Копипаста:Программирование — Lurkmore

Разнообразная копипаста, связанная с программированием, живущая преимущественно в соответствующих разделах борд.

Зачем вы вообще, блядь, сюда залезли?

Пока вы, блять, будите лезть в этого говно, оно никогда не сдохнет. Я понимаю еще, человек написал: "Посоны, я угорел по гейдеву, потому решил выучить плюсы, чтобы написанные мною игры летали! Посоветуйте годноту". А то пришел, ему видите ли "Нужно понятие о классах и работой с памятью", так пойди попробуй мелочи стрельнуть у бритых пацанчиков в шапках-годонках, они тебе объяснять по понятиям. Может выясниться, что тебе ни классы ни память не нужны. Неужели, блять, так сложно прикинуть, за каким хуем ты вообще лезешь в отрасль. "рассчитано на грамотных людей" - уебывай нахуй. Сколько вас таких тут было: "Хочу сдать программистом! Какой язык выбрать для начала?", "Хочу выучить язык %langname%. Посоветуйте литературы". Кто-то подрывается и отвечает вам, может даже по хардкору палит годноту. А вы, все те самые, которые поматросят и бросят. Дальше хелло-ворда дело не идет, не говоря уж о работе с памятью Неужели вы и вправду думаете, что программирование/кодинг/хакирство стильно-модно-молодежно и, того глядишь, на практике пригодится? Напряги свое серое вещество и подумай, нахуя тебе это. Просто оно тебе не надо, иначе бы ты не создавал здесь очередной хуевый тред, а загуглил бы давно нашел статьи на хабре/рсдн/100_других_ресурсов. Люди составили тысячи подборок с описаниями, рецензиями - выбирай и читай. Но нет, блять, вместо этого очередной хуй припиздовывет в кодач и устраивает симуляцию деятельности. Нахуя, скажи блять, подбирать тебе книгу, если ты ее даже не прочтешь? Ф пизду вас, мудаков!

Забей на SICP/TAPL/HtDP парашу

Забей на SICP/TAPL/HtDP парашу. Сразу читай Lambda The Ultimate, затем OnLisp, наконец плавно переходи к Introduction to MIPS Architecture - идеальная архитектура для реализации своего первого Lisp-компилятора на основе CPS. Если интересуют структуры данных, то только MIT-овский 6.851 Advanced Data Structures. Виртом и Ахо-Копрофтом можешь сраку подтереть. Кнута не читай, ему есть адекватная замена - книга Hacker"s Delight.

Ответ

Забей на Lambda The Ultimate/OnLisp/Introduction to MIPS Architecture парашу. Сразу читай SICP, затем TaPL и наконец плавно переходи к HtDP - идеальное руководство для реализации своего первого первого полноценного проекта. Если интересуют структуры данных, то только Аховский Data Structures and Algorithms. Маккарти и Тернером можешь сраку подтереть. Уоррена не читай, ему есть классическая замена - книга The Art of Computer Programming.

Dive into Python

> *Основная статья:* ***Копипаста:Python***

Завтра ищешь в интернете книжку Dive into python. Похуй если ничего не поймешь. Затем идешь на python.org и изучаешь стандартную библиотеку от корки до корки. Потом зубришь, именно, сука, вызубриваешь конвенцию по написанию питоньего кода - PEP8, чтобы от зубов отскакивало. Когда напишешь свою первую имиджборду, по пути изучив верстку на html+css, скачиваешь и изучаешь любой питоний асинхронный вебсервер, рекомендую Tornado или Gevent. Как переделаешь имиджборду, чтобы выдавала по крайней мере 5 тысяч запросов в секунду, можешь идти дальше - тебя ждет увлекательный мир хайлоада. Apache Hadoop, сверхбыстрые асинхронные key-value хранилища, MapReduce. Отсос хиккующих выблядков / просто неудачников типа рейфага или сисярп/джава-хуесосов, которые сосут хуй по жизни не заставит себя ждать и уже через пол года ты будешь получать такие суммы, что любая баба будет течь[1] при одном упоминании твоей зарплаты.

Компиляция

Забей на SICP/TAPL/HtDP парашу. Завтра ищешь в интернете книжку Dive into python и перепидорашиваешь под себя, что грузовой бобрик будет течь при одном упоминании капустный бархан. Ебать, если вы не понимаете.

Только хардкор!

> *Основная статья:* ***Копипаста:Мытищи***

- ПАЦАНЫ, Я КОРОЧЕ ШЕЛ СЕГОДНЯ ПО ОФИСУ И УВИДЕЛ ПРОГЕРА В МАЙКЕ "C# IS THE FUTURE", НУ Я ПОДСКОЧИЛ К ЕГО КОМПУ И РЕЗКО НАПИСАЛ "std::cout << std::endl;" И ПОЯСНИЛ ЭТО ОДНОСТРОЧНЫМ КОММЕНТОМ, ПОТОМУ ЧТО Я УГОРЕЛ ПО ПЛЮСАМ, ПАЦАНЫ ДУХ ООП ЖИВЕТ ТОЛЬКО В МНОЖЕСТВЕННОМ НАСЛЕДОВАНИИ, ГДЕ В ПЕРЕГРУЗКЕ ОПЕРАТОРОВ НА ОДИН ПАРАМЕТР МЕНЬШЕ, ГДЕ ЕБАШАТСЯ ПО СТАТИЧЕСКИМ МАССИВАМ, ГДЕ ПАЦАНЫ ЖИВУТ УКАЗАТЕЛЯМИ, ЧИСЛЕННЫМИ ЗНАЧЕНИЯМИ В УСЛОВИЯХ И ЕБУТ СБОРЩИКИ МУСОРА В РОТ! ТОЛЬКО C++, ТОЛЬКО ХАРДКОР!!! СТРАУСТРУП ХАРДКОР C++!!! пацаны ебашьте дельфикодеров, шарперов, пехапешников, жаверов, угорайте на компиляции в машинный код, любите Страуструпа, плюсокодеров и IDE! ГОВОРИТЕ ОТКРЫТО И СМЕЛО ПРЯМО В ЛИЦО! C++!
- ПАЦАНЫ, Я СЕГОДНЯ ШЁЛ КОРОЧЕ ПО ОФИСУ И УВИДЕЛ ЗАПУЩЕННУЮ СПЕРМЕРКУ А РЯДОМ КЛОУНА В МАЙКЕ "ENTERPRISE PROGRAMMER", НУ Я ПОДСКОЧИЛ И РЕЗКО ДЕИНСТАЛЬНУЛ НА МАШИНЕ NET ФРЕЙМВОРК К ХУЯМ: И ПОЯСНИЛ ЕГО КРИКОМ "НЕ ЛЮБЛЮ УПРАВЛЯЕМЫЙ КОД", ПОТОМУ ЧТО Я УГОРЕЛ ПО ДИСТРИБУЦИИ РЕГИСТРОВ, ПАЦАНЫ ДУХ СТАРОЙ ШКОЛЫ ЖИВЁТ ТОЛЬКО В СТАТИЧЕСКИ ЛИНКУЕМЫХ ЛИБАХ, ГДЕ ИНКРЕМЕНТИРУЮТ УКАЗАТЕЛИ, ГДЕ КОДЕРЫ ЖИВУТ KERNEL.DLL, USER32.DLL И ЕБАШАТ АНСЕЙВ КОД! ТОЛЬКО ПУР СИ, ТОЛЬКО ФАСМ!!! ЮНИТИ УЛЬТРАХОРДКОР mov edx, dword [esp+4*eax+8]!!! Пацаны, компиляйте в нейтив, дебажте идой, прописывайте относительные смещения, сбрасывайте регистры флагов, цените свободу! ПИШИТЕ БЛОКНОТОМ СМЕЛО И ПРЯМО В БИНАРНИК! 0xDEADBEEF!
- ПАЦАНЫ, Я СЕГОДНЯ ШЁЛ КОРОЧЕ ПО ГОРОДУ И УВИДЕЛ ЧЕЛА В МАЙКЕ "Я ЛЮБЛЮ СИШАРП", НУ Я ПОДСКОЧИЛ И РЕЗКО ПЕРЕЕБАЛ ЕМУ В ЩЩИ С ВЕРТУШКИ И ПОЯСНИЛ ЕГО КРИКОМ "НЕ ЛЮБЛЮ ООП", ПОТОМУ ЧТО Я УГОРЕЛ ПО ЛИСПУ, ПАЦАНЫ ДУХ СТАРОЙ ШКОЛЫ ЖИВЁТ ТОЛЬКО В ФУНКЦИОНАЛЬНОМ ПРОГРАММИРОВАНИИ, ГДЕ ЕБАШАТСЯ ПО ХАРДКОРУ, ГДЕ ПАЦАНЫ ЖИВУТ ЛИНЕЙНЫМИ СИМВОЛАМИ, ХВОСТОВОЙ РЕКУРСИЕЙ И ЕБУТ СИСТЕМУ В РОТ! ТОЛЬКО ЛИСП ТОЛЬКО ФУНКЦИИ, ТОЛЬКО СКОБКИ!!! ЮНИТИ УЛЬТРАХАРДКОР ЛИСП!!! пацаны ебашьте дотнетоблядей, сишарперов, одинесников, формошлепов, быдлокодеров, угарайте по рекурсии любите Лисп, списки и Скобки! ГОВОРИТЕ ОТКРЫТО И СМЕЛО ПРЯМО В ЛИЦО! ЛИСП!
- ПАЦАНЫ, Я СЕГОДНЯ КОРОЧЕ ШЁЛ КОРОЧЕ ПО ОФИСУ И УВИДЕЛ ЗАПУЩЕННУЮ RATIONAL ROSE ENTERPRISE А РЯДОМ КЛОУНА В МАЙКЕ "WATERFLOW MODEL FTW", НУ Я ПОДСКОЧИЛ И РЕЗКО ДЕИНСТАЛЬНУЛ НА МАШИНЕ РОЗУ К ХУЯМ: И ПОЯСНИЛ ЕГО КРИКОМ "НЕНАВИЖУ НЕАГИЛЬНЫЕ МЕТОДЫ РАЗРАБОТКИ", ПОТОМУ ЧТО Я УГОРЕЛ ПО ЭКСТРЕМАЛЬНОМУ ПРОГРАММИРОВАНИЮ. ПАЦАНЫ ДУХ УСПЕШНЫХ ПРОЕКТОВ ЖИВЁТ ТОЛЬКО В ИТЕРАТИВНОЙ РАЗРАБОТКЕ, ГДЕ КОДЕРЫ ЖИВУТ ИТЕРАЦИЯМ, СПРИНТАМИ, ЕЖЕДНЕВНЫМИ СТАНДАПАМИ И ЕБУТ ДОКУМЕНТАЦИЮ В РОТ! ТОЛЬКО СКРАМ, ТОЛЬКО ХАРДКОР!!! КЛИНКОД, ХАРДКОР XP!!! пацаны ебашьте архитекторов, бизнесаналитиков, менеджеров, угорайте по общению с клиентом, а не с контрактом, цените свободу!! ГОВОРИТЕ ОТКРЫТО И СМЕЛО ПРЯМО В ЛИЦО! АГИЛЬНОСТЬ!
- ПАЦАНЫ, Я СЕГОДНЯ ШЁЛ КОРОЧЕ ПО РЫНКУ И УВИДЕЛ ЗАДРОТА В МАЙКЕ "HASKELL", НУ Я ПОДСКОЧИЛ И РЕЗКО ПЕРЕЕБАЛ ЕМУ В ЩЩИ С ВЕРТУШКИ И ПОЯСНИЛ ЕГО КРИКОМ "НЕ ЛЮБЛЮ ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ", ПОТОМУ ЧТО Я УГОРЕЛ ПО ООП, ПАЦАНЫ ДУХ СТАРОЙ ШКОЛЫ ЖИВЁТ ТОЛЬКО В КЛАССАХ, ГДЕ НАСЛЕДУЮТ И ПЕРЕОПРЕДЕЛЯЮТ МЕТОДЫ, ГДЕ ПРОГРАММИСТЫ ЖИВУТ ИНКАПСУЛЯЦИЕЙ, ПОЛИМОРФИЗМОМ И ЕБУТ ЛЯМБДА ИСЧИСЛЕНИЕ В РОТ! ТОЛЬКО C++, ТОЛЬКО ЯВА!!! пацаны ебашьте лисп, хаскель, эрланг, угорайте на питоне, любите свой Smalltalk, Ruby и Perl! ГОВОРИТЕ ОТКРЫТО И СМЕЛО ПРЯМО В ЛИЦО! ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ!
- АНОНИМНЫЕ ЭКСПЕРТЫ, Я СЕГОДНЯ ШЕЛ ПО ИНСТИТУТУ И ВДРУГ УВИДЕЛ ПИТОНОЁБА В ФУТБОЛКЕ "I LOVE PYTHON", НУ Я РЕЗКО ПОДСКОЧИЛ К ДЕКАНАТУ, РЕШИТЕЛЬНО ЗАБРАЛ ДОКУМЕНТЫ И ПОЯСНИЛ ИХ КРИКОМ "НЕ ЛЮБЛЮ ПРЕСМЫКАЮЩИХСЯ"! ПОТОМУ ЧТО Я УГОРЕЛ ПО RUBY, ПАЦАНЫ, ДУХ Smalltalk ЖИВЕТ ТОЛЬКО В РУБИ, ГДЕ ФУЛЛ ООП, ГДЕ РУБИСТЫ ЖИВУТ МЕТАПРОГРАММИРОВАНИЕМ И ЕБУТ ОБГВИДКОВ В РОТ! ТОЛЬКО RUBY, ТОЛЬКО ДИНАМИКА, ТОЛЬКО ООП!!! ПРИМЕСИ ИТЕРАТОРЫ ОБЪЕКТЫ!! РУБИСТЫ, не учитесь со ПЕТОНОЁБАМИ, УГОРАЙТЕ в настоящем ЯЗЫКЕ, любите РУБИ, ОТКРЫТО И СМЕЛО ОТЧИСЛЯЙТЕСЬ, ПЕРЕВОДИТЕСЬ НА ДРУГОЙ ФАКУЛЬТЕТ!

О разработчиках

Я считаю, что разработчик - тот кто может решать практические задачи и получать за это деньги. Ильхам Зюлькорнеев - разработчик. Павел Дуров - разработчик. Борис Нуралиев и Евгений Касперский - разработчики. А /c/ целиком состоит из теоретизирующих задротов, которые никогда не достигнут таких высот и не заработают столько денег, как Паша или Ильхам.

Причем фундаментальная ошибка заключается в отнесении себя к некой категории "небыдла". Дело в том, что разработка ПО никогда не являлась самоцелью. Использование небыдлоязыка или красивого архитектурного решения оправдано до тех пор, пока его применение находит отражение в потребительских качествах продукта, и имеет тем большую ценность, чем большей аудитории улучшение потребительских качеств будет доступно и понятно. Т.е. я хочу сказать, что продукт должен быть наоборот максимально быдло-ориентирован, учитывать его интересы, сделан под него. Это касается и разработки средств и библиотек, а также к выбора средств разработки. CMS система, написанная на PHP и работающая на нищебродских хостингах обладает гораздо большим рыночным потенциалом, чем какая-нибудь астральная фигня на лисповском uncommon web-е. Если эта еще и интегрируется с 1С а также имеет маркетинговую поддержку от производителя, но позволяет разработчикам самостоятельно зарабатывать на лицензиях (т.е. получать деньги фактически ничего не делая) - то её потенциал еще

выше. Деньги всегда были и будут в "быдло-технологиях", то, что произошло с Viaweb (кстати, "илитность" его заключаась только в выбранном языке, сам проект - типичное веб-говно для быдла) - нелепая случайность, одна на весь мир, возникнувшая на перегретом американском IT-рынке до краха доткомов. Так что учите PHP и 1C, вся ваша математическая хуита и небыдлоязыки никому даром не нужны.

Многопоточное программирование

ОП, вкуривай join calculus, pi-calculus, temporal logic of actions и прочую годноту. Программируй на Хаскелле, используй STM, DPH, CHP, пиши свои реализации. Не зашкварься о кресты или интелевские либы, эта хуита для быдла, которое не умеет ничего, кроме C/C++ и ни на что большее, чем написать #pragma omp for чтобы фор распараллелися не способно.

Erlang

Эрланг уже просто не нужен в свете последних событий. Весь мир понял что ассинхронность нужна в 10% мест и это отлично делается твистедами, ивентмашинами, нодежсами. В эрланге же 90% мест это ссанина типа send.. receive
Ккакие там есть мифы про эрланг - распределённость из коробки ? да нихуя каждому поцу приходится лепить свою инфраструктуру. всегда будет ботлнек в виде один процесс который один на много процессов и хранит какой то стэйт.
Отказоустойчивость ? Какая блядь отказоустойчивость в языке который поощряет код типа ok = huita() который валится в стектрейс и никого больше не ебёт. Какая блядь отказоустойчивость в языке в котором один из главных принципов Let it fail и мы перезапустим. ага блядь перезапустили закрывшийся по сигналу сокет, упавшую по сигналу таблицу с данными... Ссаный рэббитэмку который выглядит как сырая поделка и падает от бэдматч персистер! ссаный ежабир. ой блядь.
паралельность ? а нахуй если можно взять zeromq.

1C

Говорят, что программист 1с нужен на предприятии лишь затем, чтобы осуществлять проводки "задним числом" и править базу в обход его (1с) родных алгоритмов (нестандартные, но востребованные, операции и суровая реальность этой страны), со всем остальным тупо справится админ, да и с этим тоже, но ленится/брезгует/не хочет брать на себя чужой геморой. Грубо говоря - живой компонент системы (обезьяна). Иначе просто не объяснишь их востребованность там, где готовых решений больше, чем снега зимой. Кстати, большинство из них и работают за зарплату, на которую реально можно купить только миску супа и оплатить коммунальные услуги на одного человека, хорошо хоть работенка у них не пыльная особо и можно взять что-то по совместительству (например, мыть полы в офисе).

Естественно, он может попасть под раздачу, если главбуха и директора начнут ебать за всякую нелегальщину, допущенную случайно или намеренно (это его вторая основная функция в предприятии, хорошо ему, что органы у нас ленивые, а начальство похуистичное, но бывает влетает так, что мама не горюй! Особенно если он молодой и глупый). Ну уж о том, что ему каждый день приходится подставлять жопу и убирать погадки за каждой тупой, ЧСВшной и капризной планктониной, говорит сам Капитан Очевидность.

Встроенный в 1с ЯП - это глумление над мозгом русского человека, а для русского программиста - это глумление над мозгом в квадрате. Кто видел, тот поймет, остальные - остерегайтесь! Радует то, что программировать там особо нечего не надо, кроме см. выше.

Ну а во всём остальном (читай во внешности) типичный 1с-программист похож на типичного айтишника (в представлении неайтишников): небритая рожа, мятый костюм, грязный свитер, пивной перегар и геморой на полжопы. Но в душе он бухгалтер! Конечно, бывают совсем молодые, только что с вуза/техникума, но они быстро исправляются.

Функциональщина

Точка зрения любителей ФП.

Языки программирования

Мэйнстрим

(область с заторможенным временем, в которой работает 99% программистов. Время заторможено потому, что в серьезном бизнесе сделать за предсказуемое время важнее, чем быстро. Сейчас в мейнстриме примерно 1975-ый год):

- C (няшная сишка) - самый простой и убогий язык из тех, что используются на практике. Более убогий - только брейнфак. Единственное выразительное средство - копипаст, для автоматизации которого есть даже специальный второй язык-препроцессор. Делает решение любой задачи нетривиальным,

так что его решение задач с его помощью может требовать высокой квалификации. Тем не менее, типичная няшаблядь ничего не знает и не умеет. Даже дибиловатый обгвидок знает, помимо гвидопыха, еще и сишку, но сишкаблядь не знает ничего кроме нее. Языком владеют почти все, но только няшаблядь этим знанием гордится, остальные стыдливо скрывают. Также няшаблядь может ошибочно считать что знает C++ или несуществующий язык C/C++. Благодаря C компьютерные программы - самое ненадежное из всего созданного человеком.
- C++ (кресты) - самый сложный и уродливый (и то и другое временно, появился быстро развивающийся соперник) из языков, использующихся на практике, но выразительность его невелика. Усовершенствованная сишка. Сложность использования повышает ЧСВ крестобляди до заоблачных высот, но вменяемая крестоблядь понимает, на каком говне пишет и постоянно пытается себя загипнотизировать. Поэтому любое упоминание крестов в негативном ключе вызывает атомный батхерт крестобляди и обвинения в некоем "неосиляторстве". Насчет неосиляторства крестоблядь, обычно, права - это вообще распространенное явление. Интересно, что до недавнего времени неосиляторами были даже разработчики большинства крестокомпиляторов. Тем не менее, по мере "осиливания" крестов, мнение о них может меняться только к худшему.
- Java (жаба) - кобол нашего времени. Убогий и примитивный, но простой язык. Хорошо подходит для написания зиллионов строк кода быстро-быстро, пока солнце еще высоко. Наряду с крестами - один из самых массово используемых языков.
- C# (сисярп) - зародился в недрах микрософта из-за анальной копирастии Сан и отсутствия стандарта жабы как сплав жабы и сишки, но впоследствии стал самым уебищным, но единственным в мейнстриме ФЯ. Уверенно движется к тому, чтобы отобрать у крестов лавры самого сложного и уродливого языка. 80% программистов на нем знают и используют только его 20% подмножество - жабу. Попытка микрософта насильно осчастливить жава- и крестоблядей для последующего захвата и порабощения, о чем неустанно предупреждают ведущие мозолееды. Кресты двухтысячных.

Немейнстрим

(не используются на практике, программисты знающие эти языки, на работе пишут на других, указанных в конце):

- ML (стрелочное эмелеговно) - старейший ФЯ, многое в нем напоминает современные языки, но сделано впервые, а потому криво и уродливо. Уебищность эмеля и его производных часто связывают с практичностью, потому как все языки используемые на практике (мэйнстрим) непереносимо уебищны, что позволяет надеятся на то, что и эмелеговно тоже скоро к ним присоединится. Любой программист на эмеле - будущий программист на хаскеле, который пока боится привыкнуть к хорошему и блевать кровью на работе. Типичная эмелеблядь пишет на работе на крестах или сисярпе.
- Haskell (хаски) - условно современный ФЯ, являющийся, разумеется, говном, но несколько менее вонючим, чем остальное здесь перечисленное. Имеет подмножество Haskell 98 для обучения второкурсников, на котором даже факториалы и фибоначи нормально не напишешь и раковую опухоль, которая его убивает - стандартную прелюдию. Несколько сниженная в сравнении со средним уровнем говенность хаски - основная мишень для критики. Дело в том, что он, по всей видимости, недостаточно уебищен для мейнстрима, да еще и вызывает головные боли и тошноту у знающих его программистов, когда те отрабатывают свой кредитный фокус-покус на работе. Типичная хаскиблядь зарабатывает на жизнь программированием на сисярпе, крестах.

Скрипты

(применяются для обучения, администрирования и лепки гостевух, но есть и единичные примеры использования не по назначению, для написания программ. Вместо программы, впрочем, получается гигантский скрипт на выброс)

- LISP (скобочное говно) - один из самых старых скриптов, до начала семидесятых даже использовавшийся в качестве эрзац-ЯП, за неимением ничего лучшего. На заре его существования был игровой площадкой для монстров CS, которые вскоре положили на него хуй и пошли лепить алгол. Последний крупный проект (то, что сейчас называется Axiom/Open CAS) стартовал в 70-ом году. В настоящее время форсед-мем школьников и первокурсников. Типичный скобкошлеп берет деньги у мамы.
- Scheme - скобочное говно, сделаное правильно. К алголу приделали скобки и получился лучший скрипт для обучения первокурсников и, возможно, лучший скрипт вообще, но в этом имеет сильного соперника. Типичный схемоеб получает стипендию и денежные переводы от мамы из Крыжополя.
- SmallTalk - скрипт, придуманый для обучения программированию детей-дибилов и лучший скрипт всех времен и народов. IBM решили, что для индустрии это как раз то, что надо и пытались его пропихнуть. FIAL. Тем не менее, индустрия таки заразилась от него всеми спидами. Баззворды "ООП", "паттерны проектирования", "юнит-тесты", "рефакторинг", "MVC", да и все остальные пришли в мейнстрим из смолтока. Типичный смолтокоеб на работе пишет на жабе.

