каждой сессии возможно задать несколько задач для создания копий целого диска, отдельной папки/папок или одиночных файлов. В программе имеется встроенный планировщик. GRBackPro поддерживает запись копий на сетевые средства хранения информации, дискеты, жесткие диски, CD, DVD, DVD-RW, DVD-RAM, CD-RW…

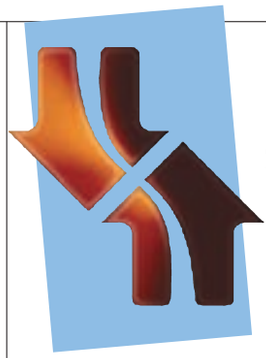


Windows Updates Downloader 2

В Сети появилась новая версия программы для скачивания обновлений продуктов компании Microsoft. Скачивание всех программных обновлений, патчей и заплаток ведется непосредственно с официального сайта компании.

Eudora 7

Eudora — почтовый клиент, который считается на западе признанным лидером среди подобных программ, являясь наиболее функциональным продуктом в своем классе. Особо нужно отметить прекрасно реализованные и в таком объеме нигде больше не встречающиеся возможности для работы с почтой и веб-серфингом на КПК. За последние годы фирма Qualcomm, разработчик Eudora, сделала ее стандартом для всех без исключения почтовых клиентов.



Dupeguru v2

DupeGuru — небольшая быстрая утилита для поиска дубликатов файлов на компьютере. Интерфейс программы предельно прост: нужно только выбрать область поиска дубликатов («Мои документы», все диски, отдельный каталог/типы файлов). Программа просканирует диск и выдаст результат в виде наглядной таблицы. DupeGuru может просмотреть имена файлов и их содержание. Просмотр имени файла показывает четкий алгоритм соответствия, который может найти двойные имена файла, даже когда они не точно похожи. Очень эффективна в работе.

BitSpirit 3.2

Мощный и удобный в использовании клиент BitTorrent, который не только работает по этому протоколу, но также имеет ряд дополнительных возможностей. Среди них особо следует выделить несколько аспектов: удобный и понятный интерфейс, возможность работы сразу с несколькими закачками, механизм кэширования данных на диск, быстрое восстановление работы, поддержка выбора файлов, удобный файловый менеджер, поддержка опции обмена сообщениями, компрессия по алгоритму GZip, работа по расписанию, интеграция в IE, поддержка HTTP/SOCKS4/5 Proxy, малая требовательность к ресурсам ПК.

Actual Reminder 2

Actual Reminder — это мощная и удобная говорящая программа-календарь для любых видов напоминаний: встречи, важные события, праздники, дни рождений, ежедневные и еженедельные напоминания. Используя горячие клавиши, ты сможешь добавлять или просматривать добавленные напоминания, смотреть календарь на текущий год. Одним нажатием легко выставить время выключения, перезагрузки компьютера, завершения сеанса пользователя или выполнить эти операции немедленно. Ты сможешь выставить время и выбрать изображения для автоматической смены обоев на рабочем столе.

GeeXboX v1

Добрый дяденька Админ перекрыл установку программ? Как? И даже пиво не пьет?! А так хочется послушать музыку или даже посмотреть фильм в отсутствии начальства, но, увы, в системе не установлены кодеки, проигрыватели и прочая лабуда? Теперь это не проблема: загрузись с этого диска и будет счастье!

Crew

e-mail

ПИШИТЕ ПИСЬМА! SPEC@REAL.XAKEP.RU
DR. KLOUNIZ

zelenkov5@mail.ru
Зеленков Александр
помогите управится с мобилой

Здравствуйте, SPECы.
У меня возникла такая проблема — захотел купить хороший сотовый телефон, но денег особо небыло купил БУ SonyEricsson T630.
Во время покупки из-за нейзвестного защитного кода скинул цену :-).
Теперь проблема с разблокировкой. Плиз подскажите прогу для подбора кода (бесплатную или демку).
Прошение к всия руси экспертам: Если ответите на вопрос немогли б вы скинуть ответ на мыло. Трудно достать ваш журнал особенно СпецХакер

С уважением, Зеленков

Уважаемый Александр! Несмотря на явную потерю денег, ты получил два огромных жизненных урока :). Первый из которых был освещен еще А.С. Пушкиным в «Сказке о попе и работнике его Балде». Я имею ввиду фразу: «не гонялся бы ты поп за дешевизной» ;). Ну и, конечно же, русскую пословицу «не носи ношенное и не е-и брошенное» тоже стоит вспомнить. Выход один — пойти к метро к мужичкам с плакатиками «куплю мобилу б/у, ДОРОГО» и подозрительно бегающими глазками, заплатить им сэкономленное бабло за разблокировку. Либо — читать наш Спец «Мобильный взлом», но там ты не найдешь готового рецепта :).

gnom_88@mail.ru
Алексей Форманчук
***

Я понимаю что мне до тебя далеко но я пытаюсь тоже програмировать и кстати получается но у меня слишком мало опыта можеш посоветовать исходник как зделать так чтобы прога заходила на сайт (неважно какой) и искала изображение ну типа введите число на картинке или буквы сюда. И чтобы прога работала на опредиленное время а я вдолгу неостанусь просто обидно. В интернете столько жуликов и все хвалятся что программа будет работать я же хочу попотеть и зделать не кидалова а настаяшую прогу которая будет зарабатывать хоть и копейки но всеже будет работать. Помоги не впадлу.
С уважением Oleg

Второй раз читаю данное сочинение, написанное высоким штилем и преисполненное своего рода романтикой. Уважаемый Алексей! Я уже ответил тебе на форуме нашего журнала. А за это письмо мы премируем тебя бонусной вещью (не благодари, мы умеем угадывать желания). Итак, внимание — «Учебник русского языка» всенародно известного специалиста Дитмара Эльяшевича Розенталя. Читай на здоровье.

mypochtan3@mail.ru
Энтри
номер «Админ всея сети»

Здравствуйте.
Подскажите, какой нужно купить сейчас номер и тему журнала СПЕЦ'а, чтобы на диске лежал pdf-журнал на тему «Админ всея сети», по-моему, за июль.

Многие люди интересуются: «Уважаемый Александр, когда Вы уже устанете отвечать на вопросы про то, где можно купить, как достать и можно ли скачать на халяву?» Этим людям я обычно отвечаю: «Не иссякну никогда, ибо этот вопрос — главный и животрепещущий». Так вот, купить надо СЛЕДУЮЩИЙ СПЕЦ.

flex-mx@yandex.ru
flex-mx
бэдные хакеры

Привет, хаккеры.
Я недавно прочитал, что вы, оказывается, помираете с голоду, потому что никто не покупает ваш журнал: все его електронную версию читают. Поэтому я решил, что мне не нужен бесплатный номер аля Мурзилка. А то что же это будет: вы совсем обессилите без еды, не сможете ничего больше написать. Мне нечего будет читать. Ваш журнал закроется! А вслед за вашим и другие журналы закрываться начнут! Потом люди перестанут смотреть телевизор! Телевидение обанкротиться и тоже закроется. Нам не будут говорить правды! Мы ничего знать не будем! И тут настанет апокалипсис, полтрГейтс! Отовсюду, изо всех щелей полезут эти проклятые капиталисты американцы со своими платными программами и китайцы, много, много китайцев, которые будут продавать нам глючные компьютеры. А хакеров не будет, не будет взломанных программ, не будет кряков, не будет ключей, не будет серийников, не будет лекарств.
КАКОЙ УЖАС!!! Этого нельзя допустить! Что же делать!!! Я сегодня купил 15 номеров ваших журналов. Я копил на велосипед, но я же не могу допустить, чтобы хакеры умирали от голода. Может вы номер web-money опубликуете, мы бы вам WMZ отправляли, а?

Мужик! Вот это подвиг, достойный настоящего Мальчиша-Кибальчиша, Люка Скайуокера и доброго Терминатора. Ты совершенно правильно выявил причину грядущего апокалипсиса, причину нарастания энтропии вселенной и, что самое главное, правильным и точным ударом сокрушил эту самую причину. Теперь госпожа Белладонна, я так думаю, будет вести себя прилично, а у поросят появятся домики. Ура!

noreplay@antivir.ru

ДиалогНаука

***Человек признался в том, что шантажировал школьниц с помощью шпионской веб-камеры

Уважаемые Дамы и Господа!
Предлагаем вашему вниманию новость по теме информационной безопасности от компании Sophos. Человек признался в том, что шантажировал школьниц с помощью шпионской веб-камеры.
8 сентября 2006 г.

«ДиалогНаука», официальный партнер компании Sophos в России, сообщает, что эксперты SophosLabs ™ , глобальной сети аналитических центров компании Sophos по анализу вирусов, шпионского ПО и спама, предупредили пользователей компьютеров о кибер-преступниках, которые шпионят за детьми с помощью веб-камер, — после того, как злоумышленник признался в шантаже школьниц.

Более подробная информация в приаттаченном файле в формате doc. Информация представлена компанией «ДиалогНаука», официальным партнером Sophos в России.

Более подробная информация? Посмотрим…
«Безработный Рингланд использовал информацию, похищенную из компьютеров детей, а также свою способность осуществлять контроль за компьютерами (при помощи таких способов, как удаленное открывание лотков приводов компакт-дисков), в целях шантажа жертв, принуждая их высылать все больше и больше откровенных фотографий».

Ага. Из этого текста, любезно присланного нам «ДиалогНаукой», можно сделать вывод, что во всем виноваты педофилы-извращенцы и зловредное программное обеспечение. Но на самом деле это не так! На самом деле во всем виноваты две вещи — БЛОНДИНКИ и ЛАМЕРСТВО. Точнее, наоборот — ЛАМЕРСТВО и БЛОНДИНКИ.

