на чем писать, как писать, с кем писать

крис касперски ака мыщъх, no-email

в России учат быть гениями, но не учат быть просто профессиональными работниками

Давид Ян, компания ABBYY

современные программные комплексы уже не программируются на одном языке и представляют конгломерат гибридного типа. правильный выбор технологии программирования очень важен. без преувеличения, он определяет судьбу продукта, сроки реализации, трудоемкость разработки, расширяемость и т. д. оставьте священные войны "Паскаль против Си" или "Си против Ассемблера". перед нами стоит вполне конкретная задача: создать конкурентоспособный продукт, нацеленный на рыночный успех. программирование "для души" здесь не рассматривается, поскольку искусство и маркетинг несовместимы.

на чем писать

Писать целиком на Ассемблере — это выпендриваться и заведомо обрекать проект на провал. Ассемблер оправдывает себя лишь в системно-зависимых модулях и критичных к быстродействию участках кода. Стандартные сишные библиотеки для работы со строками и блоками памяти чаще всего пишутся именно на ассемблере. Еще в ассемблере нуждаются программы, использующие мультимедийные расширения процессоров Intel Pentium и AMD Athlon. Уродовать программу ассемблерными вставками — дурной тон (нет, это просто преступление!) и весь ассемблерный код должен быть вынесен в отдельный объектный модуль. В противном случае программа рискует оказаться непереносимой и привязанной к своему компилятору. Впрочем, если жизненный цикл продукта невелик и свобода передвижений ему не требуется, использование ассемблерных вставок и специфичных для данного компилятора расширений вполне допустимо. Это сократит срок разработки. Кстати, у некоторых компиляторов, в частности у Intel C++, есть так называемые инстриксы – возможность прямого вызова SSE-команд. Красота!

Существует множество бесплатных трансляторов с ассемблера, но реально используется только два из них (речь, разумеется, идет о платформе x86, мы умышленно не затрагиваем сферу встраиваемых систем, поскольку это отдельная большая тема): морально устаревший MASM (Microsoft Assembler), входящий в состав свободно распространяемого DDK, и стремительно развивающийся FASM (Flat Assembler), созданный в рамках проекта Open Source (http://flatassembler.net/).

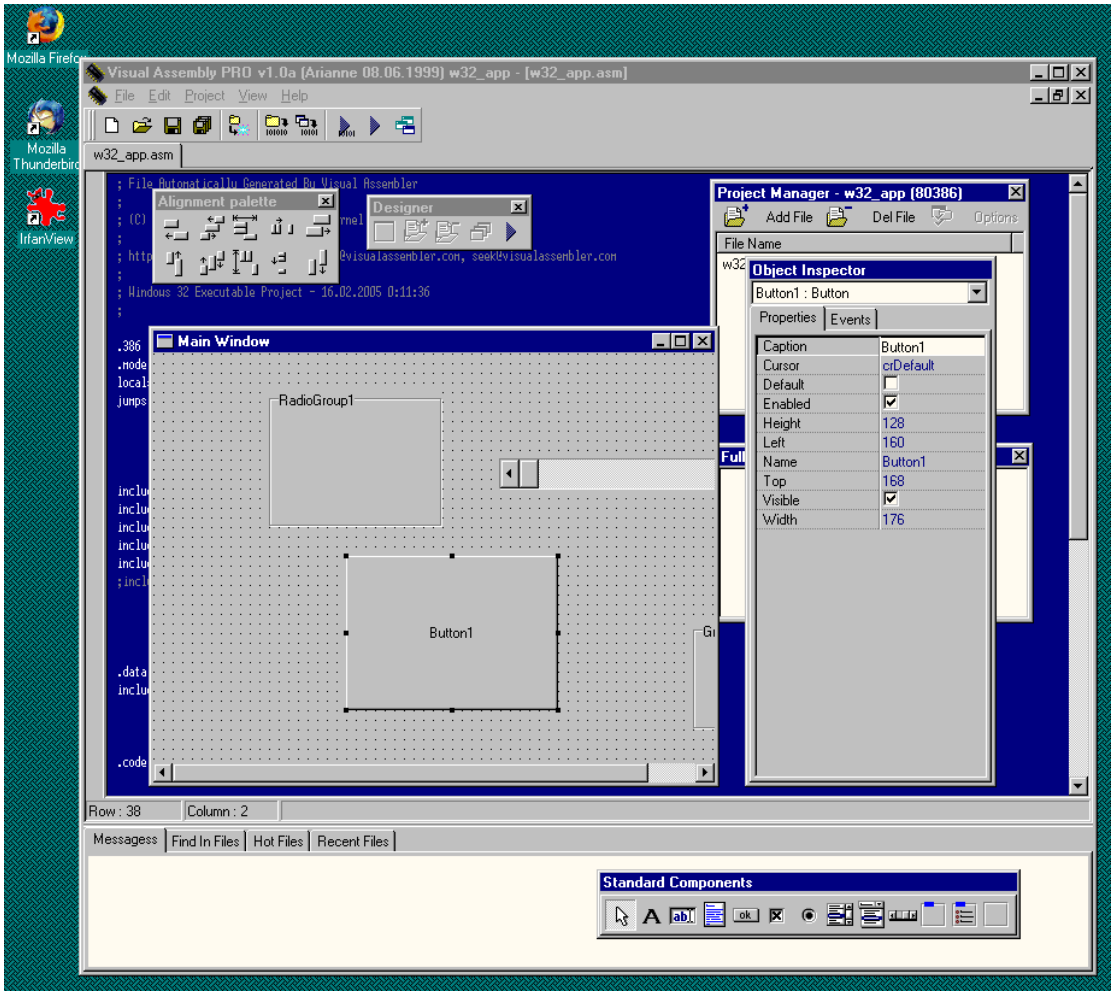

Рисунок 1 Visual Assembly Pro – визуальный ассемблер под Windows с кучей мастеров и удобным редактором диалогов (но TASMED от Solar Designer'а все равно круче)

Но ассемблером можно только побаловаться. Для большинства из нас, основной язык разработки это Си, в "конституции" которого задекламирована приближенность к аппаратуре (а, значит, и высокая эффективность). Фактически это мега-ассемблер, предоставляющий программисту полную свободу и без предупреждение отстреливающий ногу вместе с детородным органом, даже если тот совсем не это имел ввиду. Изначально ориентированный на чисто "хакерские" цели, Си завоевал признание миллионов программистов, использующих его для решения самых различных задач, как-то: низкоуровневое и высокоуровневое системное программирование, встраиваемые системы, финансовые и научные расчеты, общее прикладное программирование и т. д. При всех присущих ему недостатках (сходите на SU.C-CPP, почитайте Харона), это лучше средство для выражения программисткой мысли в наиболее естественной для него форме. Конструкции в стиле: `x == (flag?sin:cos)(y)` здесь вполне законны и являются нормой. В этом смысле Си очень похож на спектрумовский Бейсик, везде допускающий подстановку выражений, где это только возможно. Отсутствие встроенных средств для работы с массивами вкупе с доминирующей небрежностью проектирования, приводит к многочисленным ошибкам переполнения (buffers overflow), а "демократичность" работы с указателями — к утечкам памяти. Писать сетевые приложения на Си категорически не рекомендуется. Но ведь пишут же. Отсюда берутся черви, атаки на удаленные системы и прочие коварства виртуального мира.

