Функциональщина — Lurkmore

← обратно к статье «Копипаста:Программирование»

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.