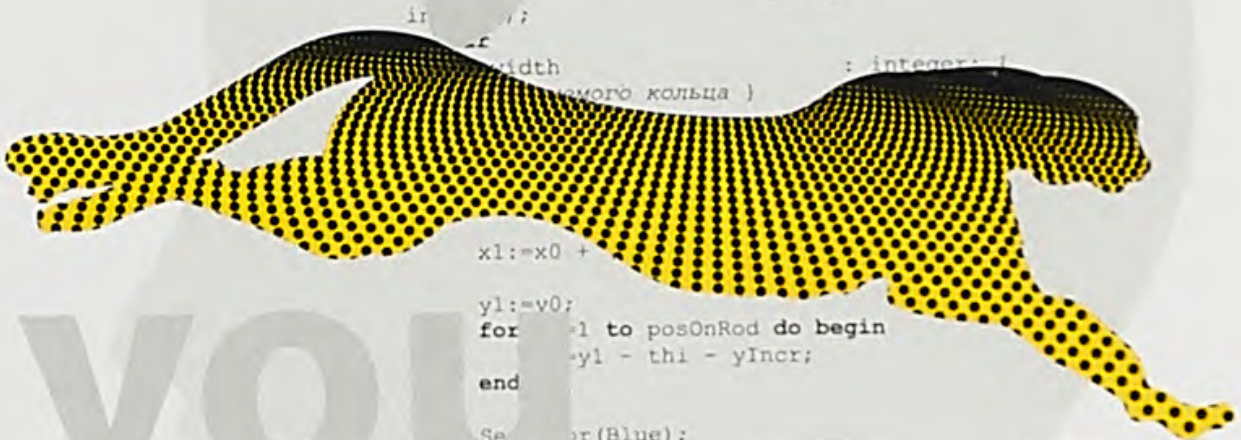
Серия «Программирование»

«СОЛОН»

Рик Гаско

Простая Математика для Простых Программистов



Серия «Программирование»

Рик Гаско

Простая Математика для Простых Программистов

Москва
СОЛОН-Пресс
2018

УДК 681.3
ББК 32.973-18
К 63

Под редакцией Н. Комлева

Рик Гаско

Простая Математика для Простых Программистов. — М.: СОЛОН-Пресс, 2018. — 260 с.: ил. (*Серия «Программирование»*)

ISBN 978-5-91359-283-5

Книга простая математика в первую очередь для программистов, но не только.

Для программистов в книге дано самое необходимое, то, без чего программисту будет трудно. Материал представлен в самом простом и правильном для немедленного употребления программистом изложении.

Настоящее применение математики программистом начинается тогда, когда он, имея перед собой задачу, сам, без ансамбля, догадывается, какую именно отрасль математики надо для такого случая вспомнить, какой конкретно метод из неё взять и как его запрограммировать. И всё это сам, без посторонней помощи.

Цель книги не в том, чтобы научить программировать математические алгоритмы. Цель в том, чтобы взять задачу и понять какой математический алгоритм к ней применить.

Прочитайте и применяйте.

Для не программиста эта книга – короткое введение в основы высшей математики.

Особое внимание уделено базовым понятиям теории вероятностей.

Straight Math for Straight Programmers. — Moscow: SOLON-Press, 2018. — 260 p. (*The "Programming" series*)

По вопросам приобретения обращаться:
ООО «СОЛОН-Пресс»
Тел: (495) 617-39-64, (495) 617-39-65
E-mail: kniga@solon-press.ru, www.solon-press.ru

ISBN 978-5-91359-283-5

Посвящается
тем непростым программистам,
которые всё-таки знают математику

Содержание

Вступление

Здравствуйте, простые программисты!

В словах этих ничего обидного нет, я и сам, в сущности, простой программист. Но только я ещё и математик, так написано в моём, а возможно, и вашем, дипломе, хотя, скорее всего, нет. Да, может быть, у вас в дипломе написано, что вы отучились на факультете прикладной математики, да и то не факт. Но по специальности вы, наверное, какой-нибудь специалист по информатике, информационной безопасности, безопасной экономике или экономичному веб-дизайну.

Я где-то уже говорил, но я повторю, я упорный. В моё время прикладных математиков очень плохо учили программированию и тому, что сейчас называется Computer Science или IT. Количество учебных часов было таким же как и сейчас, но на IT, как я сказал их выделялось много меньше, что, конечно плохо. Это минус к советской системе образования. Плюс к советской системе образования в том, что мы нигде не работали, кроме каникул и второй половины пятого курса. В результате, волей неволей, гораздо больше времени оставалось на собственно математику и, вообще, учебный процесс.

Не то, чтобы этим временем я очень хорошо воспользовался:

В те дни, когда в садах Лицея
Я безмятежно расцветал,
Читал охотно Апулея,
А Цицерона не читал
© Наше Всё Александр Сергеевич

Цицерон, если что, это такой древнеримский общественный деятель, депутат и вообще болтолог, во всех отношениях. Апулей наоборот, писатель тоже древнеримский, но более непристойной направленности. В советские времена Апулея издавали массовыми тиражами, потому что иначе нельзя, классик всё-таки, но отдельные фразы были совсем непристойны, их так и печатали, на древнеримском языке. Я латынь имею в виду, если кто-то не понял шутку юмора. Я латынь тогда не знал, да и сейчас не знаю.

Это лирическое отступление намекает на тот факт, что в рамках учебного процесса я злостно пренебрегал функциональным анализом и уравнениями математической физики. Мне нравилось программировать. Я программировал программы, которые мне надо было программировать для получения зачётов и сдачи курсовых. Поскольку этого мне было мало, я программировал программы для девушек из нашей группы.

Так вот, я – математик, и не только потому, что у меня в дипломе написано *Математик* орфографически правильно. Я математик потому, что думаю как математик, смотрю на мир как математик, рассуждаю как математик, и это не лечится. Я очень плохой математик, я могу взять только простейший интеграл и не могу решить даже простейшего дифференциального уравнения, но я всё равно математик. Сам не похвалишь, никто не похвалит.

Всё предыдущее было аргументированным объяснением того факта, что эту книгу должен был написать именно я. И почему именно я напишу её лучше всех.

Upd. В процессе работы над книгой выяснилось, что не так давно окончившие прикладную математику коллеги катастрофически не разбираются в предметах, которые нельзя сейчас и немедленно запрограммировать. Например – теория вероятностей. Поэтому я не стал углубляться в дебри – хотя, какие это дебри - и предпочёл несколько, много раз повторить основы. Это не потому, что я ничего больше не знаю. Это не я такой, это жизнь такая.

О чём эта книга?

Это очевидно из названия. Книга эта о математике для программистов. Надо только уточнить, что такое *простая математика* и кто такие *простые программисты*. Но сначала о том, как вообще связана математика с программированием.

Бывает, что программисту выдают какой-то детально расписанный математический алгоритм и требуют запрограммировать его, ни на шаг не отступая от задания. Это скучно, неинтересно и, главное, никакого отношения к математике не имеет. С тем же успехом программисту могли

бы выдать и детально расписанный алгоритм из какой-нибудь другой области знания – бухгалтерии, астрономии, игры в очко. В большинстве программ приходится хоть что-то считать. Должен заметить, что программист, которому выдают детализированный алгоритм, обычно получает не очень много денег – деньги у него отнимает тот, кто алгоритм для него детализирует.

Бывает, что программисту выдают *не вполне* детализированное задание. К примеру, там может быть требование «вычислить расстояние между точками» или «найти минимум на интервале». Предполагается, что программист понимает, о чём идёт речь, как минимум, и умеет это сделать, как максимум. Такой программист ценится выше.

- *Граф сто́ит больше!* © Собака на сене

Настоящее применение математики программистом начинается тогда, когда он, имея перед собой задачу, сам, без ансамбля, догадывается, какую именно отрасль математики надо для такого случая вспомнить, какой конкретно метод из неё взять и как его запрограммировать. И всё это сам, без посторонней помощи.

Основная проблема тривиальна, банальна и очевидна – если вы чего-то (из математики) не знаете, то не сможете этого применить. Что хуже, вы даже не догадаетесь, чего именно вы не знаете и не полезете искать это в Интернете. Нет, всё ещё хуже – вы даже не догадаетесь, что вы чего-то не знаете и вам надо что-то искать. Ещё раз и другими словами. Цель моя не в том, чтобы научить программировать математические алгоритмы. Цель в том, чтобы взять задачу и понять какой математический алгоритм к ней применить.

Ещё бывает просто нужная математика, сама по себе, настолько нужная, что адекватный программист даже её не замечает. Ведь никто не требует от программиста знать таблицу умножения. Он просто должен знать её и всё! Точно также программист должен уметь перевести из одной системы счисления в другую ($FF_{16}=255_{10}$) и знать основы математической логики, кажется, это называется *исчисление предикатов* - заменить условие (((x>=0) **and** (y<=pi)) **or** pofig) на противоположное. Если вы ешё не в курсе, это повседневное занятие программиста.

И ещё одно уточнение. Есть такой раздел математики – вычислительная математика. В Википедии можете не смотреть, там какая-то неведомая чепуха. Смысл вычислительной математики в том, что у нас есть какая-то мутная формула, а нам задают конкретный вопрос – сколько грамм тротила класть. Вычислительная математика позволяет получить разумный ответ на разумный вопрос – в пределах плюс-минус полкило, конечно.

Теперь о том, почему у нас математика именно *простая*. Потому что математика будет действительно простая, очень простая, минимально простая, настолько простая, насколько я сам в состоянии понять. А почему для *простых* программистов? Потому что я сам просто программист и эта книга написана программистом для программистов. Потому что, кто, если не я?

А будет ли здесь у нас то, что называется Computer Science? Нет, не будет, потому что я считаю присутствие *её* и *здесь* странным и нелепым.

Как устроена эта книга?

В каждой отдельно взятой главе будет рассматриваться один отдельно взятый раздел науки математики. Глав предполагалось изначально ровно двенадцать, так что на каждую главу приходилось страниц двадцать-двадцать пять. Это нелегко, пересказать весь функциональный анализ на двадцати страницах, но я и не собираюсь. В смысле, о функциональном анализе в этой книге не будет ни слова, ни к чему он простому программисту – это хорошо. Плохо то, что и остальные разделы математики я не смог упаковать в заранее заданный объём, поэтому пришлось урезать и сокращать. И, снова хорошо, то, что не влезло в эту книгу, я затолкаю в следующую. В этой книге будет больше математики. А в следующей книге, неотъемлемо связанной с этой, будет больше алгоритмов. В первой книге будет больше о том, как понимать, во второй книге – больше о том, как делать.

Каждая глава состоит из математической части, в смысле формул, из практической части – как это запрограммировать, из прикладной части – куда это можно применить, и из онтогенетически экзистенциальной – а нужно ли *это* применять. Не обязательно именно в таком порядке.

UPD – в процессе написания главы растут, увеличиваются, раздуваются и лопаются – уступают место другим, более скромным главам. Естественный отбор, однако.

И ещё раз UPD. Я совершенно не стараюсь выглядеть умным. В процессе написания вроде бы простой программы, или применяя вроде бы простую формулу, или тем более решая вроде бы элементарную задачу самостоятельно, я регулярно ошибаюсь. Большинство этих неверных поворотов и зигзагов ликвидированы, спрямлены и в тексте книги никак не отражаются. Тем не менее, некоторые непонимания и промахи, имеющие типичный и назидательный характер, я честно пересказал – в педагогических целях.

На каком языке мы будем программировать?

На Delphi, естественно. Почему-то, когда дело доходит до изложения математических и не совсем математических алгоритмов, все резко понимают, что Паскаль и производные языки от него очень удобные и приятные для этого средства.

Самый великий в мире программист Дейкстра программировал – для своих книг только – на неведомой фигне собственного изобретения и был счастлив. Фигня эта подозрительно напоминала Паскаль, но с хитрыми подвывертами.

Что почитать?

Обычно я даю список основной литературы сразу, немедленно, во введении. Здесь подход другой – литературы будет много, если перечислить её сразу и всю, - всё равно, немедленно забудут. Поэтому ссылки на необходимые книги будут выдаваться по мере необходимости, в каждой главе.

В процессе написания этой книги я прочитал, врать не буду, огромное количество математических книжек. Теперь вам их читать не обязательно, а которые обязательно надо, я перечислю особо. Многие из этих книг я прочитал даже на бумаге, и все они были куплены за настоящие деньги (!), причём, как правило, мои личные деньги.

А здесь я перечислю несколько книг, что называется, вообще, не относящихся к конкретным разделам математики.

Г. Г. Харди «Апология математика», Ижевск, 2000
G.H,Hardy "A Mathematician's Apology', Cambridge, 1967

Харди, знаменитый английский математик, работал вместе с другим знаменитым английским математиком, Литлвудом, который тоже написал книгу на свободную тему:

Литлвуд Дж. «Математическая смесь», М. Наука
J.E.Littlewood "A Mathematician's Miscellany", Cambridge University Press

Эти две книги себя взаимно дополняют и объясняют. Но книга Литлвуда написана с юмором, а книга Харди грустна и печальна. Причина понятна – автор перенёс что-то типа инсульта, вполне реабилитировался, но у него начисто отшибло все математические способности. После чего он и написал книгу. Когда я читал книгу Литлвуда, то удивлялся прорывающейся нелюбви автора к прикладным математикам. Чего нас не любить, ведь мы белые и пушистые? Книга Харди это объясняет. Прикладные математики применяют математику к чему-то *полезному.* А всё полезное является потенциально военным. Поэтому заниматься надо абсолютно бесполезными предметами – теорией чисел и атомной теорией. С атомом всё ясно, что до теории чисел, напомню, что если кто-то изобретёт быстрый способ разложения числа на простые множители, у спецслужб возникнет проблема. Им немедленно станут доступными миллиарды накопленных ими и не ими зашифрованных алгоритмом RSA сообщений, проблема только в том, с чего начать чтение?

— Девушка, вы что будете, водку или самогон?
— Ой, ну я прям даже не знаю, все такое вкусное.....
© Анекдот. Вы заметили, насколько аккуратно я соблюдаю авторские права?