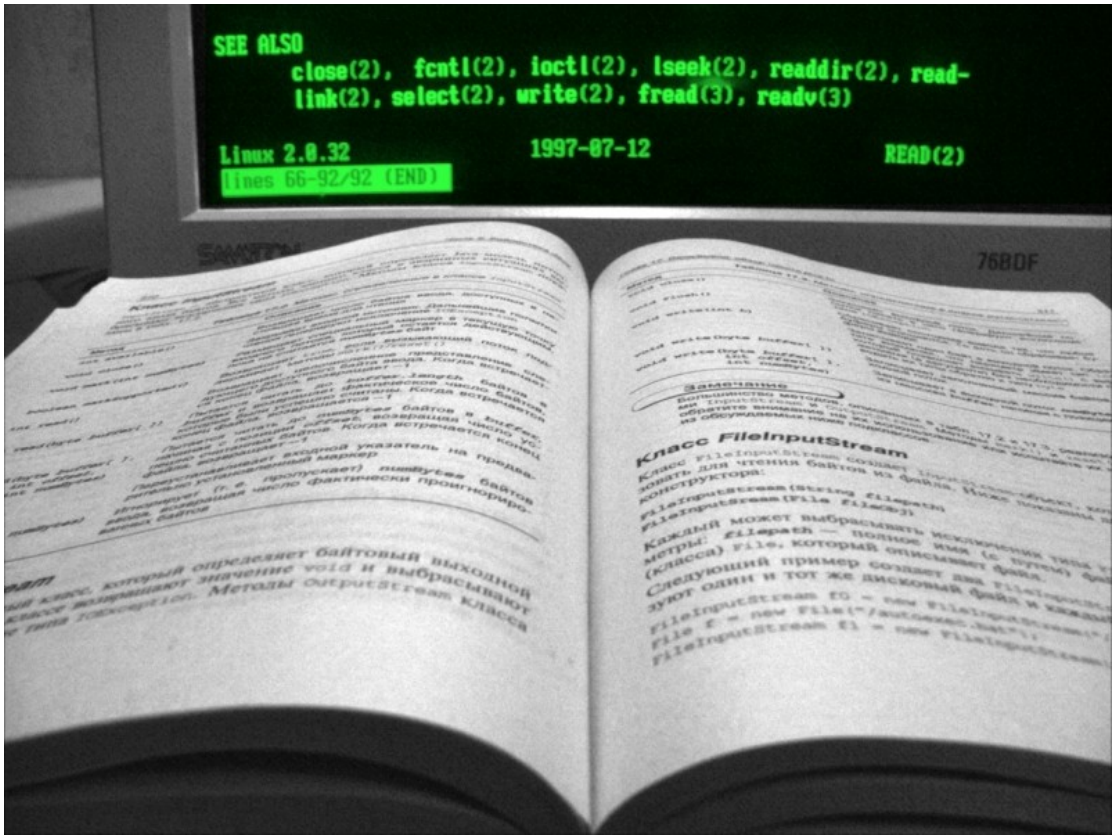

Рисунок 2 мир сишного программиста

Теперь структурное программирование считается достоянием истории. [Свое господство установил ООП] В моду вошел ООП. Си++ завладел умами программистов. Объективно-ореентированный подход пропагандируется единственно возможным способом программирования вообще и на приверженцев классического Си смотрят как на чудаков или недоучек. Прямо насилие какое-то получается! На самом деле, преимущество ООП над процедурным программированием весьма спорно и возложенные на него ожидания так и не оправдались. Ошибок не стало меньше, сроки разработки только возросли, удачных примеров повторного использования кода что-то не наблюдается, а требования к квалификации разработчиков взлетели до не небес.

Но ведь Си++ поддерживает не одну, а целых три парадигмы программирования: структурное программирование в духе улучшенного Си, абстрактные типы данных, позаимствованные с Ада, и, наконец, объективно ориентированный язык в стиле Simula. Вот, что говорит по этому поводу Бьерн Страуструп (прозванный "Дохлым Страусом"): "*При создании программы всегда есть некоторое количество вариантов, но в большинстве языков выбор сделал за вас проектировщик языка, В случае Си++ это не так — выбор за вами. Такая гибкость, естественно, непереносима для тех, кто считает, что существует лишь один правильный способ действий. Она может так же отпугнуть начинающих пользователей и преподавателей, полагающих, что язык хорош, если его можно выучить за неделю. Си++ к таким языкам не относится. Он был спроектирован как набор инструментов для профессионалов, и жаловаться на то, что в нем слишком много возможностей, — значит уподобляться дилетанту, который, заглянув в чемоданчик обойщика, восклицает, что столько разных молоточков никому не понадобиться*".

Приплюснутый Си это целый мир. Богатый ассортимент языковых возможностей еще не обязывает или пользоваться. Объективный подход бесполезен в драйверах. Сколько программисты ни пытались найти ему применения — так и не получилось, а вот парадигму "улучшенного Си" (объявление переменных по месту использования, а не в начале программы и т. д.) используют многие. Правда в драйверах (равно как и модулях сопряжения со средой) жесткая типизация приплюстного Си порождает дикий кастинг (от англ. casting – явное преобразование типов), уродуя исходный код и отнимая массу времени. Автоматической сборки мусора в Си++ нет, а, значит, от утечек памяти он не спасает (даже если используются "умные"

указатели и прочие извращения). Механизмы для работы с строками переменной длинны как будто бы появились, но переполнения буферов с завидной регулярностью встречаются и до сих пор. Так что Си++ это не панацея, а всего лишь раздутая рекламная компания. Дохлый Страус оценивал количество пользователей приплюснутым Си в 5 тыс. программистов по всему миру. Навряд ли он ошибался. Феноменальная популярность плюсов вызвана скорее высокой себестоимостью его компиляторов и вытекающей отсюда раскруткой (надо же как-то возвращать вложенное), чем техническими достоинствами.