Пыхоплеяда

Несмотря на то, что схемка и смолток делают остальные скрипты совершено ненужными, массы динамических петушков выбирают ПЫХОПЛЕЯДУ (Perl, PHP, Python, Ruby). ПЫХОПЛЕЯДА - это высеры

ГСМ-ов и неграмотных долбоебов, которые проделали большую работу изобретя колесо (квадратное) - чукча не читатель, блеать. Пыхоплеяда состоит из протопыха (слишком переподвыподвывернут для петушков, известно, что новейшую версию первоначально удалось реализовать только на хаскеле), пыха - классика гостевушного жанра, гвидопыха и джапопыха. При этом, если гвидопых отличается от пыха только ЧСВ гвидопыхеров, упивающихся своей невъебенной илитностью, и наличием у хуесосов харизматичного фюрерка, то джапопых действительно несколько более продвинут, и в мокрых фантазиях джапопыхеров является смолтоком. Знатоки пыхоплеяды лепят гостевухи за доширак и заправляют картриджи.

Хаскель

Хаскель - простой и понятный язык для нормальных людей вроде физиков и математиков, отчищенный от байтосодомии и делающий специально выведенных компьютерных опущенцев-программистов ненужными. Да, вы не ослышались. После того, как этот язык для нормальных людей получит распространение, байтоспарта закаленных еблей в жопу архитектурой фон Неймана и еблей в рот аппликативным порядком боевых пидарасов-программистов потеряет всякий смысл. Что останется делать после этого жалким недочеловекам, у которых пять лет в шараге вымывали из мозгов все человеческое? Остается только прерывание своего жалкого существования.

Плюсоёбство

Плюсоебство - это как православие. Когда протестантизм завещает работать над собой и создавать блага, славя величие Господа, православие требует от последователя искупления греха, посредством непрерывного страдания, как страдал Иисус неся свой крест. Стрелять себе в ногу и нажираясь потом водкой, позерски ноя о тяжестях нищебродской жизни -- что может быть лучше для православного плюсоеба?

Байтоёбство

Байтоёбство включает в себя:

1. Императивный стиль программирования как начало байтоёбского пути.
2. Дрочка на машинно-ориентированные типы данных (собственно, основной симптом байтоёбства) и последующее за ней закономерное возмездие байтомудакам в виде big endian vs. little-endian, особенности обработки чисел с плавающей точкой и.т.д.
3. Предтерминальные стадии байтоёбства - интринсикоёбство и его более тяжёлая форма - инлайн-ассемблероёбство. Подсадка начинается с убеждённости поциента в необходимости ручками использовать SIMD -инструкции.
4. Терминальная стадия, как итог п.3, тру-ассемблероёбство и "хроническая низкоуровневая оптимизация головного мозга"

В нашем мире, к счастью, подобные симптомы с распространением java, C# и прочей "замещающей терапии" встречаются реже, однако остались две отрасли, входящие в зону риска:

1. Гейдев. Байтоёбство в гейдеве берёт своё начало в 70х-80х, поскольку именно тогда зародилась традиция байтоёбства в геймдеве. Обязаны этим, в основном, восьмибитным соснолям и домашним компьютерам, которые, обладая малым объёмом ОЗУ и имея скудные средства программирования, требовали делать на них ёба-игры. Эта традиция продолжилась и далее, всё благодаря тем же консолям, на которых консолерабы должны были выпускать игры 5 лет, задрачивая их убогие байтоёбские архитектуры по полной. К сожалению, подобная практика перешла и на ПК, где байтоёбство, в общем-то не так оправдано. К слову байтоёбам-игроделам дали шанс выбраться из этой трясины в 2002 году, когда майкрософт запилила дуднет и менеджед дайректикс к нему. Но байтоёбы остались верны своим указателям, плюсам и байтоёбской оптимизации. Зашоренность, верность привычкам, безыдейность, десу.
2. Эмбеддед. Причины почти всё те же, что и в гейдеве. Маломощное железо, пара сотен байт озу, деревянные игрушки, прибитые к полу и.т.д. К этому прибавляется огромное количество разных железок разномастных архитектур, для которых нет толковых тулчейнов. Из хорошего - в последние годы байтоёбство в этой сфере потихоньку излечивается, спасибо дядям из ARM co ltd, сделавшим свою архитектуру более менее распространённым стандартом среди всех архитектур и огромному количеству появившихся фреймворков и компиляторов языков для этой платформы.

Также распространено ложное утверждение что байтоёбство крайне необходимо в системном программировании. На самом деле этого легко избежать. Рассмотрим среднестатистическую аппаратную платформу. Краеугольными камнями любой аппаратной платформы являются:

1. Процессорная архитектура
2. Memory map - адресное пространство, в которое отображаются RAM, ROM и внешние устройства
3. Протоколы управления этими самыми внешними устройствами.

Так вот, всё вышеперечисленное вполне можно вполне декларативно описать обычным конфигурационным файлом, не прибегая к программированию вовсе. Затем, скормить этот файл

генератору платформ и на выходе получить готовый фреймворк-скелет нашей операционной системы, доступ к которому можно получить из любого языка программирования. Вот так вот просто, если бы байтоёбство гологного моска не мешало.

Выбирайте Хаскелл

Хаскелл на данный момент является лучшим языком для новых проектов. Исключительная выразительность языка и мощная система типов позволят Вам быстро писать элегантный и надежный код. Язык еще не столь распространён. пока ваши конкуренты используют устаревшие технологии на базе нетипизированных лямбла-исчислений или императивного подхода с элементами динамической типизации, вы сможете в разы поднять свою эффективность, задействовав System F - последнее достижение науки в области статической типизации. Но это еще не все. В жизни любого стартапа наступает момент, когда он превращается в продукт и сопровождению проекта привлекаются дополнительные разработчики. На этом этапе распространённость и доступность языка начинает играть решающую роль. Благодаря активной популяризации Хаскелла и функционального программирования в среде коммерческих программистов, а также поддержке этого языка со стороны лидера производства оффисных приложений и операционных систем - корпорации Майкрософт, Вы можете быть уверены, что в будущем Вам не придется переписывать свой проект на C++, как это было с печально известной разработкой Пола Грэма. Хаскелл обеспечит вам гарантии успеха и стабильности Ваших начинаний. Выберите Хаскелл сейчас и через несколько лет Вы сможете наслаждаться результатами своих трудов - успешным проектом, выполненным с учетом всех современных технологий и индустриальных стандартов. Хаскелл - Ваш проводник к успеху в мире разработки программного обеспечения. Выбирайте Хаскелл.

О разработчиках

Я считаю, что разработчик - тот кто может решать практические задачи и получать за это деньги. Эрик Мейер - разработчик. Саймон Пейтон Джонс - разработчик. Филип Уодлер и Пол Худак - разработчики. А /pr/ целиком состоит из теоретизирующих задротов, которые никогда не достигнут таких высот и не заработают столько денег, как Саймон или Эрик. Причем фундаментальная ошибка заключается в отнесении себя к некой категории "быдла". Дело в том, что разработка ПО никогда не являлась самоцелью. Использование быдлоязыка или некрасивого архитектурного решения оправдано до тех пор, пока его применение находит отражение в потребительских качествах продукта, и имеет тем большую ценность, чем большей аудитории улучшение потребительских качеств будет доступно и понятно. Т.е. я хочу сказать, что продукт должен быть наоборот максимально небыдло-ориентирован, учитывать его интересы, сделан под него. Это касается и разработки средств и библиотек, а также к выбора средств разработки. Монадическое IO , написанная на Хаскелле и работающее на нищебродских компах обладает гораздо большим рыночным потенциалом, чем какая-нибудь астральная фигня из лиспа типа CPS. Если эта еще и интегрируется с Agda2 а также имеет маркетинговую поддержку от производителя, но позволяет разработчикам самостоятельно зарабатывать на верификации (т.е. не писать горы юнит тестов) - то её потенциал еще выше. Деньги всегда были и будут в "небыдло-технологиях". Так что учите Curry и Coq, вся ваша императивная хуита и клиппед понос никому даром не нужны.

И это программисты...

Ничего не понимаю. И это программисты. Говно какое-то. Пидоры, блядь. Блядь, Шейнфинкель с Карри им дали комбинаторы. Комбинируй, комбинируй комбинаторы, блядь, "не хочу! хочу жрать говно!" Что такое? Это программирование? Это программирование? Суки. Мудачьё. Программисты. SCIP прочитали. Говно жрут. Придоры блядь ёбаные.

Критика ООП

Кодер, а с чем ты работал кроме ООП? Нет, не надо писать про процедурное программирование, мейнстримовое ООП - это и есть расширение процедурного программирования, добавляющее в процедурные языки первоклассные модули, иначе называемые объектами.

В итоге местами получается код ради кода или хаки на преодоление препятствий собственной архитектуры.

Это всё от бедности элементной базы. Правило одно - все есть объект. Dixi. Операции композиции объектов не определены (в отличие от ФП, где композиция функций имеет смысл) из чего сразу вытекают следующие прелести:

1. Объектная декомпозиция - чёрный ящик. Тип композитного объекта не может раскрыть его составные части. Увидеть, что скрывается за абстракцией не посмотрев в код невозможно, использовать автоматику для частичной спецификации композитного объекта на основе его составных частей невозможно.
2. Создание комбинаторов объектов невозможно. В отличие от ФП, где комбинаторами являются те же функции, в ООП объекты комбинируются процедурным быдлокодом. Пытаясь вынести его в другие объекты, ты, по сути, ничего не получаешь, потому что в итоге тебе приходится комбинировать большее число объектов тем же быдлокодом.
3. Паттерны не могут быть первоклассными объектами. Следует из отсутствия комбинаторов, т.к.

паттерны являются некими стандартными комбинациями объектов.

Дополнительный вброс:

1. Неконтролируемая мутабельность добавляет к этому дополнительные прелести, вроде отсутствия ссылочной прозрачности, проблем с вариантностью, еще больше усложняет автоматический вывод спецификаций.
2. Сабтайпинг изрядно усложняет систему типов, не добавляя к ней особой выразительности. Технически, сабтайпинг - это ограниченная универсальная квантификация, тайпклассы позволяют добиваться того же эффекта, только с гораздо более простым выводом типов ок, тут я могу ошибаться, у вас есть шанс поймать меня на некомпетентности, питушки!

Вывод - ООП полный фейл. Во всём. ООП создаёт некую иллюзию, что можно работать программистом нихуя при этом не зная, но вообще, во всёх вузах детям выёбывают мозг ОО-программированием. Не знаю, что бы они делали, если бы их всех поголовно не оттрахали в жопу виртуальными деструкторами. С другой стороны, если ты хоть что-то знаешь, мейнстримовые ОО-языки не дадут тебе воспользоваться своими знаниями.

Правильный стиль программирования на Haskell

Нужно написать тутор по монадам "Изучите хаскель чтобы всех нагибать". С тезисами:

1. Забудь про do-нотацию и тем более list comprehensions - становится слишком мало символов и быдлу понятно.
2. let и where - для лохов, поставляй все в огромную лямбду, пусть все видят, какой у тебя крутой однострочник.
3. Функции нужно называть не иначе как f и f', а выглядеть они должны только как f a b c d = a b (c d). Никаких аннотаций и тем более комментариев.
4. Pointfull - для лохов, все должны видеть, как ты умеешь ((.)$(.)).
5. Не забудь использовать fix вместо рекурсии, все поймут, что ты правильный пацан.
6. Участвуй во всех спецолимпиадах, где требуется посчитать ряд с помощью Integer. Другие задачи решать не надо, потому что тогда ты приближаешь момент, когда хаскель будет мейнстрим и не получится всех нагибать.

Scala и Clojure

какой же благородный дон станет писать на досуге на Scala или Clojure? Ведь всякие Scala, Clojure, F#, OCaml и прочие эрзацы нормальных языков програмирования нужны благородным донам только для того, чтобы использовать их работе, где им приходится считаться с мещанскими вкусами остальных работяг-програмистов. поэтому на досуге благородный дон будет пописывать на Хаскелле, Агде или Эпиграмме, и почитывать алгебраическую топологию или теорию категорий. А на Scala и Clojure на досуге пишут только некоторые представители люмпен-пролетариата, котрые подсмотрели это занятие за благородными донами на работе, и думают, что ритуальное копирование поведения благородного класса делает их самих благороднее.

О "школах жизни"

Есть такая порода людей, которая считает, что каждый, кто на что-то претендует, просто обязан хлебнуть говна. Например, "настоящий мужчина" обязан сходить в армию или отсидеть в тюрьме, ведь что тюрьма, что армия - школа жизни; "настоящий программист" обязан писать на крестах, ПХП или тому подобном, ведь сами они на нём писали, а кто не хапнул этого говна, тот и не программист вовсе. С одной стороны понять их можно - механизм примерно такой же как у дедовщины в уже упомянутой армии или прописки на малолетке ("меня опускали, теперь я буду опускать" - ну просто обучение и сделование паттерну, через пиздюли навязанному авторитетом, детская психика так устроена), с другой - сами они со временем так нихуя и не понимают, и любые попытки объяснить им, что все эти боль и унижения, через которые они прошли, в общем-то лишние, встречают у них агрессию и отрицание.

Пандорический захват

Делаешь пандорический захват, лифтишь в монаду, потом строишь рекурсивную схему (здесь подойдёт зигохистоморфный препроморфизм) как монадический трансформер из категории эндофункторов, и метациклически вычисляешь результат. Любой второкурсник справится. А если делать на анафорических лямбдах — так задачка вообще на пять минут.

Дискуссия по монадам - с объяснением очевидного завязывать

Любая «дискуссия» по монадам переходит в терминальную стадию именно вот так. Пациент видит do-нотацию, от которой у него случается когнитивный диссонанс. И непонятно что лучше... То-ли в Applicative-стиле писать (\x y -> SetTime $ x-y) <$> GetTime <*> GetTime, то-ли с объяснением очевидного завязывать...

Monad в хаскеле, не для do-нотации. Вся фишка Monad, в волшебных пузырьках хитрожопом комбинаторе μ :: m (m a) → m a и наборе комбинаторов с типом a → m a, не являющимися return. Все вместе они задают логику работы «монады». Причем, через крайсивый и простой интерфейс: Functor + Monad + MonadPlus + Applicative + Alternative + некий MonadFoo — специфичный для данной монады тайпкласс, куда включены специфичные комбинаторы.

Как результат, тип выражения выбирает какой именно код будет исполняться, и позволяет управлять логикой работы всей, возможно весьма-весьма нетривиальной, монадической машинерии.

Т.е. в примере quasimoto GetTime, и SetTime не определяют каким именно образом будет производиться искомое действие. Вся логика определена в Monad, точнее, в неявном аналоге runIO :: m a → a — instance Show.

У тебя же — сплошной, махровый, императивный, сайд-эффектный ad-hoc. Со всеми вытекающими.

Печальные даты

Знаете, некоторые печальные даты надолго остаются в памяти людей: 11 сентября, 17 августа, 1917-й год, 1941-й. К ним стоит добавить 1995-й - год появления JavaScript, PHP, Ruby, ну и Java тоже. Кому-то захотелось по-быстрому добавить динамизма в веб-странички, и он за пару недель наговнякал интерпретатор, встроив его в браузер Netscape. Кому-то захотелось оживить свою домашнюю страничку, добавить счетчик посетителей, еще что-то, и он на коленке сделал такой вот изменятель страничек на стороне сервера. О больших проектах тогда никто не думал, personal home page назывался тот изменятель. А когда делаешь интерпретатор, проще всего сделать его на динамической типизации. Это банально очень просто. О системе типов вообще можно не задумываться, не говоря уже об их выводе. К сожалению, на фоне тогдашнего мейнстрима (Си, ранние плюсы, что там еще было?) эти скриптовые языки выглядели очень выигрышно, писать мелкие куски кода на них было намного проще. Что такое нормальная система типов тогда мало кто знал: хаскель был еще в пеленках, ML'и традиционно не выходили из университетов. Так что люди эти скрипты подхватили, стали добавлять все новые функции. Менять систему типов стало поздно. В итоге выросло то, что выросло. С тех пор одна масса людей занята тем, чтобы делать все более сложные интерпретаторы, которые бы не так тормозили, другая масса придумывает 121-й способ добавить в JS типы, а третья на динамических языках пишет и плачет в бложиках о том, как грустно им делается. И проблема не только и не столько в скорости, сколько в maintainability кода и усилиях на необходимые тестирование и отладку при росте проектов. Единственная реальная причина появления динамически типизированных языков - лень и недальновидность авторов. Эволюционно динамические языки - тупиковая ветвь, хоть они и обречены рождаться вновь и вновь просто потому что их делать проще, а делать языки люди любят. Сегодняшняя популярность некоторых из них - случайность, исторический казус, следствие контраста между этими языками и мейнстримом начала 90-х. То, что много идиотов используют идиотские языки, говорит лишь о том, что идиотов много. Сегодня, когда есть языки с нормальной статической системой типов, никаких реальных преимуществ у динамической больше нет. Только я имею в виду действительно нормальные статически типизированные языки - как минимум с параметрическим и ad hoc полиморфизмами, с выводом типов. Не Си с джавой. Хаскель, окамл, скала - такого уровня. У этих конкретных языков могут быть свои проблемы, часто инфраструктурные, но речь сейчас не о них, речь о динамической vs. статической типизации в целом.

Трагичная история Вирта

История Вирта очень трагичная, братюни. Начиналось все хорошо:
Хоар сказал: «В Алгол нужно добавить ссылки с нуллами»
А Вирт сказал: «Мы с Хоаром считаем, что в Алгол нужно добавит ссылки, да с такими-то нуллами»
Потом Хоар сказал: «Алгол 68 сосет»
И Вирт, конечно, сказал: «Мы с Хоаром ваш Алгол 68 в рот ебали, пидоры комитетские!!111»
А потом Хоар внезапно сказал: «Анаморфизм, катаморфизм, иломорфизм, параморфизм, наконец»
И Вирт сказал:«Ана-што? Иломорфизм? Хуе-мое! Так, падажжи, ебана, Карри же был моим научруком, параморфизм? Ах тыж ебаный ты нахуй!
АЛГОЛ 68 СОСЕТ!!!111 ССЫЛКИ ДА С ТАКИМИ-ТО НУЛЛАМИ!!!111 АЛГОЛ 68 СОСЕТ!! СОСЕТ АЛГОЛ 68 !!11111»
Ну, так с тех пор и повторяет.

Лисп

Контрамоты

Я понял!!! Динамические петушки - контрамоты. Они движутся во времени назад. Как было на самом деле: протоцепепе - ЛИСП 50-80 гг. C++ 80-90 гг. Haskell - новая версия цепепе. 90-00 гг. С точки зрения лиспопетушка сначала был Хаскель, в следующей версии он деградировал до цепепе, а в лиспопетушином будущем цепепе деградирует до лиспа.

А я все ломал голову, как человек может так ненавидеть будущее, так привязаться к прошлому, что пишет

на лиспе? Но для лиспопетушка-контрамота лисп - это и есть будущее. А дальше ВВ2, индустриальная контрреволюция, религиозные войны, рабовладельческий строй, отрастание хвостов и подъем на деревья. Скай из зе лимит!

Контрамоты 2

Общеизвестные факты. Вы, разумеется, не можете этого знать, раз для вас это далекое будущее, но в 70-80 ведущие деятели компьютер сайенс переживали серьезное разочарование в Лиспе. Новая эра разработки языков программирования началась с признания того, что ЛИСП - говно, и нужно делать все иначе, с типами, первоклассными функциями и так далее. В 90-е стало окончательно ясно, что последняя попытка гальванизировать труп лиспа - Схема - провалилась. Вы сбиты с толку ореолом романической дымки прошлого окружающей лисп в наши дни. Современный человек может даже восхищаться лиспом, как восхищается пирамидами Гизы - романтика веков, епт. Но строить бессмысленные пирамиды блоков сейчас дураков нет. В 80-е эти лиспопирамиды все воспринимали просто как горы говна. Еботня с лиспом еще была свежа в памяти.

Консерваторы 1

Лисперам чужды изменения и динамика. Не забывай, что когда белый программист слез с лиспа и перешел на алгол в поисках новых горизонтов, лисперы были теми ретроградами и консерваторами, что остались сидеть на скобках, как обезьяны. Поэтому менеджер Гугля был прав, когда выеб очередную лиспомразь, пытающуюся протолкнуть свое говно.

Консерваторы 2

Динамические педерасы не понимают этого нифига. Для них компьютерная наука застыла в 1960-х годах, в лучшем случае. Все, что было после этого — ниибацца до чего сложно и непонятно. И поэтому способы написания скриптов не менялись с 1970-х. А вот если из той хуйни, которой мучают программистов, удалить совсем уж бессмысленную и никому нахуй не нужную тупую хуйню, типа динамической типизации, паттернов-хуяттернов, указателей-пиздолизателей, оставшимся вещам можно обучить нормальных школьников за год-два, либо на первом курсе. А дальше учить людей полезному, теории категорий, Хаскелю, Агде-2 и прочим простым и красивым языкам, без которых программист это не инженер, а просто гнида.

Правила поведения 1

Дискутируя со скобочной мразью ты даешь ей почувствовать себя человеком. Это ошибка. Лиспомразь следует игнорировать. Если ты дискутируешь с лиспомразью, чтоб продемонстрировать как она «соснет», то это бессмысленно. Лиспомрази сосут просто публикуя любой свой пост - в этом плане они самодостаточны.

Правила поведения 2

Разговаривая с лиспером, никогда не следует пытаться убедить его в своей правоте, апеллируя к логике, фактам и доводам - он их все равно не воспримет, ибо ему попросту нечем... Вообще в процессе разговоров перед глазами обычно возникают некие "имиджи" или лучше, говоря по-русски, "условные образы" людей-собеседников. Так вот: беседуя с лиспером, ни в коем случае не следует обманываться, награждая его, даже условно, человеческим образом. Представьте его говорящим глистом и тогда все сразу станет на свои места. И да хранит вас господь.

C++ лучше чем Лисп

Как нет? Все ошибки лиспа (динамическая типизация, макросы, вырожденный синтаксис) кроме одной (байтоебная энергичность) были исправлены в цепепе. Вот только с первого раза получилось плохо. Типизация вышла с израдной долей петушения, макросы кое-как замаскировали под параметрический полиморфизм, но метапрограммирование все равно изо всех щелей торчит, синтаксис вышел куда лучше чем в лиспе, но все равно говенный. Ну и энергичность не поправили. Пришлось предпринять еще один сеанс исправления ошибок. И Хаскель получился уже гораздо лучше.

Мнение каталог-куна

LISP давно уже не язык. LISP - просто прилипчивое говно на ботинках анонимного эксперта, тонким слоем размазанное по всему /c/. Анононимный эксперт совершил ошибку, растоптав LISP. Теперь мучается, пытаясь отчистить подошву от этой вонючей дряни. Засело крепко. Понимаешь, просто больше нечего добавить к тому, что уже было написано здесь. Нормальному человеку таких унижений достаточно, чтобы почувствовать всю ничтожность говна, которое он форсит. Но когда имеешь дело с лиспофанбоем — это совсем другой разговор. Он будет отравлять своим LISPом воздух еще долго после растаптывания.

Пидарасы

А мне наплевать на пидарасов и их парады. Они мне совершенно не мешают. А вот лисперов я ненавижу за то, что они, как православные и прочие угоревшие по религии, тормозят прогресс, сковывают производящие силы общества и делают мою жизнь хуже, снижают ее качество. При этом лисперы - самая мерзенькая категория религиозных фанатиков и врагов прогресса. В той же категории находятся амиши, например. Но я сравниваю лиспопетушков с пидарасами потому, что предполагаю, что такое сравнение для них обиднее, чем сравнение с амишами. Сам же я считаю, что лиспер - существенно хуже пидараса, более того - сравнение с лиспером скорее оскорбление для пидора, чем наоборот.

Список лисперов /c/

Лисперы це: Общешлюхи: 1) Золотце. Шизофазный фрик без работы и образования. 2) Андрюша. Студент младших курсов. Схемобляди: 1) Собакоеб. Имеет опыт аудита кластеров парадигм на пыхе и питоне. 2) Варкрафтёр-полуебок. Студент. Судя по его рассуждениям о хвостовых рекурсиях и системах типов регулярно упарывается. 3) Альфаслесарь. Пишет на схемке после заправки всех картриджей. Я что-то пропустил?