Стюарт Й. "Величайшие математические задачи", М. Альпина, 2015

Содержание этой книги далеко выходит за пределы моей книги. Очень далеко, настолько далеко, что вроде бы уже и бесполезно. В этом и польза – посмотреть с высоты птичьего полёта на наш муравейник – или

курятник – и осознать, насколько мелко и убого всё то, что мы здесь обсуждаем. Кроме того, автор обладает талантом очень просто излагать очень сложные вещи. Эту книгу я ещё вспомню – но не подобострастно, а осуждающе.

И ещё о пользе от Интернета, Word заставляет писать это слово с большой буквы. От Интернета польза в математическом отношении, безусловно, есть. Раньше как было? Идёшь в библиотеку, берешь книгу по математике, читаешь. Сейчас всё не так.

- Как будут размножаться люди при коммунизме?
- При помощи света и пара.
- То есть?
- Заходит пара в комнату, гасит свет, а дальше всё как при социализме ©
Дряхлый советский анекдот

То есть, теперь вы идёте в Интернет (с большой буквы) и скачиваете книгу по математике в формате DjVu. Дальше всё как раньше. В библиотеку ногами больше идти не надо, другого счастья от Интернета нет.

Чуть не забыл, но в интернетах напомнили – исполнилось тридцать лет со дня выхода самой главной книги:

Edsger W. Dijkstra "A Discipline of Programming" 1976

На русский её перевели через два года. Как меланхолично заметил напомнивший о юбилее мыслитель из Интернета – в сущности, за тридцать лет ничего и не изменилось.

И, в последний момент, обнаружилась неплохая книжка для общего математического развития:

Ф.Журдэн «Природа математики», Одесса, 1923, гос. Типография имени тов. Троцкого

Что почитать. Особый случай. Кнут

Применительно к теме этой книги вся литература делится на две категории – Кнут (Donald Ervin Knuth) и всё остальное. Кнут – это такой специально

обученный американский компьютерный мужик. Когда-то давно, когда даже я был маленьким, Кнут начал писать многосерийную книгу «Искусство программирования» и стремительно написал три тома. Я их читал, ещё, когда был студентом. Потом процесс заглох, и к настоящему времени он смастерил только ещё половину четвёртого тома. Книги от издания к изданию постоянно дорабатываются, перерабатываются и дополняются, но книги настолько огромные, а процент того, что дойдёт до благодарного читателя, настолько невелик, что абсолютно безразлично, какое издание читать. Официально это называется так:

Дональд Кнут «Искусство программирования»
Donald Knuth "The Art of Computer Programming"

Разумеется, автора можно размазать по отдельным темам, но ввиду глобальности и необъятности личности предпочтительно рассмотреть вообще и сразу.

Любой современный автор, пишущий о математике для программистов, использует в своём сочинении какой-то язык программирования. Авторы древнего замшелого прошлого зачастую ограничивались одними формулами, но это не наш метод. Язык программирования бывает или реальным, или выдуманным лично автором специально для этого случая. Реальный язык обычно Паскаль, выдуманный – что-то подозрительно на него, Паскаль, похожее. Так поступил Великий Дейкстра в своей *Дисциплине программирования*.

Кнут пошёл другим путём. Кнут изобрёл машину MIX. Ещё Кнут изобрёл машинный язык для своей машины по имени тоже MIX, и, чтобы два раза не вставать, заодно и язык ассемблера для машины MIX под названием MIXAL. Вы знаете, что неправильно говорить *программировать на ассемблере*, без уточнения того, на ассемблере какой именно машины мы программируем? Но и это ещё не всё. Кнут придумал ещё и язык программирования высокого уровня – PL/MIX, замечательно мощный, изящный и понятный даже мне. На этом хорошие новости кончаются. Описание его будет опубликовано в десятой главе многотомного произведения, не моего произведения, кнутовского, а применение языка высокого уровня начнётся, видимо, в одиннадцатой. Даже черепаха Фиделя Кастро столько не живёт.

Изобретённые Кнутом машина, язык и ассемблер очень настоящие, сто пятьдесят с лишним команд, регистры разной длины и разного назначения. Чтобы мало не показалось, Кнут изобрёл для своей машины байт собственной, кнутовской, длины. Потом, много лет спустя, ему стало стыдно (за нестандартный байт), и он стал оправдываться, что в его время байт у каждого был своей персональной длины, а у некоторых, более того, даже длины переменной.

Я не самый умный, самые умные уже высказались раньше меня. Тогда, когда я читал Кнута, я был студентом третьего курса и верил всем его аргументам - почему машинный язык это самое лучшее, что может быть для обучения искусству программирования. Так что все мои претензии к нему – это всего лишь претензии того, кто не умнее автора, потому что умнее, а всего лишь кажется умнее, потому что прошло уже много лет. С другой стороны, в последних изданиях (которые я читал) девяностых годов, автор признаёт, что MIX устарел, и обещает перейти на MMIX, гораздо более продвинутый и основанный на архитектуре RISC. Вы помните архитектуру RISC? Я помню. Мне сразу показалось, что это (RISC) очередное фуфло, наподобие корма для кошек, который стал ещё вкуснее, так оно и вышло.

// фрейдистское объяснение того, для чего Кнуту это надо
Я играю в игрушки и мне не стыдно, поскольку я старенький, я играю в старенькие игрушки, некоторые даже из-под DOS, например Panzer General I. Просто запустить под Windows эти игры давно уже невозможно, приходится пользоваться эмуляторами, например DOSBox (это бесплатная реклама). Для запуска игрушки надо в окне ввода DOSBox'а набрать такие команды:

```
mount k e:\games\pg-1
k:
pg
```

Для чего разработчикам программы это нужно? Неужели они не могли реализовать это одной строкой для пользователей программы, ведь *они тупые* © М.Задорнов? Или, после первого запуска, с полным набором пути, запомнить навсегда, где находится это самое `pg`? Могли, разумеется, разработчики ведь не тупые. Зато теперь любой пионер, набравший эти три строки, чувствует себя крутым хакером и начинает любить программу, возвышающую его в этот статус.

Вот и с Кнутом случилось что-то подобное. Он решил почувствовать себя программистом и даже больше чем программистом – Разработчиком Машинной Архитектуры.
// конец Фрейдистского объяснения

Вы имеете право задать резонный вопрос – как же так, если Кнут такой неумный, зачем я выделил для него отдельный подраздел? Да потому, что кроме глупого выбора языка иллюстрации алгоритмов, во всём остальном он прав. Читайте и перечитывайте. И если вам кажется, что он неправ – то он всё-таки прав. А недостатки – ну что недостатки…

Толпа жадно читает исповеди, записи etc., потому что в подлости своей радуется унижению высокого, слабостям могущего. При открытии всякой мерзости она в восхищении. Он мал, как мы, он мерзок, как мы! Врете, подлецы: он и мал и мерзок не так, как вы, — иначе! © А.С.Пушкин

Кнут прав даже в отношении своего компьютера MIX, прочтите и запомните: *Характерная особенность компьютера MIX состоит в том, что он является двоичным и десятичным одновременно. Программисты MIX на самом деле даже не знают, с какой арифметикой они программируют – двоичной или десятичной.*

Вот над этим местом надо долго думать, или не думать вообще. Думать – программистам старшего поколения, у которых в голове прошито, что компьютер считает в какой-то загадочной двоичной системе. Не думать – программистам помоложе, потому что некоторые из них упорно пытаются думать, и пристают в лучшем случае с вопросом: *откуда растут байты*, а в худшем – *слева или справа в байте младший бит?*

Кроме того, у нас с ним куча общего, только он об этом не догадывается. Он, как и я, пообещал написать кучу книг и не написал (пока). Он, как и я, считает своим достоинством то, что не учит пользоваться чужими программами, а учит писать свои. А ещё он, как и я, выделяет формулы и код пустыми строками сверху и снизу, и я считаю это правильным!

Я, я, снова я… © День сурка

И, как и я, Кнут пишет только о вечном. О недостатках – Кнут скучноват. То есть, это о втором томе можно сказать, что он скучноват. Первый том откровенно скучный. У Пушкина есть героиня, решившая прочесть роман модного английского автора Ричардсона. Её честно предупредили, что первые шесть томов скучноваты, зато потом пойдёт трэш и угар. Она честно прочитала первые шесть томов, и внезапно оказалось, что все умерли, и вообще – конец. Оказалось, что в каждом *физическом томе*, который она читала, было напечатано два *логических тома*. Чисто программистская ситуация и терминология, кстати.

Со мной гораздо веселее и пишу я сжато. Моя книга намного короче не потому, что мне нечего сказать, и не потому, что я умнее. Я обладаю тем преимуществом, что могу не объяснять многие вещи, которые он должен был объяснять, потому что за прошедшее время они, эти вещи, стали очевидными и общеизвестными.

Замеченные очепятки и ашипки. Короче, работа над ошибками

Нет, это не об ошибках в этой книге. Они, само собой тут есть, но ещё пока скрываются. Они есть в предыдущих книгах, и я должен за них извиниться, потому что я очень скромный, аккуратный и самокритичный.

В моей первой книге, той, которая про Паскаль всех видов - примерно двух – была, да так и осталась, масса опечаток. Причина в том, что я написал учебник программирования применительно к языку Турбо Паскаль. Оцените тонкую разницу – не учебник программирования на языке ТП, а учебник программирования, в котором, учебнике, в качестве языка используется ТП. Большая разница.
Затем я переписал этот учебник программирования с ориентацией на язык и среду программирования Pascal ABC. Результат имел некоторый успех и переиздавался. Сейчас тренд качнулся куда-то вбок, и я готовлю к изданию исходный вариант, в улучшенном и расширенном виде, разумеется.

Так вот, переписывание книги с Turbo Pascal на Pascal ABC заняло ровно две недели, включая осознание факта существование этого самого ABC и его освоение.

Дело для нас новое, незнакомое © Операция Ы

В связи с этим на доработку текста напильником времени катастрофически не хватило.

Во второй книге, по ОПП, я был гораздо внимательнее и аккуратнее. Я вообще очень аккуратный. Позже я обнаружил, что некоторые концепции и конструкции в Delphi я описал так, как они *должны* работать. Оказалось, что по факту работают они *не так*.

Основная ошибка третьей книги – я, по глупости, написал слово *оседелец*. Тут же, как мухи на мёд, слетелись очень грамотные люди, начавшие объяснять, что так это слово пишут только те, кто пишет слово *бандеровец* через «е» - *бендеровец*. Так это слово пишут абсолютно неграмотные москали и, не побоюсь этого слова, угрокацапы, не знающие, что бендеровец через букву «е», это уроженец молдавского города Бендеры, которых бандеровцы через букву «а» животворяще аннигилировали в ходе вразумительных погромов.

Зато теперь я точно знаю – Карл Маркс и Фридрих Энгельс – не четыре человека, а два. А Бандера, Бендера и Бандура – целых три.

Да теперь я знаю, что правильно писать *оселёдец*, потому что проверочное слово *селёдка*, которую этот предмет физиономии и напоминает при внимательном рассмотрении.

Ещё должен признаться, что посвящение к предыдущей книге (*Полезное программирование*) было немного другим. Изначально там красовалось *Посвящается жалким ничтожным людишкам, которые ждут от программирования только пользы*. Издатель™ решил, что это может обидеть потенциального читателя. Я согласился и заменил - *Посвящается тем, кто ждёт от программирования пользы и денег, лучше, конечно, денег*. Ну, здесь-то что обидного? Но Издателю™ и это не понравилось, в результате там теперь то, что там теперь есть. А почему я про это здесь так смело пишу? А потому, что до этого места Издатель™ всё равно не дочитает.
Шутка © Операция Ы

И, совсем последнее замечание – меня спрашивают, к чему там, в *Полезном программировании* ссылка на Ферсмана? Мне кажется, что это

место у Ферсмана в чём-то весьма близко описывает подготовку к процессу поиска ошибки в программе.

И совсем самое последнее замечание и цитата:

Однажды я предложил Харди найти опечатку на одной странице нашей общей работы. Он не мог её найти. Опечатка была в его собственном имени "G.H.Hardy" © Литлвуд "Математическая смесь"

Ложка дёгтя в моей бочке варенья

Длинная и печальная цитата:

Сущность заключается в том, что математическая подготовка почти не нужна в компьютерном программировании. Тот тип организационного мастерства и аналитических способностей, который нужен для программирования, связан полностью с гуманитарными науками. Логика, например, преподавалась на философском факультете, когда я был в университете. Процесс, используемый при проектировании и написании компьютерных программ, почти полностью идентичен тому, который используется, чтобы сочинять и писать книги. Процесс программирования совсем не связан с теми процессами, которые используются для решения математических уравнений.
Здесь я делаю различие между информатикой (compiter science) – математическим анализом компьютерных программ – и программированием или разработкой программного обеспечения – дисциплиной, интересующейся написанием компьютерных программ. Программирование требует организационных способностей и языковой подготовки, а не абстрактного мышления, необходимого для занятий математическим анализом (в университете меня заставляли проходить год на лекции по математическому анализу, но я никогда из него ничего не использовал ни на занятиях по информатике, хотя для них матанализ был необходимым условием, ни в реальной жизни).
© Голуб Ален И. "Веревка достаточной длины …"

Это что же получается, для программирования математика вообще не нужна, и эта книга абсолютно напрасна? И всё зря? И надо выпить яду и убиться об стену? Только не подумайте, что я перешёл на уже заплесневевший молодёжный стиль языка, чтобы к ней, молодёжи,

подмазаться. Когда я был относительно молодой и местами в *сети друзей* ФИДО, там все так изъяснялись.

Некоторые замечания с моей стороны, почему автор, конечно, почти абсолютно прав, но, с другой стороны, всё-таки не совсем прав. Тонкие импортные различия между computer science и собственно программированием мы посылаем в игнор.

Чтобы программировать, математику знать не нужно, если понимать под математикой высшую математику. Арифметику знать надо, по любому, и даже в очень расширенном варианте, и основы математической логики. Это для всех программистов.

Дальше начинаются нюансы. Если вы сферический конь в вакууме – к примеру, что-то связанное с базами данных – то, пожалуй, математика и не понадобится. Но стоит выйти за узкие рамки в мир реальности, как математика ползёт и лезет изо всех щелей. Когда я работал в системе ГУИТУ, большинство обходились арифметикой и шестнадцатеричной системой счисления – без неё нигде и никак. Но конкретно мне повезло – всех смыслах – заняться задачами оптимизации заказов и транспортировки продукции. Использовалось для этого линейное программирование.

Извините, если это для вас очевидно, но линейное программирование, нелинейное программирование, целочисленное программирование – это вообще не программирование, а методы оптимизации.