Чистых компиляторов языка Си уже давно не существует и сейчас они поставляются вместе с плюсами. На одной лишь x86 платформе их насчитывается более десятка. Среди них есть как бесплатные, так и коммерческие, причем бесплатных значительно больше. По качеству кодогенерации лидируют Microsoft Visual C++ (входящий в состав халявных Platform SDK и Visual C Toolkit, правда IDE там нет) и Intel C++ (версия под Windows – условно-бесплатная, а под Linux – бесплатная для некоммерческого использования, однако, никто не запрещает нам компилировать системно-независимые куски кода в объектные файлы под Линухом и линковать их с Windows-приложениями). WATCOM C++, когда-то оптимизирующий круче всех, прекратил свое существование и теперь развивается в рамках проекта Open WATCOM, который по свидетельствам очевидцев больше глючит, чем работает. Borland C++ так же бесплатен, однако, с качеством кодогенерации у него кранты. Это худший оптимизирующий компилятор из всех! В мире UNIX большой популярностью пользуется GCC, портированный в том числе и на Windows, однако, под окнами он чувствует себя неуютно и особого резона в нем нет.

Рисунок 3 Бьерн Страустрап — создатель языка Си++

Паскаль, получивший второе рождение в среде DELPHI, изначально задумывался как "студенческий" язык, демонстрирующий основные концепции структурного программирования. ООП в него перетащили уже потом, да и то криво. Получилось что-то вроде морской свинки. И не свинки, и не морской, зато от одного названия сдохнуть можно. Подход, исповедуемый Паскалем, находится между Бейсиком и Си, поэтому многие называют его "игрушечным" языком программирования. Но именно такой язык и нужен разработчикам интерфейсов! Не зря Borland остановила на нем свой выбор. DELPHI намного удобнее появившегося вслед за ним C++ Builder (хотя тут можно и поспорить), но как бы там ни было, это коммерческий продукт и

он хочет денежку. Приложения, разработанные в DELPHI, с некоторыми ограничениями можно откомпилировать бесплатным транслятором Free Pascal, однако, для разработки с нуля Free Pascal не пригоден, поскольку у него нет соответствующей IDE. То есть, пригоден, конечно, но только не при визуальном подходе.

Остальные языки программирования используются намного реже. Java в основном применяется во встраиваемых системах, например, тех же сотовых телефонах. Одни производители распространяют SDK бесплатно, другие высылают его за деньги. В Web-программировании до недавнего времени активно использовался Perl, но сейчас он начинает сдавать свои позиции, уступая PHP. Оба интерпретатора бесплатны, но их дальнейшая судьба под угрозой. На рынок сетевого программирования легла грозная тень, надвигающейся эпохи .NET, за которой стоит Билл-разрушитель, а "Bill always win". Базы данных, в России традиционно писавшиеся на Шкипере и Фоксе, сейчас реализуются на встроенных макроязыках типа Visual Basic'a, обслуживающего монстров вроде Access или Excel. Про 1С-бухгалетерию я уже и не говорю. Но ведь это тоже программирование! Пускай и в наиболее извращенной его форме. Короче, языков много. Хороших и разных.

Рисунок 4 каллейдоском языков программирования

>>>> врезка мнение человека с форума

> …понятно дело, что в конечном итоге можно все к логике ИЛИ-НЕТ или И-НЕТ свести. Народ, а вы в курсе, что уже давно есть объектный ассемблер? А Си — это совсем не макроассемблер. Далековато сишному exe до ассемблерного и по объему и по скорости. Вот паскаль — это да. Это — круто. Это – офигеть. Си хорош тем, что имеет 7-8 операторов, десяток операций — и все. Его учить — плевое дело. Однако он не для трусов. Си — это свобода плюс ответственность. Почему многие так и остаются на всю жизнь на Паскале? Потому что готовы пожертвовать свободой, лишь бы ответственности поменьше. А у Паскаля настоящий тоталитаризм: шаг в сторону – расстрел. Си – это настоящая демократия. Разгильдяйство и воровство тут не проходит. Однако для людей, скажем так, с совестью – полная свобода. C++ высшая форма демократии (пока что). Ассемблер – это коммунизм. Туда дорога еще меньшему количеству народа, чем в Си. VB – гнилой капитализм. Вот почему: меньше вложить – больше заработать, пару тыков мышой – а у вас офигенное приложение, медленное – значит солидное; и еще для его приложений надо иметь крутую тачку, а крутая тачка — это престижно.

Рисунок 5 языки программирования были всегда поводом для войн и раздоров

>>> врезка мнение человека с формума

НК> …у Си++ по сравнению с Си намного больше плюсов!
ну да, ровно на два.

>>> мнение человека с формума

Як мені казав мій викладач з асму : Сі - це ассемблер для лінивих.
Хто вчив асемблер зрозуміє про що я говорю.
З цього виходить : а чого ви чекали?

>>> врезка эволюция программиста

"*Программист долен обладать способностью первоклассного математика к абстракции в сочетании с эдисоновским талантом сооружать всё, что угодно, из нуля и единицы. Он должен сочетать аккуратность бухгалтера с проницательностью разведчика, фантазию автора детективных романов с трезвой практичностью экономиста... В отношении к машине у добросовестного программиста есть одна особенность: он относится к ней как хороший жокей к своей лошади; зная и понимая возможности машины, он никогда не позволит себе компенсировать леность ума беззаботной тратой ресурсов ЭВМ*" А.П. Ершов.

"Гибель российской компьютерной промышленности и нашествие IBM PC "смыло" целое поколение талантливых разработчиков программного обеспечения, "одевавших" отечественные PDP-совместимые машины. Большинство из которых так и не смогло самореализоваться в "новой жизни". Это было уже второе "потерянное поколение" в советском программистском сообществе — первое образовалось несколькими годами раньше в связи с утерей актуальности "больших машин". Вместе с этими поколениями в значительной степени ушла и культура программирования, мигрировав в маргинальный мир UNIX-систем. Засилье IBM PC, компьютеров с предельно упрощенной архитектурой и "заточенным" под "конечного пользователя" интерфейсом привело к появлению нового поколения разработчиков — малограмотных

и амбициозных. Может быть, Автор здесь проявляет снобизм, но, по его мнению, ***человек, с детства избалованный интерактивным отладчиком, не в состоянии научиться как следует программировать"*** Акопянц Андрей Хоренович

как писать

В правильно спроектированной программе можно выделить три независимых уровня: ***слой сопряжения со средой*** (оборудованием/операционной системой), ***"вычислительная" часть*** и ***пользовательский интерфейс***. Эти уровни предъявляют различные требования как к языкам программирования, так и непосредственно к самим программистам. Обычно они реализуются различными людьми, образующими программистскую команду. Конечно, утилиту в несколько тысяч строк исходного кода можно сварганить и самостоятельно, но мы ведь не об этом сейчас говорим. Рассмотрим внутреннюю структуру типичного приложения (десятки и сотни тысяч строк исходного кода) во всех подробностях.