Всем же известно, что выходить в интернет без базовых знаний — то же самое, что выходить на оживленный перекресток без представлений о дорожном движении (помнишь идею про «сдачу» прав на пользование интернетом, гулявшую одно время по секлабу? Я ее полностью одобряю :)). Поэтому скажу прямо: мне ничуть не жалко бедных девочек-блондиночек, которых злой дядя разводил на частное ню-фото. Ведь если бы они развивали свой слабый мозг перед тем, как идти тусить в чате с разными сомнительными персонажами, они бы и сами протроянили этого гнусного извращенца, дернули с его компа откровенные фотки и сдали бы его с потрохами отдельному батальону Конной Полиции. Отсюда мораль. Дорогие девочки! Читайте доки, они — рулез! Интернет — не для слабых духом, и проигравшие бой на его просторах никогда не попадут в Валгаллу!

prokaznik81@mail.ru

Nik V

***Тема для Журнала — Прокси серверы

Здравствуйте!
Читаю Ваш журнал постоянно. Очень нравиться. Спасибо за интересное содержание. Но хотелось бы чтобы вы посвятили один из журналов теме прокси серверов. Usergate, Wingate, Kerio, SquidNT (в особенности его — почемуто единственный проксик который мне не удалось установить- тямы не хватает пока :)).

Успехов Вам ХакерСпец, будте такими же Спецами, как и всегда, не останавливайтесь на достигнутом, ищите, публикуйте и будет вам счастье через благодарных поклонников(чтецов).
Удачи. ProKazNik

Привет, шалунишка!
А вообще, как ты мыслишь, чем заполнить целый номер про прокси? Мне кажется, что это может быть 3 статьи максимум. Тем не менее, мы всегда рады новым темам и новым идеям, исходящим от наших читателей, поэтому — велком! Кстати, отзывы о журнале на почту и форум мы тоже приветствуем, и очень огорчаемся (плачем в подушку, когда никто не видит), когда читатели не рассказывают нам о своем мнении и желаниях, ведь для кого мы работаем? :)

А что там, кстати, насчет поклонников? Я слышал, что наша литред, Ася Глухова, не отказалась бы от парочки поклонников, исполняющих у ее окна свои серенады на мандолинах и катающие ее по Москве-реке на гондолах. Так что давай, если знаешь несколько горячих итальянских парней, фанатеющих от нашего журнала — шли контакты Насте. Да, опять же — поклонницы. Поклонницы нужны нам всем! Сделай так, чтобы они осаждали нашу редакцию толпами и забрасывали Аваланча своим нижним бельем — и родина тебя не забудет!

nikolay87@mail.ru

Куда девается СПЕЦ?

Как ни прихожу в киоски — ваще не могу его купить, че за фигня?

Дело в том, что мы — настоящие Бамбуча и нас, как фанты, никогда не бывает много! Борись за СПЕЦ, рви конкурентам волоса и бей их по почкам. А мы постараемся увеличить тираж :) **C**

Story

форс-мажор

ГОЛУБИ ПЕРЕПОЛОШИЛИСЬ, ЗАВИДЕВ НА СВОЕЙ ТЕРРИТОРИИ ЧУЖАКОВ. ВЗЛЕТЕЛИ, ЗАМЕТАЛИСЬ ПО ЧЕРДАКУ, ЗАВОРКОВАЛИ ГРОМКО, ТРЕВОЖНО. НЕСКОЛЬКО ПТИЦ ВЫЛЕТЕЛИ В ОКНО, ЗАНЯВ ВЕТКИ РЯДОМ С КРАЕМ КРЫШИ.

NIRO (NIRO@REAL.XAKEP.RU, WWW.NIRO-DE-ROBERT.LIVEJOURNAL.COM)

— Сколько их тут! — поразился Дима, бережно прижимая к себе сумку с фотоаппаратом и в очередной раз нагибаясь, чтобы пролезть под трубами отопления. — Сотни, наверное. А ведь раньше люди по ним с ума сходили! Покупали друг у друга за бешеные деньги, голубятни строили. Я помню, у нас где-то в районе есть парочка старых, заброшенных… А сейчас… Опа! — он ударился головой о какую-то перекладину. — А сейчас, говорят, голуби — потенциальная угроза обществу.

— Это как? — Левка отстал от него ненамного, метров на пять, но голос Димы уже терялся здесь среди пыли, хлопанья крыльев и скрипа балок. — Ты громче говори, не слышу.

— Да они какую-то заразу переносят! — почти крикнул Дима. — Говорят, даже могут быть даже биологическим оружием. Ты только зарази их чем угодно — а они всем табором донесут куда захочешь. По всему земному шару. Поэтому их временами отстреливают. Иногда. Правда, наверное, без толку…

— Кино такое было, — Левка, оказывается, был в теме. — «Охота на гения». Там какие-то гады хотели что-то подобное изобразить. Так что запросто… Далеко еще?

Дима остановился, посчитал, сколько труб они прошли, потом взглянул в сторону ближайшего окна и сказал:

— Здесь тоже можно — но из следующего однозначно лучше видно. Ты что, устал?

— Да дерьма голубиного уже полные карманы, — смахнул с лица паутину Левка. — Ты в своей камере уверен? Не зря тут пыль глотаем?

— Уверен, — продолжая продвигаться дальше, отозвался Дима. — Я ей даже Луну фотографировал. Представляешь, некоторые крупные кратеры видно…

— Ну, нам такая мощь не нужна, — Левка покачал головой. — Кратеры… Сколько тут до дома напротив?

— Метров семьдесят. Дистанция — лучше не придумаешь. Был бы киллером — из этого окна можно полквартала расстрелять.

— Давно тебя такие мысли посещают?

— Да с прошлой недели — как на пересдачу по английскому попал. Представляешь, препод запросил две сотни — или в армию, сапоги топтать.

— Так ты здесь за этим? — Левка подошел к окну, возле которого стоял Дима. — Думаешь, денег срубить? Кто купит-то? Ты пока продашь, уже два года службы закончатся.

— Полтора, — не оборачиваясь, сказал Дима. — Уже полтора.

— Да какая разница? Это же какие связи надо иметь, чтобы…

— Не бухти, — Дима осмотрелся. — Тут у меня где-то старый матрац запрятан… Я, когда в первый раз сюда забрался, чуть перо в бок не получил. Какая-то компания здесь шмаль варила… Ацетоном на весь чердак несло. Я полез в люк — смотрю, лежат, обдолбанные… Один меня увидел, за нож схватился…

— А ты? — Левка тревожно оглянулся.

— Папа не дурак… — улыбнулся Дима. — Мне же чердак важнее. В драку ввязываться не стал, хотя накостылял бы этому наркоше без вопросов — он на ногах держался только потому, что падать вокруг некуда было. Спустился на улицу, позвонил куда следует — приехал наряд и вычистил весь чердачок за полчаса. А матрац остался. Даже два — но второй, знаешь ли, не супер… Воняет, короче. Я его какими-то досками забросал.

— Тут, наверное, шприцев вокруг набросано… — Левка посмотрел себе под ноги. — Козлы… А если они вернутся? Вот прямо сейчас — возьмут и залезут сюда зелье свое варить? Что делать будем?

— Они уже сюда не пойдут, — Дима уверенно замотал головой. — Чердаков в городе, что ли, мало? Не переживай ты — сейчас такое увидишь, про все забудешь!

Он поднялся по маленькой лесенке, встал в скошенном окне, чтобы увидеть дом напротив.

— Ну, где вы там?..

Он, не отрывая взгляда от окон на той стороне, медленно расстегнул сумку и прикоснулся к камере. Подарок отца. Хорошо, когда родители не разбираются во всех этих пикселях, матрицах и прочей цифровой информации. Просто замечательно! А иначе — как бы он убедил отца, что ему нужна именно эта машинка, достойная телескопа Хаббла?

— Ну, видишь что-нибудь? — нетерпеливо спросил Левка. На чердаке к тому времени стало значительно тише — голуби освоились с присутствием здесь людей, успокоились, вернулись на свои трубы.

— Пока — только окна. Еще рано, — он спустился обратно, присел на матрац. — Вот минут десять пройдет — а потом только успевай…

Левка поставил возле лесенки сумку с ноутбуком, присел рядом, продолжая искать на полу признаки присутствия наркоманов. Парочку шприцев с гнутыми иглами он все-таки нашел, показал на них Диме — тот отмахнулся от него, как от назойливого комара.

— Да хрен с ними, ты же не босиком здесь ходишь. Ничего не трогай — и все обойдется… Вот уж не думал, что для тебя все это так страшно. Ты же должен быть в курсе…

Левка молча кивнул. Он панически боялся всяких неизвестных и страшных болезней типа гепатитов, СПИДа и прочих, еще не описанных в учебниках, страстей-мордастей. И все потому, что кому-то знание приносит успокоение и уверенность, а кому-то — страх.

Левка, будучи студентом медицинского института, боялся…

— Так, ну все, — Дима посмотрел на часы, встал, вынул камеру, включил. — Заряда хватит надолго. Снимем все и даже больше. Давай парочку пробных кадров для истории.

Вспышка выхватила Левку, который успел закрыться ладонью. Голуби вновь ринулись в хоровод, хлопая крыльями над головами.

— Ну и на хрена? — Левка щурился оттого, что вспышка успела врезать ему по глазам. — А вдруг с той стороны заметят? Знаешь, как эта штука ярко полыхает!

— Да никто ничего не заметит, — Дима махнул рукой. — Ты, когда на

балконе стоишь, много чердаков напротив рассмотрел? То-то же… Ладно, я пошел…

И он снова сделал несколько шагов по лесенке, уперся локтями в раму слухового окна и осмотрелся.

Внизу шумела улица — но видно ее не было, мешала крыша. А вот третий и четвертый этажи дома на противоположной стороне были как на ладони — двенадцать окон на одном и столько же на другом. Почти все жалюзи закрыты; два на третьем и одно на четвертом открыты полностью.

Дима посмотрел на эти окна через экран, включил зумминг. Рамы рванулись навстречу. Дима разглядел в окне четвертого этажа несколько кресел, два компьютера, большой зеркальный шкаф. Остальное оставалось вне поля зрения.

