Введение

Несколько лет назад я ехал на подъемнике вместе с нашей шведской студенткой по обмену. Я спросил ее, думала ли она о том, чем будет заниматься после выпуска. Она сказала, что подумывает об инженерии и в прошлом году ходила на курсы программирования. Я поинтересовался, чему они учат. Она ответила: «Java». Я инстинктивно отреагировал: «Очень жаль».

Почему я так сказал? Мне потребовалось время, чтобы понять это. Дело не в том, что Java — плохой язык программирования; он на самом деле довольно неплох. Просто он (как и другие языки) обычно используется для обучения программированию *без передачи каких-либо знаний о компьютерах*. Если вам это кажется странным, моя книга — для вас.

Язык программирования Java был создан в 1990-х годах Джеймсом Гослингом (James Gosling), Майком Шериданом (Mike Sheridan) и Патриком Нейтоном (Patrick Naughton) из Sun Microsystems. Частично он был смоделирован по образцу языка C, который широко использовался в то время. Язык C не поддерживает автоматическое управление памятью, и потому связанные с этим ошибки были настоящей головной болью. Java намеренно исключил данный класс ошибок программирования; он скрыл от программиста управление памятью. В том числе поэтому он очень хорош для начинающих. Но для подготовки компетентных специалистов и создания качественных программ требуется нечто гораздо большее, чем просто хороший язык. Кроме того, оказалось, что Java привнес целый класс новых проблем программирования, которые труднее поддаются отладке, включая низкую производительность из-за скрытой системы управления памятью.

Как вы увидите в этой книге, понимание памяти — ключевой навык для программистов. Учась программировать, легко выработать привычки, от которых потом трудно избавиться. Исследования показали, что дети, которые выросли, играя на так называемых «безопасных» площадках, чаще травмируются в старшем возрасте, чем остальные (предположительно потому, что такие дети не знают, что падение — это больно). Аналогичная ситуация с программированием. Безопасная среда программирования снимает страх перед началом работы, но, помимо этого, нужно подготовиться к реальным условиям внешней среды. Эта книга поможет осуществить такой переход.

Почему важно программировать хорошо

Чтобы понять, почему сложно преподавать программирование без обучения тому, как работают компьютеры, сначала подумайте, как прочно компьютеры вошли в нашу жизнь. Цена на них упала настолько резко, что компьютер стал самым дешевым способом создания множества вещей. Например, для вывода на приборную панель автомобиля устаревшего аналогового циферблата дешевле использовать компьютер, чем настоящие механические часы. Это результат того, как производятся компьютерные микросхемы; они печатаются огромными партиями. Выпустить чип, содержащий миллиарды компонентов, больше не составляет труда. Заметьте, что я говорю о цене самих компьютеров, а не о цене объектов, в которые они включены. В целом компьютерный чип сегодня стоит меньше, чем упаковка, в которой он поставляется. Доступны компьютерные чипы, которые стоят копейки. Скорее всего, наступит время, когда будет сложно найти устройство, в котором нет электроники.

Огромное количество компьютеров, выполняющих массу задач, означает множество компьютерных программ. Поскольку компьютеры так широко распространены, программирование невероятно разнообразно. Подобно врачам, многие программисты становятся узкими специалистами. Можно сосредоточиться на таких областях, как техническое зрение, анимация, веб-страницы, телефонные приложения, промышленное управление, медицинские устройства и многое другое.

Но что самое странное — в отличие от медицины, в программировании можно стать узким специалистом, не являясь универсалом. Скорее всего, вам не нужен кардиохирург, который никогда не изучал анатомию, но среди программистов подобное — норма. Так ли это важно? На самом деле, множество фактов свидетельствуют о том, что это не очень хорошо, учитывая почти ежедневные сообщения о нарушениях безопасности и отзыве продуктов. Случалось, что люди, осужденные за вождение в нетрезвом виде на основе данных алкотестера, выигрывали суды о пересмотре кода алкотестера. Оказывалось, что код содержал массу багов, и обвинения в таких случаях снимались. Недавно антивирусная программа вызвала отказ медицинского оборудования во время операции на

сердце. Проблемы с конструкцией самолета Boeing 737 MAX привели к человеческим жертвам. Большое количество подобных инцидентов не внушает особого доверия.