на глубине

Слой сопряжения со средой абстрагирует программу от особенностей конкретного окружения, сокращая трудоемкость переноса на другие операционные системы и языки программирования. Если же заранее известно, что этого заведомо не требуется, слой сопряжения со средой частично или полностью "вживляется" в вычислительную часть, что упрощает ее проектирование и программирование.

Для абстрагирования от операционной системы, на все API-функции надеваются "обертки". Хорошим примером тому служит стандартная библиотека Си — fopen и malloc работают и в Windows 3.x/9x/NT, и в UNIX и даже в MS-DOS, в то время как CreateFile и HeapAlloc — только в Windows 9x/NT. Тем самым, Си частично абстрагирует нас от операционной системы, а DELPHI/C++ Builder идут еще дальше. Слагающие их библиотеки образуют что-то вроде операционной системы в миниатюре и изучать win32 становится необязательно (только, чур я вам этого не говорил!). Простой перекомпиляции достаточно, чтобы перенести программу на Линух, а с некоторыми ограничениями и на другие операционные системы.

По понятным причинам штатные библиотеки ориентированы на сравнительно небольшой круг стандартных задач, а все, что находится за его пределами, требует тесного взаимодействия с операционной системой. Прочитать сектор с диска, намылить приятелю шею, выдернуть лоток CD-ROM и т. д. Проблема в том, что в каждой версии Windows (не говоря уже про Юних и полуось), эти задачи решаются по-разному. Часто — с применением ассемблера и прочих хитрых хаков. Их разработка требует хорошего знания операционной системы, помноженной на высокую квалификацию, но заниматься разработкой самому — необязательно. Существует масса готовых библиотек (в том числе и бесплатных), однако, качество реализации большинства из них оставляет желать лучшего, к тому же они тянут за собой заранее неизвестное количество подозрительных драйверов и вспомогательных библиотек, ожесточенно конфликтующих между собой. Но это уже — издержки цивилизации. Либо учитесь создавать свой собственный код, либо используйте то, что удалось найти в Сети.

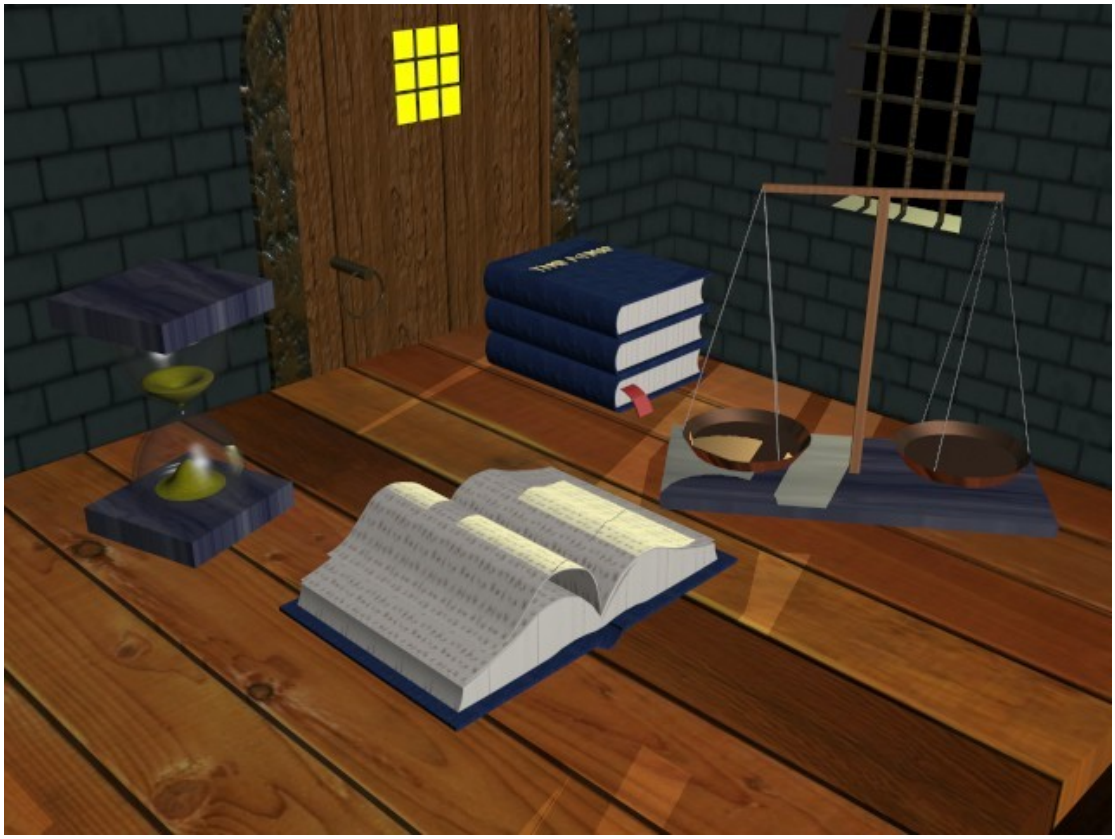

Рисунок 6 системщик должен очень много читать (и все — на английском языке), специалисты — это люди, похоронившие себя заживо в одиночестве четырех стен

Работу можно считать законченной когда в вычислительной части не останется ни одного прямого вызова API-функции. Обертки в основном пишутся на Си с редкими вкраплениями ассемблера и оформляются как DLL или статически компонуемый модуль. Опытные программисты используют одну и туже библиотечку для всех своих проектов, поэтому DLL в большинстве случаев все же предпочтительнее (но не забывайте, что большое количество явно прилинкованных DLL ощутимо замедляет загрузку приложения). Использовать Си с плюсами здесь в общем-то нежелательно. Во-первых, исходный текст получается более избыточным, а во-вторых, "манглеж" приплюснутых имен не стандартизирован. Динамическую библиотеку, скопированную Borland C++, к Visual C++ проектам подключить совсем не просто (впрочем, как и наоборот). Паскаль здесь категорически непригоден — отсутствие нормального интерфейса с операционной системой, вынуждает ходить окружным путем, то есть через жопу.