— Давайте, я готов, — тихо сказал он сам себе, вынул из кармана сумки маленький пятнадцатисантиметровый штатив, прикрутил его к камере. Аккуратно, в буквальном смысле слова балансируя над пропастью, установил фотоаппарат, нацелив объектив на то самое окно, что было ему интересно — на четвертом этаже.

— Шнур не забыл? — спросил он, не оборачиваясь.

— «Папа не дурак…» — передразнил Левка Диму. — Не забыл. Держи.

Дима протянул руку за спину, поймал провод, воткнул в камеру, проверил, что наведение не сбилось, и медленно спустился назад.

— Тихо… — сам себе прошептал он. — Не шумим, не топаем… Если камера завалится, будем надеяться на шнурок… Теперь осталось только заглянуть в сумку и узнать, что дистанционку я забыл дома на диване.

— Такое в принципе возможно? — Левка поднял брови.

— В принципе, возможно все, — подмигнул ему Дима и вытащил пульт. — Вот он, родимый. Батарейки вчера куплены. Ошибок быть не должно. В ноутбуке аккумуляторы заряжены?

— Под завязку, — Левка кивнул. К своей части работы он тоже подошел со всей ответственностью. — Ты же меня знаешь…

— Потому и спросил, — хмыкнул Дима. — Ты у нас тот еще мастер… Включай свою балалайку.

— «Балалайку…» Нечего его оскорблять, а то возьмет и откажет в самый неподходящий момент. Зависнет там или еще чего-нибудь…

— Типун тебе на язык! — Дима начал уже нервничать. — Быстрей давай, я же не могу по лестнице скакать туда-сюда каждый раз! Навел уже, все нормально, давай экран!

Ноутбук включился резво — Левка не жалел денег на «железо». Провод в нужный разъем — и камера уже транслировала изображение на экран компьютера. Комната, увеличенная до размеров семнадцатидюймового эк-

ГОЛЛИВУД… — ШЕПТАЛ ОН. — КАКАЯ ГРУДЬ… НЕТ, ТЫ ВИДИШЬ, ЛЕВКА, КАКАЯ ГРУДЬ! ТРЕТИЙ РАЗМЕР…

рана, могла быть осмотрена со всей тщательностью.

— На фига им компьютеры? — спросил Левка. — Вроде бы совсем им там не место… Ты уже разобрался, что к чему? А то слишком уж узнаваемые получатся снимки. Вычислят точку, с которой фотографировали, потом вычислят нас…

— И застрелят! — скорчив жуткую рожу, прохрипел Дима. — Но вначале заставят сожрать все фотографии!

— Ты — идиот, — Левка покачал головой. — Я реальные вещи говорю… Придется редактировать, а это уже не есть хорошо. Как ни крути, а «Фотошоп» всегда «Фотошоп» — опытный человек разберется…

— Хватит трепаться! — вдруг махнул на него рукой Дима. — Первая ласточка…

Левка посмотрел на монитор. В комнату вошла девушка. Молодая симпатичная девушка лет двадцати в белом костюме.

— Обалдеть… — выдохнул Дима.

— Чего тут балдеть? — пожал плечами Левка.

— Смотри… — и, не отводя взгляда, махнул пультом в сторону камеры. Зумминг приблизил девушку настолько, что казалось, можно дотронуться до нее рукой.

— Сильно круто, — сам себя подкорректировал Дима. — Чуть отъедем…

Дама, войдя в комнату, первым делом подошла к зеркальному шкафу, отодвинула одну из створок, сняла с себя костюм и повесила на одну из свободных вешалок.

— Тоже вариант, — широко раскрыв глаза, комментировал Левка. — Помнишь, у Хазанова — «Осталась неглиже…».

— Пусть повернется… — Дима держал палец на кнопке спуска. — Вот так…

Первый снимок. Девушка подошла к окну, стряхнула волосами, потянулась. Второй снимок. Оглянулась на дверь. Третий снимок.

— Какой ракурс! — восхищенно пробормотал Дима. — Пулитцеровская премия!

— Вторая входит!

— Вижу!

— А они дальше раздеваться будут? — Левка посмотрел на Диму. — Чего же ты мне сразу не сказал? Я-то думал — легкая эротика…

— Индюк тоже думал, — Дима смотрел в экран. — Вот так… И вот так…

Он делал снимок за снимком. На экране две красивые девушки в белом белье — разговаривают друг с другом, сидят в креслах, смотрят телевизор, курят…

— Твою мать! — шумно вдохнул Левка. — Смотри, они раздеваются!

Дима едва успевал ловить моменты.

— Голливуд… — шептал он. — Какая грудь… Нет, ты видишь, Левка, какая грудь! Третий размер…

— Я в размерах не разбираюсь, — отмахнулся Левка. — Лишь бы нравилось. Мне нравится — значит, все нормально.

Девушки, действительно, были чертовски хороши. Длинноволосые, с точеными талиями, крепкими бедрами — оставшись в одних стрингах они, тем не менее, ходили по комнате на высоченных каблуках.

— Голливуд… — повторил Дима. — Откуда такие только берутся?!

— А чего в тех окнах вообще такое? — вдруг спросил Левка. — Зачем они там? Ходят, коньяк цедят, да еще голые?

— Там — модельное агентство, — не переставая снимать, ответил Дима. — А я разве не сказал? Ну, брат, извини. А тетки эти — модели…

— Да уж, на фотографов они не похожи, — Левка понял, что рот полон слюны, плюнул под ноги. — А дальше что?

— А дальше — еще круче. Ты такого точно не видел. Сейчас придут два мужика…

— Групповуха?

— Мелко плаваешь, Левка, — посмотрел на его Дима. — Ты думаешь, для чего там компы?

— Даже не буду угадывать. Сам скажешь?

— Ничего я тебе говорить не буду. Смотри…

Левка уселся поудобнее, потом спросил:

— В камере места хватит?

— Пока суть да дело, скинь на комп фотографии, — согласно кивнул Димка. — Потому что сейчас начнется самое интересное…

В комнату вошли еще два человека — парни, которые и не собирались раздеваться. Девушки приветливо махнули им руками, потом практически синхронно потушили сигареты, откинули спинки кресел и неподвижно замерли.

Парни тем временем подошли к компьютерам, включили, потом из зеркального шкафа один из них достал пару приспособлений с проводами — все это напоминало какие-то шлемы для снятия энцефалограмм, — Левка видел такие в институте. Девушки послушно дали надеть их себе на головы, на лица каждой из них легла маска телесного цвета, превратив их в мумии фараонов. Один из парней что-то спросил — девушки показали большой палец, дав понять, что все нормально.

— Что это? — тихо спросил Левка. — Ты ведь не первый раз видишь.

— Смотри, смотри… — Дима загадочно улыбнулся. — Я, когда увидел, подумал, они киборги какие-то из будущего… Фильмов слишком много смотрел…

Парни опустились в кресла, каждый за своим компьютером, пощелкали «мышками» — тут Дима подработал зумминг по максимуму — и на экранах появились заставки программы с неизвестным названием.

— Какой-то Morphing, — пробурчал Левка, пытаясь прочитать, что написано на экранах. — А, вот уж слово Make-Up я знаю — это так называют макияж. По-английски. Сергей Зверев, парикмахер — по телевизору все время «мэйк ап», «мэйк ап» — как будто русских слов нет…

— Помолчи, а! — дернул его за рукав Дима. — Сейчас увидишь свой морфинг…

На экранах возникли фотографии девушек крупным планом. Парни перекинулись парой фраз, один засмеялся. Потом стали щелкать «мышками» по фотографиям на своих компьютерах — Дима проследил провода, каждый из парней отвечал только за одну девушку. На снимках что-то менялось — эти мелкие изменения были видны нечетко, просто насыщенность снимков цветом несколько увеличилась.

— А если цифровым зумом? — спросил Левка, не отрываясь от ноутбука.

— Не поймешь ни хрена! — ответил Дима. — Тебе оптического мало?

— Хочется же знать, что они делают. Что за шлемы, что за маски…

— Увидишь. Потом.

Один из парней закончил явно быстрее — он отъехал от компьютера в сторону, посмотрел в экран напарника, на что-то указал пальцем. Тот быстро внес какие-то исправления, вопросительно взглянул. Вроде все пришлось ко двору.

И они синхронно нажали на какую-то клавишу, каждый у себя.

Маски слегка засветились — это было видно даже днем. На состоянии девушек, похоже, это никак не отражалось — одна из них закинула ногу на ногу и покачивала на кончиках пальцев босоножку. Вот только делала она это очень аккуратно — словно боясь вмешаться в процесс, который сейчас происходил под маской. Парни тем временем закурили и о чем-то не спеша разговаривали между собой.

Левка вдруг почувствовал себя человеком, подглядывающим за какой-то ужасной тайной. Ему вдруг на несколько секунд стало страшно — намного страшнее, чем было в состоянии справиться его сердце. Он вздрогнул и ощутил биение пульса где-то за грудиной, там, где ему совсем было не место. Захотелось глубоко дышать, двигаться, бежать куда-то…

Дима заметил эту возню, положил ему руку на коленку.

— Не дрейфь, прорвемся. Представляешь, сколько эти фотки могут стоить? А ты думал — порнуху снимать идем? Нет, и порнуху тоже, клиенты найдутся всегда. Выложим в интернете для примера парочку — может, кто и клюнет. А вот с этим морфингом — тут у меня далеко идущие планы…

Левка кивнул, практически не услышав ни единого слова. Ему все это очень не нравилось.

Процесс шел уже около двадцати минут; босоножка таки упала. Девушка пошарила вокруг ногами, но зацепить ее снова не смогла. Один из парней — тот, что отвечал за ее маску — встал с кресла, подошел к ней и вернул туфельку владелице. И Дима понял, что сделал он это не по дружбе — а чтобы прикоснуться к ее ноге.