А Слава КПСС – вообще не человек! © банальность

Программировать всё это богатство мне, разумеется, не пришлось – для этих целей на просторах Советского Союза единообразно применялся пакет программ ЛП АСУ. Но мне неизбежно пришлось понимать, что же он мне пишет в ответ на поставленную задачу – а писал он много, и в расчёте на человека, который понимает. Но на тот момент и этап, то есть во время работы в ГУИТУ, мои математические знания намного превосходили потребности моей работы.

Кстати, а вы знаете, что такое ГУИТУ?

Брежнев в Ташкенте.

— Салям алейкум! — кричат ему трудящиеся.
— Алейкум салям! — отвечает натасканный по такому случаю вождь.
— Салям алейкум! — кричат ему.
— Алейкум салям! — отвечает он.
— Архипелаг Гулаг! — кричит подскочивший диссидент.
— Гулаг архипелаг! — отвечает Леонид Ильич.
© старый добрый советский анекдот

Архипелаг ГУЛАГ сейчас вроде бы изучают в школе по литературе, так что юному читателю это гипотетически должно быть знакомо. Этот самый ГУЛАГ в реинкарнации моего времени назывался как раз ГУИТУ, потом как-то по-другому, и ещё как-то по-другому, а теперь уже совсем по-другому. Но он никуда не исчез, да и исчезнуть не мог – ни одна страна не может жить без структуры по имени ГУЛАГ.

Глава 1
Просто цифры.
А также системы счисления и кодировки

Зачем что-то ещё читать о цифрах?

Эта, самая первая глава, не совсем о математике. Точнее, начнётся она как сугубо математическая, но потом свернёт немного вбок, на привычную компьютерную тематику. Зато все следующие главы, обещаю, никуда сворачивать не будут. Можно провести аналогию с Великим Кнутом – он в самом начале предупреждает, что хотя книга о программировании, в первой главе программирования будет очень мало – так надо. Вот и у меня, то же самое, только наоборот – хотя в заголовке обещана математика, сначала будет программирование. В свою пользу должен заметить, что у Кнута первая глава под четыреста страниц, а у меня заметно меньше.

- *Ведь я же лучше собаки?* © Малыш и Карлсон

Глава будет немного аморфная, без того, чтобы сначала изложить теорию, потом объяснить, куда эта теория применяется, потом что-то запрограммировать для примера и так далее. Обещаю, в следующих главах всё будет структурировано чётко.

Сначала поговорим о системах счисления, потом о представлении чисел в памяти и на бумаге, а потом о представлении и других типов данных. Да, повторюсь, это не дотягивает до математики, но где-то же надо это обсудить. Главное, это очень несложно, очень полезно и, почему-то, не все это знают.

Разумеется, о системах счисления всё знают все. По крайней мере, всё знают все программисты. В процессе углубления в тему я прочёл книгу

А.Савин «Математические миниатюры», М. «Детская литература», 1991

Название издательства как бы намекает, на кого эта книга ориентирована. В книге для малолетних школьников быстро и понятно рассказывается всё о системах счисления. Я приуныл, и решил, что писать об этом не стоит. Впрочем, в книге так же легко и просто рассказывается всё о проективной

плоскости и о теоремах Дезарга и Паппа. Вы о таких людях раньше слышали? Я таки нет.

После чего я заглянул в Википедию и понял, что я о системах счисления знаю одну десятую, не больше. Интересно заметить, что чем проще тема, тем больше нового и интересного узнаёшь по ней из Википедии. Обратное тоже верно – о *литологическом расчленении* я узнал мало, точнее, ровно ничего, что в русской Википедии, что - в английской. На всякий случай – *литологическое расчленение* не имеет никакого отношения к лондонскому Джеку-Потрошителю. Так что я решил, что смело могу пересказать известные мне десять процентов, которые, конечно *там* (в мире Википедии) есть, и, главное, добавить к ним свои комментарии, которых *там, конечно,* нет.

Теория номер раз

Вкратце напомню, какие вообще в математике бывают числа, и что им соответствует в программировании, тоже вкратце, подробнее потом, по мере надобности.

Натуральные – 0,1,2.3… В программировании это целое беззнаковое. Конкретно в Delphi это word и тому подобное.

Целые – то же, что и натуральные, плюс отрицательные. В программировании вообще и в Delphi в частности, это то, что объявляют, не задумываясь – integer. Если объявляют, задумываясь, то всплывают ещё и shortint, smallint, int64 и странная штуковина longint. На любителя ещё есть cardinal.

Далее в математике нас ожидают рациональные, действительные и трансцендентные числа. Человечеству потребовались века, чтобы сформулировать эти категории и осмыслить, чем они отличаются друг от друга. Программирование на все эти тонкости плюёт и вводит понятие числа с плавающей точкой. На самом деле, число с плавающей точкой понятие несколько более широкое, но какая разница?

Что такое вообще система счисления? У нас есть число, натуральное конечно, хотя бы для начала, которое мы обозначим N. Пусть есть набор закорючек, которые мы называем цифрами. Количество закорючек пусть

будет A, где A называется основанием системы счисления. Самая первая загогулина всегда обозначает ровно ноль, а самая последняя – A-1. Напоминаю очевидный факт – загогулины для значения A нет вообще. В результате, наше число N представляется вот в таком виде:

$$x = \sum_{k=0}^{n-1} a_k b^k$$

Для десятичной системы эту формулу даже и объяснять дальше трудно, настолько всё очевидно. Тем не менее, далее пример:

128 (натуральное число) = закорючка 1 * 10^2 + закорючка 2 * 10^1 + закорючка 8 * 10^0

А теперь конкретно про конкретные системы счисления. Чем они принципиально отличаются друг от друга? Принципиально - ничем, кроме, разве что, системы счисления по основанию один, да и то не факт. А самый важный вопрос - какие системы счисления мы встречаем в программировании?

Двоичная. Это атавизм. Атавизм, если вас в школе не учили анатомии, а вас, наверняка, и астрономии не учили, это тот печальный факт, что у человека есть маленький хвост, только его снаружи не видно, а надо щупать физически. Проверьте на себе или на подруге, он там есть. Достался от обезьяны. Вот и с двоичной кодировкой примерно так же. Считается неотъемлемой частью любого компьютера, ни в одном компьютере в упор не видна, но иногда всплывает в каких-то древних алгоритмах. Имеет только два символа – 0 и 1. Выглядит число вот так – 1111000, это 240 десятичное. Как записать в Delphi? Да собственно, никак. Далее из какого-то форума, от безымянного автора. Взгляд, конечно, варварский, но верный.

- Что ж Борланд так лоханулся?
Почему лоханулся? Ни один нормальный программист не нуждается в двоичной записи, если есть шестнадцатеричная. А что касается студентов - так Delphi не для них предназначена :

Восьмеричные, мои любимые числа. Сейчас их уже не осталось. Жили они на платформе PDP. Цифры в диапазоне от 0 до 7. 1111000 восьмеричное –

это 2396160 десятичное. Имеют исключительно археологический интерес. В Delphi отсутствуют.

Шестнадцатеричная. Жива и активно применяется. Если вам понадобится посмотреть содержимое файла или области памяти, то вы получите их именно в таком виде. Просто потому, что никакого другого вида нет. Восьмеричный не подходит, двоичный – то же самое, только хуже. Десятичный – числа во внутренней десятичной кодировке умерли очень давно, в языках программирования, имею в виду. Учите шестнадцатеричный. И китайский. В Delphi почему-то записываются с долларом впереди, вот так:

```
N:=$FF  // N = 255
```

Всё это для программирования традиционно имеет какое-то религиозное значение, особенно двоичная система счисления. В книгах, в старых – во всех, а в современных только очень популярных, обязательно пишут, что компьютеры (они же ЭВМ) работают и думают в двоичной системе счисления.

Да какая нам разница, как оно там внутри на самом деле устроено! Есть оперативная память и её физическое отображение в виде файлов и прочего. Есть потоки данных, в широком смысле этого слова, движущиеся от точки выхода до точки входа. Всё это адресуется в байтах. Один байт – восемь бит, а пять старушек – рупь. Но переменные, которые мы объявляем программе, если они целого, в программном смысле этого слова, типа, в точности соответствуют математическому понятию целого числа. Реже натурального – в программировании этому соответствует целое беззнаковое, повторюсь, мало кому нужное.

// Воспоминание
В языке PL/I на котором я программировал порядочное количество лет – сначала в Университете, а затем на первой работе – были реально десятичные числа. В один байт помещалось две десятичные цифры, что было, разумеется, несколько расточительно. При этом числа эти были как целыми, так и дробными, причём с задаваемым количеством знаков.

Напоминаю, что PL/I расшифровывается как Programming Language Number One.

// конец Воспоминания

// Другое Воспоминание
Существовала единственная в мире машина, работающая в троичной системе – советская *Сетунь*. Я всегда считал это какой-то странностью и глупостью, а недавно всё-таки соизволил ознакомиться с предметом. Никакая это не глупость. Разумное, хотя, конечно, экзотическое решение, дающее много преимуществ для разработчика. Впрочем, дальше университетов эта штуковина не пошла. У нас, к сожалению, такой машины не было.
// конец Другого Воспоминания

Пока мы работаем с переменными как с математическими числами, нас совершенно не интересует, в какой системе счисления они записаны. Просто потому, что в математике никаких систем счисления нет. А вот когда мы захотим посмотреть, что же там внутри – в памяти, а чаще в файле, нам придётся вспомнить и о байтах, и о системах счисления.

Суммарно – в программировании используется двоичная система, но это никому не важно и не нужно. Реально раньше была восьмеричная система – на PDP, но теперь PDP нет и про восьмеричную систему можно забыть. Осталась только шестнадцатеричная, которая в глубине души, разумеется двоичная. Я не очень сложно излагаю? Вы ведь всё понимаете?

Простой пример

У нас есть простая программа. Программа сама производит данные, или, что сложнее, программа не сама производит данные, а получает их откуда-то. Возможно, от другой программы, возможно, от какой-то железяки, а, возможно, непосредственно от зелёных человечков с планеты Нибиру. Мы трактористы, нам – всё равно.

Данные эти сохраняются в файл. Если что-то записывается, то, как опыт учит нас, только для того, чтобы позже быть прочитанным. Некто читает наш файл, чтобы эти данные в каком-то виде вывести. Некто выводит на экран вместо красивого синуса – который предположительно записали мы – какие-то царапины от когтей Фредди Крюгера. Возникают вечные русские вопросы – *Кто виноват?* и *Что делать?*

Разумеется, если по счастливой случайности автором обеих программ – и пишущей и читающей – является один и тот же трудоголик, то проблема удачным образом закольцовывается и самоликвидируется. Но чаще эти две программы пишут два разных человека. которые немедленно начинают сваливать вину друг на друга. Ничего личного, только программирование.

Одно из двух. Или данные в файл записаны неправильно и виноват первый. Или данные неправильно читаются вторым и виноват, соответственно, он. На самом деле, всё, как минимум, немного сложнее. Кроме первого, который пишет, и второго, который читает, есть ещё и нулевой, тот который присылает первому данные. Его может и не быть, но если программа реальная, то он есть. Возможно, кривые цифры приходят от него.

Если эти данные непосредственно пишутся в файл, то отследить ошибки легко. Если первый перед записью хоть как-то обрабатывает эти данные, то возможно проблема кроется на этапе обработки. В этом случае необходимо отследить данные, непосредственно приходящие от источника, в сыром, необработанном виде. Возможно, для этого их тоже придётся сохранить в рабочий, отладочный файл, сразу после прихода.

Но мы рассмотрим простейший случай. Первому программисту данные приходят в заведомо правильном виде, просто потому, что он сам их генерирует. Пусть его программа рассчитывает факториалы и пишет в файл числа парами – само число и его факториал. Задача, конечно искусственная, но только в части никому не нужного факториала, а в остальном вполне жизненная.

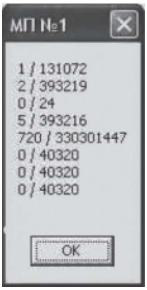
Предположим, что мы отвечаем за эту, записывающую файл, программу. Закодировали мы её вот так:

```
type
   TFactRec = record
      arg                    : smallint;
      fact                   : integer;
   end;
var
   F                         : file;
   rec                       : TFactRec;
   numOf                     : integer;
   i                         : integer;
```

```
{.....................................................}
function Factorial(     n : integer) : integer;
   var
      k                          : integer;
begin
     result:=0;
     if n >= 1 then begin
        result:=1;
        for k:=2 to n do
           result:=result * k;
     end;
end;
{.....................................................}
begin
     ChDir(ExtractFilePath(ParamStr(0)));

     AssignFile( F, 'fact.fact');
     ReWrite( F, 1);

     numOf:=8;
     BlockWrite( F, numOf, 4);

     for i:=1 to numOf do begin
        rec.arg:=i;
        rec.fact:=Factorial(i);
        BlockWrite( F, rec, SizeOf(rec));
     end;

     CloseFile(F);
end;
```

Комментарии. Факториал *принципиально* не запрограммирован рекурсивно. Жутковатая строчка после основного **begin** делает каталог, из которого запущена программа, текущим. То есть, именно туда и будет записан наш файл. Оформите эту строку в виде процедуры, не пожалеете, пригодится ещё много раз.

Итак, мы записали файл. Теперь сотрудник, который нас раздражает - *праативный, ухаади!* - его читает и выводит наши факториалы на экран, вот так, текст в заголовке *МП №1* расшифровывается как *Математическая Программа №1*:



После чего так называемый коллега выкатывает нам не бочку варенья, а бочку претензий. Мы уверены, что у нас всё правильно, он уверен, что правильно всё у него, а у нас наоборот. Каждый предлагает другому посмотреть в его код и убедиться, насколько там всё гениально и даже правильно. Но кто же будет смотреть в чужую программу, тут бы в своей разобраться. В конце концов решают сойтись на относительно нейтральной территории и заглянуть в файл. Что же там записалось? Кстати, правильный программист немедленно заметит, что количество факториалов совпадает с исходным, их ровно восемь.