Иногда возникает необходимость обратиться к защищенным ресурсам, доступным только из режима ядра (например, вызвать привилегированную команду процессора). Программисты старого поколения в этом случае пишут псевдодрайвер. С точки зрения операционной системы псевдодрайвер выглядит как обыкновенный драйвер, но в отличии от него не управляет никакими устройствами, а просто выполняет привилегированный код, общаясь с прикладным приложением через интерфейс IOCTL. Псевдодрайвера пишутся на Си с небольшой примесью ассемблера. Чистый ассемблер более элегантен, но экономически невыгоден. Приплюснутый Си содержит подводные камни. Не используйте его, если точно не уверены, что именно вы делаете. Для облегчения разработки драйверов фирма NuMega выпустила пакет Driver Studio и хотя многие от него без ума, общее впечатление отрицательное. Лучше купите "Недокументированные возможности Windows 2000" Свена Шрайбера. Там вы найдете готовый скелет псевдодрайвера, который легко адоптировать под свои нужды, не отвлекаясь на изучение посторонней херни [посторонних дисциплин]. Начинающие могут использовать готовые библиотеки, дающие с прикладного уровня доступ к портами и прочим системным компонентам (исходный текст одной такой библиотеки приведен в моей книге "Техника защиты лазерных дисков от копирования"). Естественно, такой подход небезопасен и совсем не элегантен.

Возникает дилемма — либо пыхтеть над DDK (тысячи страниц и все на программистском международном — он же английский), либо поручить эту работу системному

программисту. Разумеется, не бесплатно. Но самостоятельное изучение режима ядра обойдется еще дороже. Эта не та область, которую можно постичь за месяц или два. В программировании драйверов есть множество неочевидных тонкостей, известных только профессионалам. Умение совмещать в себе проектирование баз данных с разработкой низкоуровневых компонентов даровано не каждому. Сосредоточьтесь на чем-то одном. Том, что у вас получается лучше всего. В противном случае, вы впустую протрахаете время, не получив ни денег, ни удовольствия.

Рисунок 7 программисты любят пиво

в ядре

"Вычислительная" часть (так же называемая "движком") — сердце любого приложения и основной "интеллект" сосредоточен именно здесь. "Вычисления" — это не только математические расчеты, но и любая обработка данных вообще. В частности, вычислительная часть Тетриса трансформирует фигуры, посчитывает очки, убирает заполненные строки, обрабатывает наложения спрайтов и т. д.

Чаще всего движок пишется на приплюснутом или классическом Си, реже — на Фортране, Паскале и других экзотических языках. Выбор определяется личными пристрастиями программиста с одной стороны и возможностями языка с другой. А язык это не только компилятор, но еще и его окружение — среда разработки, отладчик, верификаторы кода, библиотеки, системы обнаружения утечек памяти и т. д. Многие останавливаются на Visual Studio только из-за ее IDE. Интегрированный отладчик с перекомпиляцией на лету, автозавершение имен функций, удобные мастера. С непривычки все это здорово заводит и возбуждает, но не доходя до оргазма эйфория кончается. Мастера тесно завязаны на MFC, а MFC использует нестандартные расширения и вообще говоря непереносим. Можно, конечно, писать и без мастеров, но что тогда остается от IDE? А текстовой редактор можно найти и покруче. Системы контроля и визуализаторы данных находятся в глубоко зачаточном состоянии. Основным лекарством становится отладчик, из-под которого не вылезаешь ночами, но с ошибками синхронизации потоков он все равно не справляется. В мире UNIX, где в основном используется "тяжелая" многозадачность, этих проблем просто не возникает, да и ассортимент инструментальных средств там побогаче. Есть мнение, что дешевле

запрограммировать вычислительную часть на Линухе и затем перекомпилировать под Винды, чем писать на Visual C++. Трудности разработки с лихвой компенсируется легкостью отладки. Правда, для этого следует проникнуться идеологией GDB – самого "правильного" отладчика в мире. Он совсем не похож на Turbo-Debugger и намного более продвинут чем Soft-Ice (правда, совершенно непригоден для взлома, но взлом — это дело другое).

Прежде, чем опускать лапы на клаву, мучительно соображая чтобы такого сейчас написать, обязательно обшарьте все уголки Сети на предмет поисков уже готового кода. Программирование возникло не сегодня и не вчера. Все что только было можно написать, уже написано! Допустим, вам потребовалась своя версия кодека G.729 для создания мини-АТС или организации телеконференции. Писать все с нуля? Мы что, рехнулись?! Открываем Гугла, выясняем, что стандартизацией данного протокола занимается комитет International Telecommunication Union, представленный сайтом www.itu.int, где (правда не забесплатно) можно заказать не только описание самого алгоритма сжатия/разжатия, но и исходные тексты кодеков с комментариями. Зная конкретно что именно мы ищем, файлы легко добываются в Осле (правда, их там 600 метров с гаком, но Осел животное терпеливое, и не такие объемы перетаскивал). Как вариант, можно скачать библиотеку Intel IPP в состав которой входит несколько версий кодека, оптимизированных под MMX и SSE. Помимо этого, в процессе поисков обнаруживается добрый десяток "студенческих" реализаций вполне приемлемого качества. Доступность исходных текстов большинства UNIX-приложений, превращает программирование из творческой задачи в азартных поиск готовых кусков кода, которые порою обнаруживаются в самых неожиданных местах. Конечно, при этом всегда существует риск нарваться на чью-то ошибку (и тщательно замаскированную закладку в том числе), но… искушение всегда побеждает соблазн.

Разработчик движков — хороший алгоритмист и отчасти даже математик. Знания ассемблера и устройства операционной системы приветствуются, но в общем-то необязательны. Зато свой непосредственный язык программирования, разработчик должен знать от и до, используя все предоставляемые им возможности на полную катушку.