Ничто человеческое парням было не чуждо.

Одна из масок — на той девушке, что лежала дальше от окна — погасла. Она сделала вопросительный жест — ее оператор подошел к ней, отстегнул маску. Снял.

Она потянулась в кресле. Парень о чем-то попросил ее — похоже, не торопиться вставать. Она показала пальцами «о'кей», прикрыла глаза.

— Чего-то я ее не узнаю, — пожал плечами Левка, немного пришедший к этому времени в себя. Все-таки любопытство у людей в крови — оно способно затмить даже инстинкт самосохранения. — Она… Она изменилась? Или у меня проблемы с памятью? А ты вообще снимаешь или нет?

Дима опомнился, сделал несколько снимков, стараясь максимально приблизить лицо девушки. К тому времени она уже, по-видимому, могла подняться — и сделала это с удовольствием.

Встала, потянулась еще раз, щелкнула пальцами.

И повернулась к окну.

Дима сделал снимок и сказал:

— Я понял.

— Чего ты понял? — спросил Левка, глядя на остановленное в кадре обновленное лицо девушки. Сам он видел лишь, что у нее неизвестно откуда взялся обалденный макияж с губами, сверкающими ярко-алой помадой, длинными ресницами и отменным цветом лица.

— Фабрика грез по-русски. Ты чего, не узнаешь?

— Кого?

— Девицу эту. Это же Лара Крофт, — Дима даже разозлился немного на несообразительного друга.

— Какая Лара?… — спросил Левка и вдруг понял. — Анжелина Джоли… Ну точно…

Перед ними возле окна стояла точная копия голливудской актрисы. Девушка тихонько дотронулась до своих щек, улыбнулась уголками рта и что-то изобразила на лице — то ли кокетливый взгляд, то ли удивление. Левка с трудом понимал женщин, тем более красивых… Тем более стоящих в открытом окне с обнаженной грудью — уж таких в его жизни пока еще точно не было.

Он сделал еще пять-шесть снимков.

— Левка, ты копируешь? — спросил он.

— А как же, — отозвался тот. — Все уже на компе. Ждем рождения второй бабочки.

Дима усмехнулся. А что — неплохое сравнение. Маска — как куколка. И кем же будет вторая?

Хлопанье крыльев вывело его из ступора. Пара голубей резвились точно возле камеры — похоже, собираясь устроить возле нее если не гнездо, то, как минимум, брачные игры.

— Кыш! — взмахнул рукой Дима и вскочил с матраца. — Пошли прочь, нечего здесь делать!

Голуби отпорхнули в сторону от окна метра на два, но с определенной настойчивостью стремились вернуться. Выполняя какие-то немыслимые кульбиты, они сновали возле камеры, но грозные Димкины крики не давали им опуститься на крышу.

— Дима, там еще кто-то пришел, — вдруг сказал Левка. — Какой-то странный мужик…

Голуби сразу же стали неинтересны. Дима прильнул к экрану ноутбука.

В комнату, действительно, вошел новый участник действа — человек в черном кожаном пиджаке — и, похоже, его прихода никто не ожидал. Парни встали со своих мест, а девушка схватила с кресла покрывало и прикрыла грудь. Человек недолго стоял в дверях — он вытащил что-то из-за пояса…

— Пистолет! — одновременно вскрикнули Дима и Левка. — Зачем?

В следующие несколько секунд стало ясно — зачем. Один из операторов был отброшен выстрелом в упор в другой конец комнаты. Звука выстрела до чердака не донеслось — пистолет был с глушителем.

Дима смотрел на все это и снимал, снимал непрерывно. То, что происходило сейчас в окне дома напротив, могло дорогого стоить…

Тем временем оставшийся в живых оператор метнулся к своему товарищу, потом схватил кресло и швырнул его в стрелка. Тот был человеком явно тренированным, он легко увернулся и выстрелил еще раз.

И второй оператор тоже упал на пол.

— Дима, надо валить! — вдруг сказал Левка. — Валить по полной программе!

— Чего, в штаны наделал? — Дима не выпускал из руки пульт, ладонь стала потной от напряжения. — Ты хоть понимаешь, что мы свидетели преступления? Что у нас эти снимки купит или милиция, или преступники, или эта модельная контора? Я такой шанс из рук выпускать не буду.

— Дима, там двух человек только что убили! — Левка вскочил и отступил на пару шагов назад. — Убили на хрен, ты что, не понимаешь?! Это не сотовый в переходе рвануть, не лохов в интернете разводить!

— Убили только одного, не ори, — оборвал Левку Дима. — Второй шевелится — ему, вроде бы, в ногу стрельнули.

Левка посмотрел на экран. Второй оператор с перекошенным от боли лицом отползал куда-то в угол и должен был вот-вот выйти из поля зрения объектива. Дима немного прибрал увеличение, раздвинул кадр. Человек с пистолетом подошел к девушке, лежащей до сих пор под маской (которая к тому времени тоже перестала светиться, процедура морфинга была окончена), ткнул стволом в живот около пупка. Она вздрогнула и попыталась сама снять маску, но киллер что-то сказал ей и надавил стволом. Девушка замерла.

— ПИСТОЛЕТ! — ОДНОВРЕМЕННО ВСКРИКНУЛИ ДИМА И ЛЕВКА. — ЗАЧЕМ?

Тогда он махнул пистолетом в сторону Анжелины Джоли и о чем-то попросил ее. Та как-то боком, словно краб, не выпуская из виду ствол, прошла к шкафу, выбрала там какую-то накидку и бросила своей напарнице. Киллер поправил складки ткани, скрыв грудь девушки. Потом подошел к убитому, присел, посмотрел ему в лицо; видно было, что он что-то бормочет. Потом вышел из кадра к раненому.

— Сливай кадры, Левка, — нетерпеливо бормотал Дима. — Так, стоп, хватит. Ставлю на видео режим. Но ты уж тут следи, сейчас карта будет заполняться на глазах.

— Ты время от времени паузы делай, разбивай на файлы. Блин, куда мы вляпались? — махнул рукой Левка, следя за потоком информации.

Теперь изображение на экране не билось на кадры — они смотрели на непрерывную историю нападения на модельное агентство.

— Димка, кто он? Конкурент? Или там проблемы с девушками? — Левка выдвигал версии прямо на ходу. — Может, их кто-то заказал — в смысле время с ними провести, а они не дали? А вдруг они…

Он не договорил. Киллер снова возник в кадре — он помогал парню, действительно, раненому в бедро, добраться до компьютера. Оператор был бледен, держался только за счет силы рук ранившего его стрелка. Штанина была вся мокрая от крови — Дима сомневался в том, что он протянет еще хотя бы час.

В кресло он просто упал в прямом смысле слова. Киллер встал между ним и компьютером, что-то спросил, внимательно посмотрел в глаза и выслушал ответ на свой вопрос. Вроде бы его все удовлетворило, он отошел в сторону.

Оператор положил руки на клавиатуру, нажал несколько клавиш. Потом попытался воспользоваться «мышкой», но это у него не получилось — рука соскользнула со стола, он покачнулся и повалился на пол. Киллер пытался его подхватить, но не успел.

Девушка, стоявшая в углу, закричала. Дима видел, что до этой минуты она держалась из последних сил — лицо ее становилось то ярко-пунцовым, то его посещала крайняя степень бледности. Она просто должна была сорваться — и она сорвалась.

Человек с пистолетом не ожидал этого вскрика. Он присел над упавшим оператором, шевельнул его стволом — и в этот момент за спиной раздался пронзительный девичий визг.

Левка был уверен на сто процентов — киллер сам не ожидал от себя подобной реакции. Он сложился практически пополам, сделал быстрый кувырок через упавшего оператора и выстрелил на голос. Девушка с лицом Анжелины Джоли взмахнула руками, получив пулю в грудь, и упала на пол — так получилось, что она целиком осталась в кадре. Дима приблизил ее лицо — и они с Левкой увидели, как контуры лица меняются на глазах; оно оплывало, словно свеча; голливудские черты уступали место славянским…

— Дима, паузу сделай, — попросил Левка. — Не влезает уже на камеру.

Пока скачивался файл, киллер пришел в себя от неожиданного поворота событий, встал, подошел к убитой модели. И только потом, по-видимому, понял, что остался наедине со второй девушкой, до сих пор скрытой под маской. Он подошел к ней, что-то спросил.

— Как она там в маске разговаривает? — удивился Левка. — Чего-то я уже совсем понять не могу, зачем он приперся. Цель не вижу в упор…

— Всему свое время, — сказал Дима. — Он нам сам сейчас все объяснит. Не просто же так он стрелял налево и направо. И оператор ему был нужен живой. Как-то непрофессионально все это.

— Ты, что ли, профессионал? — прищурившись, спросил Левка. — Леон-киллер. Ну, стрельнул в ногу. Попал в бедренную артерию. Все, каюк. Пишите письма, шлите переводы.

— ДА, ЗАСИДЕЛИСЬ, — ПОКАЧАЛ ГОЛОВОЙ ДИМА, ВСПОМИНАЯ ПРИСТАЛЬНЫЙ ВЗГЛЯД УБИЙЦЫ

— Профессионал бы не попал. Профессионал нашел бы место более безопасное для такого выстрела. Не знаю, плечо, например, голень… — Дима категорически был не согласен с Левкой. Если по условию задачи оператор должен быть жив — значит, перед нами какой-то дилетант.

Тем временем киллер подошел к компьютеру, который был соединен с маской оставшейся в живых девушки. Сел в кресло, положил пистолет рядом с собой, взглянул на экран.

— Дима, в кого ее-то хотели превратить? Понять можешь?

— Нет, под слишком острым углом мониторы стоят, ничего толком не разглядеть…

— А если позвонить в это агентство? — вдруг спросил Левка. — Сказать, что у них на четвертом этаже людей убивают?