OCaml

Я вот тут от балды написал прикладной http сервант на ocaml, что бы несколько спустя дней сыпя проклятиями переметнуться на хаскелл, напиать от балды сервак на нем и успокоиться. Так что не все так однозначно, особенно учитывая то, что уже haskell 6.12 имеет конкурентный сборщик мусора и сам из коробки умеет использовать несколько ядер.
1) Откровенный, признанный в рассылке баг в Lwt, собранный с ним ocsigen, фикс есть, но надо почти весь окамловый мир пересобрать. Даже на ubuntu 10.10 он еще есть.
2) Если жить с этим багом, то это приводит к тому, что сервер становится блокирующим. Это вообще неприемлемо, при сколь угодно маленьких нагрузках
3) Собранный хаскеллем бинарник будет жить на любом вообще почти линуксе --- там динамические зависимости от малого количества системных библиотек. Его не поломать сменой энвайромента
4) На окамле так теоретически можно, на практике за половину дня у меня не получилось (может я тупой, но я лучше буду на хаскелле код писать, чем бодаться со сборкой и путями к библиотекам и чтением манов на ocamlfind)
5) Happstack очень хороший и он проще в использовании, чем Ocsigen, хотя и не такой пока навороченный. Но лучше все равно.
6) 90% из того, что я пишу в окамле руками, в хаскелле уже есть
7) В компиляторе хаскелла правильный подход к компиляции, к представлению значений в памяти, и люди над компилятором работают, а не занимаются черти-чем. Это значит, что момент, когда хаскелл станет уделывать окамл по вычислениям --- неизбежно наступит, это только вопрос времени. Представление чисел в Окамле --- это какой-то позор, при том, что есть SML/MLTon и Эндрю Аппель. Работа с этими числами и их типами при отсутствии тайпклассов --- это тоже позор. Так что окамл быстро работает только на 31-битной арифметике (ну или 63-битной, но я не уверен, что в 64-битной сборке примитивные типы 63-битные) Может, в хаскелле оно вообще сейчас боксед всё, я не знаю, но динамика развития компилятора говорит о том, что это временно
8) Темпы развития компилятора и рантайма хаскелла - в нём уже есть конкурентный сборщик и масштабирование по ядрам, а не отмазки, что это все сложно и потребует больших переделок. Работает само (на примере хаппстека). Т.е. теже драйвера HDBC --- уже сами по себе написаны в неблокирующей манере и прозрачно работают с ForkIO, а не требуют запуска в нативных тредах или переписывания, как окамле (я не говорю про качество самих драйверов еще).

Эмель сдох

Вообще, эмель сдох. Никакого развития нет и не планируется. Тот же Лерой прямо писал в конфочку, что да, ГЦ окамла сосет, но ему похуй, ничего делаться не будет. А раньше раньше развитие было разным в разных реализациях, т.е. приличная оптимизация в Млтон, свежие языковые фишечки в нью-джерси или москоу, человеческое лицо в элис, в окамле - традиционнно сосание хуйцов бесплатно. Флагманской реализации так и не появилось, язык распался на диалекты. Итого, большинство реализаций и диалектов сдохли или в коме, комьюнити нет, библиотек меньше чем в хаскеле, эмелеебы переходят на хаскель хлопая дверью (могу доставить свежую сочную пасту), заглянувшие на огонек хаскелеебы охуевают от того, что в Окамле все делаешь, а ничего нет. При этом окамл позиционируется как практичный, но практичность окамла это: нет многопоточности, нет нормальных числовых типов, нет нормального ГЦ, нет нормальных оптимизаций в компиляторе, нет либ, нет инфраструктуры, нет комьюнити, на главном сайте есть надпись, что окамл - практичный язык.
Помимо пиздежа про практичность, окамлофанбои любят похвалится производительностью, но это тоже пиздежь. Идеоматический код на окамле сосет не разгибаясь, более-менее сносную производительность можно получить только написав на окамле фортраноподобную говнолапшу. Да и то, даже хаскелист, почитывающий бложек какого-нибудь дона стюарта и знающий пару ключей компилятора и настройки рантайма, имеет все шансы оставить окамлоеба глотать грязь на обочине.
В общем-то окемл -- это такое легаси для компиляции Coq. Язык так себе, практически не развивается (0.5-1 man-power на поддержке), но умеет быстрые символьные вычисления (сам Ксавье переключился бы с него на что-нить еще, было бы на что).

Кроме большей скорости на некоторых задачах и большей простоты обучения, у кемла нет никаких

особых достоинств. А вот недостатков куча.

Приведенный пример еще не так страшен. Понадобилось всего две вспомогательные ф-ии (drop/takeWhile, ужасающе кривой inputlist с mutlist не в счет). А вот при разработке больших проектов кемл показывает себя во всей красе -- иерархических модулей нет, автоматом они в repl не подгружаются. Мутабельность, которая по-началу местами что-то упрощает, потихоньку превращает проект в кашу. Отсутствие ленивости по-умолчанию делает всё менее модульным. Уж не говорю про отсутствие type class-ов, задания приоритетов операторов, убогие функторы. В итоге получается конечно получше, чем C++/java/c#/python (все таки замыкания, вывод типов, алгебраические типы, pattern-matching), но остаются все их проблемы связанные с неконтролируемой мутабельностью.

Разговор 1

(папрашу не путать православный мл с мутантом кемлом. и не приписывать ВСЕМ вариантам мл неустранимые преимущества имеющиеся только у кемла. в отличие от кемла мл-щики хорошо относятся к параллелизму и он в нескольких реализациях есть. и хотя в некоторых при этом сборщик мусора еще не параллелен, но в poly-ml например он уже запилен. и полька масштабируется отлично, хорошо ублажает изабеллу. ходят слухи и про sml-sharp. про mlton - пока не уверен. говорите мл не развивается? при наличии достаточно полного стандарта мл97 есть планы и на новый стандарт. просто старый стандарт был продуман настолько годно что до сих пор неплохо работает. что тут говорить - в хаскеле до сих пор нет нормальной модульности а только уродство убогое тайпклассное. что толку от иерархий если самого главного нет - изоляции, достаточной чтоб дать гибкость? в хаскеле все влияет на все. скорость хаскеля вообще отдельный разговор - она то есть то ее нет. и нуб ни в жись не ускорит хаскель если ему не повезло нарваться внезапно на тормоза. так что с хаскелем так - либо за пять минут нашел готовое решение... либо споткнулся о кирпич и тогда сиди кури бамбук месяцами ищи обходы. полюбуйтесь на высера на шутауте. И почуствуйте разницу между недостатками МЛ и недостатками хаскеля. на МЛ все недостатки типа - "еще не написано но то что есть вполне годная база а остальное никто не мешает дописать". а на Хаскеле - "кирпич! все механизмы уже написаны для чего-то другого именно так, и хрен их поменяешь или отключишь". Если не умеете писать на мл иммутабельно и нормально использовать модули так учитесь. Ленивость никак не связана с модульностью или возможностью обрабатывать бесконечные структуры данных, а функции высокого порядка есть и в МЛ. Так что пилите свой хаскель лучше до надежно работоспособного состояния, хотя бы как у "старых" МЛ, а не занимайтесь всякими недостаточно обоснованнями срачами тут с умным выражением лица)

Разговор 2

http://www.linux.org.ru/news/doc/9813153/page1?lastmod=1384429057571#comment-9818578

Да, Лерой подзабил на OCaml давно, но как раз-таки в последние пару лет OCaml действительно стал развиваться в связи с появившейся возможность не только INRIA, но и членам так называемого «консорциума Caml» (то есть людям, в отличие от Лероя сильно заинтересованным в развитии языка) участвовать в разработке языка.

По поводу многоядерности и плохого GC, на это забил именно Лерой, причем по относительно разумным причинам, он это дело в рассылке обосновывал. Проблема только в том, что реальность поменялась, многоядерность — сегодняшний путь развития процессоров, и потребность у пользователей в использовании этой самой многоядерности только растет, поэтому уже давно пытаются что-то сделать, пока, правда, для продакшена это не готово, но да ладно, не это главное. Поглядите на весьма популярные Python и Ruby, у них с параллельностью не лучше, и ничего, вполне себе здравствуют.

Далее, насчет именно функциональных языков программирования. Из семейства ML OCaml действительно самый практичный. Пользователей у него больше, чем у остальных диалектов ML, библиотек, соответственно, тоже гораздо больше. Инструментарий, пусть и довольно скудный, опять же богаче. Да, если отвлечься от ML, то у того же Haskell пользователей и библиотек еще куда больше, но Haskell по заявлению самого Пейтон-Джонса не предназначен для продакшен. Да, его используют там, но Пейтон-Джонс позиционирует и развивает его как именно исследовательский язык. Ну и, что куда важнее, как язык Haskell _гораздо_ сложнее, порог вхождения в него гораздо выше. Оба языка достаточно маргинальны, чтоб легко найти разработчиков на них, но тот же Лев Валкин вполне себе набирает просто толковых людей себе в команду, а там за пару недель на OCaml оные начинают вполне себе писать. С Haskell это определенно не пройдет. Ну и остались F# и Scala, которые посовременнее, популярнее и развиваются быстрее, но они завязаны на .NET и JVM, что для кого-то будет несомненным плюсом, а для кого-то — совсем наоборот.

Дальше, производительность. Она вопреки заявлениям фанбоев все же не на уровне C/C++, особенно если не писать жутко императивный код, но весьма приемлема, выше, чем у тех же Python/Ruby на порядок. Да, в отличие от сложных оптимизирующих компиляторов MLton и GHC тут все куда проще, но это сознательное проектное решение: это дает относительно небольшой проигрыш в производительности, но зато предсказуемость для среднего разработчика эффективности тех или иных языковых конструкций. Одно из важнейших достоинств C в его прозрачности: глядя на любую конструкцию легко представить, какой примерно машинный код за ней стоит. В OCaml с этим похуже без сомнений, язык все же высокоуровневый, но куда лучше, чем у того же Haskell, намного лучше. Писать на Haskell эффективный код — отдельный и довольно нетривиальный скилл. Тут действительно, если сразу с приемлемой

производительностью не заработало, начинается трах, тем более долгий, чем менее опытен разработчик.

Насчет свежих языковых фишечек, опять повторюсь, как раз в последнее время они добавляются. Из крупнейших добавлений — модули первого класса и GADT, например. То есть язык вновь начал развиваться, так что заявление «развития нет и не планируется» совсем не верно для текущего состояния дел.

В общем да, далеко все не идеально, и скорее язык так и останется в таком маргинальном положении, в каком находится сейчас. Но и не все так мрачно, как описано, язык развивается, у него есть пусть небольшое, но стабильное коммьюнити и, главное, есть пользователи в индустрии, которые зависят от этого языка, поэтому сильно заинтересованы в его улучшении, а с недавних пор могущие в этом улучшении участвовать. Вот и книжечка современная вышла, хорошее дело.

satanic-mechanic ★ (14.11.2013 15:26:22)

Python

Питон сделан некачественно, делают его какие-то сомнительные, судя по высказываниям, люди. У него плохой сборщик мусора. Плохой рантайм вообще. GIL. Плохо (никак вообще) писать DSL-и. Он многословен. У него есть неявные, трудноконтролируемые динамические зависимости (всякие sitedefaults) которые подгружаются из разных мест, и могут привести к разному поведению в разных средах. Ладно, еще если на одном сервере крутится одно приложение. А если их десяток?
Попытки сделать из него что-то приличное (Unladen swallow) потерпела эпический, эталонный фейл. "Эволюция" языка довольно смехотворная, привела к зоопарку версий. Его трудно поддерживать в продакшене. Он медленный, в конце концов. Нафиг он вообще нужен? Щас сюда придут и расскажут про "песочницы", линты, "умение готовить" и всю прочую стандартную документацию. Я все эти доводы знаю, так что прошу сразу перейти к следующей части --- а зачем, ради чего весь этот гиморой

Руби

Ruby sucks

Ruby is (Руби это):

- когда в языке нету недостатков, но всякие пхпшники сумели приебаться и найти следующие 5 пунктов
- когда йоба-ооп заменяет здравый смысл;
- когда пишут "begin end begin begin end end..." вместо кода;
- когда программа тормозит настолько, что за время подсчёта произведения 2 матриц 10x10 можно сходить попить чай;
- когда нормальным считается писать вещи типа 12.5.integer? или 3.times {puts "Ruby sucks!"}
- когда скорость увеличения и так небольшого количества библиотек падает на глазах - ведь есть гораздо более удобный Python

Ruby - замена Питона

Ruby - это замена Питона, которого скоро не будет. Perl стал не модным и питоновцы, воспользовавшись случаем, так долго мучили программистов тупым, абсолютно неудобным и уродливым синтаксисом, что когда пришел Ruby, идейно вышедший из старого доброго Перла, то сразу полетели щепки и питоновцам только локти осталось кусать, видя, как радостные программеры сматываются с этого дерьма на нормальный язык.

Java

Критика Java

Ёбаным жабоиндусам, напоминаю разницу. Зоопарк вопросов:
- где замыкания?
- где свойства?
- где шаблоны? Разработчики Sun вынуждены только облизываться. Даже генерики, введённые в 5-й версии Java — не более, чем syntactic sugar. Дотнетовские генерики это реально поддерживаемые платформой типы, которые расширяются на лету при загрузке, котрые оптимизируются JIT-компилятором. Для Java генерики существуют только в коде и ни JIT, ни загрузчик классов их никогда не видит. Поэтому проблемы боксинга, преобразования типов в runtime просто скрыты от программиста.
- где делегаты/евенты?
- где partial-классы?
- где детерминированное освобождение ресурсов (ключевое слово using + интерфейс IDisposable)?
- где оптимизация JVM для расширений процессоров?
- где аналог linq и в частности удобные мапперы?
- где расширения методов класса?

- где скрытая имплементация интерфейсов?
- где перегрузка параметров функций?
- где нормальное потребление памяти приложением?
- где быстрая работа приложения?
- где нормальные иде, с полноценными дизайнерами?
- где пользовательские value types?
- где методы у инстансов value types?
- где var и анонимные типы
- где перегрузка операторов?
- где оптимизиции хвостовых вызовов? (в свете фунционального хайпа это должно вызывать некоторый батхёрт)
- Где чёткое разделение домены и сборки? Это не учитывая, целый ворох технологий недоступный понимаю жабоиндусов, такие всякие сильверлайты/вин-веб/формочки, впф, XNA, список можно продолжать бесконечно, как впрочем и список ущербности жабы...

Всего, чего нет в жабе, автоматически объявляется хуитой, как только это появляется в жабе, это автоматически становится нехуитой. При этом, требуется сделать вид, что хуитой это называл кто-то другой.

Java лучше C#

На шарпе нет ни нормального IoC, ни билд-фреймворка, ни нормальных webMVC, ни ORM. Только ебаный .НЕТ ФРЕЙМВОРК и кусок бинарного говна - silverlight, вообще охуеть. Это не бро.

Джава же это отличный язык для сложных и надежных систем, для embeeded разработки, для веба, даже для реалтайма. Джава отлично работает на нормальных операционных ситемах - posix, да что там говорить она работает на любой ОС и платформе - даже на симкартах и контроллерах. У нее есть куча охуенных спецификаций на все случаи жизни. Большой и активный коммюнити, который всегда поможет. Большинство кода на джаве - опенсорсный, поэтому это отличный язык для обучения джуниоров - ведь поняв как написан код в крупных и серьезных проектах человек учится и его уровень как программиста растет. На джаве есть куча охуенных фрейворков - play, akka, spring,hibernate,maven,harmony,tapestry. На JVM есть множество языков на любой вкус - Scala, Ruby, Python, Groovy, JavaScript, Clojure и все они отлично взаимодействуют между собой и можно использовать библиотеки написанные на джаве. Это твой бро.

Есть ли в природе хоть один жавапидорас

Есть ли в природе хоть один жавапидорас, который знает что-нибудь кроме джавы? Ну хотя бы ту же Скалку, работающую на jvm. Причем `знает' не в смысле запускал 2 раза и посчитал рекламную хуйню с сайта, а действительно хорошо знает. Работал минимум полгода, знает, решению каких проблем адресован язык, знает, зачем он нужен и какие преимущества имеет перед джавой, хорошо понимает, какие принципы заложены в систему типов и нахуя они вообще туда заложены, знает, как она жутко она фейлит на оптимизации хвостовой рекурсии, знает, как она затирает типы и вызывает методы через рефлексию, знает, почему на уёбищной jvm по-другому не сделать, знает, почему неленивые языки - говно, знает, что такое abstraction penalty и почему любой джаваынтырпрайз адово тормозит, несмотря на превосходные результаты на шотауте, самый лучший мусоросборник и hotspot до которого всему немейнстиму в говне плыть и плыть, знает, почему джава - не язык программирования для нормальных людей, а говно для переквалифицировавшихся таксистов? Нет таких? Ни одного? Только студентота и дрочеры на абстрактные фабрики абстрактных аннотаций виртуальных плагинных шин? Так я и знал. И это программисты. Говно какое-то. Пидоры блядь.

Пример хорошей архитектуры на Java

1. Абстрактный синглтон базы данных.
2. Синглтон mysql
3. Фабрика баз данных
4. Регистри, хранящая фабрику баз данных
5. Абстрактный строитель вывода
6. Строитель вывода из SQL
7. Абстрактный наблюдатель базы
8. Наблюдатель за mysql
9. Посредник, принимающий сообщения от наблюдателя
10. Абстрактная стратегия выбора баз данных
11. Стратегия выбора Mysql
12. Абстрактный декоратор стратегии выбора базы данных
13. Фабрика стратегий выбора баз данных
14. Посетитель, запускаемый посредником при получении сообщения от наблюдателя
15. Заместитель, принимающий посетителя и решающий, делегировать ли команду вывода виду
16. Абстрактный вид
17. Вид CLI
18. Абстрактная фабрика видов

19. Фабрика видов, возвращающая инстансы видов CLI
20. Прототип CLI-вида, используемый CLI-фабрикой
21. Абстрактная модель
22. Абстрактный ORM
23. ORM для Mysql
24. Модель, работающая с MySql-ORM сущностями
25. Абстрактная активная запись
26. Активная запись для mysql-ORM моделей
27. Абстрактный синглтон хранения моделей
28. Синглтон, хранящий активные записи
29. Абстрактный синглтон хранения видов
30. Синглтон, хранящий CLI-виды
31. Абстрактное мементо
32. Мементо активной записи Mysql-ORM модели
33. Абстрактная команда
34. SQL-команда

Таким образом довольно просто мы получаем быстрое и гибкое приложение, пригодное для дальнейшего расширения функционала и рефакторинга.

Попытка создания учётной системы на Java

Подведу-ка я итоги полугодового эксперимента по разработке прототипа некоей учётной системы. На данной предметной области специализируюсь более 10 лет, так что опыт постановки задач мною набран вполне достаточный, и хотелось поднабраться опыта именно в реализации.

Требования к архитектуре стандартные: трёхзвенка, включающая промышленную СУБД + слой бизнес-логики + клиент. Сверху торчит подсистема аутентификации, авторизации и аудита, а сбоку – подсистема интеграции с учётными системами третьих производителей, включая складские и бухгалтерские системы. СУБД должна быть непременно 1) популярной 2) промышленного уровня, что автоматически исключает из перечня подходящих СУБД всякую экзотику типа DB2, поделки вроде MySQL и сырые штуки наподобие Derby. Собственно, остаются только Oracle и MS SQL. Несмотря на то, что оба этих продукта являются проприетарными, производители предлагают разработчикам бесплатные, хотя и урезанные дистрибутивы в виде Oracle XE и MSDE соответственно. Очень хотелось, чтобы система была кроссплатформенной и могла функциклировать под управлением Линукса, поэтому я с сожалением отложил в сторону Visual Studio и обратил взор на Java.

Итак, с СУБД вопрос решён, осталось покумекать над клиентом и middle-tier.

Броузер в качестве тонкого клиента был смело отвергнут, потому что: 1. Веб был изначально задуман для просмотра порнухи, а вовсе не для создания функционально насыщенных учётных систем. Мне не нравится, что в стандарте HTML до сих пор нет места деревьям и редактируемым таблицам. Не нравится, что элементарное событие – случайное закрытие окна пользователем, который перед этим битый час вводил данные - невозможно отследить единообразным способом. Бесит, что в Опере нельзя использовать самодельные контекстные меню.

2. Отрадно, что есть на нашей планете ремесленники, которые всё-таки умеют делать конфеты из говна (Ext-JS, GWT и т.д.). Но блядь такое количество интерпретируемого и плохо читаемого JavaScript-кода должно настораживать и пугать. Я просто не в состоянии понять, как работают эти кортежи вложенных конструкций и замыканий. Я не могу отдебажить ошибку, сообщаемую мне броузером, потому что даже если я буду знать имя файла и номер строки, где случилось исключение, я хуй чё там пойму.

В общем, выбор сделан в пользу толстого клиента. Какого? А не важно, главное, чтобы на Java. И я взял первый попавшийся под руку фреймворк – а именно Eclipse RCP – и закопался в документацию по RCP, SWT и Jface.

Документация в мире Java – предмет отдельного разговора. Я даже не хочу упоминать о тех пидорасах, которые вообще не документируют код, или, того хуже, хуячат свои манулы прямо в PDF (который тоже предмет ещё одного разговора). Чё, вам сложно привести пример использования прямо в аннотации пакета, класса или метода? Откройте блядь MSDN и посмотрите, как должна выглядеть нормальная документация. Впрочем, ладно – на худой конец, если юзаемая библиотека реально работает, я готов смириться, свалить все мануалы до кучи и натравить на них Google Desktop. Жить можно.

Спустя три месяца я через мат-перемат научился ваять на SWT вполне сносные окошки, как настоящий брутальный программист – инициализируя лэйауты и контролы прямо в коде. «А что, в Java визуального дизайнера нет?» - спросите вы. А это смотря какую визуальную библиотеку вы используете и в какой среде разработки. Если взялись за Swing – то самый классный редактор UI есть в Netbeans. Но в Eclipse, для SWT - такого нет. А есть плагин Visual Editor для редактирования гуя. Но вот незадача – его актуальная версия почти всегда отличается от актуальной версии Eclipse на один пункт. То есть плагин 3.2. не работает в Eclipse 3.3. Для того, чтобы выяснить, почему ЭТО не работает, вам придётся убить как минимум час времени на гугление и, что самое неприятное, в конце концов отказаться от заманчивой идеи визуального проектирования формочек. Или, как это принято в мире UNIX, кочать сорцы и

затачивать их рашпилем под свои нужды. Уф.

Далее. После того, как SWT был освоен и первый плагин был торжественно запущен, было принято решение его почикать-декомпозировать на несколько самостоятельных Eclipse-плагинов. Сказано-сделано. Поначалу всё работало прекрасно. Пока я не столкнулся с комплексной проблемой конфликта манифестов. Её суть в том, что формат манифеста плагина Eclipse RCP должен соответствовать неким скверно документированным правилам, гласящим, что в манифесте нужно прописывать как все зависимости на другие плагины, так и все выставляемые плагином наружу пакеты (это делается, слава аллаху, с помощью визуального редактора манифеста). Но. Если в этом же Workspace существует Java project, не являющийся плагином, то его подключить к другому плагину НЕЛЬЗЯ. Нужно либо скомпилировать этот проект и подключить к плагину jar-файл, либо переконвертировать проект в RCP. Но в последнем случае, если мы попытаемся подключить RCP-плагин к, скажем, проекту веб-приложения, то с вероятностью в 50% получим плавающий ClassNotFoundException при попытке обратиться к любому из классов, определённых в проекте плагина. Подозреваю, это происходит потому, что эксплуатируемый контейнер сервлетов и инфраструктура Eclipse используют разные классloадеры.