Если вы очень серьёзный программист, для просмотра файла можно использовать какой-нибудь серьёзный шестнадцатеричный редактор. Если вы программист вроде меня, то можно применить FAR, или, того хуже, Total Commander. Вне всякой зависимости от применённого спецсредства, увидим мы вот такое:

```
00000000: 08 00 00 00 01 00 01 00 | 00 00 02 00 02 00 00 00
00000010: 03 00 06 00 00 00 04 00 | 18 00 00 00 05 00 78 00
00000020: 00 00 06 00 D0 02 00 00 | 07 00 B0 13 00 00 08 00
00000030: 80 9D 00 00
```

Эта штука традиционно называется *дамп*, от английского *dump*, помойка. Слева для удобства выведены смещения от начала файла, шестнадцатеричные, разумеется. Справа обычно присутствует ещё одно поле, где коды отображаются в виде соответствующих им символов. Обычно смысл это имеет только для символьных данных, поэтому я это поле откусил, чтобы не отвлекало.

Теперь совсем немного математики, точнее, совсем немного простой, честной арифметики. Как понимать стоящее слева в последней строке

смещение 30? Хотя система счисления не указана, число это по основанию шестнадцать. Понимать его надо в соответствии с приведённой ранее формулой. $30_{16} = 3*16^1 + 0*16^0 = 48$. Здесь числа без указания системы счисления уже десятичные. Если бы вы захотели сохранить в файл больше значений, то где-то дальше встретили бы смещение 120, например. $120_{16} = 1*16^2 + 4*16^1 + 0*16^0 = 320$. Я так подробно объясняю такие тривиальные вещи потому, что дальше будет хуже. Наши 40 и 120 – это нормальные шестнадцатеричные числа. Впереди нас ждут ненормальные.

Почему 120_{16} число нормальное? Потому что слева находятся самые значимые/весомые цифры. Чем правее, тем меньше их вес, всё как в жизни, всё как у традиционных десятичных чисел. Но эти смещения, вещь сугубо вспомогательная, теперь перейдем к основной части.

Что мы пишем в файл в самом начале – правильно, количество записей, оно равно восьми. В дампе видим $08_{16} = 0*16^1 + 8*16^0 = 8$. Всё сходится. Немного дальше мы видим последовательность байт `04 00 18 00 00 00`. Четвёрка означает, что далее последует факториал от этого числа. Напоминаю, 4! = 24. Проверяем. $18_{16} = 1* 16^1 + 8*16^0 = 24$. Опять всё хорошо. Немного смущает тот факт, что все числа мы пишем как четыре байта, а нужные значения оказались в самом первом, слева.

Проверим на чём-нибудь сложнее, на факториале восьми, он как раз в конце файла. 8! = 40320. В файле у нас `80 9D 00 00`. Можете сами трудолюбиво перевести это в десятичное число, если результат не понравится, повторите через калькулятор Windows (инженерный режим). Вам всё равно не понравится. Почему? Ведь всё начиналось так хорошо.

Потому что мы имеем здесь непростую встречу суровой физической правды и грациозной математической условности. В математике, в арифметике, или кто там вообще отвечает за позиционные системы счисления, *условились*, что старшие цифры находятся слева, а младшие – справа. В оперативной памяти младшие байты – слева, старшие справа. Разумеется, в нашем контексте слова *слева* и *справа* синонимичны словосочетаниям *в начале* и *в конце*. Это потому, что мы пишем слева направо. У евреев и прочих семитов Ближнего Востока всё наоборот. Дейкстра вспоминает, что его студент из Сирии всегда начинал перебор массива с последнего, правого элемента. Как-то я три дня сражался с настройками своего собственного монитора. Всех настроек было четыре

кнопки – реальных, не виртуальных. Я потерпел позорное поражение и был вынужден прочесть руководство. Оказалось, настройку надо всегда начинать с крайней *правой* кнопки, а я, понятное дело, сначала давил на крайнюю *левую*.

При этом, каждый байт, имеющий, напоминаю, ровно 256 значений, кодируется двумя цифрами по основанию 16. Эти цифры пишутся, как и положено, от старшего (левого) разряда до младшего (правого). Какое всё это имеет практическое значение, или, как нам правильно прочитать дамп?

Общее правило – если в дампе имеем последовательность байт, и эти байты вместе представляют собой четырёхбайтовое целое, конкретно `B1 B2 B3 B4`, то результатом будет

$B4*256^3 + B3*256^2 + B2*256^1 + B1*256^0$

После чего значения каждого байта рассчитываются традиционным арифметическим способом, как мы их и рассчитывали раньше и подставляются в формулу. Исходная последовательность `80 9D 00 00`. Меняем порядок на `00 00 9D 80`. Первые два байта нам неинтересны, далее имеем:

$256^1 * (9_{16}*16^1 + D_{16}*16^0) + 256^0 * (8_{16}*16^1 + 0_{16}*16^0)$ = то, что надо.

Если вам это всё понятно и очевидно, то я рад за вас, но большинство почему-то смущается и путается. Дальше в этой главе математической теории не будет, только программистская практика.

Как бы это всё упростить?

Однако, мы отвлеклись от нашей увлекательной задачи – отыскания виновного. В нашей части явной ошибки нет – файл мы пишем правильно.

Теперь пробежимся по нескольким смежным практическим вопросам. Целые бывают не только четырёхбайтовые (integer). Также существуют, в терминологии Delphi, shortint (один байт), smallint (два байта), int64 (восемь байт). Smallint мы тоже используем для хранения аргумента функции, видимо из жадности – в целях экономии двух байтов. Внутри у этих типов всё то же самое. Кроме того, есть ещё byte, word, и longword.

Это беззнаковые целые, что как бы намекает – ранее перечисленные типы – знаковые. Занудства ради, есть ещё longint и cardinal, но то, чем они отличаются от integer и longword соответственно никакого отношения к арифметике на имеет, а является вопросом сугубо политическим. Лучше разберёмся со знаковыми и беззнаковыми – как их читать из дампа. Это вопрос не политический, а всего лишь арифметический.

Сначала присмотримся в чём отличие этих переменных с точки зрения программиста, на примере однобайтовых типов byte (беззнаковый) и shortint (знаковый). Переменная типа byte вмещает в себя числа в диапазоне от 0 до 255, shortint переваривает от -127 до +128. Интуитивно понятно, надо же где-то хранить информацию о знаке числа, а она у нормальных людей занимает ровно один бит.

Кстати, если вы думаете, что в других языках программирования лучше, то вы ошибаетесь. В других языках хуже, по крайней мере, во всех других практически применяемых языках.

Хотя переменные без знака применяются гораздо реже, с ними всё просто. С ними всё именно так, как я раньше и объяснял. Это со знаковыми хуже.

- *У тебя всё хорошо?*
- *У меня всё хорошо. (Вешает трубку) — У тебя плохо.*
© кино «Брат-2»

Проведём опыт. Объявим переменные, присвоим им значения, запишем в файл и почитаем дамп.

```
var
   sh                    : shortint;
   by                    : byte;

   sh:=127;
   by:=127;

   BlockWrite( F, sh, 1);
   BlockWrite( F, by, 1);
```

Поскольку число 127 попадает в диапазон обоих типов, результат предсказуемо будет одинаковым:

```
00000000: 7F 7F
```

Теперь запишем в файл числа 128, 255, -1, -128. Для этого один из операторов присваивания обязательно придётся закомментировать. Нельзя переменной типа byte присвоить -1, а переменной типа shortint +128. В результате получим вот такую табличку:

128	X	80
255	X	FF
-1	FF	X
-128	80	X

Таблица демонстрирует подозрительную симметричность, которая объясняется использованием для кодирования отрицательных чисел так называемого *дополнительного кода*. Как записать в этом коде отрицательное число, например -1 (минус один)?

1. Взять число по модулю. Получаем 1.
2. Записать в двоичной системе. Получаем 0000 0001
3. Инвертировать. Получаем 1111 1110
4. Прибавить 1. Получаем 1111 1111

В результате имеем FF, которое, собственно, мы и так имеем. Но всё это нам совсем не интересно, всё это делает за нас компьютер. Нам интересно из дампа прочитать и понять отрицательно число. Легко. Всё то же самое, но в обратном порядке. В дампе 80.

1. Записать в двоичной системе. 1000 0000
2. Вычесть единицу. 0111 1111
3. Инвертировать. 1000 0000
4. Вернуть в десятичную. 128
5. Поменять знак. -128

Подумайте, можно ли было обойтись без перевода в двоичную систему счисления и остаться в родной шестнадцатеричной.

// Воспоминание, как я играл в игрушки

Не то, чтобы я сейчас совсем уже не играл в игрушки. Но – *нет уже пороха в пороховницах и ягод в ягодицах* © Народное. Помните ли вы Цивилизацию? Просто Цивилизацию, без номера. С тем же успехом можно спросить, помните ли вы компьютеры на базе 286-го процессора и

DOS 5.0. Или это был DOS 3.3? На этом богатстве она, Цивилизация, и работала.

Теперь цитата к месту из Википедии:
Неправильное использование беззнаковых целых может приводить к неочевидным ошибкам из-за возникающего переполнения.
Мысль столь же бесспорная, сколь и очевидная. Не могу утверждать с уверенностью, что у них там было внутри, знаковые или беззнаковые, и с чем именно разработчики не справились, но симптомы были такие.

Последняя Цивилизация, в которую я играл, была третья. Там, как и в первой, была возможность подкупить вражеский город засылкой шпиона с деньгами. *Осёл, гружённый золотом, возьмёт любую крепость* © какой-то Македонский, то ли Филипп, то ли Александр. В третьей Цивилизации у меня это ни разу не получилось – денег банально не хватало.

В первой был шанс. Чем больше был город, тем больше была его стоимость. Шпион время от времени подходил к городу и спрашивал, за сколько они готовы продаться. Как только цена переваливала за 32787 случалось маленькое чудо. Цена города становилась отрицательной. То есть, можно было получить и город даром и ещё денег в придачу. Я понимаю, что это нечестно, но ведь все так делали.
// конец Воспоминания, как я играл в игрушки

// Угрызения совести
Да, я играю в игрушки до сих пор. Но, как сказал по аналогичному вопросу – курил ли он марихуану – президент Клинтон, *курил, но не затягивался.* И я так же. Играю, но только в правильные игрушки. Civilization III, Panzer General, Heroes of MM 3.5 (In WOG wee trust, вы меня конечно понимаете), Fallout II и Morrowind. Иногда, впадая в маразм, дохожу до Railroad Tycoon Deluxe.

Впрочем, обо всём этом, какие игрушки правильные, а какие нет, и как правильные игрушки программировать, я пишу увлекательную книгу в трех томах. Как только я её допишу, *они* её немедленно напечатают.
// Угрызения совести закончились

Однако! Где же ошибка?

Мы начали с поиска причины нечитаемости нашего файла, но куда-то уклонились. Наш файл записан безусловно правильно. Поскольку чудес не бывает, из этого следует, что он неправильно читается. Почему он неправильно читается? Причин я могу придумать множество, это нетрудно, придумывать причины, по которым программа не работает. Это моя работа в конце концов. Возможно запись объявлена не так, как у нас, например поля немного другие:

```
TFactRec = record
   arg                  : integer;
   fact                 : smallint;
end;
```

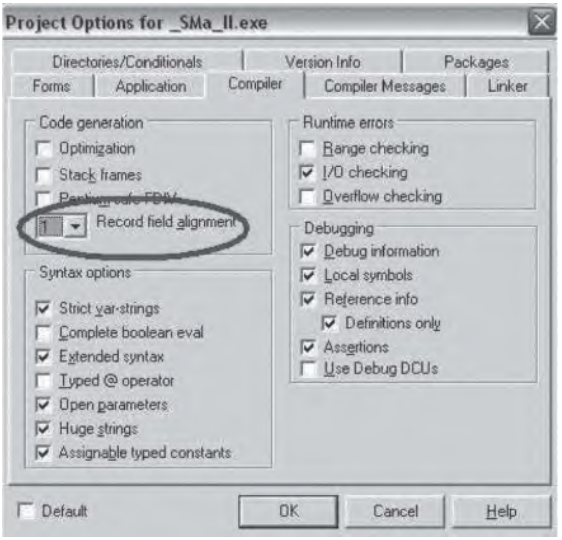
Этого достаточно, чтобы сокрушить программу. Возможно, файл открывается чуть по другому: `ReWrite( F, SizeOf(rec));` Тоже неплохо. Особо альтернативно одарённые обязательно спросят у автора первой программы, каким именно способом он пишет в физический файл, и, узнав, что через нетипизированный файл, принципиально будут читать через файловый поток и, разумеется, неправильно. Такие сотрудники есть, и в ассортименте. Однако коллега предъявляет текст и там мы видим, в сокращении, вот такое:

```
type
   TFactRec = record
      arg                  : smallint;
      fact                 : integer;
   end;

  AssignFile( F, 'fact');
  ReSet( F, 1);

  BlockRead( F, numOf, 4);

  stroka:='';
  for i:=1 to numOf do begin
     BlockRead( F, rec, SizeOf(rec));
     stroka:=stroka + IntToStr(rec.arg) + ' / ' +
                      IntToStr(rec.fact) + #13#10;
  end;
```

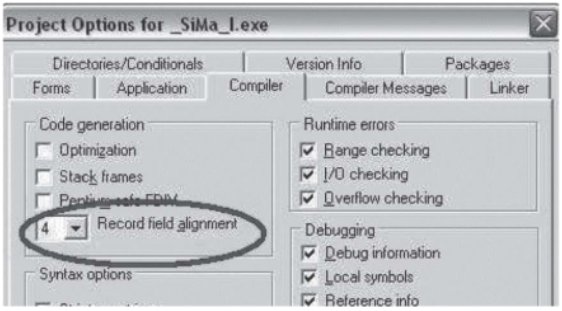
Наблюдаем полную симметрию чтения с записью и отсутствие какого либо криминала. Так где же собака порылась? Возможно, вы сочтёте пример немного искусственным, а я возражу. Что ситуация абсолютно

реальная, я видел почти всё, а это я точно видел. К текстам программ (исходному коду) претензий нет. Они одинаковые. К исполняемому коду (то, что с расширением .EXE) претензии есть. Они разные. Почему?