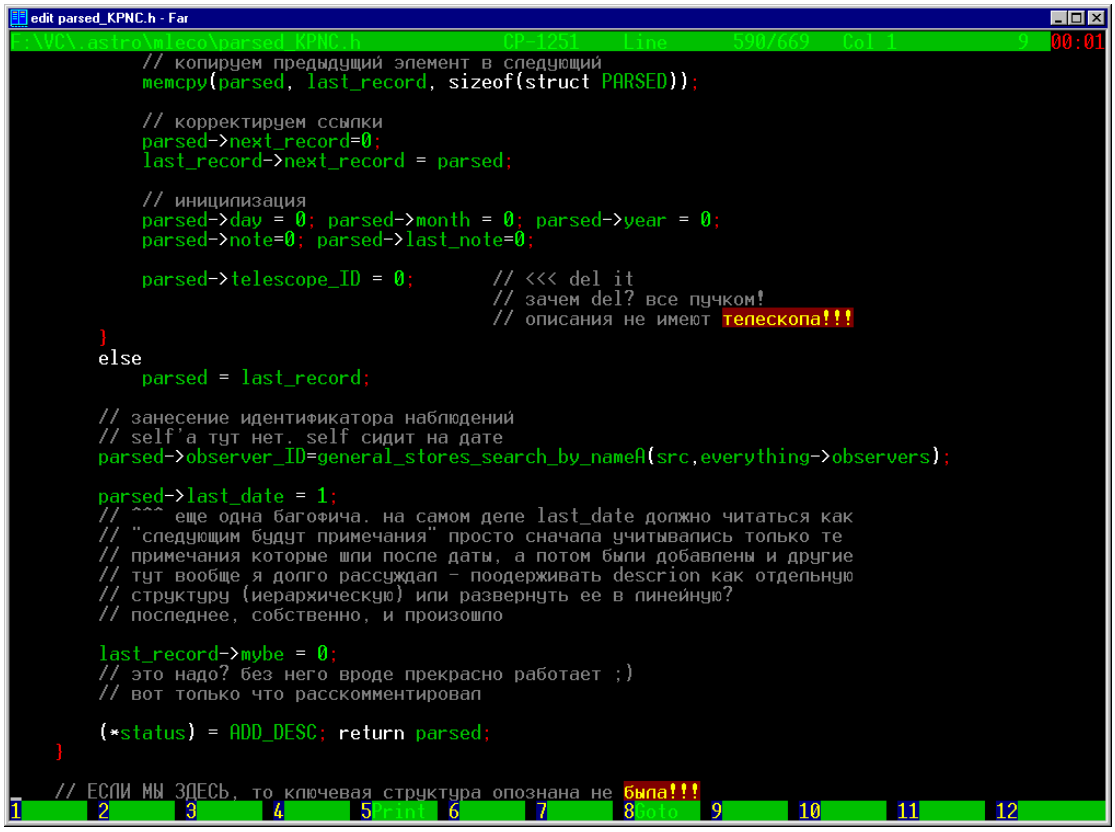

Рисунок 8 FAR – самое правильное IDE с точки зрения системщика и продвинутого прикладника

на поверхности

Пользовательский интерфейс — это лицо и одежка программы, зачастую отнимающее более половины общего времени разработки проекта (а некоторые приложения, например, бух или склад, из одного интерфейса и состоят, "вычислительного" кода там очень немного). Интерфейс необязательно должен быть графическим. Командная строка и консольный режим здравствуют и по сей день, однако, сфера их применения ограничена узким кругом (или можно даже сказать "клубом") матерых профессионалов и кворума мы не наберем, а выйти на массовый рынок без иконок нереально (вот она, скрытая религиозность!).

Эту интеллектуально-непритязательную, но трудоемкую работу целесообразнее всего поручить "пионерам" — начинающим программистам с дизайнерской жилкой. Ведь разработчик интерфейсов в первую очередь художник, а уже потом программист. Использование готовых пиктограмм не только безвкусно, но и пошло. Всякая программа должна иметь свой собственный, легко узнаваемый "фирменный" стиль, выполненный в единой цветовой гамме и объединенный общей идеей. Стандартные ресурсы, входящие в комплект штатной поставки Visual Studio, ни на что не годятся (программы, написанные "для себя", мы в расчет не берем).

Интерфейс быстрее всего пишется на DELHI/C++ Builder/Visual Basic/Visual C++/.NET и прочих системах быстрой разработки приложений. Компакность кода и его быстродействие, конечно, оставляют желать лучшего, ну да кто на них обращает внимание? Главное — опередить конкурентов, не дав им первыми выйти на рынок. DELPHI тем предпочтителен, что под него можно найти любые готовые компоненты на все случаи жизни. Позиция .NET несмотря на масштабный маркетинг выглядит как-то неубедительно и программисты все еще осторожничают с переходом. Почему? Главным нововведением в .NET стала виртуальная машина (.NET Framework) с промежуточной компиляцией приложений в P-код. Идея далеко не нова (даже на ZX-Spectrum, кажется, было что-то подобное). Теоретически все выглядит блестяще — программист пишет программу на Visual Basic'e, Visual C++ или C# (клон Java), и она выполняется на любом процессоре и под любой операционной системой, для которой эта виртуальная машина существует. Что за херня! Снимите лапшу с ушей! Если есть прямые вызовы API-функций или управление оборудованием о переносимости можно забыть. Если же их нет, исходный код, написанный в ANSI стиле, транслируется любым ANSI-совместимым компилятором, без всякой виртуальной машины, кстати говоря, съедающий львиную долю производительности. Под .NET существует несколько достойных библиотек для создания Web-приложений, взаимодействия с серверами баз данных и т. д. и т. п., но до изобилия готовых компонентов, которыми славится DELPHI, она явно не дотягивает. Возможно, со временем ситуация и переменится (а, учитывая рьяную озабоченность Microsoft, она переменяется наверняка), но в настоящий момент времени, .NET выигрывает лишь на тех задачах, на которые ее сориентировали — то есть на сетевые приложения. Но вернемся к переносимости. Платформа .NET позиционируется как среда открытых стандартов. Язык виртуальной машины описывается документом ECMA-335 (бесплатная pdf-версия которого может быть скачена с http://www.ecma-international.org/publications/standards/Ecma-335.htm), а C# – ECMA-334 (http://www.ecma-international.org/publications/standards/Ecma-334.htm). Любой производитель может создавать собственную реализацию платформы .NET, не спрашивая у Microsoft разрешения и не платя никаких отчислений.

Известно по меньшей мере два проекта переноса .NET в среду UNIX. Первый, спонсируемый Free Software Foundation (или сокращенно — FSF), носит название DotGNU (http://www.dotgnu.org/) и развивается в рамках одноименного проекта, частью которого является компилятор GCC. Второй проект называется Mono (http://www.go-mono.com/) и спонсируется компанией Ximian, распространясь под лицензий GPL/LGPL и MIT License, что в ряде случаев намного предпочтительнее. Найти его можно в дистриьюбтиве Федоры. Знакомство с обоими проектами вызывает лишь разочарование. Для реальной работы они непригодны. Из любви к искусству, конечно, можно и пострадать, истязая свою задницу замученную бесконечным сидением у монитора, но программист с нормальной половой ориентацией не задумываясь выберет DELPHI/Free Pascal или Visual C++ с MFC, бесплатные порты которой под UNIX уже имеются.