Но самые большие разочарования постигли меня с Middle-tier. Суть идеи была проста – использовать обычную модель оторванных данных, по схеме: выполнить запрос, закачать данные в Transfer Objects, отдать сервисом клиенту на редактирование, принять от клиента изменённый пакет данных, выполнить обновления в базе. В .NET это делается при помощи датасетов и веб- или remoting-сервисов с полтыка. Аналога System.Data.DataSet и вообще ADO.NET в мире Java я не нашёл. В поздних JDK в пакете javax.sql появились несколько классов, реализующих интерфейс RowSet, которые, по заверениям товарищей из Sun, гарантированно являются сериализуемыми наборами данных и их можно гонять между процессами и по сети. Первая же попытка получить WSDL из класса сервиса при помощи инфраструктуры Axis, один из методов которого возвращал JoinRowSet, привела к тому, что мастер публикации веб-сервиса честно предупредил меня о том, что JoinRowSet не является Java-бином и привёл его к типу Object.

Ладно, подумал я, - а как же обстоят дела с модными нынче ORM/POJO? Hibernate оставил на сладкое и взял в руки Apache IBatis. Название блять в точку. Но ибатился я недолго, хватило воли вовремя взять себя в руки и переключиться на Хибернейт.

Хибернейт – вообще вещь в себе. Альфой и омегой X прежних версий, насколько я знаю, являются XML-файлы, в которых описываются всякие там параметры ORM-мэппинга чуть ли не для каждой сущности. Вообще я ненавижу XML. Точнее, ненавижу нестандартный XML – это когда каждый задрот придумывает свой гениальный формат описания чего-то-там и пытается донести его до общественности. Реально работающих стандартов на основе XML в мире ровно три – это XSLT, XSD и WSDL. Всё остальное – это кал, в котором и ковыряться-то неохота. Поэтому раздел документации Hibernate, посвящённый описанию XML-схем маппинга, я пропустил и сразу перешёл к разделу, описывающему реализацию EJB 3.0 – то бишь аннотациям. Тестовый проект, включающий в себя десяток таблиц, в том числе и с достаточно сложным мэппингом типа «многие-ко-многим», как ни странно, на аннотациях заработал сразу и без нареканий. Но работал он ровно до тех пор, пока я не подключил хибер к… догадались? Да, плагину эклипса. Это был просто праздник. Начнём с того, что в режиме отладки плагина конфигуратор Хибера упорно не желал находить файл persistence.xml, если тот находился не в папке META-INF, которая в свою очередь ОБЯЗАТЕЛЬНО должна была лежать в папке src проекта. Это я победил. Далее, если плагин, дергающий хибер, вызывался из другого плагина, то даже такое хитрожопое размещение persistence.xml не помогало – я ради развлечения создал десяток его копий и распихал во все возможные места – не помогало, хибер его упорно не находил. Тогда я захардкодил конфигурирование Hibernate в коде java. Это помогло. Но. Встала проблема автоматического вытягивания зависимых объектов по связям. Если я хочу сериализовать объект и передать его в «оторванном» виде на другую машину по сети, то есть риск того, что за этим объектом высосутся полбазы данных, если мы все связи пометим для ранней загрузки (earlier fetching) или риск того, что объект придёт клиенту «пустым» в случае использования lazy fetching. Такой риск минимален, если клиент и сервер живут на одной машине - например, в архитектуре веб-приложения все связи можно помечать lazy и не бояться, что хибренейт выкачает что-то лишнее, но если мы используем disconnected-модель и «толстого» клиента, то хибернейт для задачи формирования пакетов данных, скажем прямо, не пригоден.

В общем, серебряную пулю я так и не нашёл и вчера торжественно отправил Java на помойку.

О жава и жавамакаках

ДИМОЧКА пришёл в офис очень несчастным и подавленным. Всё, что он хотел, это просто поесть в абсолютной тишине наедине с собой. — Что стряслось? — прервал раздумья ТИМЛИД.
— Мы программируем, как мудаки — нервно ответил ДИМОЧКА, — У всех программистов есть по машине, крутому телефону и толпе шлюх! А у нас нету нихуя совершенно!
В офисе нависла тишина. Через несколько секунд ТИМЛИД прервал её:
— Послушай, — обратился он к ДИМОЧКЕ , протянув ему ОПЕРАТИВКУ.
— Что это?
— Да послушай говорю.
ДИМОЧКА прислонил ухо к ОПЕРАТИВКЕ и услышал шипение.
— Ну и что? — спросил ДИМОЧКА .
— Шипит?
— Шипит.

— Это она так нагрелась от нашего кода! — весело сказал ТИМЛИД.
— И что?
Немного помолчав, ТИМЛИД сказал:
— Не важно, сколько у тебя денег, ДИМОЧКА . Зачем тебе они, если у тебя на лице кусок расплавленной пластмассы, а я выпишу денег на новую оперативку и распилю бюджет, только для этого ты и нужен, мудак...
После этих слов ДИМОЧКА повесился.

Новый жавист

К нам в айти отдел пришёл новый сотрудник. Нужно сказать, что у нас в отделе работают почтенные крестогоспода. Новичка посадили за компьютер, но не успели даже дать задание, как он начал кодить. Начальник из любопытства подошёл посмотреть, что он там написал. В течении секунд тридцати он побледнел, затем посинел, затем покраснел, а потом трясущимся от нескрываемого гнева голосом сказал:

- Это же Абстрактная фабрика! На чём ты кодил до этого?

- На Джаве.

- Жабапидор! - в один голос заорали все 20 человек.

- Жабапидор! Жабапидор! Жабапидор!

Кто-то включил сирену. Над дверьми замигали красные лампочки тревоги. На окнах мгновенно сомкнулись плотные жалюзи. В офисе одновременно бывает два отдела человек по сорок. На обеде вся эта толпа собирается на первом этаже, где яблоку негде упасть. А поэтому, как охранники ни пытались вырвать джавапидора из рук разъяренной толпы, им это не удалось. По всему офису стоял сплошной рев:

- Жабапидор!

В коридоре его сразу же сбили с ног. Используя галстук как поводок, его тащили через весь коридор, передавая из рук в руки. Поэтому получалось так, что никакого движения в коридоре не происходит, но и джавапидора тоже нет. Его заволокли в каптерку под лестницей, где хранятся ведра и швабры с тряпками, и там закрыли. Под конец рабочего дня он всё же появился. За получасовой обед его изнасиловали несколько человек. Сопротивляться было бы бесполезно. Через день на нем чистым оставалось одно лицо, а на теле не было живого места. Он превращался в мразь, в животное. Его били все, даже дизайнеры и уборщицы. Его заставляли есть говно и опарышей. В очко ему совали битые лампочки, живых птиц и змей. Он стал «дельфином» – в нужнике пятнадцать дырок, он ныряет в первую, выныривает, ныряет во вторую... И так – до конца. От него постоянно воняло. С ним невозможно было рядом находиться. Был такой случай: к нам устроился работать Степаныч. Степаныч сидит на толчке, а кто-то снизу через очко хватает его за яйца. Степаныч с воплем вылетает в коридор без штанов. Напротив - айти отдел. Смех не стихал долго... Потом Степаныч забил его кирзовым сапогом насмерть. Менты как узнали, что сдох джавапидор даже дело заводить не стали.

Пыхопроблемы

На похапэ можно писать довольно быстрый код. Конечно, не такой резкий, как заточенный под конкретную задачу инстанс nodejs или, не дай бог, что-нибудь для веб на няшной с ассемблерными вставками.

Можно банально придерживаться паттерна mvc и не погрязнуть в паутине спагетти-скриптов с сотнями инклюдов. Код будет хотя бы структурирован и изолирован локальными кучками говнеца. Это идеальное состояние, если большую часть рабочего времени вы добавляете в общую свалку новые, независимые друг от друга конвертики с тухлятиной.

Можно написать классы объектов, там где они необходимы и наполнить их методами. Опять же, всё ради структурирования кода, для вполне сносной и быстрой навигации по разрастающейся выгребной яме проекта.

Можно вооружиться профайлером, раскурить исходники ядра фреймворка, который вам предписало начальство, и частично переписать его, снизив время выполнения этого хитросплетения пиздеца на 80%. Вырезать конфиг веб-приложений, сделанный в xml. Уничтожить миллионы вызовов __call() и call_user_func(), от которых кровоточат глаза. Большинство макак знает, что обычное веб-приложение на похапэ инициализируется каждый раз с нуля. Поэтому уменьшить на 90% время инициализации - это очень хорошая идея.

Можно искать узкие места и куски рендерера, где хтмл генерится недостаточно быстро. Вооружиться memcached и реализовать грамотные схемы самообновляющегося блочного кеширования. Избавиться от пары дюжин лишних запросов к бд на каждый чих. Получить 80% страниц, выхлоп которых отрабатывает без запросов к бд вообще.

Можно заняться очередями сообщений и перенести на них особенно тяжёлые куски процессинга

картинок, видео, музычки, почты и прочего хлама, чтобы всё упиралось в длину очереди, количество воркеров и машины, эти очереди разгребающие, а не в число клиентов и их терпение к времени отклика от сервера.

Можно навесить плюшки в виде аякса, где это уместно, и местами перенести генерацию контента вовсе на клиент, вместе с тем сэкономив десятки тяжёлых запросов на отрисовку страницы целиком.

Можно взять сверхбыстрое простое хранилище типа redis и использовать его для сегментов системы, которые создают большую плотность не очень важных запросов к бд, типа учёта баннеропоказов, трекинга статусов online и логирования всякой поебистики.

Можно придти к мысли, что mysql с её слоупочными table locks и transactional safety и с её возможностью масштабирования только при помощи анальных расширителей не очень-то, собственно, и нужна в большинстве задач. Потратить 2 месяца и перенести огромную смердящую кучу наваленных друг на друга небольших пакетиков с говном на mongodb, на небольшой, но няшный кластер из нескольких replica sets по тройке лёгких машин. Ощутить невесомое изящество, с которой она похрустывает сотнями тысяч записей, прелесть schema-free и отсутствие дрожи в коленях, когда раньше ты запускал alter table на рабочей копии бд, глубокой ночью, потому что оно кладёт сервер на час-другой. А потом часами напролёт в умилении смотреть на графики munin, которые резко перебежали из погранично-красной зоны в самый низ зелёной. Финально включить eaccelerator и наслаждаться запасом в сотни запросов в секунду на отдельно взятом сервере начального уровня.

Можно дополнительно озаботиться настройкой nginx, убрать из конфига логгирование для файлопомойки, включить пяток жизненно-важных параметров, указать нормальные значения для буферов. Окончательно уничтожить апач, для которого был прописан reverse proxy для некоторых урлов. Выкинуть SATA-винты на помойку. Поставить дополнительно недорогих SSD и развернуть на них кэш для самой мелкой статики.

Только это всё не нужно. Ваш сайт, результат вашей работы никогда не получит хоть какой-то нагрузки. Когда на ресурс заходит 10 человек в день, а 90% хитов совершают боты гугла, можно хуярить страницы на 50, и даже на 150 SQL-запросов, ведь все таблицы бд влезают в оперативку, и страница даже на каком-нибудь позапрошлогоднем zend framework без твиков соберётся менее, чем за секунду. Да какой там фреймворк! Какой там MVC! Проще дёргать по необходимости разнородные готовые куски, часть кода бросить голодным доширак-макакам, и склеить всё воедино лишь-бы-работало спагетти-кодом. Ведь проект нужно было сдать ещё вчера, а завтра он будет навсегда забыт. И останется крутиться на задрипанном, надолго предоплаченном vps, в cron которому прописана ежедневная перезагрузка.

Я кончил.

Программирование для чайников

Тут будут храниться всевозможные ответы из "Посоны, посоветуйте, с чего начать программировать" тредов.

Вариант 1

Ебать, саентач, ты меня расстраиваешь. Уже в /pr/ на плюсы батхерты прекратились, а тут все то же самое. На C++ батхертят неосиляторы и задроты с ЛОРа. C++ - универсальный язык, а не только промышленная копипаста, его даже ЦЕРНе как очень быстрый скрипт используют для обработки гигабайт статистики со всяких коллайдеров, и такие же эффективные альтеративы этому намного менее удобные.

Сейчас будет поток сознания, потому что я не верю, что кто-то пойдет по этому пути - слишком сложным он кажется, хотя это не так, а времени нужно всего ничего, по сравнению с тем, что проебывается в шараге.

Итак, ОП, у тебя есть лето, несколько часов в день и желание быть в теме. Нужно проходить несколько предметов сразу. Один день одно, другой день другое, третий - попить пивка. И не париться по поводу того, что ты учишься программировать:
>То есть, юзер, который максимум, что может, это переустановить винду и почистить реестр, покрасноглазив наедине с учебником, сможет писать ПО на этом языке?
Чувак, программировать может даже бабушка, которая ничего не понимает в винде, зато очень хорошо понимает в БЭСМ. В компах разбираться не обязательно, другой вопрос, что если ты до сих пор не захотел программировать, с чего ты решил, что это твое. Это как рисование, не всем дано. Но, почитать книги ниже полезно даже если ты совсем дундук в программировании, но собираешься связать свою жизнь с компами, администом тем же.

1. Изучи javascript по какому-нибудь краткому справочнику. Пусть это для тебя будет бейсик такой, посчитай там квадратные уравнения и т. п. Дело в том, что javascript - это почти что scheme с другим синтаксисом, а scheme тебе понадобится для следующего уровня.
2. Энтрилевел - SICP + любая книжка по алгоритмам, на какую не жалко времени. Самая тонкая - Алгоритмы и структуры данных Вирта, заодно можно взглянуть на модулу, не порча себе вкус сями. Читать желательно одновременно, потому что в SICP используется динамическая типизация, которая

разжижает мозг, а Вирт плохому не научит, но он гораздо менее обширен и глубок. Вместо Вирта можно взять любой другой модный в этом сезоне путеводитель, типа Кормена, но я не могу посоветовать книгу, которую не читал, я и Вирта я читал давно, а Кнута советовать не буду.
На этом этапе можно не пытаться программировать, читать как художественную литературу, и пытаться познать дзен, что синтаксис не главное (SICP в этом поможет, ага). Примеры из SICP можно делать на javascript, а можно и поставить ракету.
Познав дзен, можно дальше думать, а что делать собственно. Программирование ради программирования заведет в тупик.

3. Выбор языка - динамическая типизация (python) + статическая (c#). Книжки - спросишь, как осилишь первый левел. Мне купленные мной книжки не понравились. Динамическая разжижает мозг и прививает хуевое мышление, но позволяет быстро набросать код на выброс, писать различные скрипты для автоматизации и прочее. Знать это надо. Тут выбор python или ruby. Ruby более красивый и цельный, но под него, кроме rails нихуя нет, так что учи python второй версии. Руби - это не лисп, как тебе написал какой-то непонятный чувак, это такой веобу smalltalk. Еще есть matlab для всяких инженеров, но учи python лучше, в нем с numpy индексы с 0 начинаются.
Статическая типизация позволяет думать по-человечески, но для быстрой разработки слишком много накладных расходов. Подойдет любой язык с управляемой памятью, то есть C#, потому что не VB.NET и Java выбирать же. Научишься формошлепствовать заодно и гуглить говно в библиотеках.
4. 32-х битный асм x86 + Керниган и Ричи. Назад, к машине. Машина твой друг и помошник. Асм нужен не ради асма, а ради понимания железа - протектед мод, плоская модель памяти и т. п. - в современных книжках по асму это есть, я читал какого-то русского автора.
5. C++
На это тебя точно не хватит. В принципе, последний пункт, что бы у тебя не было батхерта, а по жизни C# покроет 95% твоих задач, если ты не скриптовик-числодробитель, как я. А там уже можно устроиться на полставки куда-нибудь.
Выглядит дохуя, но за лето вполне все это освоить без особых напрягов, одна книжка станет нудной - можно продолжать другую, и т. п. На самом деле за месяц, если не ставить себе цель научиться программировать (нужно поставить себе цель научится ставить себе нормальные задачи - а программировать ты под это дело сам научишься, я так думаю, не научишься ставить задачи и будешь делать упражнения из учебников - будет хуже) можно все прочитать, поняв, что тебе по душе (некоторые железо любят, другие абстрагируются от машины в ленивые миры).

> А где можно ДОСТУПНО почитать об азах компьютерных сетей, их устройстве и управления ими?
Таненбаум - Компьютерные сети. Очень доступно и очень много.
Сорри за сумбур, пойду спать.

Админ, а чо нельзя зайти на твой блёваный сортирный сайтик с lurkmore.to? У тебя опять очко кровоточит по внутренним причинам?

Вариант 2

> Ясен нужно поступать, получать диплом

Спорный вопрос. С одной стороны окончание вуза - это хоть какая-то гарантия, что человек умеет доводить дела до конца. С другой стороны рашкинское высшее образование не все хвалят. С третьей стороны есть http://ocw.mit.edu/index.htm, другое открытое высшее образование, есть НМУ, хотя последний по слухам экстремально сильное колдунство, о которое редко кто может не опалить крылышки.

А с четвертой стороны, оп, постарайся понять, почему ты хочешь стать именно программистом, а не строителем, не конструктором самолетов, не доктором, не бизнесменом, например. Потому что из "хочу стать программистом" непонятно, приветствовать ли твое желание или помочь тебе найти другой путь. Каждый хочет ту или иную профессию по-своему, как например и космонавтом: кого-то завораживает возможность полета с огромной скоростью, кого-то то, что он побывает там, где мало кто был, кого-то почет и уважение.

Не стоит идти в программисты, если хочешь денег или почета и уважения, существуют более рациональные пути. Для того, чтобы побывать там, где мало кто еще был программирование тоже мало подходит - для этого надо становиться ученым, геологом или космонавтом. Так же не стоит идти программистом в надежде прохикковать, всю жизнь отсидеться на стуле, как премудрый пескарь. Если программист, как и любой другой сотрудник не будет бегать и скакать в течение всего рабочего дня по этажам и вообще всему городу, то он станет офисной крысой, будет плакать на дваче, ругать плохое государство и злое начальство - участь его незавидна.

Наверное только если тебе нравится работать с системами, сложными иерархическими сущностями, строить такие системы - это будет показанием к программированию. Также хорошо, если ты чувствуешь в себе качества "врача" - человека, умеющего ставить технические диагнозы и исправлять неправильно работающие системы.

> И еще, пока сижу на семерке, но вот подумываю пересесть на линукс, потому что быть не таким как все я слышать, что линукс настраиваем чуть менее чем полностью, а я просто обожаю кастомизацию.

Пересаживание на линукс и его изучение занимает достаточно много времени. В сутках 24 часа, так что

учись расставлять приоритеты для своих интересов уже сейчас. В любом случае, советую всегда оставлять себе дорогу назад - то бишь старый работоспособный виндоус, в который всегда можно вернуться, если ленуск заглючил.

Ещё вариант

В качестве альтернативы осмелюсь предложить почитать перед сном следующие книжки:
Раймонд Исскуство программирования для unix[2] - даже если вы не собираетесь кодить в линуксах и прочем эту книжку стоит прочитать, у Спольски есть отличная рецензия, если кто интересуется
Роберт Гласс Факты и заблуждения профессионального программирования[3] - это великая книга, можно смело противопоставить ее отвратительному рекламному чтиву, время от времени высираемому апологеотами тест-дривен девелопинга, скрумов, паттернов, рефакторинга, etc
Том ДеМарко Deadline - об управлении проектами, опять же стоит противопоставить ее остальной мути, которую время от времени почитывают наши рп (естественно, ни к чему хорошему это не приводит)
Это можно читать просто как художку вечерами.
Ну и само, собой, надо тупо программировать - в процессе стараться осовить синтаксис, семантику языка, примерно представлять, какие есть готовые библиотеки и для чего. Также читать код чужих программ (да, да, тупо читать и разбираться).
Вообще, потозреваю, лучший способ научится кодить - это путь торвальдса. Что он по сути сделал? У него на руках была написанная unix спецификация (ну то есть то, что потом превратилось в posix и single unix specification, то что уже десятки лет придумывали и проктировали бородатые дядьки в научных лабораториях) и он тупо начал реализовывать ее.
Предлагаю сделать то же самое. Например, реализовать PKI на сертификатах формата x.509, который бы удовлетворял российским законам о цифровой подписи. На самом деле, там будет просто кодерская работа, ведь все форматы и протоколы определены. Начать плясать можно от rfc4491 - Using the GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms with the
Internet X.509 Public Key Infrastructure

Гейдев

1. Алгоритмизация. Владение алгоритмизацией. Научиться думать алгоритмически. Ты должен будешь изучить какой-нить простой язык. Чаще это Паскаль или Бейсик. Однако я всем знакомым советую Python. И для обучения прост, и в будущем пригодится.
   Уже на данном этапе ты должен уметь без проблем использовать ЯП для имплементации своих замыслов и идей вроде "написать качалку всех пикч из треда на двачике", "написать проигрыватель музыки" etc.
   Из литературы хз что посоветовать. Сначала следует понять что такое алгоритмизация хоть на примере псевдоязыка, посмотри курсы "для новичков-кодеров" на INTUIT.ru.
   Для изучения самого языка тоже хз. Смотри INTUIT.ru и официальные буки.
2. Владеть хорошим ЯП. Если в п.1 особо не важно какой язык учить и главная цель только понять как оно работает, то здесь необходимо изучить нужный язык. Уже хз сколько лет работает цепь C->CPP. Да, сначала учишь ANSI C, а затем изучаешь парадигмы, основы ООП и как их применять в CPP.
   По си сразу бери ANSI C. Только издание по новее, а то в старом такое старьё описано... В общем, лучше 3тье(уже не помню точно какое последнее).
   После изучения си уже сам выбирешь себе "сенсея"(в смысле книгу по ООП), ну или создашь тред.
3. OpenGL, OpenAL, OpenCL, GLUT, GLSL, DirectX. Хватит и обычного C чтобы начать изучать DirectX и OpenGL. Мой первый движок, который я написал назаказ для одного проекта, был на C(т.е. не на CPP, т.е. без ООП) с использованием OpenGL + GLUT.
   OpenAL нужен для звука, а OpenGL для графики. OpenCL не так важен, он только начал развиваться. Это технология выполнения кода на графическом процессоре, эдвенсед шейдеры, если быть точнее. GLSL это язык шейдеров. Я использую его в своих проектов. Так проще. Нет проблем с совместимостью etc.
   DirectX стоит просто изучить чтобы понимать отличия от OpenGL. Да, я пизываю использовать OpenGL и только.
4. Изучить построение готовых движков. Здесь самое интересное. Здесь придётся разбираться в компонентах и устройстве других движков, чтобы со своим не сфейлить.
   Как только ты напишешь свой мини-движок, будешь понимать отличия Fragment-shader от Vertex-shader, будешь знать зачем юзают FBO, что такое bump-mapping и как он работает.. тогда тебя можно будет назвать ещё одним разработчиком компьютерных игр, которого не берут на работу из-за кризиса.

Опытный кодер, разработчик игр, питоноблядь, перепрофилировавшийся в веб-кодера-кун

Haskell

Начинай с The Haskell School of Expression дальше читай Typeclassopedia (http://www.haskell.org/wikiupload/8/85/TMR-Issue13.pdf), дальше статьи по ссылкам в тайпклассопедии, викикнигу (http://en.wikibooks.org/wiki/Haskell), а точнее отдельные главы. Ну и вики на haskell.org.
Некоторые няшечки могут посоветовать Душкина (беги от этой книги, как от чумы), Грема Хаттона или "Изучи себе хаскель во имя великой справедливости" - не слушай их, только напрасно потратишь время.
Ну а после изучения нормального языка программирования уже сам решишь, нужен ли тебе этот самый

питон.

Тонко подвести к Хаскелю

Начинай с дискретной математики. Во-первых, она действительно используется на практике (практически всё, здесь перечисленное http://ru.wikipedia.org/wiki/Дискретная_математика в той или иной мере используется при составлении алгоритмов), во-вторых, это подготовит тебя к изучению любых теорий, оперирующих значками. Можешь взять что-то ориентированное на программистов, например, книжка совсем для чайников так и называется: Дискретная математика для программистов. Еще тебе понадобится что-то по алгоритмам, например Алгоритмы: построение и анализ Кормена. Если это слишком сложно, еще один автор, который популярно пишет для самых маленьких - это Вирт - Алгоритмы и структуры данных. Дальше язык программирования. В случае книжки Вирта это будет Паскаль, которому ты и научишься по ходу чтения.