Научиться писать код — только начало

Одна из причин такого положения дел в том, что не так уж и сложно написать программу, которая *кажется* работоспособной или выполняется без проблем бóльшую часть времени. Возьмем для сравнения изменения в музыке (не диско!) 1980-х годов. Раньше людям приходилось потрудиться, прежде чем писать музыку. Изучить теорию музыки, композицию и освоить музыкальный инструмент, натренировать слух и провести много часов за практикой. Затем появился стандарт цифрового интерфейса музыкальных инструментов (Musical Instrument Digital Interface, MIDI), предложенный Икутаро Какехаши (Ikutaro Kakehashi) из Roland и позволивший любому человеку создавать «музыку» на компьютере — без усилий. Я считаю, что лишь малая доля таких «произведений» является музыкой на самом деле; по большей части это шум. *Музыку* создают настоящие музыканты вне зависимости от того, применяют они MIDI для записи основы или нет. В наши дни программирование во многом похоже на использование MIDI. Больше не нужно потеть от усердия, или тратить годы на практику, или даже изучать теорию, чтобы писать программы. Но это не значит, что они будут хороши или надежны.

Ситуация может стать еще хуже, по крайней мере в Соединенных Штатах. Корыстные инвесторы, такие как владельцы компаний-разработчиков программного обеспечения, лоббируют закон, обязывающий изучать программирование уже в школе. В теории это звучит великолепно, но на практике это не лучшая идея, потому что не у всех есть способность к программированию. Мы не требуем, чтобы все учились играть в футбол, потому что знаем, что он не для всех. Вероятная цель этой инициативы не в том, чтобы готовить отличных программистов, а в том, чтобы увеличить прибыль компании-разработчика за счет наводнения рынка множеством неквалифицированных специалистов, что приведет к снижению заработной платы. Люди, стоящие за этой идеей, не очень заботятся о качестве кода — они также настаивают на принятии закона, ограничивающего ответственность за некачественные продукты. Конечно, вы можете программировать для развлечения так же, как и играть в футбол. Просто не ждите, что вас выберут на Суперкубок.

В 2014 году президент Обама сказал, что научился программировать. Он действительно переместил пару элементов в превосходном инструменте визуального программирования Blockly и даже напечатал строку кода на JavaScript (язык программирования, не связанный с Java и изобретенный в компании Netscape, предшественнице Mozilla Foundation, поддерживающей многочисленные программные пакеты, включая веб-браузер *Firefox*). Как вы думаете,

он действительно научился программировать? Подсказка: если вы ответите да, вам, вероятно, следует вдобавок к чтению этой книги поработать над оттачиванием навыков критического мышления. Наверняка, он кое-что узнал о программировании, но *нет, он не научился программировать*. Если бы он мог научиться программировать за час, это бы значило, что программирование очень простая штука и его не нужно преподавать в школах.

Низкоуровневые знания важны

Интересное и несколько отличающееся от общепринятого мнение о том, как обучать программированию, было выражено в статье из блога Стивена Вольфрама (Stephen Wolfram), создателя Mathematica и языка Wolfram, под названием «Как научить вычислительному мышлению». Вольфрам определяет такое мышление как «формулирование инструкций с достаточной ясностью и достаточным систематическим подходом, чтобы передавать их компьютеру». Я полностью согласен с этим определением. Фактически оно во многом мотивировало меня на написание этой книги.

Но я категорически не поддерживаю позицию Вольфрама, согласно которой будущие программисты должны развивать навыки вычислительного мышления с помощью мощных высокоуровневых инструментов (таких, как те, которые он разработал), а не изучать лежащие в основе дисциплины фундаментальные технологии. Например, из растущего интереса к статистике, а не к расчетам становится ясно, что «обработка данных» — это развивающаяся область. Но что происходит, когда люди просто загружают груды данных в причудливые программы, принцип работы которых не понимают до конца?