— Звони, — безразлично ответил Дима, продолжая наблюдать за происходящим. — Думаю, пока они там поймут, что это не шутка, неделя пройдет. Звони, звони, не думай, что я тебя отговариваю…

Левка вытащил мобильник, повертел его в руках и спрятал обратно.

— Смотри, — позвал его Дима — Он чего-то в этом соображает…

Киллер нажимал какие-то клавиши, делал на экране непонятные штрихи «мышкой», временами что-то громко говоря и размахивая свободной рукой. Пока было непонятно, ладится ли у него процесс или нет.

— Представляешь, он там сейчас что-нибудь подправит и превратит ее в царевну-лягушку… — Левка, выпучив глаза, посмотрел на Диму.

— В Крэйзи Фрога, — подмигнул ему тот в ответ. — Тоже лягушка — вот только не царевна. Наше дело не задачку решать, а снимать. Потом все в деталях дома рассмотрим, не пропустим ничего.

Девушка в кресле попыталась снять маску — на экране сразу же замигал какой-то тревожный красный значок. Киллер вскочил со своего места, за долю секунды оказался рядом и схватил ее за шею, вдавив в кресло. Она тут же затихла будто парализованная. Тогда человек, уже не надеясь на ее послушание, стащил с груди девушки накидку, разорвал ее пополам и привязал руки к подлокотникам. Проверил узлы, после чего встал и что-то сказал — короткое и грубое.

— Вроде сукой назвал, — пожал плечами Дима. — Жалко девчонку… Чего он там сейчас наколдует?

— Думаешь, он ее оставит в живых? — Левка спросил Диму как-то жалостливо, так, что тот на несколько секунд оторвался от просмотра и оглянулся.

— Ты еще заплачь, мальчик, — развел он руками, глядя на Левку. — Мы-то тут при чем?

— Мы можем вызвать милицию… И тогда ее спасут.

— Спасут? Ты уверен? — Димка рассмеялся. — Да они никого спасти не могут. Поставь уже на них на всех крест. По крайней мере, мы на этом деле точно должны подняться. Либо поможем этого киллера поймать — либо поможем ему скрыться. Кто больше заплатит. А может, и само агентство даст объявление — типа «Нужна информация за вознаграждение, анонимность гарантируем». Мы еще подумаем, куда себя подороже продать! Левка, такой шанс бывает раз в жизни, грех им не воспользоваться…

— Ты хотел денег заработать, чтобы английский сдать…

— Да я теперь… Какой к черту английский, Левка! — Дима взмахнул руками. — Так, не будем отвлекаться, — остановил он сам себя. — Что у нас там происходит?

Киллер тем временем вернулся за компьютер, поработал еще немного, после чего достал из кармана листок бумаги, расправил его рядом с собой на столе и, заглядывая в него и сверяясь с какими-то инструкциями, закончил работу. Сцепив вместе пальцы рук, он размял их над клавиатурой, оглянулся на привязанную девушку и нажал на клавиатуре пробел.

Поначалу ничего не происходило — по крайней мере, Дима с Левкой ничего не заметили, как ни старались. Примерно через полминуты девушка на кресле стала дергаться, крутить головой и пытаться освободить руки, но это ей не удавалось. Тогда она начала биться, как под действием высокого напряжения.

— Он ее что, током лупит? — в ужасе спросил Левка. — Он там что-то изменил — и теперь это работает, как электрический стул?

Дима молча смотрел в экран.

Тем временем из-под маски стал виден дымок — поначалу легкий, потом все сильнее и сильнее. А еще через несколько секунд у нее вспыхнули волосы.

Парни, оказавшиеся невольными свидетелями зверского убийства, смотрели на все это, затаив дыхание. Девушка сопротивлялась еще примерно минуту, после чего ее тело неподвижно замерло в кресле.

— Все, — шепнул Левка. — Конец фильма.

Киллер подошел к убитой, разогнал руками дым, осторожно прикоснулся к маске. Пальцы ему пришлось отдернуть практически сразу же — но он был к этому готов. Взял ее запястье, пощупал пульс, бросил руку.

— Точно говорю — умерла, — повторил Левка, хотя это было понятно безо всяких слов. — Твою мать…

И он сел на матрац, обхватив голову руками и раскачиваясь из стороны в сторону.

— За каким дьяволом я поперся с тобой на этот проклятый чердак! — бормотал он себе под нос. — Да еще со своим ноутбуком! Теперь на моем винчестере криминала — на два пожизненных! Или на одно вот такое кресло!

— Ты что, припадочный? — Дима подошел, положил руку ему на плечо. — Все будет нормально…

Хлопанье крыльев заставило его взглянуть в окно. Та самая пара танцующих голубей, что не сумела сесть на камеру в первый раз, со второго таки удачно приземлилась. Одна из птиц опустилась прямо перед фотоаппаратом, вторая — сверху.

И как только они своими крыльями прикрыли фотоэлемент, автоматически сработала вспышка.

Все остальное случилось за какие-то мгновенья.

Птицы, напуганные внезапным потоком света, взлетели, баламутя воздух своими крыльями. Камера, не предназначенная для подобных нагрузок, покачнулась; штатив наклонился, и она упала с оконной рамы. Провод, соединяющий ее с ноутбуком, резко натянулся.

Дима замер.

— Стоим спокойно… — сказал он сам себе. — Лева, контролируй тот конец, что в буке… Я полез…

Он поставил ногу на первую ступеньку; она скрипнула — едва слышно, но у Димы екнуло в груди. Он представлял, как там за краем окна болтается на проводе его дорогущая камера… «Лучше не думать об этом, — сказал он сам себе, взбираясь на вторую ступеньку. — Отец не переживет…»

Он, не отрываясь, смотрел на натянутый провод, который служил гарантией того, что по ту сторону слухового окна камера ждет его на расстоянии вытянутой руки.

— Если друг оказался вдруг… — прошептал он, ставя ногу на третью ступеньку; отсюда уже можно было видеть то самое окно… — Парня в горы тяни, рискни…

— Дима, — донеслось снизу. — Дима…

— Цыц, — отозвался он. — Не время… Еще две ступеньки, потом руку протяну — и валим отсюда, только нас и видели…

— Дима…

Но звать его сейчас было без толку. Ему было абсолютно все равно, что происходит вокруг — надо было спасать фотоаппарат. Он перегнулся через бортик и увидел, как камера слегка покачивается на проводе, цепляя треногой штатива крышу. Голуби сидели на дереве метрах в десяти и наблюдали за происходящим с явным интересом.

— У, твари… — погрозил им кулаком Дима. Потом он протянул руку к камере — и встретился взглядом с человеком по ту сторону улицы.

Киллер стоял у окна и внимательно смотрел на него. Дима на мгновенье забыл, зачем он здесь — так захотелось спрятаться, взлететь, как голубь, превратиться в маленькую точку на небе…

«Вспышка, будь она неладна», — сразу понял он. А иначе чего бы ему пялиться сюда, в сторону чердака. Наверняка камера сверкнула так, что не заметить ее было невозможно. «Вот так попали», — подумал он, протягивая руку за фотоаппаратом. Схватив камеру, он втянул ее в окно и спустился вниз.

— Дима, ты посмотри, — как-то испуганно произнес Левка. — Я тебя звал, звал…

На экране застыл стоп-кадр — человек с пистолетом стоит у окна и смотрит прямо в объектив.
— Как вспышка сработала, так он сразу к окну метнулся, — прокомментировал Левка. — И камера, прежде чем упасть, его засняла. Он же нас вычислил! А мы до сих пор здесь…
— Да, засиделись, — покачала головой Дима, вспоминая пристальный взгляд убийцы. — Так чего сидим, кого ждем?

Он сунул камеру в сумку; Левка упаковал ноутбук, даже не выключая его. И они рванули в лестнице.

Голуби вновь разлетелись в разные стороны; парни мчались сломя голову, не замечая, как собирают рукавами вековую пыль, а головами — паутины.

«Быстрее, быстрее», — подгонял себя Дима, вспоминая, как можно было уйти отсюда дворами, быстро и незаметно. Приходя сюда в прошлый раз, чтобы определиться с точкой съемки, он поставил себе задание — найти пути отхода на всякий случай. Правда, он не мог и предположить, что случай будет именно ТАКОЙ. Конечно же, он забыл выполнить данное самому себе обещание…

Левку же подгонять было не надо. Он, прижав к себе портфель, пробирался между трубами и перегородками с небывалой прежде скоростью. Страх гнал его вперед лучше любого допинга.

Они с грохотом спустились по лестнице с чердака на площадку, Дима помог Левке, принял его портфельчик с ноутбуком — а сам постоянно посматривал вниз и прислушивался ко всем звукам, что слышались в подъезде. Левка, сам напуганный до смерти, спрыгнул чуть ли не из самого люка, отбил ноги и даже не заметил этого; схватив портфель, он помчался вниз. Перепрыгивая через несколько ступенек, они с Димой попеременно становились лидерами этой гонки. Сумка с фотоаппаратом было явно поудобнее, чем портфель — хвататься за перила и поворачивать на площадках Диме было сподручнее. Но Левка уделывал его своим ростом — прыжки через четыре-пять ступенек были Диме не под силу.

Грохот они подняли страшный; на пятом этаже, там, где Левка выпрыгнул из люка, открылась дверь и вслед им донеслись ругательства, выкрикнутые дребезжащим старушечьим голосом — но они не разобрали ни слова, каждый из них видел перед собой внимательный взгляд киллера, остановленный камерой, и ощущал себя на прицеле.

На втором этаже прямо перед Димой внезапно открылась дверь — и он влетел в железную преграду, издав непонятный звук. Дверь захлопнулась; с той стороны донесся какой-то крик. Дима остановился и прижал ладони к лицу. Левка успел притормозить, держа ноутбук на отлете, чтобы не ударить им о двери, стены и перила.
— Нос… — прогундел Дима. — Не стой… Дальше…

Из-под пальцев показалась тоненькая струйка крови. Он сделал несколько нетвердых шагов вперед; Левка пробежал еще пролет и оглянулся. Дима стоял, держась за перила и задрав голову кверху. А потом стал медленно опускаться на ступеньки.
— Кружится… — тихо сказал он. Дверь, о которую он ударился, открылась вновь, на площадку выскочила молодая овчарка в намордник, попыталась гавкнуть, но получилось очень неубедительно. Внизу хлопнула входная дверь.