После того, как ты научишься писать алгоритмы и начнёшь самостоятельно ориентироваться в индустрии (т.е. сможешь понять любую задачу, найти и разобраться в любом нужном тебе алгоритме, изучить и начать использовать любую библиотеку или фреймворк, выбрать прочитать специализированную книжку) можно переходить к осуществлению твоей ебанутой мечты. Для этого тебе понадобится изучить хороший язык программирования. Хороший в том плане, что находясь в тусовке, ты всегда сможешь развиваться и изучать что-то новое, как фундаментальных, так и в прикладных направлениях. Сейчас таким языком является Haskell. Изучая его ты естественным образом придёшь к изучению теории типов, углублению своих знаний во многих разделах абстрактной алгебры, мат. логики, теории категорий, а также в области дизайна программ и языков программирования. Впрочем, как я уже писал, изучение его требует самостоятельности, а следствием самостоятельности будет то, что ты сам выберешь себе специализацию. Например тот же теорвер, если он тебе нравится (мне, например, нет), тогда Haskell будет языком программирования для структуризации и записи алгоритмов, а теория категорий - инструментом, помогающим спроецировать знания предметной области на конструкции языка программирования.

Высшее образование в IT своими руками v2

Итак, салаги, вы пришли к старым морским волкам в /pr/ и хотите стать программистами. Надеюсь, вы знаете, что хотите, так как это нелёгкий путь. Позвольте разъяснить вам несколько моментов: а) Большинство людей, которые делают сайты – не программисты. Если вы хотите делать сайты, но не хотите быть программистом – берите в зубы учебник «PHP5 в подлиннике» и начинайте делать свою первую гостевуху. Вопросы решайте в гугле, /s/ и форумах. Здесь другая тематика. Эту пасту дальше читать не надо, мы будем долго разговаривать, а вам нельзя терять время. б) За 21 день освоить программирование не получится. Никак. Совсем. Если срочно нужны деньги, то присоединяйся к товарищам, которые встали и ушли после пункта а. Нормальный объём академических часов в высшем образовании – 8000 штук. Это три года хорошей учёбы. Для того чтобы освоить материал хорошо надо ещё больше. Если параллельно бухать в общаге, то можно и в пять лет не уложиться. в) Я не Попов, магических способов изучения программирования за два DVD-диска не знаю, и учить им не буду. Я худший наставник, чем Кормен или Ахо, и буду только указывать вам направления деятельности. Готовься искать информацию сами. В каждой книге читайте, по крайней мере, оглавление. Задавайте вопросы. г) Программирование не есть изучение языков программирования. Хотя мы начнём его изучение с нескольких языков, они не являются самоцелью курса. д) Если вам не нравится паста – пишите конструктивную критику и предлагайте лучшие решения. От попёрдывания в лужу паста лучше стать не сможет.

Итак, надеюсь тут остались только те, кто хотят учиться. Если вам надо учиться, но вы не хотите, значит надо не вам. Наслаждайтесь. Вы станете настоящими программистами. Я надеюсь, что вы знаете математику и информатику на уровне 9 класса. Если не знаете, то перечитайте учебники. Курс от /pr/ состоит из модулей, каждый модуль состоит из двух частей: а) Матчасть. В матчасти перечислены моменты, которые надо изучить и книги, которые надо читать при изучении модуля. Все книги есть в Интернете. Если позволяют деньги, можно заказывать печатные варианты. Лучше читать на английском, но если не получается – используйте хороший русский перевод. Читайте так, как вам нравится. Если ничего не понимаете – читайте вперёд и перечитывайте после. Можно начать другой модуль. Можно заняться практикой. Можно почитать другую книгу похожей тематики. б) Практика. На практике надо писать программы. Ну, или, по крайней мере, составлять алгоритмы. Я буду предлагать небольшие проекты, которые охватывают материал из модуля. Но писать надо то, что нравится.

Модуль первый, введение. Задача: получить мотивацию и базовые знания, которые потребуются для освоения дальнейшего материала. Матчасть: информатика, программирование на языках высокого уровня, базовые понятия программирования: итерации, рекурсия, процедуры, функции, абстракции, классы, объекты, методы, переменные, присваивание, замещение, цикл, ветвление. Вначале советую читать SICP. Не весь. Вычисления на регистровых машинах можно отложить на потом. Нужно понять и прочувствовать принципы работы схемы (язык программирования, который используется в этой книге): это простой и одновременно мощный язык. Поначалу будет сложно, так как схема не похожа на бейсик, паскаль или что вы там изучали в школе. Но если вам удастся ухватиться по крайней мере за половину того, что написано в SICPe дальше будет легко и приятно. Писать на схеме сложные приложения невозможно. Это чисто учебный язык и вы никогда не будете его использовать на практике. Поэтому далее надо выучить кое-что посерьёзнее. Обычно первокурсники в России изучают язык си. Это не очень плохая идея в той части, что большинство языков имеют си-подобный синтаксис. В части байтоебли и

плохого ООП на крестах (так я буду называть язык C++) это плохая идея. Поэтому откройте толстенный учебник Дейтелов и хорошенько изучите его ровно до конца шестой главы. Это где-то 1/3 часть учебника. Дальше можете не читать, так как рискуете навсегда испортить себе вкус. Но можете и прочитать. На си можно писать сложные программы, но так тоже никто не делает. Поэтому большинство программ из курса я рекомендую писать на Java и Python.

Отвечаю на недовольный гул в аудитории: Java потому, что java легче. Изучая что-то другое на этом этапе, вы просто запутаетесь в особенностях языка. Особенно это касается шарпа (хотя на нём можно писать, как на джаве, только вот ведь не захочется), крестов (там сложно не запутаться) и хачкеля. Python потому, что некоторые задачи легче решать на скриптовом языке. Кроме того, в питоне есть некое подобие функциональщины, и если рано припечёт, то можно будет посмотреть и её. Хорошо ориентируясь в этих языках (на это не нужно слишком много времени – это не кресты, которые нужно учить годами) можно потом достаточно быстро изучить другие языки. А можно и не изучать, так как оба этих языка (в сущности, плохих) широко применяются до сих пор. Не заворачивайтесь на IDE, компиляторах и прочем инструментарии: вы всё равно перепробуете все доступные. Не дожидайтесь, пока вас заебёт первая рекомендованная среда, а сразу поставьте все распространённые и выберите понравившуюся.

Книги: Философия Java Эккеля, читать по мере необходимости. Не занимайтесь особым оверинжинирингом. По крайней мере многопоточность следует отложить до лучших времён. Не забивайте себе голову паттернами. Книга номер два - в глубь языка Python. Кстати, я знаю, как пишется «вглубь», просто использую русский перевод с официального сайта. Опять же – изучайте разделы по мере необходимости. Сомневаюсь, что тёлки будут течь при одном упоминании каких-то ваших характеристик, но изучить основы этого языка можно очень быстро. Кроме того, попробуйте почитать «Конкретную математику». Пока не станет скучно. Я рассчитываю, что скучно станет весьма быстро, хотя книга (как и TACP Кнута) написана с характерным юморком. Асимптотику лучше отложить до алгоритмов. Если чувствуете, что идёт совсем плохо (не Кнут с Паташником, а вообще), то читайте школьные учебники. Лучше старые, советские. Можно почитать книги для совсем маленьких детей «А я был в компьютерном городе», «Занимательная информатика» и.т.п. – это просто весело. Вам должно быть интересно читать. Если на этом этапе вам скучно, то дальше будет вообще кромешный непролазный пиздец. Ещё не поздно пойти писать гостевуху. Да, это была самая сложная часть. Если вынести из неё ещё и знание английского, то всё остальное покажется лёгкой прогулкой.

Практика: из всех учебников, которые я перечислил, задачи есть только в SICP'e и Дейтелах (ну и в конкретной математике, конечно). Их нужно решать. Освойте все простые конструкции, напишите несколько несложных игр, для одной из них напишите ИИ. Порешайте задачи для школьников, которые просят сделать за них лабу в /pr/ - но обязательно пишите на другом языке.

Теперь можно перейти к дискретной математике. Задача: понимать язык, на котором написаны остальные книги. Нет, это не самый занудный раздел. Теория трансляции будет зануднее. Матчасть: Открываете любой учебник, в котором есть: множества, алгебры, отображения, графы. Хорошо подойдут университетские методички. Можете видеокурсы с интуита посмотреть. Учишь. Плюс нужна элементарная матлогика – кванторы, законы де Моргана, таблицы истинности. Семиотику пока трогать не надо. Практика: Выполняете задания. Доказываете теоремы.

Традиционно далее изучаются базы данных. Базы данных есть в любом мало-мальски сложном приложении. Даже в компьютерных играх есть. Даже в ссаных гостевухах, которые сейчас пишут оставившие нас несколько абзацев назад «коллеги». Поэтому базы данных надо знать. Сейчас используются исключительно реляционные базы данных. Некоторые люди поговаривают про key-value хранилища (непременно асинхронные и сверхбыстрые), ну так вот, они концептуально тоже реляционные. Но вы с ними обязательно разберитесь отдельно. Матчасть: идёте по учебнику Кристофера Дейта и изучаете темы. Идти до конца не надо: читайте выборочно и смело бросайте около 17 главы. Изучить надо реляционное исчисление, ER-модель, транзакции, SQL. SQL лучше изучать не по Дейту, а по какому-нибудь практическому учебнику – обратите внимание на книжку Моисеева и его сайт с задачами. Практика: проектировать базы данных. Быстро. В уме. Таблицы должны интуитивно получаться сразу в 3NF. Пишите запросы на сайте у Моисеенко. Напишите приложение, которое активно использует базу данных – многим студентам такое барахло нужно на курсачи и дипломы, можно даже найти заказчика за деньги. Посмотрите на ORM (SQLAlchemy, Hibernate и.т.п.), почитайте статейки. Узнайте, какие сейчас используются базы данных, и обязательно прикрути парочку к своим приложениям.

Архитектура ЭВМ. Задача: знать, как работает компьютер. Дабы не делать ляпов. По крайней мере, глупых ляпов. Матчасть: Читаете Таненбаума, про архитектуру ЭВМ. Лёгкое и интересное чтиво. Знать: что такое вентиль, что из них составляют: там очень подробно описано по разделам. Не путаться в шинах. Знать про адресацию памяти, прерывания. Практика: Спроектировать простейший "железный" компьютер из блоков. На бумажке. Чтобы выполнял программу, записанную в память. Спроектировать всякой хуйни в эмуляторе схем. Дешифратор для семисегментного индикатора, например. Ассемблер лучше особо не трогайте, познакомишься с ним для интересу у Кнута, а писать на нём вам всё равно не придётся. Во всяком случае, я до сих пор я пытался оградить вас от низкоуровневого программирования. Знаний там очень много, но они все совсем не фундаментальные и изучать их надо под конкретную должность. Начните читать Кнута, по крайней мере, разберитесь с его компьютерами (MIX и MMIX) и напишите для них несколько программ на бумажке. Сделайте свой виртуальный компьютер, но не такой старый и сложный, как у Кнута. Сделайте для него ассемблер и напишите пару простых программ.

Наконец переходим к алгоритмам. Задача: понимать, как оценивается скорость алгоритма, почему существует много алгоритмов, как выбрать нужный. Знать базовые алгоритмы. Знать структуры данных и связанные с ними алгоритмы. Хорошо знать! Их много самых разных. Всякие связные списки из массивов вы должны уметь реализовывать стоя у доски с маркером. Книги: Вирт, Ахо по алгоритмам и структурам данных. Тут вот читать надо всё, очень пригодится дискретка. Опять же, Кормен. Там очень много материала, разбирайтесь в нём постепенно. Можно вернуться к конкретной математике, раз уж вы её бросили. Практика: реализуйте алгоритмы, про которые читаете. Вряд ли в реальном мире вы будете использовать их в чистом виде, однако вы должны знать хорошие решения. Да, эта бодяга надолго. Изучайте параллельно что-нибудь ещё, следующие разделы лёгкие и богатые на практику.

Сети. Задача – научиться писать сетевые приложения. Матчасть: Таненабум наш друг и товарищ на все времена. Осиливайте модель OSI, читайте спецификации нескольких сетевых протоколов. Например, http и smpt. Особенно http – разберись с хедерами, сжатием и.т.п. Долго и хорошо почитайте в Википедии про современные системы свзяи. Посмотрите алгоритмы, которые используются в маршрутизации, разберись, чем пакет отличается от кадра. Практика: делаем сокет-сервер, например, для чата. Разберитесь с XML, HTML, JSON. XML особенно. Освойте XPath.

Операционные системы. Задача состоит не столько в изучении операционных систем, сколько в изучении принципов распределения ресурсов компьютера. Тут же надо разобраться с многозадачностью, которую я вам как-то отсоветовал изучать сразу. Матчасть: опять же Таненбаум. Разберитесь с алгоритмами для планирования процессов, организацией памяти, файловыми системами, ядрами. Есть толстенный учебник Дейтелов. Помните, вы по ним си изучали? Так вот, ещё есть и по ОС учебник. Отдельно изучаете многозадачность: синхронизацию, пайпы, семафоры, мониторы. В жабе всё это дело есть из коробки и писать программы, которые реализуют такую функциональность будет просто и приятно. Если вы бросили Эккеля на этом месте – самое время начать читать опять. Одного Эккеля мало, используйте гугл. Хотя, наверное, к этому времени вы уже сменили язык. Практика: многопоточные приложения. Сделайте свой компьютер многопоточным. Это весьма занимательно.

Формальные языки и методы трансляции. Да, вот она вершина, с которой видно весь остальной курс. Если вы досюда добрались, то у вас железные яйца. Жму руку. Хотя и написано, что теория трансляции, надо обратить внимание на синтаксически управляемую обработку данных вообще. Матчасть: начинаем разогрев с главы учебника по дискретке про семиотику. Продолжаем Ахо и Сети, Книгой Дракона. Введение по дискретке там есть, но бедное. Нужно осилить грамматики, языки, иерархию Хомского и соответствующие автоматы. Кстати, автоматы в конце SICPa есть. Изучаем работу компиляторов и интерпретаторов. Изучаем оптимизации. Отдельно про регулярные выражения. Что такое регулярное выражения вы поймёте при изучении иерархии Хомского. Но регулярные выражения – это уже прикладная область, и чтобы их составлять нужно быть знакомым с синтаксисом, обозначениями и.т.п. – учебник по дискретной математике вам этого не даст. Прочитайте книгу О'Рейли про регулярки. С совами на обложке. Практика: написать несколько сложных регулярок, компилятор, интерпретатор. Да, чёрт подери, настоящий оптимизирующий компилятор простого языка.

Стандарты в программировании: всё самое сложное вы уже осилили, осталась сущая малость. Во-первых, стили разработки. Юнит-тесты, UML, рефакторинг, всякие совершенные коды. Уже пора изучать язык, на котором будете работать, и изучать классические труды о его устройстве, стандартных библиотеках и методах. Для прихода к просветлению можно таки выучить модный хачкель. В книжках, которые я рекомендовал есть моря ссылок на другие труды. У вас уже должен быть большой кругозор. Думаю, к этому времени вы уже знаете, что делать.

Байтоёбство

Машина - не шлюха

Машина должна и будет служить людям, она не шлюха, чтобы люди исполняли её прихоти. Отсюда байтобляди (а так же сочувствующие им императивные пидорасы, надрачивающие на показатели System.currentTimeMillis() - start) - пиздолисы, которые опускаются до полного говноедства, лишь бы ублажить её регистры и микросхемы. Альфапрограммисты, как и положено альфам, если машина не выполняет положенных ей задач и требует пресмыкаться перед ней и ублажать её байтами, просто берут и за патчкорды, ебашат с вертушки по передней панели и списывают машину на мороз, купив взамен ту, которая не будет выёбываться и выполнит код в сроки и без выебонов, будь там хоть 1000% неоптимизированного оверхеда. И настоящего программиста не волнуют вопросы выдрачивания и быстродействия - он решает важную задачу из предметной области гораздо более сложной, чем низкоуровневое дрочево, и отвлекаться на всякую подзалупную хуету вроде осоьбеннойстей какой-то там архитектуры ему некомильфо.

Рабы машины

Двачую, байтоёбы рабы во всём - рабы машины. рабы предубеждений, рабы производительности, рабы стереотипов, рабы обрабатываемых штеудом x86 типов данных - для них всё, что не кратно 2 байтам и больше 16 байт не может быть примитивным типом, хотя число - это просто число, оно может быть целым,

дробным, рациональным, комплексным, но не "в 2 байта в 4 байта в 8 байт". Да, байтобляди были актуальны пару-тройку десятков лет назад, когда кроме этого пресловутого отлизывания регистров и микросхем не было способов заставить машину быстро решать задачу. Но теперь-то в нашем распоряжении оптимизирующие компиляторы, многоядерные процессоры с параллелизацией, которые производительнее машин 20летней давности в сотни тысяч раз. Жаль, что программирование было поглощено стереотипным быдлом, не могущим в думать, и способным работать лишь по зазубренной инструкции, написанной кровью и потом сотен павших хомячков-байтоёбов до него. Настоящее, полноценное программирование, благодаря подобным обмудкам, мало теперь где востребовано. Хотя там где оно востребовано, можно кататься как в масле сыр и получать в три раза больше не то что сениор-байтоёба, а ёбанного заместителя директора быдлоконторы в которой этот байтоёб работает. С другой стороны это и хорошо - в космическую промышленность, Data mining и прочие сложные и непосильные для императивных байтохомячков сферы попадает лишь элита.

Типобезопасность C++ не нужна

У меня есть коллега, который презирает уровни доступа C++. Он работает на низком уровне, использует C++, как C с классами. Драйвера, софт для embedded и т.п. Когда я ему начинаю говорить про типобезопасность, инкапсуляцию, он говорит, что возьмет и получит нужные данные по смещению. Ну и что ему возразишь? Ничего. Он прав. Если кто-то захочет - он сделает. Даже в мейнстримовых песочницах можно. Вопрос цены только.

Конечно это не значит, что нужно все это забыть. Но категоричность сбавить можно было бы :)

Переписать проект на Си без сервисов, клаудов и вообще каких либо сторонних библиотек

Я принимал участие в одном проекте очень интересном проекте.

Существующие решение: на C++, STL, Boost, сервис ориентирования архитектура, cloud хранилище. Одна проблема система больше падает, чем работает. Каждую неделю проблемы, требующие вмешательства программистов, чтобы вернуть систему к жизни.

Архитектор принимает решение переписать проект на Си без сервисов, клаудов и вообще каких либо сторонних библиотек.

Результат: кодовая база уменьшилась в 50 раз, производительность возрасла 3000 раз.

Си для людей умеющий думать. Для тупиц которые читают Страуступа, Саттера и думають, что в их книгах правда написана как надо код писать не понять. Пошишите сначала код 5-7 лет, а потом вернитесь на Си и сравните результат.

Учитесь код писать у людей которые действительно доказали, что умеют это делать и не верьте тому что в блогах пишут. Примеры у кого учиться: Линус Торвальдс, Игорь Сусоев.

Критика Си

Указывать сиблядям на проблемы языка бесполезно. Кроме сишки сиблядь нихуя не знает и не умеет, а на любое обвинение у сибляди есть универсальный ответ - "криворукость". Этим сиблядь как-бы намекает, что что все вокруг криворуки - т.е. сотрудники микрософта и интеля, пишущие кривые драйвера и библиотеки, прыщебляди, пишущие дырявое ведро своей системы вот уже не первый десяток лет, просто другие сибляди из соседнего подвала полусвовковой шаражки, в которой сиблядь работает. А вот сама сиблядь - сука граф Шарль Ожье де Бац де Кастельмор д’Артаньян среди педерастов, владеющий техникой левитации, предсказания будущего и написания небыдлокода на сишке. К сожалению, простым смертным едва ли не удастся увидеть творения сенсея, так и будут они работать с глючным говном криворуких интелевских и микросовтовских инжеренов, внезапно падающим от какого-нибудь buffer overflow, несмотря на зиллионы человекочасов, проёбанных на его тестирование и отладку.

Критика C++

>C++

1. Не может в управление памятью
2. Не может в LALR грамматику
3. Как следствие, не может в человеческий синтаксис
4. Не может в денотационную семантику
5. Не может в настоящие макросы (с темплейтами отсос - не могут в квазицитирование)
6. Следствие - не может в человеческий полиморфизм (не говоря про higher-order), только убогое кодовысерание.
7. Не может в referential transparency
8. Linear typing

... и остальные миллиарды отсосов

Зато может в:

1. Нетипизированную еблю с указателями
2. Аппликативный порядок
3. УТЕЧКИ УТЕЧКИ УТЕЧКИ
4. Зловонную кучу Стандартов не реализованных в полном объеме ни одним компилятором
5. Стандарты наполовину состоящие из undefined behavior и implementation-defined
6. Следовательно, миллиард практически не диагностируемых "приятных" неожиданностей.

и тд и тп

Есть манную кашу через трубочку

Что блять? Что нахуй? Я не верю своим ебаным глазам, какой нахуй %название-технологии%, у меня не хватает слов чтоб выразить свое негодование, пошел нахуй отсяда, сраный уебок, и чтоб я тя больше не видел здесь, или я вычислю твой ойпи и ты будешь всю жизнь есть манную кашу через трубочку, ебаный дегенерат.

Возглавляется неким ..., который не знает

Оригинал

Возглавляется этим же "Гвидо ван Россумом", который не знает:
зачем был нужен reduce() (аналог foldr из Haskell) и поэтому убрал его в какую-то задницу стд библиотеки;
не знает о tail call:
и вообще, судя по убожеству грамматики пейсона, в дизайне ЯП не особенно рубит:
Язык объективно хуёвый, одно выстраивание стейтментов в лябмдах с сайдэффектами (пусть даже и локальными) с помощью логических операторов чего стоит.
Полное отсутствие выбора один-из-многих, по типу МЛ-паттернов или хотя бы няшного switch.
Хуеватый скоупинг, добавленный в язык явно не сразу.
ВНЕЗАПНО статические переменные, статичность которых зависит от способа инициализации
Убожественная система типов.
Нельзя доопределять операторы, поэтому прощайте нормальные DSL. Плюсы-хуюсы и битовые смещения рано или поздно кончатся, да и убого это — перегружать арифметические операторы.
Нет стандарта капитализации при регистрозависимых идентификаторах.
И многое другое!

Haskell

Возглавляется неким "Симоном Питоном Джонсом", который не знает:
о существовании не-ascii символов, которые успешно эскапируются в хачкеле, хотя на практике вполне читаемы с терминала.
о динамической линковке и вообще, судя по убожеству получаемых на выходе бинарников, не очень шарит в кодогенерации.
о существовании .net cli, llvm и прочих продвинутых технологий, из-за чего постоянно велосипедит бекенды и "сильные колдунства" навроде c--
Хуеватая система импорта, выблевывающая по умолчанию весь скоуп модуля в текущий контекст
Literate-programming костыли в виде доисторического кнутоговна вместо докстрингов и няшной интроспекции.
ВНЕЗАПНО Конструкторы всех дататипов в глобальном скоупе, и закономерно следующие из этого костылевысеры GlobalUniqueModuleNameMyDataType
Язык объективно хуевый, чего стоит одна система типов, требующая постоянных подсказок в виде явного объявления типа над функцией. Правда хачушки отмазываются, говорят что это традиция, повышающая читаемость кода, но мы-то с вами понимаем.
Нет постоянного стандарта на язык, а только какие-то "отчеты".
И многое другое!

Сложность

Взять Хаскель, например, там напротив всё направлено на устранение сложностей. Потому что это язык программирования для нормальных людей, вроде физиков и математиков, а его сообщество заинтересовано в том, чтобы язык был максимально удобным и эффективным.

В скриптоговне, вроде Питона или PHP, тоже никаких сложностей нет. На то оно и скриптоговно, чтобы позволить любому девятикласснику-объебосу в любой стадии алкогольного опьянения прикрутить голосовалку к постам на каком-нибудь форуме и заработать свои 20 WMZ. Однако скриптоговно нацелено на duct tape programming, им хорошо по быстрому скрутить готовые блоки, но попытка разработки

серьёзных вещей на нём ведет к неминуемым фейлам в виде гор невменяемого быдлокода, которые и создают иллюзию надуманных сложностей. Хотя на самом деле никаких сложностей в самом языке не создаётся.