Есть вероятность, что они получают интересные, но бессмысленные или неверные результаты. Например, недавнее исследование («Gene Name Errors Are Widespread in the Scientific Literature» («Распространенные ошибки в названиях генов в научной литературе». — *Примеч. ред.*) Марка Циманна (Mark Ziemann), Йотама Эрена (Yotam Eren) и Ассама Эль-Оста (Assam El-Osta)) показало, что пятая часть опубликованных статей по генетике содержит неточности из-за неправильного использования электронных таблиц. Подумайте только, какие ошибки и их последствия могут вызвать более мощные инструменты в руках большего числа людей! Особенно важно принимать правильные решения, если речь идет о человеческих жизнях.

Понимание базовых технологий помогает определить, что может пойти не так. Знание только высокоуровневых инструментов ведет к постановке неправильных вопросов. Прежде чем взять в руки гвоздезабивной пистолет, стоит научиться обращаться с молотком. Еще одна причина для изучения базовых систем и инструментов заключается в том, что это дает возможность создавать новые инструменты. Это важно, ведь потребность в создателях будет всегда, даже

если их меньше, чем пользователей. Если вы изучите компьютеры и поведение программ уже не будет для вас загадкой, вы сможете создавать лучший код.

Кому стоит прочитать эту книгу?

Эта книга предназначена для тех, кто хочет стать отличным программистом. Что для этого требуется? Прежде всего хорошие навыки критического мышления и анализа. Чтобы решать сложные задачи, программист должен уметь оценивать, действительно ли его программа решает поставленную задачу. Это труднее, чем кажется. Нередко опытный профессионал смотрит на чужую программу и язвительно констатирует: «Эта штука сложна, но она не решает даже простую проблему, которая и не проблема вовсе».

Вам наверняка знаком классический фэнтезийный образ волшебника, который обретает власть над вещами, узнав их настоящие имена. И горе тем из них, кто забывает детали. Хороший программист похож на волшебника, способного держать в уме суть вещей, не опуская мелочей.

Подобно умелым ремесленникам, компетентные программисты — люди творческие. Нередко можно встретить совершенно непонятный код — точно так же многих англоговорящих сбивает с толку роман Джеймса Джойса «Поминки по Финнегану». Хорошие программисты пишут код, который не только работает, но понятен другим и прост в обслуживании.

Наконец, хороший программист должен прекрасно разбираться в том, как работают компьютеры. Невозможно решать сложные задачи, имея лишь поверхностное представление об основах. Эта книга предназначена для тех, кто изучает программирование, но при этом ощущает отсутствие глубины знаний. Она также подойдет тем, кто уже занимается программированием, но хочет большего.

Что такое компьютеры?

Обычно в ответ можно услышать что-то вроде: «Компьютеры — это устройства, с помощью которых люди проверяют электронную почту, совершают онлайн-покупки, работают с документами, сортируют фотографии и играют». Это определение отчасти является результатом терминологической небрежности, которая распространилась, когда компьютеры стали потребительским товаром. Другой популярный ответ: компьютеры — это мозги наших высокотехнологичных игрушек, таких как сотовые телефоны и музыкальные плееры. Второй вариант ближе к истине.

Отправлять электронную почту и играть в игры можно благодаря программам, работающим на компьютерах. Сам компьютер похож на новорожденного ребенка. Он на самом деле многого не умеет. Мы почти никогда не задумываемся

о механизмах физиологии человека, потому что прежде всего взаимодействуем с личностью, которая работает на их основе, точно так же как программы выполняются на компьютерах. Например, когда вы читаете страницу сайта, вы не используете для этого сам компьютер; вы читаете ее с помощью написанных кем-то программ, работающих на вашем компьютере, на компьютере, где размещена веб-страница, и на всех промежуточных компьютерах, которые обеспечивают функционирование интернета.

Что такое программирование компьютеров?

Преподаватели обучают человека выполнять определенные задачи. Точно так же начать программировать означает стать учителем для компьютеров. Программисты учат компьютеры делать то, что от них хотят.