Идём на страницу Project\Options\Compiler. У первого автора, записывающего, она выглядит так:



А у второго, читающего, вот так:



Что это, собственно, означает? Напомню, как объявлена запись, которую мы с вами пишем в файл:

```
type
   TFactRec = record
      arg                    : smallint;
      fact                   : integer;
   end;
```

Обратите внимание, первое поле имеет размер 2 байта, второе – 4 байта. У первого программиста, который пишет, так всё и будет записано: /2 байта arg/4 байта fact/. Казалось бы, а как иначе? Однако, можно и иначе. Волшебное число 4 в настройках читающего программиста означает, что он предполагает все поля записи выровненными по адресам с кратностью 4. То есть, первый программист имеет для поля arg адрес – условно – 00000000, для поля fact – адрес 00000002, а если бы в записи были ещё поля, то третье поле имело бы адрес 00000006. Для второго программиста эти адреса будут, соответственно, 00000000, 00000004, 00000008. А что будет в двух неизвестно откуда взявшихся байтах? Нули, естественно. В нашем конкретном случае - /2 байта arg/2 нулевых байта/4 байта fact/.

Что было бы, если бы первый программист действовал в соответствии с ожиданиями второго, и установил бы те же настройки? Что оказалось бы в файле? Правильно, вот это:

```
00000000: 08 00 00 00 01 00 00 00 | 01 00 00 00 02 00 00 00
00000010: 02 00 00 00 03 00 00 00 | 06 00 00 00 04 00 00 00
00000020: 18 00 00 00 05 00 00 00 | 78 00 00 00 06 00 00 00
00000030: D0 02 00 00 07 00 00 00 | B0 13 00 00 08 00 00 00
00000040: 80 9D 00 00
```

И именно вот это второй программист и пытается прочитать, с немного предсказуемым результатом. Обрате внимание, пока мы не пишем данные в файл, никакого значения эти настройки не имеют, мы обращаемся к полям записи по имени, а транслятор сам вычисляет смещения. Хотя…

// воспоминания о коллеге

Коллега принципиально обращался к полям записи по смещениям относительно начала записи. Мотивировал это он тем, что пишет на C++. Специально для него и других мы написали утилиту, которая конвертировала объявления констант, типов и переменных из Паскаля в C++, но коллега был упорен и на провокации не поддавался. Более того, он три раза ходил к руководству и, ссылаясь на невыносимые условия работы, требовал повысить заработную плату – а иначе уволится. Два раза сработало. Сейчас он трудится в другой, суровой конторе, с жёсткой

дисциплиной. Только теперь он не программист, а, как это по-русски – системный аналитик, то есть пишет спецификации и задания для рядовых кодеров. Ему очень нравится.

В чём мораль? Морали нет.
// конец Воспоминания о коллеге

Пример может показаться громоздким и искусственным. Громоздким – да, искусственным – ни разу, совершенно обычная, штатная ситуация. Если вы надеетесь, что в C++ лучше, то вы наивная чукотская девушка – там абсолютна та же фигня.

Как это лечится? Можно лечить дисциплинарными взысканиями – порка и карцер – с тем, чтобы все-все-все выставили в настройках одни и те же циферки. Но лучше слегка изменить объявление записи, вот так:

```
type
   TFactRec = packed record
      arg                    : smallint;
      fact                   : integer;
   end;
```

Волшебное слово **packed** равнозначно полному отсутствию выравнивания, то есть цифре 1 в настройках. Всё пишется как есть, без затей. Кстати, в C++ всё то же самое, только слово другое, найдите сами.

Вам такое программирование не нравится? Извините, миром правят Рептилоиды. И не только миром. И не только они. Тему Рептилоидов я рассмотрю в главе о теории вероятностей, в разделе о теориях заговора.

Как переводить из одной системы в другую

Как видно из предыдущих разделов. переводить из одной системы счисления в другую приходится. Как показывает мой опыт, переводить приходится часто. А как?

Можно запустить казённый калькулятор из Windows. Легко, доступно, хотя есть недостатки. Только четыре системы счисления, по основаниям 2,8,10,16. На практике больше и не надо. Второй недостаток серьёзнее. Для разборки дампа это подойдёт, хотя, если вы взялись разбирать дамп, шестнадцатеричные числа давно пора уже читать в уме. Другой недостаток – иногда возникает необходимость перевода в другую систему

счисления программным образом, то есть, переводите не вы в уме, а ваша программа. Тут калькулятор не поможет.

Но пока немного о переводе без компьютера. Как переводить из любой системы в нашу, родную, десятичную, должно быть ясно из ранее приведённой формулы. Как перевести из десятичной в любую другую, проиллюстрируем на конкретном частном примере. Переведём в семеричную систему, просто потому. что о такой системе я никогда и не слышал. Это будет интересно и неожиданно и для меня и для вас. За исходное, в десятеричной системе, возьмём любимое число 255_{10}.

255 делим нацело на 7. Результат 36, остаток 3.
36 делим нацело на 7. Результат 5, остаток 1.
5 делим нацело на 7. Результат 0, остаток 5.

Поскольку результат 0, на этом всё заканчиваем и собираем результат, снизу вверх.

$255_{10} = 513_7$

Проверка. $513_7 = 5*7^2 + 1*7^1 + 3*7^0 = 245 + 7 + 4 = 255$

А теперь несколько относительно простых и честных способов для частных случаев.

Из двоичной в восьмеричную и обратно. Например, 11000011_2 перевести в восьмеричную систему счисления. Разбиваем цифры на тройки, начиная *справа* – 11/000/011. Каждую тройку, хоть слева, хоть справа, переводим в десятичное число. Всё. $11000011_2 = 303_8$. Обратное преобразование для 77_8. Каждую цифру преобразуем в двоичную тройку. $77_8 = 111111_2$.

Преобразование из двоичной в шестнадцатеричную систему немного сложнее. Надо запомнить дополнительно:

$A_{16} = 1010_2$
$B_{16} = 1011_2$
$C_{16} = 1100_2$
$D_{16} = 1101_2$
$E_{16} = 1110_2$

$F_{16} = 1111_2$

Дальше всё то же, что и с восьмеричной системой.

Перевод из восьмеричной в шестнадцатеричную и обратно несколько сложнее. Как ни печально, самый простой способ это сначала перевести в двоичную систему, а потом как обычно.

Но мы будучи всё-таки программистами хотели бы знать, как переводить из системы в систему программно. Ещё раз напоминаю (или я этого ещё не говорил?) - внутри компьютера никаких систем счисления нет, по крайне мере в более-менее современных языках программирования. Раньше – да, были. Поэтому говоря о переводе из одной системы в другую мы имеем в виду перевод из *символьного* представления числа в одной системе в числовое (без системы) или наоборот. Или из одной системы в символьном представлении в другое символьное представление. Я долго над этим думал, прежде чем сформулировать.

Для лучшего понимания этого смутного утверждения, посмотрим какие функции для перевода есть в Delphi. Там нет почти ничего, кроме двух – HexToBin и BinToHex. Параметры и результаты их строковые. Точнее, эти две функции там *гипотетически* есть. Если вам непонятно слово *гипотетически*, посмотрите в гугле, только не самую первую ссылку, а чуть пониже. От нас требуют передавать входной и выходной параметры (забудьте, что это функция) как PChar – мы терпеливые. От нас требуют заранее выделить память под результат. Мало того – от нас требуют писать входное число маленькими буковками. Не гордое ‘FF’, а позорное ‘ff’.

Тварь ли я дрожащая, или право имею? © потомственный дворянин Родя Раскольников

Вот и мы тоже, право имеем. Имеем право написать функцию перевода из любой системы счисления в любую. Почти. Можно и в любую, но фантазия наша ограничена набором символов, имеющихся в нашем распоряжении. Что может шестнадцатеричная система? Правильно шестнадцать символов, десять десятичных плюс шесть из английского алфавита – A.B,C,D,E,F. Если мы суммируем наши исконные десять и не наши, английские, двадцать шесть, то получим 36, что некрасиво. Красиво будет 32 – десять цифр плюс английские буквы в диапазоне A..V.

Напишем функцию перевода числе в пределах систем счисления по основанию не больше 32. Полезно, несложно и практично. Для краткости, проверку на корректность входной строки выкинем. Разумеется, если вы захотите использовать эту функцию в реальной, печальной, унылой жизни – проверять придётся.

Прежде чем кодировать самую простую из простых программ, надо понять, что мы от неё хотим – иначе говоря, сформулировать Техническое Задание (ТЗ), Если мы пишем функцию, мы должны заранее знать, что поступает ей на вход и что мы имеем на выходе. Если мы очень аккуратные и правильные программисты, то мы заранее напишем набор тестов для нашей функции. И не только входные данные, но и ожидаемый от них результат. Но это так, к слову.

На выходе будет строка, представляющая наше число в новой системе, это понятно. На входе тоже строка, в исходной системе счисления. И, мне кажется, ещё два целых числа – основание входной системы счисления и выходной. Если вы можете придумать что-то более изящное, тот придумайте.

```
function AnyToAny(     inStr           : string;
                       inBase, outBase : integer) : string;
```

— Если вас собьют, вы обязаны сжечь это письмо еще до того, как успеете отстегнуть лямки парашюта.
— Я не смогу сжечь письмо до того, как отстегну лямки парашюта, оттого что меня будет тащить по земле. Но первое, что я сделаю, отстегнув лямки, так это сожгу письмо.
© Семнадцать мгновений весны

Первое, что вы должны сделать, задумав написать функцию – написать для неё тестовую программу, причём ещё до того, как успеете написать саму функцию. Ну ладно, вы не сможете написать тестовую программу до того, как определите интерфейс функции. Но как только определите – так сразу. Тестовая программа очень простая, но достаточно наглядная. Главное, что мы видим и входные и выходные данные.

Первый тест будет уже проработанный руками, с семеричной системой. Объявления переменных пропускаем.

```
inStr:='255';  // 10
inBase:=10;  outBase:=7;
outStr:=AnyToAny( inStr, inBase, outBase);
ShowMessage( IntToStr(inBase) + ' --> ' + IntToStr(outBase) + #13#10
             + inStr + ' ==> ' + outStr);
```

Сама функция. Само собой разумеется, что сначала надо проверить входные данные на корректность. Если на вход можно подать что-то кривое, то именно кривое и подадут. Проверку пропускаем, напишите сами. Основания систем счисления должны лежать в диапазоне от 1 до 32, и входная строка должна содержать только допустимые символы. А вот и они:

```
const
   numOf = 32;
   syms  = '0123456789ABCDEFGHIJKLMNOPQRSTUV';
```

Объявления переменных тоже пропускаем, поверьте на слово, что все они целые. Теперь нам надо осмыслить сложный философский и методологический вопрос. Алгоритм перевода у нас всё тот же, с делением. Но делить числа в произвольной кодировке мы не умеем. Мы бы конечно смогли написать функцию, но зачем?

- *Вовочка, почему тебя вчера не было в школе?*
- *Мы с папой водили корову к быку.*
- *А что, папа сам бы не смог?*
- *Папа, конечно, смог бы, но бык лучше* © Старая шутка юмора

Это я к тому, что числа типа integer делятся как-то сами собой, без нашего участия. Поэтому мы должны привести сначала наше в произвольной системе счисления число к типу integer. Повторюсь, внутри в переменной языка программирования системы счисления нет, но присвоить ей можно только что-то десятичное или шестнадцатеричное. Это особенности нашего языка программирования, проверьте, как с этим обстоят дела в других. Следовательно, придётся сначала перевести наше число в произвольной системе счисления в десятичное (придумайте шестнадцать объяснений того, почему не в шестнадцатеричное).

Алгоритм перевода в десятичную систему счисления нам известен:

```
//  А здесь могла быть ваша проверка входных данных!!!
```

```
dec:=0;
for i:=1 to Length(inStr) do begin
   one:=Pos( inStr[i], syms) - 1;
   dec:=dec + Round( IntPower( inBase, Length(inStr)-i)) * one;
end;
```

А теперь легко и непринуждённо кодируем основную часть:

```
result:='';

rez:=dec;
repeat
   ost:=rez mod outBase;
   rez:=rez div outBase;
   result:=syms[ost+1] + result;
until rez = 0;
```

Комментарии. Нравится ли мне это код? Нет, не нравится мне этот код, потому что если поменять местами два первых оператора внутри цикла, код работает неправильно. Понятно, что нельзя менять местами строки программы, но эти два оператора, если не присматриваться, выглядят симметрично и невинно. Присмотритесь и обдумайте. И, разумеется, протестируйте функцию не менее, чем на десяти тестах. Меня-то хватило всего на четыре.

Простой пример в общем случае

В реальной жизни данные бывают не только целыми. Это, конечно, уже не совсем математика, но около математики и для программиста очень желательное знание. Как выглядит файл, в который записали не только целые, а *что-то ещё*? А что ещё бывает? Бывают числа с плавающей точкой. Они, как и целые, бывают разной длины, но, как и с целыми, принцип записи остаётся неизменным. Бывают символьные данные. С символьными данными просто зоопарк, в смысле их разнообразия.

Гуляют там животные невиданной красы © Слова не того, кого вы думали, на музыку не того, кого вы думали

// С высунутым от гордости языком
А у меня та самая пресловутая пластинка есть. И даже агрегат есть, на котором её играть.
// конец Высунутого языка

Так вот, о символьных данных. В Delphi есть просто символы – char. Есть символьные массивы. Есть короткие строки, доставшиеся в наследство от

Турбо Паскаля. И есть длинные строки, почти объекты и почти совместимые с C++. Эти типы совместимы между собой по хитрым невнятным правилам, но нам это неинтересно, потому что здесь и сейчас мы говорим о математике или почти о математике. Да и с числами с плавающей точкой не всё так однозначно, я дальше поясню. Но обо всём этом надо хоть что-то сказать, потому что в жизни пригодится. Всё остальные типы – экзотика, о которой и упомянем в следующем разделе.