В C/C++ действительно полно всяких искусственных сложностей. Но не потому что кто-то там их специально создал, а потому что с ними никто не борется (в язык бессистемно добавляют всякое новое говно не решая старых проблем, и он постепенно превращается в помойку). Такая уж философия у крестоблядей - страдать и нести свой крест. Любая попытка избежать страданий у них считается ересью и за ней сразу следуют обвинения в неосиляторстве и криворукости.

ПРИШЛО ВРЕМЯ...

По мотивам известного ролика ПРИШЛО ВРЕМЯ ПЕРЕУСТАНАВЛИВАТЬ WINDOWS

...ОСВОБОДИТЬ ПАМЯТЬ

ПРИШЛО ВРЕМЯ ОСВОБОДИТЬ ПАМЯТЬ
ПАМЯТЬ САМА НЕ ОСВОБОДИТСЯ
ОСВОБОДИ ЕЁ, ОСВОБОДИ ЕЁ ЕЩЕ РАЗ ЗАЧЕМ МНЕ НУЖЕН ХАЧКЕЛЬ, У МЕНЯ НЕТ ВРЕМЕНИ ЧТОБЫ ЕБАТЬСЯ С НИМ
ЛУЧШЕ ЕЩЕ РАЗ ОСВОБОДИТЬ ПАМЯТЬ
Я ОСВОБОЖДАЮ ПАМЯТЬ ПО 3 РАЗА В ДЕНЬ
КАЖДОЕ ОСВОБОЖДЕНИЕ ЗАНИМАЕТ ДВАДЦАТЬ НАНОСЕКУНД
Я ЖИВУ АКТИВНОЙ И ПОЛНОЦЕННОЙ ЖИЗНЬЮ
Я УСПЕШЕН И ПОЭТОМУ ЦЕЛЫЙ ДЕНЬ ВЫДЕЛЯЮ ПАМЯТЬ
А ПОСЛЕ ЭТОГО ОСВОБОЖДАЮ ЕЁ
ТУПЫЕ ХАЧКЕБЛЯДКИ ОДЕРЖИМЫ МОНАДАМИ
А Я СВОБОДНЫЙ ОТ ЗАДРОТСТВА ЧЕЛОВЕК
TEMPLATE <CLASS BAZ> BAR * FOO<BAZ>::DOWORK()

```
int s=((12<<5)&(2^(21-(4|4)*2)^1024))==0?1:0
```

ЛУЧШЕ Я ВЫДЕЛЮ ЕЩЕ РАЗ ПАМЯТЬ
И ЗАБУДУ ОСВОБОДИТЬ ЕЁ, СТАБИЛЬНОСТЬ НЕ НУЖНА
Я НЕ ОСВОБОЖДАЛ ПАМЯТЬ НЕДЕЛЮ
ПОЙДУ ОСВОБОЖУ
В C++ ВСЕ ПРОСТО И ПОНЯТНО
SEGMENTATION FAULT. ЭТО ЖЕ ОЧЕВИДНО КАК ЕЕ РЕШИТЬ
ПРИШЛО ВРЕМЯ ОСВОБОДИТЬ ПАМЯТЬ
КОКОКОКОКОКОКО
КВИКСОРТ КОНКАТЕНАЦИЯ ЗА O(1) INLINE ASSEMBLER
КОКОКОКОКОКОКО

...ОСТАТЬСЯ ПОСЛЕ РАБОТЫ

ПРИШЛО ВРЕМЯ ОСТАВАТЬСЯ ПОСЛЕ РАБОТЫ
SLA САМО НЕ ВЫПОЛНИТСЯ
ВЫПОЛНИ ЕГО, ВЫПОЛНИ ЕГО ЕЩЁ РАЗ
ЗАЧЕМ МНЕ НУЖНА СВОБОДА, У МЕНЯ ВСЁ РАВНО НЕТ УВЛЕЧЕНИЙ
ЛУЧШЕ Я ЕЩЕ ЕЩЁ РАЗ ОСТАНУСЬ ПОСЛЕ РАБОТЫ
Я ОСТАЮСЬ ПОСЛЕ РАБОТЫ КАЖДЫЙ ДЕНЬ
КАЖДАЯ ПЕРЕРАБОТКА ЗАНИМАЕТ 3 ЧАСА
Я ЖИВУ АКТИВНОЙ И ПОЛНОЦЕННОЙ ЖИЗНЬЮ
Я УСПЕШЕН И ПОЭТОМУ ВЕСЬ ДЕНЬ ЗАДРАЧИВАЮСЬ В ОФИСЕ
А ПОСЛЕ ЭТОГО ЖДУ ЧТО МЕНЯ ПОВЫСЯТ
ТУПЫЕ ПОХАПИДОРЫ ОДЕРЖИМЫ ФРИЛАНСОМ И УДАЛЁНКОЙ
А Я СВОБОДНЫЙ ОТ ДОШИРАКА ЧЕЛОВЕК
СКОЧАТЬ СПРИНГ В ДЕЙСТВИИ ТРЕТЬЕ ИЗДАНИЕ ПЕРЕВЕДЁННОЕ
КРЯК УЛЬТИМАТ КЕЙГЕН ДЛЯ INELIJ-IDEA
ЛУЧШЕ Я ОСТАНУСЬ ПОСЛЕ РАБОТЫ ЕЩЁ РАЗ
И ДОДЕЛАЮ ВСЁ ДО ДЕДЛАЙНА, ОТДЫХ НЕ НУЖЕН
Я НЕ ОСТАВАЛСЯ ПОСЛЕ РАБОТЫ УЖЕ НЕДЕЛЮ
ПОЙДУ ОСТАНУСЬ
В JIRA ВСЕ ПРОСТО И ПОНЯТНО
МОИ ПРОСРОЧЕННЫЕ ЗАДАЧИ CRITICAL BLOCKER. ЭТО ЖЕ ОЧЕВИДНО КАК ЭТО РЕШИТЬ
ПРИШЛО ВРЕМЯ ЗАДЕРЖАТЬСЯ НА РАБОТЕ
ККОКОКОКОКОКОКО
ПОХАПЭ ДОШИРАК ГОСТЕВУХИ
КОКОКОКОКОКОКО

...ПИСАТЬ КОМБИНАТОР

ПРИШЛО ВРЕМЯ ПИСАТЬ КОМБИНАТОР
КОМБИНАТОР САМ НЕ НАПИШЕТСЯ
НАПИШИ ЕГО, НАПИШИ ЕГО ЕЩЕ РАЗ
ЗАЧЕМ МНЕ НУЖЕН C++, У МЕНЯ НЕТ ВРЕМЕНИ НА ШАБЛОНЫ
ЛУЧШЕ ЕЩЕ РАЗ НАПИСАТЬ ФАБРИКУ АБСТРАКТНЫХ АБСТРАКТОРОВ
Я СЧИТАЮ ФАКТОРИАЛЫ ПО 3 РАЗА В ДЕНЬ
КАЖДЫЙ ФАКТОРИАЛ ЗАНИМАЕТ ДВАДЦАДЬ МИНУТ
Я ЖИВУ АКТИВНОЙ И ПОЛНОЦЕННОЙ ЖИЗНЬЮ
Я УСПЕШЕН И ПОЭТОМУ ЦЕЛЫЙ ДЕНЬ ЕБАШУ ФУНКТОРЫ
А ПОСЛЕ ЭТОГО ПИШУ КОМБИНАТОРЫ
ТУПЫЕ ПЛЮСОБЛЯДИ ОДЕРЖИМЫ УКАЗАТЕЛЯМИ
А Я СВОБОДНЫЙ ОТ БАЙТОЕБСТВА ЧЕЛОВЕК
СКАЧАТЬ БЕСПЛАТНО И БЕЗ РЕГИСТРАЦИИ МОКРЫЙ GLASGOW HACKELL COMPILER
КРЯК УЛЬТИМАТ КЕЙГЕН РАЗБЛОКИРУЙ MAYBE
ЛУЧШЕ Я ЕЩЕ РАЗ ПОСЧИТАЮ ФАКТОРИАЛ
И ЛЯМБДУ, УПРАВЛЕНИЕ ПАМЯТЬЮ НЕ НУЖНО
Я НЕ СЧИТАЛ ФАКТОРИАЛЫ НЕДЕЛЮ
ПОЙДУ ПОСЧИТАЮ
В HACKELL ВСЕ ПРОСТО И ПОНЯТНО
ОШИБКА PARSE ERROR (POSSIBLY INCORRECT INDENTATION). ЭТО ЖЕ ОЧЕВИДНО КАК ЕЕ РЕШИТЬ
ПРИШЛО ВРЕМЯ ПОСЧИТАТЬ ФАКТОРИАЛ
КККОКОКОКОКОКОКО
СТРАУСТРУП УКАЗАТЕЛИ STL ПИТУХИ
КОКОКОКОКОКОКО

...НАПИСАТЬ ГОСТЕВУХУ

ПРИШЛО ВРЕМЯ НАПИСАТЬ ГОСТЕВУХУ
ГОСТЕВУХА САМА НЕ НАПИШЕТСЯ
НАПИШУ ЕЕ НА PHP+MYSQL+AJAX
ЗАЧЕМ МНЕ НУЖНЫ RAILS/ASP, У МЕНЯ НЕТ ВРЕМЕНИ ЧТОБЫ ЕБАТЬСЯ С НИМИ
ЛУЧШЕ ЕЩЕ РАЗ ПРОЧИТАЮ PHP&MYSQL РУКОВОДСТВО ПРОФЕССИОНАЛА
Я ЧИТАЮ PHP&MYSQL РУКОВОДСТВО ПРОФЕССИОНАЛА ПО 3 РАЗА В ДЕНЬ
КАЖДЫЙ INDEX.PHP ЗАНИМАЕТ ДВАДЦАТЬ СЕКУНД
Я ЖИВУ АКТИВНОЙ И ПОЛНОЦЕННОЙ ЖИЗНЬЮ
Я УСПЕШЕН И ПОЭТОМУ ЦЕЛЫЙ ДЕНЬ СИЖУ НА ФРИЛАН.СРУ
ПИШУ ЗАКАЗЧИКУ ЗДЕЛАЮ ЗА ОТЗІВ
ГОТОВЫЕ ГОСТЕВУХИ ЗАЛИВАЮ НА UCOZ
ТУПЫЕ ДЖАВАМРАЗИ ОДЕРЖИМЫ EJB
А Я СВОБОДНЫЙ ОТ ЗАДРОТСТВА ЧЕЛОВЕК
<?PHP ECHO 'HELLO WORLD' ?>
$SUM = ($N & ($N % 2 ? 0 : ~0) | ((($N & 2)>>1) ^ ($N & 1)));
ЛУЧШЕ Я ЕЩЕ РАЗ СОСТРЯПАЮ ГОСТЕВУХУ
И ЗАБУДУ ПРОВЕРИТЬ НА XSS И ИНЪЕКЦИИ
Я НЕ ЧИТАЛ PHP&MYSQL РУКОВОДСТВО ПРОФЕССИОНАЛА УЖЕ НЕДЕЛЮ
ПОЙДУ ПЕРЕЧИТАЮ
В PHP ВСЕ ПРОСТО И ПОНЯТНО
Warning: Cannot send session cache limiter - headers already sent
ЭТО ЖЕ ОЧЕВИДНО КАК ЕЕ РЕШИТЬ
ПРОПИШУ В СКРИПТЕ ERROR_REPORTING(0)
КОКОКОКОКОКОКО
PHP — САМЫЙ ЛУЧИЙ ЯЗЫК ДЛЯ ВЕБ
КОКОКОКОКОКОКО

Node.js

Это серверный однопоточный джаваскрипт-движок на событиях (libev), состоящий из гугловского якобы высокопроизводительного JIT-компилятора V8 и библиотеки асинхронного ввода-вывода к нему. В библиотеке присутствует HTTP-сервер, что позволяет получить что-то в духе эрланговского MochiWeb и питоновского TornadoWeb, но позволяющее писать клиентский (браузерный/AJAX) и серверный ('скрипты') код на одном языке. Ну и конечно геморрой в стиле mod_perl + POE вам обеспечен. Тем не менее, говорят, это прогрессивно и круто. (Шутка)

Для особо одарённых, уточняю. Вышеперечисленное включает: вонючую, но встроенную вариацию memcached; невозможность без плясок с бубном, не снившихся питоновцам, задействовать более одного ядра; новые уязвимости из-за паразитной передачи данных в параллельно исполняющийся запрос; падучесть всей VM вместе с вашими фронт-эндом и бэк-эндом в стиле легендарной DOS при зацикливании или непойманном исключении в любом из обработчиков событий; возможность неправильно реализовать HTTP; феерический пул потоков для исполнения в нём unlink(); развесистые монады при вводе-выводе, не

снившиеся хаскеллистам; ну и, конечно же, необходимость писать юнит-тесты на каждый чих, потому что только джедаи в состоянии безошибочно разыменовать хеш массивов хешей хешей массивов, а а компилятор попытки присвоить ёжику зайчика не ловит.

Но и это ещё не всё! Для затягивания сроков и удорожания разработки система включает: иллюзию эрланговской изоляции посредством порождения дочерних песочниц в рамках одного потока; циклы перебора байтиков в буфере в стиле Паскаля с неявным алиасингом; отсутствие возможности читать файлы построчно.

Регресс

Пыхомакаки в своем мирке уже давно повернули стрелу прогресса в обратную сторону, надеясь в итоге залезть обратно на пальмы. Сначала начали выжигать огнём реляционки, заменяя их доступными своим имбецильным мозгам хеш-таблицами. Теперь с появлением пыха.жс убили сразу 2 зайцев: выкинули на помойку вменяемые веб-сервера вроде апача, в которые было въёбано овер9000 человекочасов, и вернулись к кооперативной многозадачности, прямо как в MS-DOS.

NoSQL

NoSQL исторически появился раньше SQL-а, собственно весь ынтырпрайз до 70-х им в жопу и долбился. Потом британский ученый изобрёл теорию РБД, появление которой привело к немедленному выметанию всего этого ёбаного хаоса с рынка, стандартизации и тотальному овладиванию SQL-а в рекордно короткие сроки. Побочным эффектом стало то, что всякое быдло начало пихать SQL туда, где он не очень-то и нужен, и очень, блядь, страдать, от того, что их гостевухи стали долго загружаться. Потом кто-то сделал фундаментальное открытие - оказывается хранить профили пользователей гостевухи в хешь-таблице в памяти и доставать их оттуда по имени гораздо быстрее, чем реализовывать EAV поверх РБД. После этого переворота в мозгах гостевушников они приняли радостно сверкать новым базвордом по своим блогам и хабро-хабрикам, радуясь, что им в очередной раз удалось повернуть стрелку прогресса вспять и укусить себя за жопу.

Oracle RDBMS

Когда начинаешь с ним работать после MS SQL, такое чувство, что тебя начинают пердолить моргенштерном в жопу. Потому что по неудобности, несоответствию стандартам, уёбищности средств разработки (oracle sql developer - это просто пездееееццц, впрочем, как и большинство программ сложнее контроллера кофеварки, написанных на джаве), надуманным сложностям на ровном месте (перенести базу на другой сервер в случае оракли - задача со звёздочкой), количеству расставленных повсюду граблей, количеству легаси-говна, хуёвости документации, кривости языка, бессистемности именования системных объектов, а также тупости и ЧСВ-шности комьюнити - оракли однозначный лидер. Зато в нём офигенные средства мониторинга. Например, ты всегда сможешь посмотреть, что было с твоей базой в любой момент времени, какие запросы выполнялись, прочитать рекомендации и запланировать их выполнение, написать хинты для конкретного запроса и заставить его выполняться так, как тебе надо, не влезая в само приложение, и всё это лениво щелкая мышкой в веб-гуйне из коробки. Для MS SQL я такого не видел. И еще все айтишные менеджеры начинают ТЕЧ когда в комнате звучит слово "оракли", поэтому оракли-свитера зарабатывают чуть больше доширака, чем их аналоги на MS SQL. Хотя в последнее время даже до них стало доходить, что оракли - это больше геморроя за те же или большие деньги, и поэтому наблюдается тенденция выравнивания в спросе на свитера и из зарплаты.

Быдлятина

Оракли - ацкая быдлятина.
Например, в 10g движок хавал селекты с /r/n в качестве перевода строки, но не хавал аналочиные апдейты. Т.е. селект и апдейт там парсился разным былокодом. Учитывая, что это говно написано на C/C++ я не удивлюсь, если там вообще используются разные библиотеки для работы со строками.
В общем да, если сравнивать СУБД с языками программирования, Оракли == C++.
Стал популярным случайно - Ларри спиздил недоделанные исходники из IBM-а и стал загонять их аж как вторую версию своей чудо-СУБД, а быдло поверило; за счет чего держится - не понятно, т.к. не обладает никакими преимуществами по сравнению с конкурентами и чуть ли не самый медленный; все от него плюются и только узкий круг ограниченных ораклисвиней, которые ничего кроме оракли не знают и не видели, кормятся опилками с распилов от неудачных внедрений и считают себя сука илитой.
И да, каждый ламерок, чинушка или олокомпьютерный свитер жаждет засунуть оракли куда только его волосатые рученки дотянутся, потому что где-то слышал, что оракли это круто. Где они все это слышат, я не знаю, как-то пора выжечь это место, чтобы зараза не распространялась.

C++

Различная паста о **НЕОСИЛЯТОРАХ**.

Повторение аргументов

Особенность неосиляторов такова, что из раза в раз повторяемый аргумент они считают побежденным.

Один из таких аргументов - про игры и C++. Неосилятор уже отбугуртил это, раны на его ньюфажеской жопе зарубцевались, и он выдумал какой-то аргумент, который предотвращает раскрытие раны.

Они уже свыклись с мыслью, что игры пишут на крестах и батхерт выходит на новый уровень - что типа все движки написали 15 лет назад, а сейчас на крестах держит только легаси, и все разработчики мечтают на самом деле копировать мир вместо присваивания. После того, как тыкнешь такого котенка в мочу новых C++ проектов, неосилятор затыкается до нового приступа, но потом он придумывает мощный аргумент (например, что новые проекты на C++ начинают пидорасы-заговорщики).

Т.е. батхертнул неосилятор на неуправляемую память, ему объяснили, что есть умные указатели. В следующий раз как только речь зайдет о неуправляемой памяти, когда ему опять скажут про умные указатели он крикнет: "ха, повторяетесь". Не смотря на то, что ничего не изменилось, этот аргумент на неосилятора уже не действует из-за взращиваемого ФГМ, защищающего жопу от старых потрясений.

ООП НА КРЕСТАХ

Ну сотри корочи. Вначале ебашили действия подряд. Потом такие: "а неплохо было бы этот кусок повторить". Запилим переходы! И запилили. И там короче заебись теперь можно даже пропускать куски! А потом ты такой хули я перехожу А ОН СУКА ОБРАТНО НЕ ВОЗВРАЩАЕТСЯ! Ну и запилили вызовы и возвраты. И теперь у нас сука ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ™. А потом ты такой бля а хули я перехожу А ОН МНЕ СТЕК ВЫЗОВОВ РАСПИДОРАШИВАЕТ?! Ну все такие ПЕРЕХОД - это зло. И такие Бёмом и Якопини пояснили их криком "ПЕРЕХОД НИНУЖЕН!11" Ну его такие выпиливают заменяя СТРУКТУРАМИ УПРАВЛЕНИЯ™. Структурное программирование у нас же! Ок. Потом ты такой а хули я глобальную переменную изменяю и потом переёбываюсь с ней по всей программе?! ИНКАПСУЛЯЦИЯ!!11 И вот уже Вирт пилит модульное программирование. Модульное программирование канеш хорошо но тебе же всё мало! Тебе надо делать массивы модулей и структуры с полями-модулями! Ну ты такой экспортируешь тип структуры из модуля и кукарекаешь на весь Bell Labs: "АБСТРАКЦИЯ!!11" как финальный аккорд ПРОГРАММИРОВАНИЯ С АБСТРАКТНЫМИ ТИПАМИ ДАННЫХ! Все почти хорошо но ты замечаешь что приходится придрачивать поля-модули к модулю внедряя в поле-модуль специфичныи функции того модуля! Ну ты такой СДЕЛАЮ НА МАКРОСАХ ЗА 20 МИНУТ!1 Сделал. А потом заёбываешься перекомпилять поля-модули под каждый чих модуля и агрясь на размер кода зопиливаешь те самые функции в отдельную структуру и переопределяя их в рантайме становишься королём ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ ПАРАДИГМЫ со страусиной типизацией!111 Вот так ёпана

Ленивое программирование

LISPеры из /s/ пользуются технологией «ленивого программирования», аналогичной en.w:lazy evaluation.

При этом программа не пишется, а сразу объявляется написанной. Фактическое формирование кода программы должно происходить при первом непосредственном обращении к ней, но поскольку никто не обращается, код тоже не формируется.

Благодаря этому LISP разработчики экономят много времени, которое могут с пользой использоваться для эффективного троллинга.

СТРАУСТРУП и ЛЯМБДА

Жил-был СТРАУСТРУП. Шел обычный, скучный день. СТРАУСТРУП занимался рутинными вещами, такими как ебля своей трехсоткиллограмовой матери в зад. Только успев кончить матери в пердак и вынуть измазанный в говне хуй, СТРАУСТРУП услышал стук в дверь. Не заметив, что его мать умерла от сердечного приступа еще 3 дня назад, он пошел открывать дверь и вышел на веранду. На веранде никого не было. СТРАУСТРУП, было, уже начал подозревать СТЕПАНОВА в очередной подъебке, но вдруг, из под крыльца что-то вылетело и понеслось на него. От перевозбуждения СТРАУСТРУП уронил скрепленнеые скотчем очки на пол и смиренно ждал, что будет дальше. ЛЯМБДА, с огромной скоростью пролетела мима СТРАУСТРУПА, квадратной скобкой отпихнула его и заползла в дом, крепко заперев дверь. ЛЯМБДА ясно дала понять, что она приняла дом в качестве аргумента, но отказывается возвращать функцию, которая принимает ПИЗДЮЛИ в качестве аргумента и возвращает дом. ЛЯМБДА заползла на стул перед компьютером и свернувшись в уютный клубок, зашла на ДВАЧ. СТРАУСТРУП знал ЛЯМБДУ. По крайней мере, это слово он точно слышал, но не знал, что оно значит. Он очень удивился, когда недавно узнал, что ЛЯМБДУ включили в НОВЫЙ СТАНДАРТ, принятый полгода назад. СТРАУСТРУП понял, что надо выгнять ЛЯМБДУ из дома, потому что желание в очередной раз залезть на мамочку было слишком велико. Будучи первоклассным инженером, СТРАУСТРУП начал искать решение проблемы. Для разминки он решил повторить таблицу умножения до 12 на 12. Он 2 часа стоял на одном месте и смотрел в никуда, потея как свинья. Пока он боролся с таблицей умножения, из за угла вышел измазанный в говне АНДРЕЙ АЛЕКСАНДРЕСКУ и осмотрелся. Рядом стояли несколько зданий, включая психбольницу для буйнопомешанных и тюрьму. АНДРЕЙ задумался и понял, что в округе нет ни одного настолько больного и гнилого человека, чтобы продать ему свою книгу. Небрежно посвистывая, АНДРЕЙ удалился. СТРАУСТРУП закончил разминку и начал думать, как прогнать ЛЯМБДУ. Вдруг его осенило. Его дом был скомпилирован последней версией GNU G++, которая поддерживает ЛЯМБДУ. Именно поэтому,

ЛЯМБДА и смогла проникнуть к нему в дом. СТРАУСТРУП понял, что ему нужна более старая версия G++, которая не поддерживала ЛЯМБДУ и тогда, при попытке компиляции дома, ЛЯМБДУ выкинет вместе с СООБЩЕНИЯМИ ОБ ОШИБКАХ. Но старую версию было негде взять. СТРАУСТРУП нанял ФУНКЦИОНАЛЬЩИКА СО ШТАНГОЙ за 5 тысяч рублей. Так как компьютера у них не было, ФУНКЦИОНАЛЬЩИК вначале написал на бумажке компьютер в 1 строчку НА ХАСКЕЛЕ:

Computer = Computer

ФУНКЦИОНАЛЬЩИК сожрал бумажку и высрал работающий системный блок с ВОДЯНЫМ ОХЛАЖДЕНИЕМ и предустановленой WINDOWS 7. На компьютере уже был установлен АЛАН ВЭЙК и ХАСКЕЛЛ ПЛАТФОРМ. Корпус был красного цвета, с наклейкой ТУРБО на прозрачной боковой крышке. Затем ФУНКЦИОНАЛЬЩИК написал в 2 строчки старую версию G++:

```
Compiler :: [C++SourceCode] -> [ExecutableFile]

Compiler source =(Link . Compile) source
```

СТРАУСТРУП взял исходники своего дома и запустил компиляцию. Компилятор начал дристать СООБЩЕНИЯМИ ОБ ОШИБКАХ. СТРАУСТРУП попытался разобрать первую строчку, но увидев такое, дальше лезть не решился(таблица умножения и так вымотала его):

```
std::map<std::basic_string<char, std::char_traits<char>, std::allocator<char> >,std::basic_string<char,
std::char_traits<char>, std::allocator<char> >,std::less<std::basic_string<char,
std::map<std::basic_string<std::map<std::basic_string<char, std::char_traits<char>, std::allocator<char>
>,std::basic_string<char, std::char_traits<char>, std::allocator<char> >,std::less<std::basic_string<char,
std::char_traits<char>, std::allocator<char> > >, std::allocator<std::pair<std::basic_string<char,
std::char_traits<char>,std::allocator<char> > const, std::basic_string<char,
std::char_traits<char>,std::allocator<char> > > > >char, std::char_traits<char>, std::allocator<char>
>,std::basic_string<char, std::char_traits<char>, std::allocator<char> >,std::less<std::basic_string<char,
std::char_traits<char>, std::allocator<char> > >, std::allocator<std::pair<std::basic_string<char,
std::char_traits<char>,std::allocator<char> > const, std::basic_string<char,
std::char_traits<char>,std::allocator<char> > > > >std::char_traits<char>, std::allocator<char> > >,
std::allocator<std::pair<std::basic_string<char, std::char_traits<char>,std::allocator<char> > const,
std::basic_string<char, std::char_traits<char>,std::allocator<char> > > > >
```

Высрав 10 000 СТРОК СООБЩЕНИЙ ОБ ОШИБКАХ, компилятор скончался от ЛЕНИВЫХ ВЫЧИСЛЕНИЙ ПРЯМОЙ КИШКИ и из монитора вылетела ЛЯМБДА. Придерживая круглые скобки квадратными скобками, ЛЯМБДА в ужасе съеблась под ближайший камень. Ей еще долго не захочется принимать и возвращать значения. Довольный СТРАУСТРУП плюнул в руку, чтобы наслюнявить хуй и уже решился залезать на мамочку, но передумал и решил вначале запостить эту историю на ДВАЧ.

