Для чайников — Lurkmore

← обратно к статье «Копипаста:Программирование»

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, админиством 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[1] - 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[2] - 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 SICPе 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'е 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 связи. 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 SICPа 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.

1. ↑ The Art of Unix Programming, Eric S. Raymond
2. ↑ Facts and Fallacies of Software Engineering, Robert L. Glass