Случаи, они разные бывают © Поручик Ржевский

Далее – пример объявления переменных, их записи в файл и как они там выглядят. Вот эти объявления и крошечная, но полезная процедура. По соотношению, как принято сейчас формулировать «цена/качество», процедура не имеет себе равных. Она записывает в файл четыре нулевых байта. В реальный файл, разумеется, такое писать на надо, но для наших обучающих целей это резко повышает наглядность и читаемость файла, разделяя его на легко различаемые сегменты.

```
   var
      F                          : file;
      fl                         : single;
      ch                         : char;
      aCh                        : array[1..16] of char;
      shStr                      : string[7];
      longStr                    : string;
{.....................................................}
procedure wZ;
   var
      fl                         : single;
begin
     fl:=0;  BlockWrite( F, fl, 4);
end;
{.....................................................}
```

Теперь присвоение переменным хоть каких-то значений и запись в файл. Короткие строки перед присвоением зачищаются нулевыми кодами – чтобы потом было лучше видно, сколько значащих символов в них хранится.

```
     fl:=0;  BlockWrite( F, fl, 4);
     fl:=1;  BlockWrite( F, fl, 4);
     wZ;  wZ;

     ch:='C';  BlockWrite( F, ch, 1);
     ch:='h';  BlockWrite( F, ch, 1);
```

```
ch:='a';  BlockWrite( F, ch, 1);
ch:='o';  BlockWrite( F, ch, 1);
wZ;  wZ;  wZ;

aCh:='Нікто мене не любить            ';
BlockWrite( F, aCh, SizeOf(aCh));

FillChar( shStr, SizeOf(shStr), #0);
shStr:='Hello';
BlockWrite( F, shStr, SizeOf(shStr));

FillChar( shStr, SizeOf(shStr), #0);
shStr:='Hello world!';
BlockWrite( F, shStr, SizeOf(shStr));
```

На немедленно возникающий правильный вопрос - а почему переменная longStr проигнорирована и не записана в файл, дадим немедленный правильный ответ.

- *Подсудимый, садитесь.*
- *Спасибо, я постою.*
- *Гражданин судья, а он не может сесть!* © Кавказская пленница

Мы не можем записать переменную типа **string**, потому что внутри она простой указатель. Простой указатель вовсе не означает, что указатель сам по себе очень простое понятие. Наш указатель указывает на совсем не простой агрегат данных, но для нас сейчас это неважно. Записать *это* нельзя. А если очень хочется – а хочется часто, то придётся перевести в **array of** char подходящей длины и его уже записывать. Ещё обратите внимание, что строку для того самого массива символов в исходном тексте программы пришлось добивать пробелами – иначе транслятор не пропускает.

Теперь поглядим, что у нас получилось на выходе. Поскольку мы записали порядочное количество символьных данных, будет уместно привести и правую колонку дампа, в который данные как раз и отображаются в символьном виде. Даже те данные, для которых такое отображение совсем не подходит и никакого смысла не несёт.

```
00000000: 00 00 00 00 00 00 80 3F | 00 00 00 00 00 00 00 00
00000010: 43 68 61 6F 00 00 00 00 | 00 00 00 00 00 00 00 00
00000020: CD 69 F5 F2 EE 20 EC E5 | ED E5 20 ED E5 20 EB FE
00000030: E1 E8 F2 FC 20 20 20 20 | 20 20 20 20 20 20 20 20
00000040: 06 48 65 6C 6C 6F 00 00 | 07 48 65 6C 6C 6F 20 77
```

Для лучшей читаемости, символьная колонка отдельно. Мысленно склейте в панораму. Вместо пробелов – знак подчёркивания, тоже для лучшей читаемости.

```
______??________
Chao____________
Hixто_мене_не_лю
бить____________
♣Hello__•Hello w
```

Интересное наблюдение - пробелы в символьной колонке не обязательно соответствуют настоящим пробелам. Они могут заменять также неотображаемые символы. Теперь по существу.

Первым в файл записано четырёхбайтовое с плавающей точкой, равное нулю. В дампе мы видим соответственно четыре нулевых байта. Это не может не радовать. К сожалению, далее нас ждёт немедленное разочарование. Вместо единицы мы видим непонятно что. Числа в плавающем формате кодируются достаточно хитрым способом. Если вы прочтёте, что вариантов кодировок ровно одна – IEEE 754, то это не совсем так. Совсем не так, точнее. Другие кодировки мало распространены, но зато присутствуют в таких важных отраслях науки и техники, что выучить их придётся, если вы туда попадёте. Но это я отвлёкся. Мораль в том, что чтение глазами плавающих чисел из файла доступно не всякому. Забудьте.

С символами интереснее и понятнее. Четыре подряд записанных переменных типа char именно так и выглядят – четыре символа подряд, что в шестнадцатеричном представлении, что в символьной колонке. Далее следует символьный массив. С ним тоже всё прекрасно. А короткая строка, то есть строка объявленная как **string**[n], где n константа, устроена сложнее. Для нашей строки n=7. Общая длина переменной равна восьми, потому что впереди находится байт, содержащий длину строки. Поскольку это всего лишь байт, ясно что длина короткой строки не может превышать 255. К этому байту можно, но не рекомендуется, обраться как shStr[0]. Когда в первый раз мы присваиваем нашей строке строковую константу длиной пять, последние два байта остаются, как и были нулевыми. Когда мы присваиваем что-то изначально более длинное, чем длина короткой строки, лишнее отбрасывается.

Настоящие, длинные строки (**string**) нельзя записать в файл просто так, только через определённые не очень хитрые манипуляции. И прочитать нельзя, без этих же манипуляций.

Всё остальное и выводы

О странном. Есть такой тип как boolean – булевский, логический. Если вдуматься, а я вдумался, то зачем он вообще нужен? Вот стоило только всерьёз задуматься и возник такой вопрос. Тем не менее, следующая глава, или одна из следующих глав, будет посвящена математической логике, так что без булевского типа никак не обойтись. Булевские типы бывают просто булевские – boolean, и с подвывертом – ByteBool, WordBool, LongBool. Про последние три немедленно забудьте, потому что даже я не знаю, зачем их придумали. Единственно правильный тип boolean интересует нас здесь только в отношение его вида в сохранённом файле. Далее фрагменты программного кода:

```
bool                      : boolean;
bool:=true;   BlockWrite( F, bool, SizeOf(bool));
bool:=false;  BlockWrite( F, bool, SizeOf(bool));
```

А вот результат:

```
00000000: 01 00                  |
```

Вывод очевиден – булевская переменная занимает ровно один байт (не бит!). True кодируется единицей, false – нулём.

Поскольку, если вы не забыли, далее будеет глава о математической логике, нельзя обойтись и без множеств, в Delphi есть такой очень экзотический тип - set.

О математическом смысле этих переменных, и о применении, весьма ограниченном, мы поговорим позже, а сейчас только о сохранении в файл и как это потом прочитать. Пусть у нас есть вот такое объявление и инициализация множества (официально инициализация множества почему-то называется вызовом его конструктора) :

```
const
   lowS  =   0;
   highS = 255;
type
   TSet255  = set of lowS..highS;
```

```
var
   a                          : TSet255;

a:=[0];
```

Далее мы традиционным способом сохраняем в файл наше множество:

```
BlockWrite( F, a, SizeOf(a));
```

Чтобы два раза не вставать, запишем ещё и множество [0,2,4]. Что получаем на выходе?

```
00000000: 01 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000010: 00 00 00 00 00 00 80 3F | 00 00 00 00 00 00 00 00
```

Первая строка для множества [0], вторая для множества [0,2,4]. Общий объём файла составляет 32 байта и это не случайно. 32*8=256, а это максимальная ёмкость множества в Delphi. Каждый элемент множества кодируется тупо и в лоб – или он есть, или его нет – или единица в соответствующем бите или ноль. Требует некоторых умственных усилий, но вряд ли вам придётся с этим встретится в реальном программистском мире.

И ещё – записать переменную типа **file** в файл нельзя. То есть, можно, но бесполезно. Точно так же, как абсолютно бессмысленно записывать в файл указатель или длинную строку. Обдумайте, можно ли записать в файл объект (экземпляр класса). И если да, то что это значит?

Какой он, этот Слонопотам?
Любит ли он поросят или нет?
И как он их любит?.. © Милн в пересказе Заходера

Как пишутся в файл агрегаты данных – массивы, записи? Записи пишутся и читаются без затей и без проблем, если вы не забудете объявить их как **packed record**. С массивами чуть интересней. С первого взгляда всё просто. Если у нас объявлен, проинициализирован и записан в файл массив вот таким образом

```
a         :  array[1..4] of integer;

a[1]:=1;  a[2]:=2;  a[3]:=3;  a[4]:=4;
BlockWrite( F, a, SizeOf(a));
```

то в файле мы увидим вот такое:

```
00000000: 01 00 00 00 02 00 00 00 | 03 00 00 00 04 00 00 00
```

Что, в общем-то, вполне ожидаемо – массив записывается в файл в точности как если бы вместо него существовали несколько, по длине массива, переменных того же типа. Всё становится более интригующим и непредсказуемым, если массив двумерный, не говоря уже о больших размерностях. Объясняю на примере.

```
a2       :  array[1..2,1..2] of integer;

a2[1,1]:=11;  a[1,2]:=12;  a[2,1]:=21;  a[2,2]:=22;
BlockWrite( F, a2, Sizeof(a2));
```

На выходе получим:

```
00000000: A1 00 00 00 A2 00 00 00 | 15 00 00 00 16 00 00 00
```

Это также не вызывает удивления, а зря. Говоря на программистском диалекте, массивы в Delphi пишутся по строкам. В C++ тоже вроде бы. Я спросил у Ясеня, тьфу, я спросили у сишника.

Ничего Петров не отвечает,
Только тихо ботами качает © Садистский стишок

Но это не всегда так. В Фортране массивы пишутся по столбцам, этот же массив, записанный в файл программой на Фортране выглядел бы так:

```
00000000: A1 00 00 00 15 00 00 00 | A2 00 00 00 16 00 00 00
```

Это касается не только тех, кто программирует на Фортране или читает записанные программой на Фортране массивы. Это касается и тех, кому придётся разбираться в фортрановской программе, и не думайте, что с вами этого просто не может случиться. Может. В Фортране, как и в более поздних языках, присутствует инициализация массивов при их объявлении или как констант. Так вот, записывается она по столбцам, что, разумеется, чисто визуально определить невозможно.

Это нельзя понять! Это надо запомнить! © из анекдота про грузинскую школу.

И не говорите, что я вас не предупреждал. Обдумайте. Выясните, как с этим в ЛИСПе. А кстати, как?

А что такое текстовый файл?

Сначала занудное уточнение, извините, если это для вас очевидно. Текстовый файл (в программировании) не есть файл, в котором содержится текст, то есть, файл который можно прочитать глазами с помощью простой или сложной программы. Файлы Microsoft Word, его же RTF, а также выползшие непонятно откуда fb2 и epub ни разу не являются текстовыми с точки зрения программиста. Если подходить к вопросу прагматически, текстовый файл – то, что можно открыть Блокнотом из Windows. Формальное определение – в нём содержатся только текстовые символы и символы форматирования. Со временем текстовые файлы окончательно деградировали и выродились и их определение значительно упростилось – текстовый файл есть то, в чём только буковки (и циферки) а также символы возврата каретки (не спрашивайте, что это) – 13 десятеричное, и перевода строки – 10.

А теперь поглядим, как это выглядит. Вам это не понадобится никогда? *Не плюй в колодец, милый мой* – А.С.Пушкин. Чтобы далеко не ходить за примером, применим того же Александра Сергеевича из того же стиха, откуда взято про колодец:

Пупок чернеет сквозь рубашку,
Наружу титька – милый вид!
Татьяна мнёт в руке бумажку,
Зане живот у ней болит.
Она затем поутру встала
При бледных месяца лучах
И на подтирку изорвала
Конечно *Невский альманах*.

//Филологический комментарий.
А.С.Пушкин, как знают все, написал роман в стихах *Евгений Онегин.* Журнал, а точнее альманах, с названием, сюрприз, *Невский Альманах,* напечатал иллюстрации к нему, на одной из которых изображена Т.Д.Ларина за сочинением письма Е.?.Онегину. К сожалению, вставить

картинку сюда я не могу. Соображения благопристойности, цензура, *а вдруг это будут читать дети?,* технологические требования издательства… В основном, конечно, технологические требования…

Кстати, были кем-то подкупленные филологи, утверждавшие, что Татьяне в романе *двенадцать* лет! Сам Пушкин в письме к Вяземском заметил мимоходом, что ей семнадцать. Филологи врут! Пушкин врёт! Они все врут! Ей за сорок. Ладно, тогда женщины старели рано, ей за тридцать пять. Посмотрите на картинку, вы найдёте, я уверен. Как теперь обязательно надо предупредить, 14+.
// конец Филологического комментария

Upd. Я всё-таки не удержался, и впендюрил аутентичные гравюры в приложение. Вас не раздражает слово *аутентичные*?

Как записать текст в файл? Тот способ, которым мы пользовались до этого (через F : **file**) не подойдет. Можно использовать либо специально предназначенную для этого разновидность файла, TextFile, либо класс TStringList. Физический результат в виде файла будет одинаковым, поэтому, не ища лёгких путей, сделаем *это* через файл.

```
   var
      TF                        : TextFile;
begin
     //  вы уже оформили это в виде процедуры?
     ChDir(ExtractFilePath(ParamStr(0)));

     AssignFile( TF, 'SimpleTextFile');
     ReWrite(TF);

     Writeln( TF, 'Пупок чернеет сквозь рубашку');
     Writeln( TF, 'Наружу сиська – милый вид!');
     Writeln( TF); Writeln( TF);
     Write(TF);    Write(TF);

     CloseFile(TF);
```

Ограничимся двумя первыми строками, величие Пушкина это не умалит (ударение на последнем слоге. Или на предпоследнем?). А зачем четыре последних оператора записи в файл? А чтобы посмотреть, что из этого получится. А получится вот что, символьная колонка отдельно:

```
00000000: CF F3 EF EE EA 20 F7 E5 | F0 ED E5 E5 F2 20 F1 EA
00000010: E2 EE E7 FC 20 F0 F3 E1 | E0 F8 EA F3 2C 0D 0A CD
```

```
00000020: E0 F0 F3 E6 F3 20 F2 E8 | F2 FC EA E0 20 2D 20 EC
00000030: E8 EB FB E9 20 E2 E8 E4 | 21 0D 0A 0D 0A 0D 0A

Пупок_чернеет_сквозь_рубашку,
Наружу_титька_-_милый вид!
```

Выводы. Символы пишутся как обычно, между строками имеем коды 0D 0A, в дельфийской нотации #13#10, на древнерусском наречии – ВК ПС, что в переводе означает «Возврат каретки» «Перевод строки». Согласно древнейшим правилам, в конце текстового файла должен присутствовать специальный символ со специальным, забытым даже Википедией, кодом , но его там никогда нет. Куда катится этот мир?

Что до наших экспериментальных операторов записи, оператор Writeln записывает коды 0D 0E. При просмотре файла в текстовом виде это проявляется как пустая строка. Оператор Write без объекта записи вообще никакого эффекта ни на что не оказывает.