АЛЕКСАНДРЕСКУ

Жил-был АНДРЕЙ АЛЕКСАНДРЕСКУ. У АНДРЕЯ всё всегда было через ЖОПУ. Мать АНДРЕЯ была наркоманкой. Список ее психических расстройств, венерических заболеваний и наркотиков, на которых она сидела, был длиннее типичного ресторанного меню. Когда пришло время рожать, каково-же было удивление врачей, когда АНДРЕЙ вылез из ЖОПЫ и каким-то образом умудрился убить и частично съесть двух медсестер. АНДРЕЙ был трудным ребенком. Когда АНДРЕЙ еще находился на лечении, на пятнадцатом году шоковой терапии и после второй лоботомии, он вдруг направил свое внимание на языки программирования. Большую часть дня, АНДРЕЙ бился головой о стену, пытался откусить кусок своего тела и ел свои экскременты. Но в перерывах между приступами, АНДРЕЙ листал книги и искал... Он прочитал про десятки языков программирования, но они не вызывали у него никакого интереса, потому что, они не были достаточно извращенными для его тонкого вкуса. Внезапно АНДРЕЙ увидел ВЫЧИСЛЕНИЕ ЧИСЕЛ ФИБОНАЧЧИ ВО ВРЕМЯ КОМПИЛЯЦИИ НА C++ и замер. В его уставшей, больной голове что-то щелкнуло - он нашел, что искал. Он начал читать книги по C++. Чем дальше он проникал в тайны C++, тем больше он понимал, что этот язык создан для него. Мерзкие извращения, которые он наблюдал на страницах, глубоко резонировали с его истерзанной и едко ненавидящей все светлое душой. Его глаза наливались кровью от удовольствия и слезы текли по щекам, от осознания, что на свете есть люди, не намного менее больные, чем он. АНДРЕЙ понимал, что скоро ему сделают третью лоботомию и тогда он вряд ли сможет написать книгу. Времени до третьей лоботомии оставалось немного и АНДРЕЙ решил начать писать книгу прямо сейчас. "THE TIME IS NOW, ANDREI", сказал он вслух самому себе на ломаном английском с выблядски кривым акцентом и начал писать. Вначале он не знал, в чем суть того, что он пишет. Но со временем картина стала ясной как день. АНДРЕЙ взял самый гнилой, уродский и омерзительный язык программирования и решил довести его до уровня сумасшествия, до сих пор невиданного в мире людей. Первый (и последний) технический рецензент его книги, сошел с ума и убил всю свою семью, после прочтения нескольких глав. Узнав об этом АНДРЕЙ смеялся, пока не потерял сознание. АНДРЕЙ понимал, что все идет как надо. Сразу после того, как он дописал последнюю главу, ему сделали последнюю лоботомию и писать книги ему больше не хотелось. Представители издателя взяли книгу АНДРЕЯ и, согласившись ее издать, спросили у него, как бы он хотел ее назвать. На ломаном, кривом английском он ответил: "MODERN C++ DESIGN: GENERIC PROGRAMMING AND DESIGN PATTERNS APPLIED BY ANDREI ALEXANDRESCU". Его акцент был настолько уебищен, что представители издателя начали ржать, с такой силой, что моча начала струиться по их ногам. Но, слишком поздно они

поняли, что это была моча АНДРЕЯ. Они не знали, что таким образом он помечает своих жертв, перед тем, как их убить. АНДРЕЙ успел убить одного, но другому удалось спастись, хоть он и лишился уха.

Через несколько лет АНДРЕЯ выпустили. 20 лет шоковой терапии и 3 лоботомии, все-таки, смогли немного успокоить его. Он, конечно продолжал убивать, но редко, и в основном мелких грызунов.

Наступил обычный, скучный день. Скучным он мог быть для кого угодно, но не для АНДРЕЯ. Ведь у него диагностировали шизофрению еще на внутриутробной стадии. Книга продавалась не особо хорошо. В мире оказалось не так уж много запредельно больных людей, готовых ее купить. Уже 2 месяца у АНДРЕЯ почти не было денег и он ел блюдо собственного изобретения - ТУАЛЕТНАЯ БУМАГА ПО ФЛОТСКИ. Блюдо представляло собой собачий корм с вареной туалетной бумагой. Роялти с продаж книги капали ему на банковский счет, но очень вяло. АНДРЕЙ уже отошел от третьей лоботомии и решил взять дело в свои руки. Он положил в сумку с десяток экземпляров MODERN C++ DESIGN и пошел на улицу, с надеждой продать хотя бы несколько. Хотя бы один. Если это удастся, то наконец можно будет купить КЕТЧУП. Подумав о КЕТЧУПЕ, АНДРЕЙ улыбнулся, но повернувшись, чтобы открыть дверь, увидел свое отражение в зеркале. Выражение лица, которое получилось из за улыбки, было настолько ужасающим, что АНДРЕЙ отшатнулся. Он вышел на улицу и стал бродить по улицам. АНДРЕЙ увидел здание, в котором было множество компаний по разработке программ и направился к нему. Там АНДРЕЯ уже знали и вызвали охрану раньше, чем он успел войти. АНДРЕЙ удивился, потому что он никогда не был здесь. Из здания вышел человек и сказал АНДРЕЮ, чтобы он убирался. Человек объяснил, что однажды, один из программистов, работавших в здании, купил себе MODERN C++ DESIGN и принес на работу. Прочитав 5 страниц, этот человек обезумел и успел убить трех коллег, до того как натолкал себе в жопу скрепок и повесился в полностью пустом помещении. После этого, запятнанную кровью книгу, подобрал другой разработчик и цепь событий повторилась. Как вирус, книга распространялась по всему зданию. В результате этой бойни, 30 человек погибли ужасными смертями, перед тем, как кто-то сообразил, что нужно уничтожить книгу. АНДРЕЙ понял, почему его не хотят пускать, но решил попытать судьбу и все-же проникнуть в здание. С раззадоренным еблом, он попытался пробежать в дверь, но охранник ударил его дубиной по еблу, выбив несколько зубов, после чего добил по яйцам, пнув достаточно много раз, чтобы наблюдающие сбились со счета. АНДРЕЙ сполз с крыльца и потерял сознание.

Очнулся он уже под вечер. Первый опыт продажи был не очень удачен, но может во второй раз повезет? АНДРЕЙ шел по улице, страстно разговаривая сам с собой и вдруг увидел двух человек. Подсознательно он узнал их, но не мог вспомнить. Трясясь от страха он подошел к ним и предложил купить книгу. ПОЛ ГРЭМ и ПИТЕР НОРВИГ взяли его книгу и стали листать. Они поняли, с кем они имеют дело. АНДРЕЙ смотрел куда-то в сторону и незаметил первого удара, который пришелся по голове. АНДРЕЙ даже в начале не понял, что происходит, потому что били его как никогда сильно. Удары сыпались со всех сторон и, услышав хруст своих ребер, АНДРЕЙ осознал, что вероятно, живым ему не уйти. Это осознание ввергло его в истерику, но он ничего не мог поделать, кроме того, как обосраться и изваляться в собственном говне. Увидев это ПОЛ ГРЭМ и ПИТЕР НОРВИГ побрезговали добивать жалкого РУМЫНСКОГО барана и оставили его в покое. Грязно выругавшись, АНДРЕЙ поднялся и пошел по улице. Завернув за угол, он увидел СТРАУСТРУПА, стоящего на одном месте и напряженно о чем-то думающего. АНДРЕЙ осмотрелся, но не обнаружив потенциальных покупателей, развернулся и пошел домой, насвистывая РУМЫНСКУЮ НАРОДНУЮ ПЕСНЮ.

СТЕПАНОВ

Жил-был СТЕПАНОВ. СТЕПАНОВ был ШУТНИКОМ. Такова была его природа. Очень любил ШУТИТЬ. В отличии от СТРАУСТРУПА и АЛЕКСАНДРЕСКУ, СТЕПАНОВ был психически здоров, и, естественно, ненавидел C++ всем сердцем. Однажды СТЕПАНОВ ПОШУТИЛ очень крепко. Он упоролся чем-то очень серьезным и написал STL. Вещь получилась комично абсурдная. СТЕПАНОВ провел много времени смеясь над тем, какое-же убогое и тупое говно - язык C++ и рассматривал STL исключительно как гротескную шутку. СТЕПАНОВ уже успел забыть про STL, но вдруг его вызвали на заседание Коммитета Стандартизации. После того как СТЕПАНОВ вышел с заседания, он начал ржать как умалишенный. "Эти уебки...", думал СТЕПАНОВ, сотрясаясь от смеха, "...приняли STL и включили ее в стандарт, мать моя женщина". Он не мог остановиться и повалился на пол. Смех его не отпустил и на полу. Звучание его смеха из обычного человеческого, превратились в какую-то неземную смесь звуков карканья вороны, со звуками блюющего носорога. Из за смеха у СТЕПАНОВА началась аритмия и пришлось вызвать врача. Вернувшись домой, СТЕПАНОВ уже не смеялся, после того, как он представил себе последствия своей ШУТКИ. СТЕПАНОВ понял, что живые люди будут писать код на STL. А вдруг кто-то вдохновится примером STL и придумает что-то еще более идиотское? СТЕПАНОВ прекрасно знал, что STL является ШУТКОЙ, но ведь могут быть больные люди, которые воспримут все это всерьез... СТЕПАНОВ не знал, что делать. Но будучи ШУТНИКОМ, он решил отвлечься и исполнить свой классический номер: ПОСТУЧАТЬ-В-ДВЕРЬ-СТРАУСТРУПА-И-УБЕЖАТЬ. СТЕПАНОВ хихикал, уже при одной мысли о такой СМЕХОТЕ, потому что СТРАУСТРУП, как безмозглый баран, каждый раз велся. Он направился к дому СТРАУСТРУПА и спрятался за кустом. Вокруг никого не было и он уже было решился подойти и постучать, но увидел как ЛЯМБДА приближается к дому. СТЕПАНОВ любил ЛЯМБДУ. Они много раз вместе пили и были хорошими друзьями. Но СТЕПАНОВ не хотел портить свою ШУТКУ и не выдал своего местонахождения. К его удивлению ЛЯМБДА подлетела к двери дома, постучала и быстро спряталась под крыльцо. Увидев это, СТЕПАНОВ немного расстроился, что ЛЯМБДА украла у него фирменный трюк. Но потом вспомнил, что ЛЯМБДА предупреждала его, что собирается так поступить, чтобы отомстить за включение ЛЯМБДЫ в СТАНДАРТ. СТЕПАНОВ вылез из кустов и пошел домой. У него было очень мрачное настроение. Он знал, что СТРАУСТРУП творит зло, но пока не решался ничего с этим поделать. "По крайней мере этот

выблядок АЛЕКСАНДРЕСКУ в психушке", думал СТЕПАНОВ. Он чувствовал, что после включения STL в СТАНДАРТ , нужно было принимать какие-то меры. Ему требовалась последняя капля... Вдруг он увидел своих друзей ПОЛА ГРЭМА и ПИТЕРА НОРВИГА. Все трое отошли в сторону и начали оживленно разговаривать, осматриваясь по сторонам. Они обменялись новостями. СТЕПАНОВ теперь знал, что АЛЕКСАНДРЕСКУ вышел из психушки, а ГРЭМ с НОРВИГОМ знали, что STL вошла в СТАНДАРТ. Они давно знали, что такое время настанет и решили пойти на самые крайние меры...

СТРАУСТРУП запостил историю на ДВАЧ и залез на мамочку. Он до сих пор не понял, что мамы уже давно нет в живых. Единственное, что он заметил - от мамы как-то странно стало пахнуть, но его это только возбуждало. К тому же мамочка стала в последние 3 дня страстно стонать, когда он особенно яростно ее трахал. Он не понимал, что эти "стоны" являются ничем иным, как газами гниения, со звуком выходящими из различных отверстий, под его напором. СТРАУТРУП почувствовал удар по голове и потерял сознание.

АЛЕКСАНДРЕСКУ, избитый и измазанный в говне, уже подходил к своему дому. Вдруг трое человек в масках схватили его. Последнее что он видел - марля пропитанная хлороформом приближалающаяся к его лицу. Затем темнота.

СТРАУСТРУП и АЛЕКСАНДРЕСКУ очнулись в темном, сыром подвале. СТЕПАНОВ заметил это и включил свет. НОРВИГ подошел к ним и снял картофельные мешки с их голов. СТРАУСТРУП и АЛЕКСАНДРЕСКУ все поняли. Осознание неминуемого конца пришло моментально. Между СТЕПАНОВЫМ и ГРЭМОМ произошел небольшой диалог:

- Выводи урода.
- Урод спит.
- Ну дак разбуди его.

ГРЭМ подошел к двери и открыл ее. За дверью было пространство метр на метр и лежал человек в кожаном костюме и маске. ГРЭМ небрежно пнул человека. Он очнулся и попросил воды. "Ну на, попей", издевательски сказал ГРЭМ, мочась ему на лицо. СТЕПАНОВ пинками выгнал человека на середину комнаты и дал в руки лопату. СТЕПАНОВ сказал : "Копай, сука." Человек начал копать яму. Прошло 10 минут и яма на одно тело была готова. СТЕПАНОВ достал крупнокалиберный магнум и наставил на человека. СТЕПАНОВ велел ГРЭМУ снять с человека кожаную маску. ГРЭМ снял маску. СТРАУСТРУП и АЛЕКСАНДРЕСКУ громко вдохнули от страха. Они знали этого человека. СТЕПАНОВ прицелился человеку в голову и выстрелил. Мозги разлетелись по всему полу и безжизненное тело упало в яму. СТЕПАНОВ смотрел на все это с улыбкой настоящего ШУТНИКА. Затем, он посмотрел на НОРВИГА и ГРЭМА и сказал:

"Закопайте САТТЕРА, а я кончу этих двух выблядков".

НОРВИГ и ГРЭМ взялись за работу, а СТЕПАНОВ, походкой Мэдсена из Бешеных Псов, подошел к СТРАУСТРУПУ и АЛЕКСАНДРЕСКУ. Они оба смотрели в пол и так и не издали не звука. СТЕПАНОВ достал КАТАНУ и с тщанием автора STL, зарубил обоих.

СТЕПАНОВ, НОРВИГ и ГРЭМ немного постояли над телами.

СТЕПАНОВ прервал молчание и сказал, чтобы трупы скормили свиньям. СТЕПАНОВ поднялся по лестнице на первый этаж дома и продолжил писать DSL на COMMON LISP.

Тимлид

Я за макинтошем. Ты за белым макбуком, и я за белым макбуком. А потом на конференцию. А потом салют... в нашу честь. Салют в нашу честь. Сначала на конференцию, а потом в коворкинг. Кодить будем. Пиво. Салют. В мою честь салют. Я тимлид, и ты — за белым макбуком. Ты меня слышишь? Посмотри, какой у меня код! Посмотри, какие у него отступы! Я его еще никому не показывал. Только тебе покажу. Посмотри! Это мой класс. Специально написал! Инициализируй, не бойся! Интерфейс? Я его поменяю! Вот он, интерфейс. Вот! Настоящий! Видишь? Тимлид. Я тимлид. Я его на гитхаб отправлю. Это мой код. Самый чистый. DRY. Что ты выгорел? Вот пойдем в коворинг, пива выпьем — и выгорать не будешь. Я, в слаке, за белым макбуком, командую репозиторием! Я тимлид! Я командую репозиторием! У меня звезды на гитхабе! За белым макбуком! Я тимлид! Я командую репозиторием!

О программировании

Программисты работают в таких огромных футуристических лабораториях, где всё белого цвета. В воздухе у них всё время летаю голограммы с зелёными нулями и единицами на чёрном фоне, они их всё время мацают руками, двигают вверх-вниз. Сами они при этом висят в воздухе, а по телепатической связи президент в мозг транслирует им сверхсекретные и особо важные задания. Друг с другом они общаются только при помощи специально оговоренных функций и только в двоичном коде. Ещё они могут силой мысли переустановить виндоус, заставить принтер работать или работать правильно, починить печаталку, создать почту, удалить баннер. Ещё если ты вдобавок ко всему хакер, то ты можешь вычислять адрес по ip, а ip - по сообщению в чате игры.

Постапокалиптика

Ты пойми, мир сдвинулся. В результате страшной катастрофы были уничтожены практически все живые существа, а выжившие давали ужасное, поразительное в своей омерзительности, потомство. Большинство умирало, 90% первого поколения были бесплодны. Но в результате жутких мутаций стало возможно межвидовое скрещивание. Из-за множественного наследования здесь можно встретить кошмарных существ: программист C++ с телом рыбы и хуем слона, труп страуса с китовым усом вместо глаз, джигурду в малиновой юбке, выдающего себя за тайскую шлюху. Здесь царят хаос и ужас с тех пор как радиоактивная пыль скрыла землю от солнца в 1983 году. Потом, когда в 1998 решили снять режим глобальной катастрофы, человечество было уже мертво, потому были законодательно закреплены права новых существ. Организмы практически избежавшие мутаций были уравнены в правах с разлагающимися на глазах и истекающими слизью трупоедами и с хищными растениями. Ты представляешь всю безысходность этого мира? Но даже здесь можно видеть лучик надежды. Существа сохранившие глаза и осколки разума тянулись, как цветы с солнцу, к ~~Роланду Дискейну~~ Хаскеллу Карри и его небольшому отряду: ~~Сюзанне~~ Алонзо Чорчу и Джону Маккарти, которые устало брели через эти пустоши. Они были ~~стрелками~~ функциональщиками. Путники несли порядок и знания именем ~~рода Эльда~~ Теории Категорий. Существа хватались за эти крупицы, неловко пытались рисовать своими шупользеподобными руками стрелки на зеленом песке, они получали квадратные скобки вместо круглых, они хотели чистоты, но лишь продолжали пачкали и без того грязные руки. Маленький отряд уходил. Хаскеллу было невыносимо больно видеть страдания этих существ, он знал, что несет ответственность. Но уходил, ведь его неудержимо тянуло к Башне. Он знал, что Бьёрн ждет его там. И там должно было все решится. Из-за Бьерна мертв Дениса. Из-за Бьерна мир сдвинулся. Хаскел шел вперед, хоть и не знал, чем это закончится, но два аппликативных функтора приятной тяжестью на поясе давали уверенность, что это лучи не будут разрушены...

ООП хорошо для GUI

О, дружище, "ООП хорошо для GUI" - это примерно такой же спид, как и "ООП хорошо для всего" - когда программисты обожглись об ООП, но проблемы гуйни глубже формошлёпства не исследовали, поэтому решили "ну ладно, ООП для %моей_предметной_области% - говно, но ведь КНОПКА - это же объект!!1". Ты ведь понимаешь, о чем я, а уёбок? Повторю другими словами на всякий случай: когда ты был маленьким и глупым и нихуя не шарил в своей предметной области, ты думал, что "всё есть (некий абстрактный) объект (про который я нихуя не знаю)" и что ООП идеально этот факт моделирует. Потом ты немного подрос и поумнел, и выделил в своей области множество взаимосвязей и закономерностей. И она наполнилась морфизмами, функторами, отражениями, естественными преобразованиями, пределами, копределами, сопряжениями и прочими знаниями, для моделирования которых убогое ООП ну никак не подходит. А гуй для тебя так и остался той областью, в которой ты нихуя не шаришь. Так вот, уёбок, не надо экстраполировать свою самоуверенность на все отрасли человеческой деятельности. Если ты охуенный спец по обработке сигналов, это еще не значит, что ты охуенный спец по чистке туалетов или лепке крудов. И если ты не видишь в лепке крудов никаких закономерностей, это еще не значит, что их там нет, и что любой виджет есть объект и не более того. Закономерностей там более чем дохуя. Я вот с ходу могу сказать, что гуйню лучше моделировать стрелками, чем объектами, а истинный спец по гуи тебе наверняка категорий пять навернёт, охватывающих визуализацию, валидацию, интернационализацию и локализацию, лейаут, помощь людям с ограниченными возможностями и хуй знает, что там еще может быть.. Поэтому пиши про то, что знаешь, а про гуй пусть лучше спецы по гуи напишут, от них то мы и узнаем, столь ли там хорошо ООП, или приходится еще каким-нибудь xml-ем в жопу поябываться.

Веб-макака - не программист

Да не писали раньше сами, уже год как есть сраный распиаренный Backbone. И он не первый. Добавили только изкоробочную синхронизацию данных с нодоподелкой.

Просто веб-макачье в очередной раз ткнули носом в MVC и паттерн observable, заодно пустив на этом волну пиара. До этого макачье не могло в простейший pubsub на 10 строчек и вместо нормального кода хуярило полнейшее спагетти из вкусной копи-пасты.

А все почему? Потому что веб-макака — не программист, а отброс, которого надо тыкать носом, пока оно не научится хоть что-то делать. Но себя макаки считают ниибаться авангардом всея технологий, поэтому ткнуть им в «мудоебы, 30 лет назад придумали Smalltalk, смотрите, блядь, как вот это говно делается просто и по-уму» нельзя (всяких штуденческих диссеров с «мы припрем в веб-браузер концепт actor'ов» было и 5 и 7 лет назад — все до единого забыты нахуй), им надо КУКАРЕКУ ПРОРЫВ В ВЕБ-РАЗРАБОТКЕ НОВЫЕ ТЕХНОЛОГИИ КОКОКО ИННОВАЦИОННЫЙ ДИЗАЙН ИСПОЛЬЗУЕТСЯ В ЛИДИРУЮЩИХ СТАРТАПАХ ДАСТИН МОСКОВИЦ ОДОБРЯЕТ КУДАХ-ТАХ-ТАХ NODEJS WEBSOCKETS HTML5 MOBILE RESPONSIBLE WEB DESIGN HATEOAS MONGODB. Тогда они обращают внимание и начинают кое-как пользоваться.