А вот для Java-программистов, открытый C# это настоящая находка (что, собственно и не удивительно, ведь его разработкой руководил Anders Hejlsberg. Да! Да! Тот самый Anders Hejlsberg, кто создал DELPHI и Турбо-Паскаль). Теперь судьба проекта не будет зависеть от правого мизинца левой пятки главы корпорации SUN! Впрочем, в программировании интерфейсов возможности и удобство языка глубоко вторичны и все опять-таки упирается в

готовые библиотеки и компоненты. В этом смысле Java застряла между голым win32 API и MFC. То есть, запрограммировать интерфейс на ней можно, но ведь это надо ***_программировать_***, то есть кодить, а не мыша по коврику гонять!

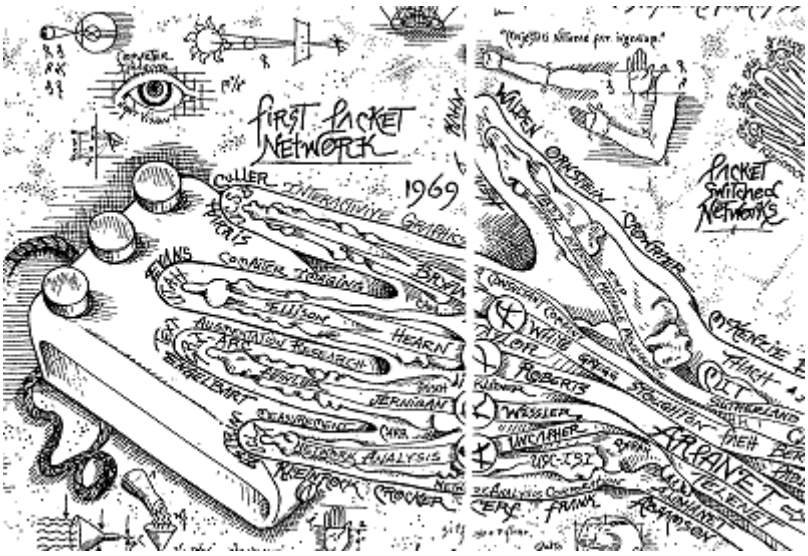

Рисунок 9 гадание по мыше

Кстати, о мышах. Визуальные технологии программирования прививают новичкам дурной стиль, от которого потом приходится долго отучиваться. Тащим кнопку на форму, щелкаем по ней клавишей, и в автоматически отрывшемся окне редактора пишем код обработчика. Это быстро, но идеологически неправильно, трудно расширяемо и совершенно непереносимо. Пока проект мал, особых проблем не возникает, но с каждым разом вносить изменения становится все труднее и труднее — даже незначительное исправление требует переделки огромных кусков исходного текста в десяти местах. Структура программного кода сворачивается в запутанный клубок, хитросплетения которого уже не удерживаются в голове и проект рушится прямо на глазах.

Учитесь программировать правильно, учитесь программировать красиво. Обращайтесь к мастерам только тогда, когда это действительно необходимо, оторвите мышу хвост, запустите текстовой редактор, встроенный в FAR, и приучайтесь кодить руками (руки — это такие штуки, которые под головой, правда, у некоторых они находятся рядом с ногами). Только так пионеры превращаются в настоящих профессионалов!

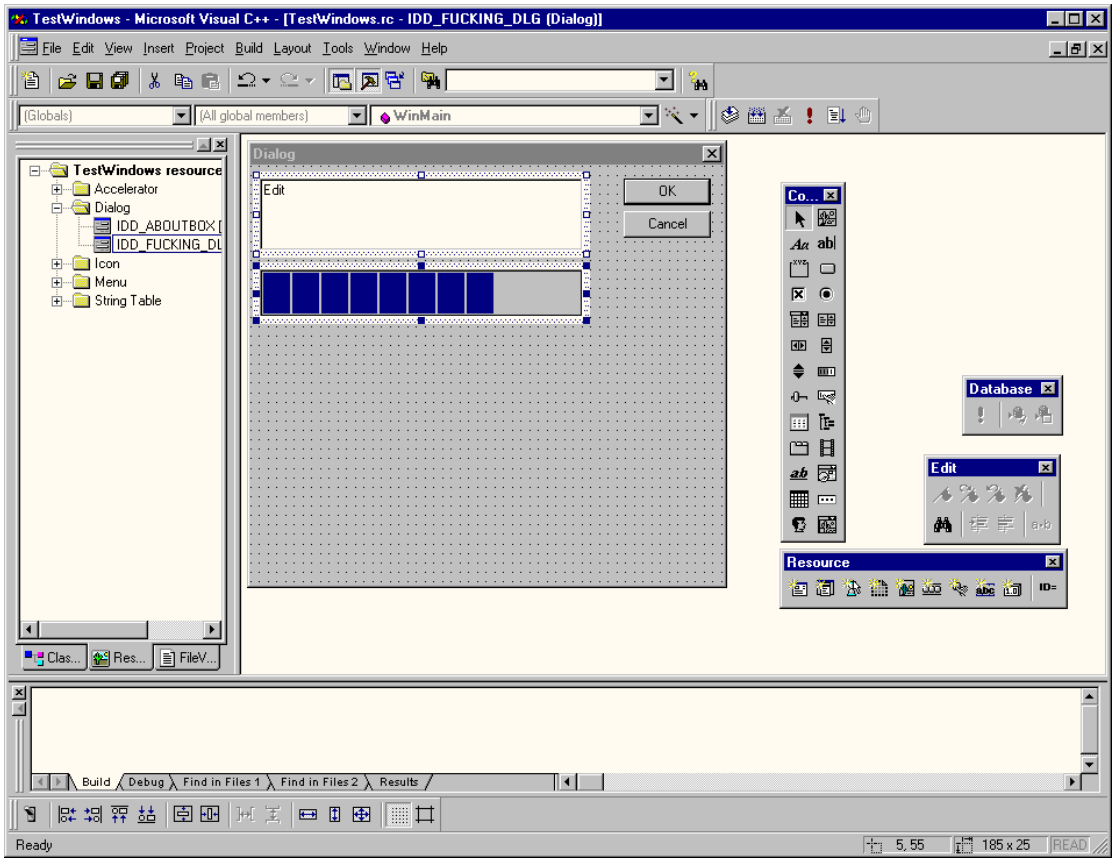

Рисунок 10 разработка интерфейсов с помощью Microsoft Visual Studio 6.0

>>>> врезка

Считается, что создание защитного механизма не на ассемблере может дорого обойтись. Это неверно. Дороже всего обходится защитный механизм, реализованный на ассемблере. Во-первых, его очень просто ломать (объем исследуемого кода очень невелик, одна строка ассемблерного кода транслируется в одну машинную команду, в то время как на языке высокого уровня можно и миллион машинных команд в одной строчке наваять). Во-вторых, защиту от "пионеров" можно состряпать на любом языке программирования (хоть Бейсике, хоть Си), а от профессионалов лучше вообще не защищаться — все равно взломают. В-третьих, правильные защитные механизмы реализуются на эмуляторах машины Тьюринга, стрелки Пирса, сетей Петри и т. д.