Левка дернулся было к Диме, но остановился на первой же ступеньке. Внизу слышались осторожные шаги — человек поднимался медленно, но не тяжело, как старик, а мягко, словно кошка.

Следом за собакой показалась хозяйка — женщина лет тридцати, державшая в руках поводок.
— Рекс, фу! — крикнула она собаке; та отступила назад на несколько шагов и принялась обнюхивать упавшие на пол капли Димкиной крови. — Господи, молодой человек, что с вами?

Дима обернулся, посмотрел затуманенным взглядом на женщину и тихо сказал:
— Дверью… прищемил…
— Это же как надо бежать, чтобы так удариться! — сочувственно всплеснула она руками. — Пойдемте ко мне, умоетесь, а потом я дам бинт… Вставайте, вставайте, нечего рассиживаться, раз уж так получилось, окажу вам помощь…

В глазах у Левки забрезжила надежда. Он подбежал к Диме, помог ему подняться и вошел с ним внутрь; друг с трудом переставлял ноги, да еще овчарка постоянно крутилась вокруг, норовя обнюхать брюки, пахнущие чердачной пылью.

Дверь закрылась. Левка привалился к ней спиной, проводив взглядом уходящих в ванную комнату хозяйку и Диму, после чего поставил на пол ноутбук, быстро повернулся и прильнул к дверному глазку.

По лестнице поднимался человек; шел он очень интересно — прижимаясь спиной к стене и норовя заглянуть наверх как можно дальше. Одна рука была заведена за спину — Левка был уверен, что там пистолет.

И еще — он был уверен, что это именно тот человек, которого три минуты назад они разглядывали через объектив камеры. Тогда он, конечно, выглядел поменьше, на лицо ложились блики от стекла, да и ракурс был не такой — но ошибиться было невозможно. Человек смотрел вверх, одновременно прислушиваясь к тому, что происходит за теми дверями, мимо которых он двигается.

Левка затаил дыхание, когда киллер подошел к двери, за которой волей случая оказались ребята. На мгновение остановившись, человек сделал несколько шагов вверх, исчезнув из поля зрения Левки, но потом вдруг вернулся.

Он стоял посреди площадки и смотрел себе под ноги. «Кровь, — догадался Левка. — Димкина кровь… Сейчас догадается…»

Киллер присел. Левке было плохо видно, что он там делает, потому что глазок расширял поле зрения за счет перспективы — человек казался довольно далеко. Было похоже, что он прикоснулся пальцем к пятнам крови, потом посмотрел на свою руку. Выпрямился, оглядел все двери на площадке, еще раз взглянул на пол.

Левка оглянулся. «Надо как-то дать понять женщине, что нельзя открывать дверь… Что вообще дома никого нет…» Он медленно отступил от двери на несколько шагов в сторону ванной — и в это время в дверь позвонили. Он вздрогнул от неожиданности — несмотря на то, что ждал этого звонка. Из дальней комнаты выскочила овчарка и бросилась к двери, издавая звуки, похожие на лай — все-таки намордник здорово мешал ей быть полноценной защитницей хозяйки и квартиры.

Левка отскочил в сторону; собака бросилась передними лапами на дверь и заскребла по железу. Из ванной раздался голос хозяйки:
— Рекс, прекрати сейчас же, а не то накажу! Рекс, фу!

Собаке, судя по всему, предупреждения хозяйки были безразличны. «А мужик, наверное, слышал, как она кричала», — решил Левка. Он отступил еще дальше по коридору и собрался было остановить хозяйку, но не тут-то было — она решительными шагами вышла из ванной, на ходу вытирая руки полотенцем, отодвинула в сторону пытающегося что-то сказать Левку, отогнала в сторону собаку и щелкнула замком.
— Вы к кому? — спросила она, увидев незнакомого мужчину и закинув полотенце на шею. — Случилось чего? Или ошиблись?
— Тут на площадке кровь, — услышал Левка тихий голос. — Что произошло? Я смотрю, что возле вашей двери пятен больше всего…
— Да вот мальчишки сверху бежали, — женщина всплеснула руками, — а я с собакой собиралась гулять, открыла дверь… И один из них нос разбил, да так сильно, похоже, что сотрясение…
— Мальчишка? — спросил мужчина. — А сколько их было?
— Двое, — ответила хозяйка. — А вам-то, собственно, какое дело? Рекс, иди сюда, — внезапно скомандовала она. Собака рванулась к ней; женщина ловким движением сняла с нее намордник. И Рекс показал, на что способен — его лай оглушил всех и разнесся по подъезду от первого этажа до последнего, заставив голубей на чердаке взлететь.

Возникла пауза. Мужчина явно соображал, что делать дальше. Рекс встал между ним и квартирой в виде зубастой преграды, которая не пропустит никого к своей хозяйке. Левка, сделав шаг за шкаф с одеждой, замер, боясь издать хотя бы звук.
— Прощаться будем? — спросила хозяйка. — Все, у меня дела…

И она стала тянуть на себя дверь, но мужчина внезапно подставил ногу и пристально посмотрел вдоль коридора.
— Да, тут, действительно, может быть сотрясение…

Из ванной показался Димка с мокрыми волосами, на которые было наброшено полотенце; он по-прежнему держал голову слегка запрокинутой и прижимал к носу несколько салфеток.

И через плечо у него до сих пор висела сумка с фотоаппаратом.

Дима остановился, посмотрел в сторону двери, где слышалось грозное рычание собаки. И увидел человека, который только что совершил несколько жестоких убийств в доме напротив.

И ОНИ ВЫЗВАЛИ МЕНЯ — ПОТОМУ ЧТО Я УЛАЖИВАЮ ПОДОБНЫЕ ПРОБЛЕМЫ

ВСЕ В ЭТОЙ КОМНАТЕ БЫЛО ПОДЧИНЕНО ЕМУ — ОН СТАЛ ПОЛНОПРАВНЫМ ХОЗЯИНОМ ПОЛОЖЕНИЯ

Сил крикнуть что-нибудь у него не осталось. Ужас сковал его; он опустил руку с окровавленными салфетками.
— Нет, так просто уйти я не могу, — произнес киллер и выстрелил в собаку. Рекс взвизгнул, не поняв, что же произошло — на оружие бросаться его не учили. Хозяйка, увидев пистолет, прекратила тянуть дверную ручку и коротко вскрикнула. Мужчина, почувствовав, что может свободно войти, переступил через раненую собаку, которая пыталась отползти в сторону, кося испуганными глазами на оружие.

Толкнув женщину в грудь, он освободил дверной проем и закрыл дверь. Женщина издала какой-то непонятный звук и ухватилась за кончики полотенца, висящего на шее.
— Что вам нужно? — нашла она в себе силы спросить.
— Этот мальчишка, — махнул человек стволом в сторону Димы. — Второй тоже здесь?

Дима машинально перевел глаза на Левку — и киллер сразу все понял.
— Все в комнату, — скомандовал он. — Кто не пойдет сразу — получит пулю. А собачка, я думаю, уже не жилец, — уточнил он для хозяйки. — Рекс… Сейчас мы ему поможем.

И он выстрелил в собаку второй раз.
— Так, — указал он пистолетом на труп зверя, — будет с каждым. Если что. Поэтому лучше выполнять мои приказы.

Дима и хозяйка отступили в сторону комнаты. Левка тоже был вынужден выйти из-за шкафа, когда киллер подошел к нему и направил пистолет в грудь. Женщина без конца повторяла «Боже мой, боже мой…» — преступник неожиданно ударил ее по щеке, она вздрогнула и замолчала.
— Прошу прощения мадам, но вы истеричка, — кивнул он ей. — А это качество сейчас для вас самое последнее. С истеричками дело имел, знаю. И не жалую своим вниманием. Так что держите себя в руках, а не то ляжете рядом с Рексом.
— Хорошо, — совершенно спокойным голосом ответила хозяйка и вдруг тихонько заплакала — с каким-то подвыванием.
— Вас как зовут? — спросил киллер.
— Марг… Маргарита, — ответила она.
— Заткнитесь, Марго, — подошел он к ней вплотную, не выпуская из поля зрения мальчишек. — Заткнитесь, прошу вас. У меня очень нервная работа… Был тяжелый день. А тут еще вот эти… Малолетки.

И аккуратно толкнул ее в кресло. Она упала, даже не обратив внимания, куда. С подлокотника на пол сорвалась книга. Киллер наклонился, поднял, протянул Маргарите.
— Читайте. Представьте себе, что ничего не произошло. Откройте там, где вы остановились, попытайтесь сосредоточиться, получайте удовольствие… А я пока пообщаюсь вот с этими сорванцами. Сели на диван, быстро! — крикнул он мальчишкам.

Они вздрогнули, оглянулись в поисках дивана и сели, не сводя глаз с дула пистолета. Почему-то именно этот предмет приковывал их внимание — не злой взгляд убийцы, не его властный голос.

Именно пистолет.

Кровь у Димы перестала течь. Нос сильно болел и уже прилично распух. Он машинально держал ладони у лица, словно оберегая свое лицо от дальнейших проблем.
— Камеру сюда! — киллер протянул свободную от оружия руку.

Дима снял сумку через голову, протянул.
— Достань!

Он достал, отдал.