Умение обучать компьютеры полезно, особенно когда вы хотите, чтобы они делали нечто новое для них, и вы не можете просто купить нужную программу, ведь ее еще никто не создал. Например, вы, вероятно, воспринимаете Всемирную паутину как должное, но она была изобретена не так давно, когда сэру Тиму Бернерсу-Ли (Tim Berners-Lee) понадобился лучший способ организовать обмен информацией между учеными из Европейского центра ядерных исследований (Conseil Européen pour la Recherche Nucléaire, CERN). За это он был посвящен в рыцари. Круто, не правда ли?

Обучать компьютеры сложно, но это легче, чем учить людей. Мы знаем намного больше о том, как работают компьютеры. И вряд ли техника когда-нибудь взбунтуется.

Компьютерное программирование включает два этапа.

1. Понять Вселенную.
2. Объяснить ее трехлетнему ребенку.

Что это значит? Невозможно писать компьютерные программы, не разбираясь в задачах, которые они выполняют. Например, нельзя написать программу для проверки правописания, не зная правил орфографии, а хорошую видеоигру — не зная физики. Итак, первый шаг к тому, чтобы стать хорошим программистом, — узнать как можно больше об окружающем мире. Решения часто приходят откуда не ждали, поэтому не игнорируйте что-то лишь потому, что оно не кажется актуальным.

Второй шаг процесса требует объяснения того, что вы знаете, машине, которая буквально воспринимает окружающий мир — как маленький ребенок. Эта детская прямолинейность очень наглядна в возрасте около трех лет. Допустим, вы пытаетесь выйти из дома и спрашиваете ребенка: «Где твоя обувь?» Ответ: «Вот она». Ребенок *ответил* на ваш вопрос. Проблема в том, что он не понимает, что

на самом деле вы просите его надеть туфли, чтобы куда-то пойти. Гибкость и способность делать выводы — навыки, которые дети усваивают по мере взросления. Но компьютеры похожи на Питера Пэна: они никогда не вырастают.

Компьютеры похожи на маленьких детей тем, что они не умеют обобщать. Они по-прежнему полезны, потому что, как только вы поймете, как им что-то объяснить, они будут делать это очень быстро и неутомимо, хотя и не будут обладать здравым смыслом. Компьютер может без устали делать то, о чем вы просите, не оценивая, правильно ли это, во многом как заколдованные метлы в отрывке «Ученик чародея» из мультфильма «Фантазия» 1940 года. Поручать компьютеру сделать что-то — все равно что просить джинна из волшебной лампы (не в версии ФБР[1]) исполнить желание. Вы должны быть очень осторожны, формулируя запрос!

Вы, возможно, сомневаетесь в моих словах, потому что компьютеры кажутся более способными, чем они есть на самом деле. Например, они умеют рисовать, исправлять орфографические ошибки, понимать, что вы говорите, проигрывать музыку и т. д. Но имейте в виду, что это делает не компьютер, а сложный набор кем-то написанных программ, благодаря которым он выполняет все эти задачи. Компьютеры функционируют отдельно от программ, которые на них работают.

Это то же самое, что смотреть на машину, которая движется по дороге. Кажется, что она довольно хорошо останавливается и начинает движение в нужное время, объезжает препятствия, добирается туда, куда нужно, ест, когда проголодается, и т. д. Но машина не действует сама по себе. Все это происходит благодаря связке автомобиля и водителя. Компьютеры подобны машинам, а программы — водителям. Не имея нужных знаний, невозможно сказать, что делает автомобиль, а что — водитель.

В общем, программирование подразумевает изучение того, что вам нужно знать для решения задачи, а затем объяснение этих вещей маленькому ребенку. Поскольку существует множество способов решить задачу, программирование — это не только искусство, но и наука. Оно предполагает поиск элегантных решений, а не использование грубой силы. Да, вы *можете* выбраться из дома, пробив дыру в стене, но, вероятно, намного проще выйти через дверь. Многие могут создать ресурс наподобие HealthCare.gov в миллионы строк кода, но, чтобы сделать это в тысячах строк, требуются определенные навыки.