Рисунок 11 хакерский взгляд на Windows

>>>> врезка

Многие путают язык с компилятором, а компилятор со средой разработки, на самом деле это три разных сущности. Язык это просто набор деклараций, описываемых тем или иным Стандартом (например, ANSI). Компилятор — штука, которая переваривает исходный текст, написанный в соответствии с декларациями языка, и переводит его в легко усваиваемую промежуточную форму или сразу в машинный код. Теоретически, исходный текст без всякой переделки должен послушно транслироваться любым компилятором, однако, в реальной жизни это не так и производители компиляторов вносят в язык свои собственные, зачастую с чем не

совместимые расширения. Свой отпечаток накладывает и архитектура процессора (векторный он или скалярный), модель памяти операционной системы (сегментированная в MS-DOS и Windows 3.1 и плоская в UNIX и Windows 9x/NT) и т. д. Наконец, некоторые производители пытаются соблазнить программистов "улучшением" языка, теми же инстиксами, например. Про специализированные компиляторы, созданные для нестандартных процессоров и микроконтроллеров, мы вообще промолчим. Никакой стандартизацией тут и не пахнет.

Выбирая компилятор, вы выбираете судьбу. Программировать в ANSI/ISO стиле (без использования нестандартных расширений и специфических особенностей данного транслятора), во-первых очень муторно и неэффективно, а во-вторых — бессмысленно. Компиляторы придерживаются Стандарта лишь в глобальных вопросах, но расходятся в мелочах. Забросьте Стандарт и руководствуетесь собственным опытом и знанием "характера" различных компиляторов. На сайте www.mozilla.org есть много статей на эту тему. Создание программы, транслируемой более чем одним компилятором, требует значительных усилий, которые окупаются в том, и только в том случае, когда вы пишите кросс-платформенное приложение, портированное на десяток операционных систем. В противном случае, лучше забить о геополитике, используя все "вкусности" и расширения конкретно выбранного компилятора (даром, что ли их разработчики создавали). Зачем стоить автомашину, легко превращающая в подводную лодку и вертолет, если все что вам нужно — раз в неделю ездить с семьей на дачу?

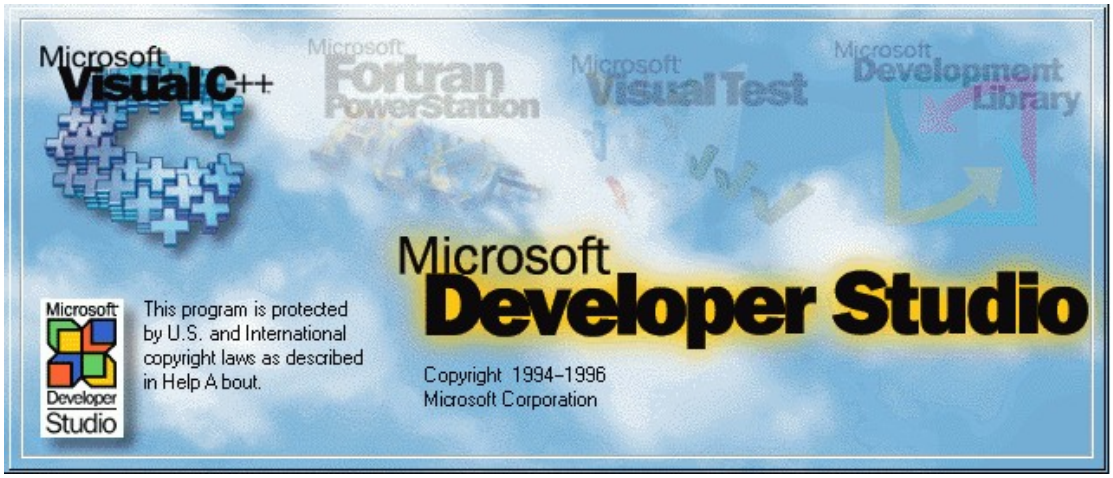

Рисунок 12 Microsoft Visual C++ 4.0 – старый но все еще рабочий

заключение

Концепция разделенного труда гласит, что над каждым компонентом проекта должен работать свой человек: дизайнер, художник, кодер, архитектор, системный программист. Еще потребуется менеджер (чтобы управлять этой шоблой, которую Word упрямо предлагает заменить на "воблу" — то ли он этого слова не знает, то ли намекает на любовь к пиву), юрист (для составления контрактов и разборок с авторским и патентным правом) и… образуется целая корпорация. А, как показывает практика, по-настоящему удачные продукты чаще всего реализуются в одиночку или тесным коллективом из двух-трех сплоченных общим духом людей. За примерами далеко ходить на надо: Си, IDA Pro или даже Olly Dbg родились именно так.

Проекты, требующие усилий семи-восьми человек еще создаются (народ встречается в курилке и обо всем договариваться), но вот написать свою собственную Windows или Office нам (русским программистам) не удавалось, и не удастся никогда. Нет, программировать мы умеем! А вот руководить — нет. Критикующие Била Гейтса просто не представляют себе какие проблемы возникают при разработке программных комплексов такого масштаба. Почитайте "Мифический человеко-месяц" Брукса — многие станет понятно.

Программирование — это не только проектирование, алгоритмизация, кодинг, etc. Это еще и управление персоналом. Без яркого лидера, авторитету которого подчиняются все члены группы, команда расколется, а продукт развалится. Поэтому, помимо обычного разделения

труда на системщиков, прикладников и дизайнеров, требуется определенная психологическая взаимодополняемость. Ведь противоположности притягиваются!

Рисунок 13 у компьютера