Мужчина сел на журнальный столик, смахнув с него какие-то журналы, положил пистолет себе на колени, включил камеру.
— Шалунишки, будь они неладны, — бурчал он себе под нос. — И как вы только узнали…
— Мы не знали… — машинально ответил Левка. — Там должны были быть…
— Заткнись, — тем же самым тоном ответил киллер — и Левка не стал продолжать. — Я знаю — там должны были оказаться только бабы с сиськами. И все. Но не сегодня. Не в это время. Как включить просмотр снимков?
— Там над экраном есть переключатель, — ответил Дима; каждое слово отдавалось болью в голове. Временами накатывала тошнота. — Надо выбрать положение с картинкой в виде кадра…
— Понял, — ответил преступник. — Так, смотрим…

Дима опустил глаза в пол и лихорадочно пытался соображать, как выйти из этого положения — но головная боль, наплывающая на него периодически, заставляла оставить все попытки думать. Удар дверью не прошел для него даром. Оставалось верить в то, что Левка что-то придумает. На Маргариту надежды не было никакой — она сидела, как восковая кукла, в кресле, держа раскрытую книгу вверх ногами, и что-то шептала себе под нос — тихо и часто шевеля губами.
— Вы меня за дурака принимаете? — вдруг встал со стола киллер и приблизился к парням вплотную. Он наклонился к Диме, стараясь оказаться к нему как можно ближе. — Здесь же ничего нет! Куда подевали фотографии?
— Как нет? — искренне удивился Дима. Он совершенно забыл в этой сумасшедшей гонке, закончившейся сотрясением мозга, что снимки сразу переписывались на ноутбук. — Не может быть — я же снимал, снимал с самого начала!
— Что ты снимал? — тряся камерой перед лицом Димы, крикнул киллер. — Куда дели фотографии, уроды? Я вас сейчас всех перестреляю!
— Да я снимал, клянусь! — Дима попытался сказать громче, но в голове что-то стрельнуло, он скривился от боли и схватился за виски. — Снимал… Сначала девиц, потом этот… Морфинг… А потом все остальное…
— Морфинг? — удивился киллер и сел обратно на стол, почему-то сразу сменив гнев на милость. — А это слово откуда тебе знакомо? Агентство закрытое, никому о своих методах работы не сообщает…
— Прочитал на экране, — ответил Дима. — Когда компьютеры включились.
— Эта штука так приближает? — покачал головой киллер. — Я, конечно, не совсем разбираюсь, но здорово, здорово… Впечатляет. Значит, насчет Морфинга вы в курсе. Что еще вам известно об этом агентстве?
— Ничего, — снова сказал Дима. Левка явно был не в теме — он понятия не имел, сколько раз его друг был на том чердаке и что именно он там успел увидеть, поэтому молчал и только слушал. — Девушки приходили, раздевались, превращались…
— В кого?
— Один раз в Ким Бессинджер, — вспоминал Дима. — Еще раз — в эту, с большой грудью… Забыл. Сегодня вот в Анжелину Джоли. Я думаю, у них возможности богатые.
— Точно, — кивнул киллер. — Так фотографии где?
— Значит, нет фотографий, — Дима пожал плечами. — Получается, я снимал, а они не записывались.
— Чушь, — отрицательно покачал головой мужчина. — Так не бывает.
— С компьютерами и цифровой техникой всякое бывает, — ответил за друга Левка. — Делаешь, делаешь что-нибудь — а оно куда-то пропадает.
— Ага, второй заговорил, — повернулся к нему киллер. — А ты там чего делал? Друга за ноги держал, чтобы он с крыши от увиденного не свалился?
— Памела, — вдруг сказал Дима.
— Чего? — не понял сразу киллер.
— Памела Андерсон… Та, которую забыл…
— Какая Памела?! Где фотки?! Или вы, ребята, их отправили уже куда-то — через интернет? Я же помню — камера висела на каком-то проводе!
— Так, для страховки, чтобы с крыши не упала, если что, — ответил Левка. — Да мы ничего никому не скажем, правда, Дима? Фоток нет, а мы как рыбы!

Дима кивнул, опустив глаза в пол. Почему-то ему казалось, что он сейчас грохнется в обморок.
— Для страховки? — киллер переводил взгляд с одного парня на другого. — Дурачить меня вздумали? «Мы как рыбы…» Живым отсюда не выйдет ни один из вас, это я вам гарантирую. Но сначала надо убедиться, что фотографий не осталось. Раздевайтесь оба.
— Зачем? — недоуменно спросил Дима.
— Я думаю, что ты заменил карту в аппарате. И флешка со снимками где-то в кармане.
— Да вы знаете, сколько стоит такая карта? — удивленно поднял брови Дима. — На две мне точно денег бы не хватило.

— Куртки сюда, — человек навел на них пистолет. — И карманы брюк выворачивайте.

Левка и Дима подчинились. Киллер вытряхнул все из курток — но ничего, кроме батареек, ключей и пульта управления камерой не нашел. В брюках тоже никаких следов карты не оказалось.

Киллер подошел к Маргарите. Та продолжала тихо разговаривать сама с собой и мелко креститься. Он поднял ей подбородок стволом и посмотрел в глаза:

— Он тебе в ванной ничего не передавал?

Она замотала головой, не в силах отвести взгляд.

— Не врешь? Вспомни, что с собачкой стало…

Маргарита сумела выдавить:

— Не вру…

— Ладно, — согласно кивнул киллер. — На чердак, думаю, подниматься не стоит. Так?

Левка кивнул. Дима через пару секунд тоже.

— Почему камера упала?

— Голуби… — тихо ответил Левка.

— На голову тебе нагадили?

— На камеру сели.

— В смысле? — не понял мужчина, привстав со стола. — И тебя не испугались?

— А я внизу был, под лестницей…

— А кто снимал?

— Я, — ответил Дима. Он вдруг понял, что они прокололись. На пустяке. На проклятых голубях.

— И тебя голуби тоже не боятся? — мужчина зло прищурился. — Или у тебя на голове они гнездо свили? И зачем тебе дистанционка?

Дима молчал.

— К чему камеру подключали?! — вдруг заорал киллер. Маргарита вскрикнула и уронила книгу на пол. — Заткнись, идиотка! — крикнул он на хозяйку. — Заткнись, пока не пристрелил!

— Не убивайте, не убивайте, — запричитала Маргарита. — У них еще портфель с собой был… там…

И она махнула рукой в коридор.

— Портфель. В коридоре. Понятно, — мужчина встал, подошел к окну, отодвинул кончиком ствола занавеску. — Понаехали уже… — сказал он сам себе. — А я тут с вами все разобраться не могу. Форс-мажор. Надо расценки поднимать… Марго, будь другом, принеси портфель из коридора — только без глупостей.

Маргарита встала, пошатываясь; она с трудом сообразила, где коридор, вышла. Оттуда донеслись ее сдавленные рыдания. Спустя полминуты она вошла с портфелем в руках.

— Ноутбук, — кивнул киллер. — Все на нем?

Левка с Димой переглянулись — молчать далее было бессмысленно. И синхронно кивнули — как ученики в школе, которых поймали за очередной шалостью.

— Хорошо, — вздохнул киллер. — Хоть что-то радует. Посмотреть дадите?

— Давайте, включу, — Левка встал с дивана и тут же ему в живот уперся ствол.

— Сидеть. Будешь с места подсказывать.

Левка, напуганный, опустился назад.

— Вы только не сломайте, он очень дорогой, мне мама на день рожденья подарила, — попросил он киллера, который к тому времени уже достал компьютер и, открыв крышку, включил его. Дима толкнул Левку коленом — мол, не влезай лишний раз.

— Где искать? — спросил киллер. Левка объяснил.

На экране появились кадры.

— А ты, парень, молодец, — похвалил Диму мужчина. — Место ты выбрал, прямо скажем, идеальное. Небось, весь чердак прополз, пока на том окне остановился?

Дима кивнул, а киллер продолжил:

— Я хорошую работу сразу чувствую. Правда, у меня-то работа другая, но место для таланта есть везде. Девицы вышли у тебя — просто загляденье! Куда думал сбывать?

— Нашел бы, — скривился от этого вопроса Дима. — В интернете места много. Тем более такие фотки — суперзвезды Голливуда в голом виде… В «Фотошопе» поправил бы комнатку…

— Тихо ты, — поднял руку киллер, не отрываясь от экрана. — Вот уже и я появился… Неплохо смотрюсь… Бум! Бум! — комментировал он то, что видел на фотографиях. — О, уже кино пошло! Ну, вы, ребята, просто молодцы! Все сняли. Мне — лет на двадцать строгого режима. Вам на медаль. Вот только придется все это стереть.

— Я понимаю, — кивнул Дима. — Куда же деваться?

— Зачем вы их убили? — вдруг спросил Левка.

— Любопытство — неотъемлемая часть глупости, — киллер отдал ноутбук Диме. — Ты пока стирай, а я объясню.

Он сделал несколько шагов по комнате, заметил, что Маргарита так и стоит в дверях, опустился в ее кресло и жестом пригласил хозяйку к себе на колени. Она как робот, приблизилась к нему, остановилась в паре шагов. Чувствовалось, что даже страх перед ним не дает преодолеть оставшееся расстояние.

— Не хочешь? Ну, не заставляю. Стой, где стоишь. Что касается вас, молодые люди…

Киллер повернулся к ним; от него исходила какая-то фантастическая уверенность в себе и в том, что он делает. Он не боялся оставить их без внимания, разглядывая хозяйку; он мог положить пистолет на стол и не следить за ним. Все в этой комнате было подчинено ему — он стал полноправным хозяином положения; закинув ногу на ногу, он спросил:

— Как вы думаете, сколько стоит та самая программа Морфинга, что установлена на компьютерах модельного агентства?

— Ни малейшего представления, — первым ответил Дима. — Такой специфический софт может стоить очень и очень дорого.