Общие, но очень полезные замечания

Почему это важно? Ведь далеко не каждый день приходится разбирать бинарные файлы? Не каждый. Более того, вполне возможно, что лично вам этим никогда не придётся заниматься, хотя это вряд ли. Но необходимость разбора данных на уровне байтов возникает намного чаще, и, не только для отладки через чтение файлов. Я буду здесь немного занудным, можно?

Первое – вам просто напросто могут вручить чужой файл и поставить задачу - обеспечить гарантированное чтение этого файла в ваш Программный Продукт. Ведь это всего-навсего ещё один формат. Если повезёт, в комплекте с файлом вы получите описание в стиле *штуки – целое, длина - плавающее, где что лежит – массив целых*. В худшем случае – догадаетесь сами, во всех смыслах. А потом объясните своему начальнику, что вы тут ни при чём.

Второе – всё вышесказанное применимо не только к файлам. Бывают ещё и потоки всех видов (в смысле stream), которые могут сохраняться в файлы, а могут не сохраняться, отчего нисколько не меняют своей сущности – это те же файлы, только в памяти. Ну, или файлы - те же потоки, только на диске. Бывают каналы (pipes), сокеты, разделяемая память (shared memory), на которой, в сущности, всё предыдущее и строится. Внутри – всё то же самое, только отлаживать труднее.

Третье, не самое уважаемое занятие – стыковаться с программой на другом языке, неважно как – через динамические библиотеки или через объектный код. Труд этот не принесёт вам ни славы, ни восхищения, ни ромашек, одна только ругань со всех сторон. Чтобы эту ругань минимизировать, желательно очень хорошо знать как выглядят данные внутри хотя бы на одном из двух стыкуемых языков.

И не забываем об отладке. Даже если ваша программа что-то тихо делает в своём углу, никого не трогая, вам всё равно придётся отлаживаться. Если ваша программа любым способом обменивается данными с любой другой программой, закодированной кем-то другим, отлаживаться придётся не то, чтобы много, а очень много. И вот тут-то всё это вы вспомните и заплачете, что не выучили всего этого вовремя.

И всегда надо думать о большом и светлом. Напишите серьёзную, годную процедуру (или программу). Если это программа, то на вход её поступает бинарный файл, состоящий из записей одинаковой – обязательно - структуры. Если это процедура, то на вход её приходит неведомо откуда взявшийся байтовый поток, содержащий то же самое. Возможно, записи приходят по одной и в реальном времени.

И ещё раз – то, о чём я говорил в этой главе, это не совсем математика.

Сравните с правилом поджаривания трёх кусков хлеба на сковородке, вмещающей только 2 куска: A_1,B_1; затем B_2,C_1; затем C_2,A_2. Это не дотягивает до математики. © Литлвуд, «Математическая смесь»

Это не совсем настоящая, большая математика. Но очень близко к ней. В любом случае, всё это очень полезно для простого программиста.

Глава 2
Простая арифметика

Зачем нам, таким умным и в белом пальто, простая арифметика?

Поговорим об арифметике. Арифметика, в пределах этой главы, есть искусство сложения, вычитания, умножения и – невероятно сложно – деления. Любой язык программирования предоставляет возможность выполнения этих операция над целыми числами (деление только с остатком) и над действительными числами. Но всем всегда хочется большего.

Обычно в качестве примера нетривиальной арифметики предлагают арифметику комплексных чисел (ударение в слове *комплексных* поставьте по вкусу). Кроме того, это очень полезно и совершенно необходимо для причастных к переменному току. Что удивительно, я не встречал ни одного электрика, который бы слышал о комплексных числах. Однажды я нашёл старый, ещё советский, учебник электротехники в двух томах. Учебник был по-советски простой и доступный. На всякий случай – эта обобщённая характеристика относится только к советским техническим учебникам. Тем не менее, начиная с приблизительно девятой страницы пошли сплошные комплексные числа. Поскольку, будучи прикладным математиком, я был лишён прослушивания курса Функций Комплексного Переменного, учебник отправился в то же место, где я его и нашёл (нет, не то, что вы подумали).

// Размышление о тенденциях
Познакомился со студентом первого курса. То есть, я с ним был знаком и раньше, но тогда он был школьником. Учится он не на менеджера (по старому – приказчика), не на маркетолога (по старому – ярмарочного зазывалу) и даже не на веб-дизайнера (подмалёвщика вывесок). Учится он, опять-таки в переводе на старорусский язык, на Инженера Путей Сообщения, то есть инженера-железнодорожника.
Юноша был очень огорчён, получив для изучения примерно тот же учебник электротехники. Впрочем, огорчение его было несколько другого характера – я в свой время огорчился, что от меня требуется знание ФКП, он огорчился, что в наше время ещё надо изучать электротехнику.
// конец Размышлений

Впрочем, комплексные числа – это действительно очень полезная вещь, но в качестве объекта тренировки программиста несколько заезженная. Попробуем две другие темы – они тоже не новые, но, с одной стороны, математически проще, с другой стороны – не менее полезны для тренировки и постижения.

Первая тема несложна – арифметика натуральных дробей, впрочем, в той же степени как несложна, она и не очень применительна. Для вычислений она не очень нужна, а нужна она разве что для красивого представления данных, да и то в редких случаях.

Тема номер два – арифметика длинных чисел. Это звучит красиво и мелодично. Арифметические вычисления с высокой точностью – уже как-то суховато.

Что скрывается за этим мы увидим в следующих разделах главы. Через один – потому что следующий, очень короткий раздел будет посвящен очень короткой, но возвышенной теории.

А кто победит – арифметика или алгебра?

Как нас учили в наше время в нашей школе, арифметика – это когда 2 + 3 = 5, алгебра это когда a + b = c. То есть арифметика оперирует с конкретными числами, а алгебра с абстрактными. Для моего возраста это значит следующее: пломбир, 19 копеек плюс пломбир, 19 копеек равняется 38 копеек. Это арифметика. А вот то, что пломбир плюс пломбир стоят как два пломбира при любой их цене – это уже алгебра.

С возрастом и получением математического образования разница между арифметикой и алгеброй стала сглаживаться – для меня, конечно. Арифметика уплыла куда-то в страну серебристо-голубых негров, как Гоблин Фродо в сказке Ролингс. Чуть позже туда же уехала и алгебра – зачем это всё прикладному математику?

Среди погрузившейся в никуда арифметической Атлантиды одинокими утёсами торчат, или торчит (гусары, молчать!) кое-что из мною прочитанного.

Д.Гильберт, П.Бернайс

Основания математики. Логические исчисления и формализация арифметики.
М. «Наука», 1979
Grundlagen der Mathemaik

Бесценную книгу эту я заполучил, как трофей, за лучший доклад (я не шучу – лучший, там так и написано!), и я её прочитал! Даже! К сожалению, для меня книга эта показалась слишком сложна и более того, вызвала полное и законченное отвращение к попыткам объяснить очень научным языком то, что и так очень просто - 2 + 2 = 4, ведь это просто? А зачем целую книгу (≈700 стр.) это доказывать?

При дальнейшем изучении вопроса, точнее, дальнейшем изучении терминологии, выяснилось, что *высшая арифметика* плавно переходит в *общую алгебру*, которая, в свою очередь, превращается, как Золушка в тыкву, в *высшую алгебру*. На данном этапе всё это словоблудие начинает наполняться некоторым смыслом.

Красивые, но бесполезные концепции

Какие бывают числа? Для простейшего программиста – целые (integer) и дробные (float). В некоторых, более извращённых языках (PL/I) были и другие, но что ушло, то ушло. Далее в этом абзаце я говорю только о программировании, а не о математике. Два целых числа можно сложить и получить в результате целее число. С вычитанием абсолютно то же самое. С умножением тоже неплохо. А вот с делением уже серьёзные проблемы. Для целых чисел есть операция деления **div**, но вот такая загогулина получается: 17 **div** 3 = 5, но 5×3 = 15, а не 17. Да, можно, конечно, включить дополнительный интеллект мозга и сформулировать, что 17 = (17 **div** 3)×3 + 17 **mod** 3 = 17. Это, безусловно, правильно, но получается, что одной операции **div** у нас соответствуют две обратных – умножение и **mod**. Что-то здесь не так.

Применяя числа с плавающей точкой, от этих размышлений мы избавляемся. В результате деления single на single, имеем, опять-таки, single. А теперь о том, как оно в математике?

Для начала в математике есть числа натуральные: 1,2,3,4.5… Не заглядывая в книжку, не могу сказать, является ли 0 (ноль) натуральным

числом. Подозреваю, что это определение меняется время от времени. Ладно, пусть будут числа не натуральные, а целые: -3.-2,-1,0,1,2.3.4… Теперь важный вопрос – что будет, если сложить два целых числа? Правильно, целое число. Вычитание – то же сложение, только наоборот. С умножением проблем опять-таки нет. Обратите внимание, мы не задумываемся, почему при перемножении двух отрицательных чисел получаем – внезапно – положительное. Почему, почему – по определению. А вот с делением всё как-то не так. $\frac{6}{3} = 2$, это хорошо и правильно, вот только $\frac{7}{3} = 2.333...$, а это уже ни разу ни целое число.

Далее математики, древнегреческие, которые ещё без штанов, объявили, что $\frac{7}{3}$ это вовсе не 2.333…, а такое специальное число - $\frac{7}{3}$. Это нельзя понять, над этим надо думать. Это называется *рациональное* число. Если поделить любое рациональное число на другое рациональное число, то опять получится рациональное число.

Рациональное в переводе – *разумное*. Но, как всем известно, если есть домашние хозяйки, то где-то должны быть и дикие. Если есть разумные/рациональные числа, то где-то должны существовать и неразумные/иррациональные. Самое известное – и нам и грекам - иррациональное число $\sqrt{2} = 1.41421...$ Здесь главное понять, что число здесь не то, что справа от знака равенства, а то, что слева. Именно так это число и задаётся, а то что справа – оно же, но пропущенное через мясорубку.

Ещё есть числа *трансцендентные*. Это наши любимые π, e и прочие порождения больного разума. Вы легко определите, какие из типов чисел математических соответствуют каким типам чисел в программировании.

Теперь немного красивых слов. Новое слово *группа*. Группа это не просто числа какого-то типа. Группа – это числа в комплекте с бинарной операций, назовём её ♣. Бинарная операция – это хорошее, знакомое программисту понятие. На самом деле, группа – это не обязательно числа, но не будем занудными. Так вот, группа является группой, если:

1. (A♣B)♣C = A♣(B♣C) Это называется ассоциативность и программисту это неинтересно
2. Существует *нейтральное (нулевое) число* ϑ. При этом ϑ♣A = A♣ϑ =A.
3. Для каждого числа существует *обратное число по сложению* A^{-1}. При этом $A\clubsuit A^{-1} = \vartheta$.

Обычный математический пример – целые числа в комплекте с операцией сложения, она же операция вычитания, нейтральное число здесь ноль.. Умножение уже не подходит – нейтральное число есть – единица, но при попытке найти обратное число мы вылетаем из множества целых чисел в рациональные. Аналог в традиционных языках программирования найти трудно – в Delphi целые и плавающие числа идут сразу с полным комплектом операций, и было бы странно работать с типом integer, для начала отказавшись пользоваться **div** и **mod**.

Развитие темы – понятие *кольца*. Кольцо – это группа к которой добавили операцию умножения, обзовём её ♦, но умножение какое-то неполноценное. Для него гарантируется только ассоциативность и дистрибутивность. В программировании убедительных неискусственных примеров нет.

То, что уже на что-то похоже, называется *поле*. Поле это, опять-таки, то же кольцо, с теми же сложением и умножением, к которому добавили единичное число (элемент) $\int$, для которого $A\blacklozenge \int = A$. Кроме того, добавляется *обратное число по умножению*, определяемого аналогично, с заменой сложения ♣ на умножение ♦. Тем самым у нас незаметным образом возникает деление – обдумайте. В математике примером поля являются рациональные числа, в программировании – наконец-то – integer и single.

Зачем всё это надо и какая от этого польза? Как любят выражаться умные люди – ответ на этот вопрос выходит за пределы этой книги. Одна польза в том, что эти понятия имеют конкретное приложение в методах целочисленной оптимизации. Разумеется, они много где ещё имеют приложение, но это именно то, о чём я пишу свою следующую незабываемую и бесподобную книгу.

Вторая польза несколько абстрактнее и маячит заманчивой целью на далёком горизонте. Мы вводим какой-то класс – в программистском смысле, класс имеет какие-то свойства и методы, с ними работающие. Если мы докажем, что эти свойства и методы соответствуют, например, определению кольца, то для нашего класса автоматически выполняются все теоремы, доказанные когда-либо кем-либо для колец. Подумайте над этим.

А теперь практическое упражнение – класс для работы с рациональными числами, они же – натуральные дроби.

Натуральные дроби – несложное упражнение

Напомню – рациональное число, это такое число, которое может быть представлено в виде дроби – 1/3, 4/3. 3/9, 15/3. Дополнительная классификация. 1/3 – правильная дробь, 4/3 – неправильная дробь. 3/9 – правильная, но упрощаемая дробь, потому что в точности равна 1/3, 15/3 – вообще чепуха какая-то.

Как вы относитесь к Википедии? Амбивалентно? Или гиперсексуально? Я сходил в это место – в Википедию, в смысле – и узнал, что если дробь записана вот так – 1/3, то черта между числителем и знаменателем называется *солидус*, а если так - $\frac{1}{3}$, то эта же черта уже называется *винкулум.*

К сожалению, Word ещё не позволяет вставлять в текст файлы формата MP3 и тем более MP4 – или уже позволяет? Да и Издатель™ не пойдёт на неоправданные затраты. Но, по хорошему, на этом месте должна быть песня из Comedy Club про то, сколько можно интересного узнать, если поехать в Таиланд с женой.

Чтобы два раза не вставать – краткий русско-английский словарик по дробям:

числитель - numerator
знаменатель - denominator
общий знаменатель — common denominator
десятичная дробь — decimal (fraction)

правильная дробь— proper fraction
неправильная дробь — improper fraction
периодическая дробь — circulator, repeater
простая дробь — vulgar fraction, common fraction, simple fraction
привести дроби к общему знаменателю — to reduce fractions to the same denomination
наибольший общий делитель — the greatest common divisor / factor

Кое-какие термины из этого списка будут присутствовать в далее следующей программе. Программа эта будет выполнять некоторые операции над натуральными дробями. Зачем это надо? А вдруг понадобится. А мне хоть раз понадобилось? А ни разу. А вот вам обязательно понадобится.