Однако, прежде чем обучать трехлетку, нужно узнать побольше о нем и о том, что он понимает. И это не обычный ребенок — это инопланетная форма жизни. Компьютер не играет по тем же правилам, что и мы. Возможно, вы слышали об искусственном интеллекте (ИИ), который пытается заставить компьютеры действовать похоже на людей. Прогресс в этой области идет намного медленнее,

[1] Отсылка к троянской программе Magic Lantern («Волшебная лампа») для чтения зашифрованной информации на компьютерах подозреваемых. — *Примеч. ред.*

чем предполагалось изначально. В основном потому, что в действительности мы не понимаем проблемы; мы недостаточно знаем о том, как работает наш мозг. Довольно сложно научить инопланетянина думать, как мы, если нам самим неизвестно, каково это.

Человеческий мозг позволяет нам совершать действия бессознательно. Мозг появился как аппаратное обеспечение, которое затем было запрограммировано. Например, вы научились шевелить пальцами, а затем — хватать предметы. После достаточной практики вы просто берете вещи в руки, не задумываясь о механизмах, которые делают это возможным. Философы, такие как Жан Пиаже (французский психолог, 1896–1980) и Ноам Хомский (американский лингвист, родившийся в 1928 году), разработали разные теории о том, как происходит процесс обучения. Является ли мозг простым инструментом или в нем есть специальные аппаратные средства для таких функций, как язык? Этот вопрос все еще изучается.

Наша невероятная способность выполнять действия бессознательно мешает учиться программированию, потому что оно требует разбиения задач на более мелкие шаги, которые может выполнять компьютер. Например, вы наверняка умеете играть в крестики-нолики. Соберите группу людей и попросите каждого самостоятельно перечислить шаги, которые должен предпринять игрок, чтобы сделать хороший ход для любой конфигурации доски. (Ответ можно найти в интернете, но постарайтесь этого не делать.) После того как все составят списки, проведите соревнование. Узнайте, чьи правила лучше! А хороши ли были ваши правила? Что вы пропустили? Правда ли вы знаете, что делаете, когда играете в игру? Скорее всего, вы не объяснили ряд условий, потому что понимаете их интуитивно.

Итак, если это еще не очевидно: первый шаг — понимание Вселенной — гораздо важнее, чем второй — объяснение ее трехлетнему ребенку. Подумайте: что хорошего в том, чтобы уметь говорить, если вы не знаете, что сказать? Несмотря на это, нынешнее образование делает упор на второй шаг. Все потому, что гораздо легче обучать механическим аспектам задачи, чем творческим, так же как и оценивать усвоение материала. И в целом учителя не имеют достаточной подготовки в этой области и работают по кем-то созданным и предоставленным им программам. Однако в этой книге основное внимание уделяется первому шагу. Хотя книга не может охватить всё вокруг, она исследует задачи и их решения в компьютерной вселенной вместо разбора точного синтаксиса программирования, необходимого для реализации этих решений.

Кодинг, программирование, инженерия и computer science

Для описания работы с программами используются разные термины. У них нет точных определений, но есть примерные.

Кодинг — относительно новый термин, ставший популярным как часть «обучения программированию». Кодинг можно в определенном смысле рассматривать как работу переводчика. Сравним это с использованием кодов Международной классификации болезней (МКБ). Постановка диагноза врачом — сравнительно простая задача. Сложнее всего перевести этот диагноз в один из более чем 100 000 кодов в стандартах МКБ (МКБ-10 на момент написания книги). Сертифицированный специалист, который выучил эти коды, знает, что, когда врач ставит диагноз «лягнула корова», этому диагнозу нужно присвоить код W55.2XA. Эта работа на самом деле сложнее, чем большая часть кодинга в программировании, потому что МКБ-кодов огромное множество. Но процесс аналогичен тому, что сделал бы кодер, если бы его попросили «выделить текст жирным» на веб-странице: он знает, какой код использовать, чтобы выполнить задание.

Стандарт МКБ-10 настолько сложен, что немногие сертифицированные специалисты знают его полностью. Медицинские кодировщики получают сертификаты по отдельным специальностям, например «Заболевания нервной системы» или «Психические и поведенческие расстройства». Это аналогично тому, что программисты становятся специалистами по отдельно взятым языкам, например HTML или JavaScript.