Просьбы

Когда я был студентотой, такие же наглые тупые и ленивые хуи как и ты, ко мне в очередь выстраивались, а я по наивности вам помогал. Потом только понял, что вам ничего не интересно, вы не хотите ни в чем разбираться и ничего изучать, вам нужен только готовый ответ и зачот. Ни на какую отдачу от вас, естественно, рассчитывать не приходится. Это не чсв, а презрение. А теперь заткни свою поганую хлеборезку и уебывай отсюда.

Установи мне программу

Ты знаешь, это очень обидно, когда какой-то уебок подходит, и говорит "ты же программист, установи мне программу". Это не работа программиста, говно.

Феерическая расстановка точек над Хаскеллем

Во-первых, Хаскель не всегда связан с желанием самоутвердится, хотя именно в этом качестве мы наблюдаем его чаще всего. Здесь можно поставить знак равенства между хаскелистами и линуксоидами, потому что манера поведения у них одна на два комьюинити. Случается, что человеку просто хочется развлечься, поиграть в Вов, запустить модель самолета или посчитать факториал на каком-нибудь особенно непривычном языке. Это просто игра, то есть создание искусственных трудностей для их преодоления. Конечно, в игры играть интересней, чем заниматься своими ежедневными обязанностями. Случается, что зарядившись адреналином от прохождения квеста, человеку очень хочется поделится этим со всем миром. В этом нет ничего страшного.

Во-вторых, модель предметной области не имеет никакого отношения к парадигме программирования, которую предполагается использовать. В любом случае, будут использованы одни и те же методы моделирования. Парадигма программирования имеет проявление лишь в мелких деталях реализации системы. Это можно увидеть на уровне тела функции, модуля или даже библиотеке в целом. Но если речь заходит о проектировании действительно сложных систем, то мыслить подобными примитивными категориями неприемлемо. Можно рассуждать только об автономных компонентах, выполняющих поставленную задачу. Это черный ящик и должен выполнять свою функцию вне зависимости от своего внутреннего устройства.

В-третьих, на мой взгляд, методика создания таких черных ящиков конкретно для Хаскеля проработана гораздо хуже. Если взять в качестве примера обсуждаемый здесь хаскель, то вместо четко очерченного публичного интерфеса библиотеки, мы увидим огромное количество методов с длинными сложными сигнатурами. При этом методы можно сгруппировать по названию, тогда будет видно, что мы имеем дело перегрузками одного метода, имеющим разные параметры и какие-то непонятные одно-трехбуквенные постфиксы (F, R и подобное). Среди хаскелистов распространено заблуждение, что введение алиасов для кортежей параметров не только сокращает сигнатуру, но и упрощает ее понимание. На деле, приходится находить объявление этого алиаса и наблюдать безмолвный кортеж, который, естественно, никак не проясняет ситуацию. Выходит, вместо того, чтобы открыть модуль, дать ему данные и сказать "работай", программист вынужден изучать внутренность этого самого модуля, пытаясь понять логику автора этой библиотеки. Именно это служит причиной изучения чужого кода среди хаскелистов, а не декларируемые "улучшить свой стиль программирования через чтение кода" или "удостовериться в качестве используемого компонента". Такой подход имеет право на жизнь для сильно формализованных проектов с относительно небольшим количеством компонентов и разделяемых сущностный. Стоит отметить, что гибкость, которой так хвалятся хаскелисты существует только на микроуровне.

В-четвертых, Хаскель достаточно простой язык программирования. В его основе лежит лямбда-исчисление, а это, пожалуй, самый простой и доступный формализм для вычислений. Хотя экспериментальный характер языка дает о себе знать. Никакой речи об ортогональности даже быть не может, потому что для решения одних и тех же задач можно использовать от 3 до 10 разных способов. Духом новаторства заражены и сами хаскелисты, потому открывая очередной модуль, готовишься увидеть что-то необычное. Единого стиля нет, все варьируется от личных предпочтений автора и фазы луны, в которой он писал код. Разглядывая большие библиотеки можно видеть, как один человек (sic!) одни и те же вещи описывает то с использованием list comprehension, то через функции вышего порядка, то используя нормальные лямбды, то исключительно безточечной нотацией, то с применением трансформеров монад, то с использованием ручного оборачивание, то с инстанцированем всевозможных фукторов и моноидов, то без. Конечно, стиль у программистов разных и что-то подобное можно встретить в любом языке, но количество сахара в хаскеле, делает код на нем абсолютно разнородным.

В-пятых, Хаскель несет с собой огромное количество новой терминологии. То есть предполагается, что программы на Хаскеле имеют прямое отношение к математике (топологии, теории категории и прочему-прочему). Однако, на деле выглядит так, как будто-то понадергали слов ради красоты. Интуитивное понимание тех же монад никто раскрывать не спешит, предполагая, что надо сначала разобраться с понятиями "категория", "моноид", "эндофунктор" и так далее. В самом же языке эти термины находят очень условное отображение, поскольку выполнение законов монад никем не проверяется, а лишь выражается надежда, что порождая новый инстанс программист удосужится их проверить. Получается, что у людей складывается ложное ощущение неподъемности и "математичности" Хаскеля. Хотя, если продемонстрировать им инстансы популярных монад в терминах, допустим, C#, то никому даже в голову не придет необходимость прочтения 2-3 книг по математике. Ко всему прочему класс типов Monad сломан, что делает ситуацию в некотором роде комичной.

В-шестых, несмотря на развитую систему типов для Хаскеля не существует ни одного достойного автопокомплита, действительно способного помочь в написании программ. Получается, что время сэкономленное на наборе текста за счет краткости синтаксиса Хаскеля полностью нивелируется, а в подавляющем большинстве случаев с лихвой покрывается необходимостью копаться с иходных кодах модуля или на веб-страничках, по непонятным причинам именуемых документацией.

В-седьмых, стремление к чистоте языка, его ленивость и отрицание состояний (как некоторые выше отписавшиеся) делают отладку потрясающим в своей увлекательности и нетривиальности занятием. К счастью, дебагеры существуют даже для Хаскеля (сторонникам позиции, что нельзя сделать слепок произвольного состояния вычислительной машины, в данном случае компьютера, стоит над этим подумать). Однако, как и в случае с пунктом 6 не образуется никакой единой системы для работы. В итоге, пишешь код в одном месте, дебажишь в другом, содержимое используемых библиотек изучаешь в третьем. При этом сами хаскелисты утверждают, что подобный разорванный контекст разработки является нормальным и отрицают важность полноценной IDE.

И последнее, вычисления в условиях ограниченных ресурсов оперативной памяти и процессора вынуждают форсить руками вычисления, заниматься какими-то иф-дефами, оборачивать этапы выполнения процесса монадами для явного задания переходов между состояниями как в императивном языке. Я не понимаю, почему имеет место отрицание императивного подхода, если сами моделируемые процессы имеют такую природу. Все это выливается в противопоставление Хаскеля популярным языкам программирования и постоянным байкам про переупорядочивание вычислений. А по сути, отличий не так уж и много, хотя, конечно, хочется сделать сенсацию из ничего.

Резюмируя вышенаписанное, скажу, что, на мой взгляд, ЯЗЫК ПРОГРАММИРОВАНИЯ и комьюнити, сложившееся вокруг него, не могут достойно представлять более крупное сообщество программистов, использующих элементы, которых принято относить к функциональной парадигме, в своих разработках. Огромное количество сказок и преданий, передаваемых хаскелистами из уст в уста, вкупе с ничтожно малым количеством вакансий и более чем скромными зарплатами способно только отпугнуть людей. Что категорически не соответствует идеям продвижения функционального программирования в массы.

Суть Хаскелля

Мне кажется, Вы совершенно не понимаете сути Haskell.

«Готовые библиотеки», «распространение», «бинарный дистрибутив», «коммерческое использование» — все эти термины в контексте Haskell просто не имеют смысла; Haskell работает по-другому.

А именно, Haskell — это: типы как объекты, каррированные функции как стрелки, HOFs как экспоненциалы, чистота, ссылочная прозрачность, аппликативность и свободные (при совпадении доменов-кодоменов) композиции, equational theory settings / reasoning как следствие, в терминах ТК, в том числе, начальные алгебры / финальные ко-алгебры как фреймворк для описания индуктивных рекурсивных данных (как выход - результаты, значения) и ко-индуктивных ко-рекурсивных потоков данных (как вход); функторы, монады, Kleisli категории - многие индуктивные [возможно] рекурсивные типы данных которые функторы (начиная с Identity, Maybe и List), также, обычные суммы, произведения и степени, то есть кортежи/записи, объединения/варианты и функции - writer, error и reader/environment, для функций более специального вида - prompt, parser, state и cont, par/conc/async как cont для fork/join/io/done языка; функторы, ко-монады, coKleisli категории - ко-индуктивные ко-рекурсивные типы данных которые функторы (простейшие потоки и деревья, например), те же произведения и степени (суммы?), указатели и изменяемые подструктуры (линзы, как функции в), зипперы; свободные монады вокруг типов данных которые функторы - iteratees (которые сами по себе потоки, то есть финальные коалгебры для соответвующих (строго-позитивных таки) функторов), разные языки (eDSL на ADT) и их интерпретаторы; ко-свободные ко-монады для типов данных которые функторы - ?; (под)категории и стрелки - линзы (категория, тензор, но не вполне стрелка), обычные функции, Kleisli стрелки, coKleisli стрелки, стрелки biKleisli категорий, функции ограниченные типом - списки-в-списки, потоки-в-потоки, деревья-в-деревья, сигналы-в-сигналы и поведения-в-поведения (как оно применяется в FRP) и т.п., автоматы, симуляции, преобразователи, некоторые языки-eDSL-на-ADT, опять же; монадические трансформеры как определённого вида натуральные трансформации для определённого вида функторов над разными монадами - WriterT, ErrorT, ReaderT, StateT, ContT, MaybeT, ListT и т.д., например, ReaderT (ConstEnvironment, MutableScope, Resources) IO - эффекты, injectable read-only / write окружение, список ресурсов пополняемый их захватами по мере выполнения и автоматически освобождаемый в конце; полугруппы, моноиды, сворачиваемые и обходимые типы и т.п. категорные и алгебраические типы и классы как «паттерны» и средства декомпозиции.

Скрытая угроза

Был такой дурачок, по кличке Страус, и решил он как-то сделать язык простой, народный, для своих пацанов. Ну и в итоге у него получилось такое неведомое лесное угребище, что люди над ним и его пацанами хорошенько посмеялись, так же как мы сейчас смеемся над денисками и бабушкиными. Посмеялись и забыли.

Обиженные пацаны с позором удалились в свои темные норы, но язык не забросили. Год за годом в своих

норах пацаны надрачивали на язык, с безумством больной психики изучали каждый его уголок, каждый косячок, каждую странность и каждую аномалию, бережно складируя накопленное богатство, в то время как весь остальной мир жил своей радужной и веселой жизнью, ничего не подозревая о нарастающей скрытой угрозе.

И так, год за годом, пацаны вырастали, проникали на производство и в образование и потихоньку подсаживали на свое обожаемое лесное угребще все больше и больше стратегически важных областей, и в итоге отомстили за обиду, скрытно разъев индустрию изнутри.

Отомстили и отомстили. Дальше жить надо, софт писать. А на угребище софт не писался. Невинные программисты вместо своих прежних повседневных дел стали вынуждены бороться с угребищем, со всеми его хитростями и подлянками. Программисты учуяли неладное и ополчились против пацанов, но пацаны, чтобы не потерять власть, выдумали свою религию, чтобы программистов усмирить, и записали все накопленные в своих темных норах косяки и аномалии в книгах, под видом священного писания. И теперь каждый, кто не потратил пол жизни, сидя в монастыре и изучая священное писание, объявлялся еретиком и маргиналом.

И называлась та религия ООП, ее священники - ООП-учеными, а те самые косяки и аномалии прозвали паттернами.

И сейчас, спустя много-много лет, мир постепенно осознает в какой беде находится и с каждой новой версией языков программирования пытается скинуть старые заржавелые оковы. А ущерб, который нанесли человечеству ООП-псведоученые, еще предстоит осознать.

Универсальный ответ

Анон, я хочу вкатиться в программирование, какой язык выучить? ***Языкнейм*** *пойдёт?*

Haskell

Языкнейм неудачный вариант для новичка. Язык сложный, возможностей мало, работы тоже мало, а на каждую вакансию десяток макак. Сейчас каждый школьник учит *Языкнейм*, а потом не знает что делать с ним. Лучше попробуй Haskell. На нём и конкуренция пониже, и зарплаты больше, и сам язык понятней. Если никогда не занимался программированием, то начинать лучше всего с Haskell - после него другие языки учатся быстрее. Работы полно, платят отлично. По книгам. Если есть хоть немного знаний программирования, читай это: http://www.ozon.ru/context/detail/id/30425643/ Если совсем новичок, пойдет эта книга: http://www.ozon.ru/context/detail/id/28346038/ Ну и куча онлайн-учебников. Вот, например: https://anton-k.github.io/ru-haskell-book/book/home.html Хороший учебник, всё расписано подробно. Сам по нему учился. Рекомендую.

Node.js

Языкнейм неудачный вариант для новичка. Язык сложный, возможностей мало, работы тоже мало, а на каждую вакансию десяток макак. Сейчас каждый школьник учит *Языкнейм*, а потом не знает что делать с ним. Лучше попробуй Node.js. На нём и конкуренция пониже, и зарплаты больше, и сам язык понятней. Если никогда не занимался программированием, то начинать лучше всего с Node.js - после него другие языки учатся быстрее. Работы полно, платят отлично. По книгам. Если есть хоть немного знаний программирования, читай это: http://www.ozon.ru/context/detail/id/135464486/ Если совсем новичок, пойдет эта книга: http://www.ozon.ru/context/detail/id/135466041/ Ну и куча онлайн-учебников. Вот, например: http://node-center.ru/docs/tutorials/item/56e336640f92fe075f53ed7c Хороший учебник, всё расписано подробно. Сам по нему учился. Рекомендую.

Go

Языкнейм неудачный вариант для новичка. Язык сложный, возможностей мало, работы тоже мало, а на каждую вакансию десяток макак. Сейчас каждый школьник учит *Языкнейм*, а потом не знает что делать с ним. Лучше попробуй Go. На нём и конкуренция пониже, и зарплаты больше, и сам язык понятней. Если никогда не занимался программированием, то начинать лучше всего с Go - после него другие языки учатся быстрее. Работы полно, платят отлично. По книгам. Если есть хоть немного знаний программирования, читай это: http://www.ozon.ru/context/detail/id/33524651/ Если совсем новичок, пойдет эта книга: http://www.ozon.ru/context/detail/id/34671680/ Ну и куча онлайн-учебников. Вот, например: http://golang-book.ru/ Хороший учебник, всё расписано подробно. Сам по нему учился. Рекомендую.

C#

Языкнейм неудачный вариант для новичка. Язык сложный, возможностей мало, работы тоже мало, а на каждую вакансию десяток макак. Сейчас каждый школьник учит *Языкнейм*, а потом не знает что делать с ним. Лучше попробуй C#. На нём и конкуренция пониже, и зарплаты больше, и сам язык понятней. Если никогда не занимался программированием, то начинать лучше всего с C# - после него другие языки учатся быстрее. Работы полно, платят отлично. По книгам. Если есть хоть немного знаний программирования, читай это: http://www.ozon.ru/context/detail/id/135794222/ Если совсем новичок, пойдет

эта книга: http://www.ozon.ru/context/detail/id/27822347/ Ну и куча онлайн-учебников. Вот, например: http://metanit.com/sharp/tutorial/ Хороший учебник, всё расписано подробно. Сам по нему учился. Рекомендую.

ООП - самый главный босс

ООП - самый главный босс. Но это не надолго. Ты пойми, мир сдвинулся. В результате страшной катастрофы были уничтожены практически все живые существа, а выжившие давали ужасное, поразительное в своей омерзительности, потомство. Большинство умирало, 90% первого поколения были бесплодны. Но в результате жутких мутаций стало возможно межвидовое скрещивание. Из-за множественного наследования здесь можно встретить кошмарных существ: программист C++ с телом рыбы и хуем слона, труп страуса с китовым усом вместо глаз, джигурду в малиновой юбке, выдающего себя за тайскую шлюху. Здесь царят хаос и ужас с тех пор как радиоактивная пыль скрыла землю от солнца в 1983 году. Потом, когда в 1998 решили снять режим глобальной катастрофы, человечество было уже мертво, потому были законодательно закреплены права новых существ. Организмы практически избежавшие мутаций были уравнены в правах с разлагающимися на глазах и истекающими слизью трупоедами и с хищными растениями. Ты представляешь всю безысходность этого мира? Но даже здесь можно видеть лучик надежды. Существа сохранившие глаза и осколки разума тянулись, как цветы с солнцу, к Роланду Дискейну Хаскеллу Карри и его небольшому отряду: Сюзанне Алонзо Чорчу и Джону Маккарти, которые устало брели через эти пустоши. Они были стрелками функциональщиками. Путники несли порядок и знания именем рода Эльда Теории Категорий. Существа хватались за эти крупицы, неловко пытались рисовать своими шупользеподобными руками стрелки на зеленом песке, они получали квадратные скобки вместо круглых, они хотели чистоты, но лишь продолжали пачкали и без того грязные руки. Маленький отряд уходил. Хаскеллу было невыносимо больно видеть страдания этих существ, он знал, что несет ответственность. Но уходил, ведь его неудержимо тянуло к Башне. Он знал, что Бьёрн ждет его там. И там должно было все решится. Из-за Бьерна мертв Дениса. Из-за Бьерна мир сдвинулся. Хаскел шел вперед, хоть и не знал, чем это закончится, но два аппликативных функтора приятной тяжестью на поясе давали уверенность, что это лучи не будут разрушены...

Адовый код

Мой тимлид ебашит вообще адовый код. Ну такая вот примерно парадигма усреднённая, потому что вариаций масса. Берется код, он не дополняется, дополнять — это не про моего лида. Он берет этот код, вываливает его в клайон и начинает рефакторить. Добавляет в него огромное количество трейтсов, специализаций темлейтов, констекспр ифов и функций СФИНАЕ! для производительности, дженерик лямбды повсюду. Все это компилится последним шлангом до дыма из системника. Потом пушится в мастер и собирается на конвейере. Потом лид скачивает бинари тестов и щедро обложив профайлами начинает замерять перфоманс. При этом запускает на мусорном инпуте шкрябая по нему фаззером. Смотрит в бенчмарки и приговаривает полушепотом ух бля. При этом у него на лбу аж пот выступает. Любезно мне иногда предлагает замерить каверейдж, но я отказываюсь. Надо ли говорить о том какие дичайшие ошибки компиляции после мёржей потом? Выхлох компилера такой, что терминал в корку падает.

Суть расточата

Ты СОВЕРШЕННО не понимаешь в чем суть расточата. расточат это не стековерфлоу «у меня не запускается хелловорлд, напишите мне 2 листа причин». расточат это не псевдоинтеллектуальные обсуждения на гитхабе. расточат это не гиттер, IRC или сраный продот. расточат это место, где люди могут побыть чудовищами — ужасными, бесчувственными, безразличными чудовищами, которыми они на самом деле и являются. В го нет дженериков, а мы смеемся. Код питонистов падает в проде из-за косяков с типизацией, а мы смеемся. Гоферы две недели не могут решить как писать простого бота в своем продоте, а мы смеемся и просим еще. Утинная типизация, фризы GC, касты к void* — мы смеемся. Тупые языки, национализм, дискриминация, ксенофобия, анальное рабство на интерпрайзных галерках, беспричинная ненависть — мы смеемся. Жаваскриптер убил своего PMа чтобы не отлаживать легаси — мы смеемся. Мы бездушно подпишемся под чем угодно, наши предпочтения не основаны на здравом смысле, бесцельные споры — наша стихия, мы — истинное лицо IT-комьюнити.

Сравнение React и Angular

В общем, меньше жри пропаганды. Выбор у тебя не большой.

Реакт

плюсы

- Нету убого хтмл, его можно засунуть в кишки и забыть про него. Далее там дойдёшь до стилизованных компонентов.
- Общая минималистичность и примитивность api. Малыми усилиями можно сделать/оборачивать что угодно. А можно вообще всё переписать.

- Следующая из предыдущего гибкость. Можно полностью уйти от навязываемой жертва пропаганды херни. Декоратор написал и пробрасываешь что и как угодно.
- Достаточно просто рендерить в 10 разных мест одновременно и независимо. Ты можешь построить/использовать нормальное api(как в десктопных гуёвых фреймворках).
- Это js - это обычная логика. Умеешь писать код - автоматически знаешь скриптухи, реакт и прочее. Ничего учить ненужно.
- В связи с этим ~идеальная поддержка туллингом. Поддержка тайпскрипта(как писать на скриптухе я даже не представляю).

минусы

- Разрабатывают его идиоты.
- Он базируется на полностью несостоятельных концепциях.
- Засилие эникеев, которые везде и всюду пытаются превратить его в хтмл. Это всякие api на компонентах, потому как макаки не могут писать код.
- Тысячи навязанного мусора и комьюнити сплошь состоящее из сектантов. Упомянутые выше сторы, редаксы и прочий бездарный мусор.
- Миллиарды проблем со связями с dom. Через жопу делается всё, даже базовый двусторонний биндинг. В 95% случаев ref через ref и пердолинг с самим реактом.
- Столько же проблем с передачей данных между компонентами.
- Их бесконечное кол-во, минусов

Ангуляр

плюсы в сравнении с реакт

- Перспективность. Концептуально он более состоятельный и писали его не совсем идиоты.
- Следовательно, если в рекате ты можешь что-то сделать только вопреки реакту(его комьюнити, подходу и тд), то здесь 50 на 50 и более.
- Комьюнити более адекватное. Да, всё равно много дошколятских фп-обезьян, но много и людей знакомых с разработкой. Людей для которых прикладное больше идеологического. Видно, что люди пытались решить проблемы, а не впаривать тебе убогую идеологически-обусловленную херню.
- Проблем с dom"ом практически нет.
- Проблем с передачей данных тоже.

минусы в сравнении с реакт

- хтмл со всем вытекающим. Этого уже достаточно, но. Следует из этого множество проблем, но основных две: 1) отсутствие какой-либо валидации. Текстовая хтмл-лапша валяется отдельно, никакие используемые там переменные и прочее нигде не определены. Никак непонятно что где меняется/используется. Перечислять можно бесконечно. 2) статичность хтмл"а и его убогость. Связанная с этим необходимость его расширять и как следствие появление в нём убогих форов, ифов, инлайн-логики и прочего. Очевидно, что зачем тебе там убогий огрызок ЯП, когда у тебя уже есть ЯП(правда такой же огрызок)?

И как уже говорилось - но. Первую проблему и частично вторую может решить туллинг, который есть. И если ты пишешь код не в каком-то мусорном блокноте - этот недостаток не является фатальным.

- Сложность. Тысячи либ говна с гигантскими api. Если в какой-то момент времени тебя что-то не устроит, то шансы твои не велики что-то сделать. Правда, как уже было сказано, писали всё это не совсем идиоты. В связи с этим шансы прийти в тупик малы и хоть через жопу, но путь существует.

Конечно, мало знаком с рядовой вебнёй и что там и как у вас, но. Вижу ангуляр куда более подходящим.

Реакт так же юзабельный, но только в связки с теми же styled components, mobx и прочим. Потенциально он более мощный, но скорее вопреки. Это вечная борьба.

Если ты не хочешь чего-то осиливать и хочешь быть рядовой макакой - иди в vue. Это примитивная, убогая херня для бедных. Но даже выбор vue говорит о том, что тебе близок ангуляр. Просто не осилил. Осиливай и не жри говно.

См. также

- Копипаста:Айтишная
- Копипаста:Python
- Хеллоуворлдщик
- Быдлокодер
- Админ
- Умение разбираться в чужом коде
- ЖРАТ

Рефоёбство

1. ↑ "теч" в альтернативной версии. Какой из вариантов аутентичен, доподлинно неизвестно.
2. ↑ The Art of Unix Programming, Eric S. Raymond
3. ↑ Facts and Fallacies of Software Engineering, Robert L. Glass