— Ты прав, мой юный друг, — хмыкнул киллер. — Я могу назвать вам примерную, очень округленную цифру. Программа морфинга вместе с прикладным оборудованием стоит почти полмиллиона долларов. Поверьте — она того стоит…

— Ни фига себе! — взглянул на друга Левка. Честно признаться, я о таких технологиях раньше и не слышал…

— Слышал, слышал, — щелкнул пальцами их собеседник. — Ты ведь частенько видел в интернете фотографии звезд, якобы сделанных папарацци. Процентов семьдесят сделаны именно таким образом. Агентство приобрело в кредит эту программу, обучило двух операторов и набрало девушек, которые согласились работать подобным образом. Дела пошли в гору, но к нужному сроку они не смогли рассчитаться с компанией, предоставившей софт. До суда дело не дошло, они отдали и компьютеры, и системы морфинга, и долги по кредиту. Ну, это и неудивительно — их продукция шла нарасхват, в основном за границу. Они выпускали и серии фотографий, и порнофильмы — якобы с участием голливудских звезд. Жили, в общем, неплохо…

— Как же — отдали? — спросил Дима. — А мы что видели?

— Вот в том-то и дело, — киллер положил руку на пистолет. — Они сделали копии программы и докопались до схем системы, сумев их воспроизвести самостоятельно — не оскудела Россия талантами… Компания, являющаяся владелицей софта, узнала об этом случайно — их агенты, отслеживающие потоки специфической информации в интернете, обнаружили новые фильмы, снятые с помощью их технологий. И они вызвали меня — потому что я улаживаю подобные проблемы… И делаю это очень и очень хорошо — вы сами видели. Я пришел, наказал — настолько, насколько мне было предоставлены полномочия. Операторы мертвы, одна система Морфинга уничтожена, на компьютерах программа удалена… Модели приказали долго жить. Короче, работа агентства прекращена надолго.

— Но ведь остались где-то диски с программой, схемы масок Морфинга… — произнес Дима. — Ведь такие вещи нельзя уничтожить совсем. Желание иметь подобные вещи даром — неистребимо.

— Тут ты в корне неправ, парень, — киллер встал, посмотрел на Маргариту, подошел к ней поближе и похлопал по щеке. — Надо же, как боится… Все-таки жалко этого брехливого Рекса — морда умная, окрас приятный… Ты уж извини.

После этого он повернулся к дивану и стал даже, как показалось Левке, выше ростом.

— Я ненавижу пиратство. Всякое. Я не люблю, когда воруют программы, музыку, игры, фильмы. Но больше всего я не люблю, когда воруют чужие мысли. Чужие идеи. Таких людей надо наказывать так, как это делаю я. И если ты думаешь, что они рискнут работать дальше, ты ошибаешься. Там на третьем этаже — кроме тех, что вы видели — лежит сейчас мертвая секретарша и еще какой-то человек, похоже, заказчик. Репутация агентства подорвана — раз и навсегда. Работа — сделана. Вот только вы — как бельмо в глазу. Я же говорю — форс-мажор… Ты все стер?

Дима кивнул.

— Ну и хорошо. А то засиделся с вами, господа…

И он выстрелил — сначала в мальчишек на диване, потом в Маргариту. Подошел к Диме, взял у него с колен ноутбук, закрыл, сложил в портфель.

— Надо уходить… — он снова выглянул в окно, отметил, где стоят милицейские «УАЗики», осмотрел квартиру — неторопливо, не оставляя лишних следов.

Возле входной двери он оглянулся, крепко сжал ручку портфеля и сказал, не обращаясь ни к кому:

— А «Фотошоп», конечно же, тоже пиратский… Ох, не люблю я это…

И тихо закрыл за собой дверь… С

ИСХОДНИКИ ВСЕЛЕННОЙ

КОЛОНКА КРИСА КАСПЕРСКИ

ЗАПИСКИ ХАКЕРА

→ сортировка списков. Списки — популярные и во многом интересные структуры данных. Они естественным образом поддерживают быструю вставку и удаление элементов, но ни в одном известном мне руководстве не затрагиваются вопросы их сортировки. Ну да, конечно, в библиотеке STL есть класс list с готовым методом stort. Пользуйся — не хочу! Но универсальное решение по определению не есть хорошее. Это раз. Не все программисты пользуются STL — это два. Наконец, в Си ничего подобного и в помине нет — это три. К тому же, было бы полным непрофессионализмом вызвывать list::sort, не задумываясь о том, как он работает. А, правда, как? Давайте, не подглядывая в исходный код STL, попытаемся реализовать сортировку списка самостоятельно.

Начнем с того, что отсортировать список тривиальным вызовом qsort не удастся, поскольку она рассчитывает, что следующий сортируемый элемент расположен непосредственно за концом предыдущего. При обработке массивов все так и происходит, но списки ведут себя иначе. Малого того, что элементы списка могут размещаться в памяти в каком угодно порядке: нет никаких гарантий, что за концом какого-то элемента находится действительный элемент! Рассмотрим следующий пример:

```
struct LIST *last_record = 0;
strict LIST *list =
(struct LIST*) malloc
(sizeof(struct LIST));

for(a=0;a<N; a++)
{
  list->val = a;
  list->next_record =
(struct LIST*) malloc
(sizeof(struct LIST));
  list = list->next_record;
} list->val=a; list
->next_record = 0;
```

Допустим, нам необходимо отсортировать список так, чтобы последующий элемент имел большее или такое же значение val. Как это сделать? Алгоритм сортировки прост как косяк: перегоняем списки в массив, сортируем его как обычно, затем либо уничтожаем несортированный список и создаем новый, либо «натягиваем» отсортированный массив на уже существующий «скелет», то есть используем уже выделенные блоки памяти. Естественно, последний способ работает намного быстрее! Демонстрационная программа для сортировки прилагается.

→ двухмаршрутные списки. В тех случаях, когда требуется осуществить ту или иную выборку элементов из списка для их последующей обработки (проще говоря, фильтрацию), можно пойти двумя путями:

1 Вернуть отфильтрованные элементы в отдельном списке (что имеет тот недостаток, что понапрасну расходует память, а при «многокаскадной» фильтрации мы рискуем окончательно запутаться в куче списков).

2 Помимо поля struct list *next_record добавить еще одно поле — struct list *next_filtred__record. В таком случае мы будем иметь дело всего лишь с одним списком, содержащим как оригинальные, так и отфильтрованные данные.

Естественно, еще потребуется поле struct LIST* first_filtred_record, содержащее указатель на первый отфильтрованный элемент списка. Оно необходимо затем, что первый элемент оригинального списка не обязательно будет совпадать с первым отфильтрованным элементом. К тому же «двухмаршрутные» списки естественным образом поддерживают «каскадную» фильтрацию. В самом деле, мы сканируем список, перескакивая по полям next_filtred_record, и, если следующий элемент не проходит через фильтр, то предыдущая ссылка перескакивает вперед. Так же очевидно, что нам потребуется два набора функций для работы с двумя маршрутами.

→ обработка ошибки выделения памяти. Постоянная проверка успешности выполнения интенсивно используемых функций во-первых, слишком утомительна, во-вторых, загромождает исходный текст, и, в-третьих, приводит к неоправданному увеличению объема откомпилированного кода программы.

```
char *p;
p = malloc(BLOCK_SIZE);
if (p==0)
{
fprintf(stderr,"-ERR:
недостаточно памяти для
продолжения операци\n");
_exit();
}
```

Кстати говоря, следующий код «p = malloc(x); if (!p) return 0;» даже хуже, чем отсутствие проверки вообще, так как при обращении к нулевому указателю Windows хоть выругается, указав на место сбоя, а такая кривая проверка просто тихо кончит программу...

Решение заключается в создании «оберток» для интенсивно используемых функций, проверяющих успешность завершения вызываемой ими функции и при необходимости рапортующих об ошибке с завершением программы или передающих управление соответствующему обработчику данной аварийной ситуации.

```
void* my_malloc(int x)
{
  int *z;
  z=malloc(x);
  if (!z) GlobalError_and_save
}
```

Между прочим, виртуальная память не безгранична, и иногда она неожиданно кончается. Попытка выделения нового блока посредством malloc дает ошибку. В этой ситуации очень важно корректно сохранить все несохраненные данные и выйти. А как быть, если для сохранения требуется определенное количество памяти? Да очень просто! При старте программы выделяем malloc'ом столько памяти, сколько ее может потребоваться для аварийного сохранения. Если нас обломают на память, то просто не запускаемся, а с ругательством сваливаем. Затем, при нехватке памяти просто сохраняемся (необходимая память у нас есть) и все!!! C

“ФУТБОЛЬНЫЙ МЕНЕДЖЕР”!

СОЗДАЙ СВОЮ КОМАНДУ ИЗ РЕАЛЬНЫХ ИГРОКОВ И ПРИВЕДИ ЕЕ К ПОБЕДЕ

ТЫ ПОЛУЧАЕШЬ $135 МИЛЛИОНОВ
на приобретение игроков российской премьер-лиги при регистрации на сайте www.total-football.ru.
Подробности на сайте **www.total-football.ru**

ГЛАВНЫЙ ПРИЗ – ПОЕЗДКА НА ФИНАЛ ЛИГИ ЧЕМПИОНОВ 2006/07

Прорыв года!

Компьютер марки <NT> AdvaNT AGE
на базе процессора Intel® Core™ 2 Duo.

Intel® Core™ 2 Duo
Процессор, опередивший время
На 40% быстрее, на 40% экономичнее*

На правах рекламы

NT computer

www.nt.ru

Компьютеры марки <NT> можно приобрести в Федеральной сети компьютерных центров POLARIS и у наших региональных дилеров: www.nt.ru тел.: (495) 363 9393

intel Core™ 2 Duo inside™

Два ядра.
Делай больше.

* Производительность измерялась с помощью эталонного теста производительности SPECint*_rate_base2000 (2 экземпляра), а энергопотребление – по значению тепловыделения (Thermal Design Power, TDP). Сравнивались процессоры Intel(R) Core™ 2 Duo E6700 и Intel(R) Pentium(R) D 960. Производительность реальной системы может отличаться. Дополнительную информацию можно получить на странице www.intel.ru/performance