Но *программирование*, то есть работа программиста, означает знать больше, чем одну-две области специализации. Врач в этом сценарии подобен программисту. Врач ставит диагноз, оценивая пациента. Это может быть довольно сложно. Например, если у пациента есть ожоги и он промок насквозь, это «неестественный внешний вид» (код R46.1) или «ожог горящими водными лыжами, первый контакт» (код V91.07XA)? После того как врач поставит диагноз, можно разработать план лечения. План должен быть эффективным; врач, вероятно, не захочет снова принять пациента, страдающего от тяжелой степени «чрезмерной родительской опеки» (Z62.1).

Программист, как и врач, оценивает проблему и находит решение. Например, есть потребность в веб-сайте, который позволил бы людям ранжировать коды МКБ-10 с точки зрения глупости. Программист определит лучшие алгоритмы хранения и обработки данных, структуру взаимодействия между веб-клиентом и сервером, пользовательский интерфейс и т. д. Это не простая «вставка кода».

Инженерия — это следующий уровень сложности. В общем, инженерия — это искусство извлекать знания и использовать их для достижения чего-либо. Можно рассматривать создание стандартов МКБ как инженерную разработку; обширная область медицинских диагнозов была сведена к набору кодов, которые было легче отслеживать и анализировать, чем записи врача. Представляет ли такая сложная система *хорошую* разработку — вопрос открытый. В качестве примера компьютерной инженерии — много лет назад я работал над проектом создания недорогого медицинского монитора, такого как те, которые вы видите в больницах. Мне было поручено создать систему, в которой врач или медсестра могли

разобраться менее чем за 5 минут без чтения документации. Как вы понимаете, для этого требовалось гораздо больше, чем просто знание программирования. И я добился цели — знакомство с моим решением занимает около 30 секунд.

Программирование часто путают с computer science. В то время как многие специалисты в области компьютерных наук программируют, большинство программистов не являются специалистами в computer science. *Computer science* — это наука о компьютерных технологиях и вычислениях. Открытия в этой сфере используют инженеры и программисты.

Кодинг, программирование, инженерия и computer science — независимые, но связанные дисциплины, которые различаются по типу и объему требуемых знаний. Специалист по компьютерным наукам, разработчик или кодер не становится хорошим программистом автоматически. Хотя книга дает представление о том, как думают разработчики и специалисты по computer science, она не сделает вас ими; для этого обычно требуется высшее образование и определенный наработанный опыт. Разработка и программирование похожи на музыку или живопись — они отчасти являются навыками, а отчасти искусством. Рассмотрение обоих аспектов в этой книге должно помочь вам улучшить навыки программирования.

Ландшафт

Компьютерное проектирование и программирование — большая область для изучения, которую я не смогу охватить здесь в полной мере. Ее можно схематично представить так, как показано на рис. 1.

Имейте в виду, что рис. 1 — упрощенное представление и что линии, разделяющие разные слои, на самом деле не такие четкие.

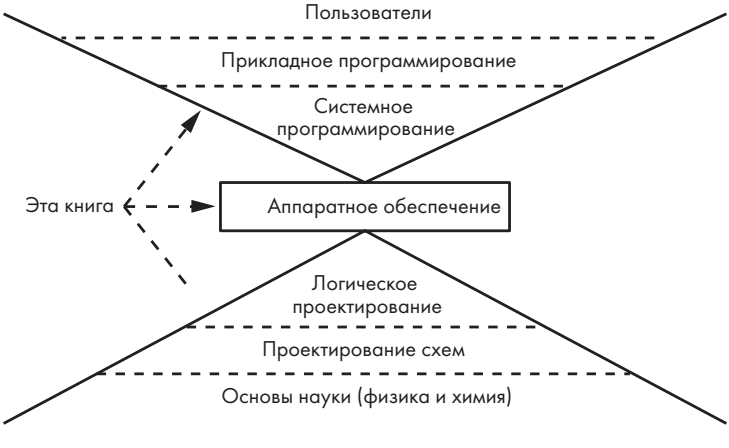

Рис. 1. Компьютерный ландшафт