Оформлено, естественно, как класс. Сначала интерфейс:

```
unit SiFrac;
{-------------------------------------------------------------}
interface
{-------------------------------------------------------------}
   type
      TFrac = class
       private
         fNum                  : integer;
         fDenom                : integer;

       public
         property num   : integer read fNum   write fNum;
         property denom : integer read fDenom write fDenom;

         constructor Create(     wNum, wDenom : integer);
         destructor Destroy; override;

         procedure Simplify;

         procedure Add(     frac : TFrac);
         procedure Sub(     frac : TFrac);
         procedure Mul(     frac : TFrac);
         procedure Dvi(     frac : TFrac);

         procedure AddImm(     wNum, wDenom : integer);
         procedure SubImm(     wNum, wDenom : integer);
         procedure MulImm(     wNum, wDenom : integer);
         procedure DviImm(     wNum, wDenom : integer);

         procedure IntPower(     pow : integer);
```

```
    //  -1 Self < frac;  0 Self = frac;  +1 Self > frac
    function Compare   (     frac : TFrac) : integer;
    function CompareImm(     wNum, wDenom : integer) : integer;

    function toStr(     wNum   : integer = 0;
                        wDenom : integer = 0) : string;
    function toStrProp : string;
    function toFl      : single;
end;
```

Комментарии. Один экземпляр класса – одна дробь. Num – сокращение от numerator. Denom – от denominator. Числитель и знаменатель задаются непосредственно в конструкторе, задавать их позже через свойства мне показалось как-то странным и неудобным. Add, Sub, Mul, Dvi – соответственно сложение, вычитание, умножение и деление. Деление называется Dvi, а не Div по очевидным причинам.

Схожие методы, но с суффиксом Imm делают то же, но вторая дробь задаётся не как класс, а непосредственно. IntPower возводит в целую степень, Compare сравнивает с другой дробью. toStr и toStrProp переводят дробь в символьное представление, с той разницей, что toStr выводит как есть, а toStrProp выделяет целую часть. То есть для дроби 11/3 первый метод так и переведёт, а второй метод напишет 3 2/3. Метод TpFl, как нетрудно догадаться, переводит дробь в число с плавающей точкой.

И самый-самый нужный метод – Simplify. Он сокращает дробь. Было 3/15, стало – 1/5.

Теперь займёмся реализацией. То, что нам понадобится из дельфийских модулей:

```
uses
   Math, SysUtils;
```

Теперь очень важная функция, не метод – определение наибольшего общего делителя для произвольного набора целых чисел. По-русски это называется НОД, по-английски, соответственно, GCD. Алгоритм не мой, какого-то Евклида.

```
function GCD(     a : array of integer) : integer;
   var
      amin                    : integer;
      ok                      : boolean;
      i,k                     : integer;
```

```
begin
     result:=1;

     if High(a) >= Low(a) then begin
        for i:=Low(a) to High(a) do
           a[i]:=Abs(a[i]);

        amin:=a[Low(a)];
        for i:=Low(a)+1 to High(a) do
           if a[i] < amin
              then amin:=a[i];

        for k:=1 to amin do begin
           ok:=true;
           for i:=Low(a) to High(a) do begin
              if (a[i] mod k) <> 0 then begin
                 ok:=false;
                 Break;
              end;
           end;
           if ok
              then result:=k;
        end;
     end;
end;
```

А почему я не вынес объявление этой функции в секцию интерфейса, вещь-то ведь универсально полезная? Вот вы и вынесите!

Гиви передаст! © Старый анекдот, у меня все анекдоты старые

Сначала конструктор и деструктор. Конструктор примитивен, а деструктор никакой, потому что незачем:

```
constructor TFrac.Create(     wNum, wDenom : integer);
begin
     if wDenom = 0 then wDenom:=100;

     fNum:=wNum;
     fDenom:=wDenom;
end;
```

Оценили изящество момента? Если знаменатель задан нулевой, дробь считается десятичной. Ещё изящнее было бы сделать второй параметр параметром по умолчания. Если его нет – то опа, десятичная дробь!

Далее методы для четырёх арифметических операций:

```
procedure TFrac.Add(     frac : TFrac);
begin
     num:=num*frac.denom + frac.num*denom;
     denom:=denom * frac.denom;
     Simplify;
end;
{-------------------------------------------------------------}
procedure TFrac.Sub(     frac : TFrac);
begin
     frac.num:=-frac.num;
     Add(frac);
end;
{-------------------------------------------------------------}
procedure TFrac.Mul(     frac : TFrac);
begin
     num:=num * frac.num;
     denom:=denom * frac.denom;
     Simplify;
end;
{-------------------------------------------------------------}
procedure TFrac.Dvi(     frac : TFrac);
begin
     num:=num * frac.denom;
     denom:=denom * frac.num;
     Simplify;
end;
```

Просто и практически очевидно. Далее аналоги для непосредственного задания дроби. Если мы не создаём объект, кто-то ведь его создаёт? Суровая правда жизни:

```
procedure TFrac.AddImm(     wNum, wDenom : integer);
   var
      frac                     : TFrac;
begin
     frac:=TFrac.Create( wNum, wDenom);
     Add(frac);
     frac.Free;
end;
{-------------------------------------------------------------}
procedure TFrac.SubImm(     wNum, wDenom : integer);
   var
      frac                     : TFrac;
begin
     frac:=TFrac.Create( wNum, wDenom);
     Sub(frac);
     frac.Free;
end;
{-------------------------------------------------------------}
procedure TFrac.MulImm(     wNum, wDenom : integer);
```

```
   var
      frac                        : TFrac;
begin
     frac:=TFrac.Create( wNum, wDenom);
     Mul(frac);
     frac.Free;
end;
{-------------------------------------------------------------}
procedure TFrac.DviImm(     wNum, wDenom : integer);
   var
      frac                        : TFrac;
begin
     frac:=TFrac.Create( wNum, wDenom);
     Dvi(frac);
     frac.Free;
end;
```

Возведение в степень оказалось немного сложнее, что немного предсказуемо:

```
procedure TFrac.IntPower(     pow : integer);
   var
      oldNum,oldDenom          : integer;
      tmp                      : integer;
      i                        : integer;
begin
     oldNum:=num;  oldDenom:=denom;

     if pow = 0 then begin
        num:=1;
        denom:=1;
     end
     else begin
        for i:=1 to Abs(pow)-1 do begin
           num:=num * oldNum;
           denom:=denom * oldDenom;
        end;

        if pow < 0 then begin
           tmp:=num;  num:=denom;  denom:=tmp;
        end;
     end;

     Simplify;
end;
```

Функции сравнения. Сначала обдумайте, как это работает с арифметической точки зрения. Потом обдумайте не было бы лучше сделать результат функции пользовательским типом, или ну его ...

```
function TFrac.Compare   (     frac : TFrac) : integer;
```

```
//  -1 Self < frac;  0 Self = frac;  +1 Self > frac
begin
     result:=0;
     if num * frac.denom < frac.num * denom then result:=-1 else
     if num * frac.denom = frac.num * denom then result:= 0 else
     if num * frac.denom > frac.num * denom then result:=+1;
end;
{-------------------------------------------------------------}
function TFrac.CompareImm(     wNum, wDenom : integer) : integer;
   var
      frac                     : TFrac;
begin
     frac:=TFrac.Create( wNum, wDenom);
     result:=Compare(frac);
     frac.Free;
end;
```

А очень важная функция Simplify оказалась очень простой, потому что всё уже сделано в функции GCD.

```
procedure TFrac.Simplify;
   var
      d                        : integer;
begin
     d:=GCD( [num,denom]);
     num:=num div d;
     denom:=denom div d;
end;
```

И, для завершённости, методы конвертирования:

```
function TFrac.toStr(     wNum : integer   = 0;
                          wDenom : integer = 0) : string;
begin
     if (wNum = 0) and (wDenom = 0) then begin
        if num <> 0
           then result:=IntToStr(num) + '/' + IntToStr(denom)
           else result:='0';
     end
     else begin
        if wNum <> 0
           then result:=IntToStr(wNum) + '/' + IntToStr(wDenom)
           else result:='0';
     end;
end;
{-------------------------------------------------------------}
function TFrac.toStrProp : string;
begin
     if num <> 0 then begin
        if (num div denom) <> 0
           then result:=IntToStr(num div denom) + ' '
           else result:='';
```

```
        if (num mod denom) <> 0
           then result:=result + toStr( num mod denom, denom)
     end
     else result:='0';
end;
{-------------------------------------------------------------}
function TFrac.toFl  : single;
begin
     result:=num/denom;
end;
```

Теперь подумает о тои, чего нет, то есть – что в этот класс – или около него - неплохо бы добавить.

Начнём со вполне бессмысленного упражнения, хотя упражнения, они все такие бессмысленные – выделим периодическую часть десятичной дроби. Для чего это нужно, я точно не знаю, но вдруг? Есть тут и ложка дегтя с бочкой мёда. Ложка дёгтя – перед собственно периодом часто идёт непериодическая часть, например – $\frac{1}{3} = 0.333... = 0.(3)$, но $\frac{1}{6} = 0.1666... = 0.1(6)$. Бочка мёда – наш алгоритм будет работать не только для десятичных дробей, но и для всех дробей с позиционной записью по любому другому основанию. Как метод класса реализовать это нецелесообразно, в классе у нас нет десятичного представления дроби, назначение класса совсем другое. Договорились, реализуем как отдельную функцию. Абсолютно универсальную функцию вы напишете сами и потом, пока напишем функцию с ограничениями. Ограничим непериодическую часть и длину периода, обе длиной 16. Здесь абсолютно не важно, каким именно числом ограничили мы эти значения, принципиально, что ограничили. Алгоритм без ограничений должен сильно отличаться. Или я не прав?

Напрашивается вот такой интерфейс:

```
   const
      maxAfter  = 16;
      maxPeriod = 16;
      minLen    = maxAfter + maxPeriod*2;

procedure FindRepetend(     d           : string;
                        var periodLen   : integer;
                        var periodAfter : integer;
                        var period      : string);
```

Красивое слово repetend, как выяснилось, обозначает повторяющуюся часть периодтческой дроби. Функция возвращает длину периода, длину непериодической части и собственно период. Десятичное представление дроби я решил представить в виде строки, мне так нравится. Очень важный момент – периодическая дробь всегда бесконечна, просто по определению, поэтому я имею наглость ожидать, что на вход нашей функции будет предоставлен достаточно длинный её кусок, не менее maxLen, По такому случаю я даже отойду от своих принципов и добавлю соответствующую проверку на входе в функцию. Кроме того, мне кажется, что достаточно на вход задать только дробную часть, то есть не '0.33333', а '33333'.

Безусловно, 16 слишком мало даже для очень несложных дробей. Но так удобнее для отладки, константы, повторю, можно заменить. Существует важное понятие – *сложность алгоритма*. Сложность, как ни странно, никак не зависит от времени, которое потребуется читателю, чтобы этот алгоритм понять. Обычно всё строго наоборот. Сложность – это приблизительная, на пальцах, где-то так, зависимость количества необходимых вычислений от количества данных. Например, у любимой мною пузырьковой сортировки сложность *квадратичная*. Это значит, что если вместо массива из 37 элементов мы захотим отсортировать 3700 элементов, то время работы программы увеличится в $(\frac{3700}{37})^2 = 10000$ раз. Это всё к тому, что наш алгоритм имеет почти линейную сложность, а лучше линейной сложности ничего не бывает. Вывод – константы можно менять смело.

Программирование чего угодно должно начинаться с программирования тестов для него. Тестировать будем следующие варианты:

Смысл	Дробь натуральная	Дробь десятичная	Длина периода
Период начинается сразу, длина периода = 1	1/3	0.(3)	1
Период начинается сразу, длина периода > 1	1/7	0.(142857)	6
Есть непериодическая часть, длина периода = 1	1/15	0.0(6)	1
Есть непериодическая часть, длина периода > 1	1/26	0.03(571428)	6

Вот тестовая программа, на все случаи сразу – временно ненужное закомментировано:

```
var
      d                         : string;
      periodLen                 : integer;
      periodAfter               : integer;
      period                    : string;
begin
     d:='33333333333333333333333333';
(*
     d:='142857142857142857142857142857';
     d:='066666666666666666666666666666';
     d:='035714285714295714285714285714';
*)
     FindRepetend( d, periodLen, periodAfter, period);

     ShowMessage( 'd = ' + d + #13#10 +
                  'periodLen   = ' + IntToStr(periodLen) + #13#10 +
                  'periodAfter = ' + IntToStr(periodAfter) + #13#10 +
                  'period      = ' + period);
```

Не считайте меня занудным и унылым, как опыт учит нас – если что-то можно понять неправильно, оно будет понято неправильно. Это я про требования к заданию десятичного представления. Кроме того, признайтесь, вы ведь даже не просмотрели этот код? Вот и я тоже, я ведь его написал.

Возвращается чукча из Москвы:

- Однако, чукчу в Союз Писателей приняли!

- Да ты же читать не умеешь.

- Однако, чукча не читатель, чукча писатель!

© Любимый анекдот чукчи

Это я к тому, в исходном коде *теста* уже содержится ошибка. Однако, бывает. Если присмотреться внимательнее, ошибок даже две. Теперь о реализации. Если вы не можете рассказать алгоритм словами, значит вы не сможете его запрограммировать. Нет, показать руками и на пальцах – не годится. Идея такая: предполагаем непериодическую часть равной нулю. Перебираем длины периода от 1 до maxLen. Если период действительно период, то Ok, если нет увеличиваем длину периода. Если всё равно не Ok, увеличиваем длину непериодической части. По-моему, всё просто и очевидно. Получилось вот такое:

```
procedure FindRepetend(     d            : string;
                        var periodLen    : integer;
                        var periodAfter : integer;
                        var period       : string);
   var
      Ok                        : boolean;
      ch                        : char;
      i                         : integer;
begin
     periodlen:=0;
     periodAfter:=0;
     period:='';

     if Length(d) < minLen then Exit;

     repeat
        periodLen:=periodLen + 1;
        period:=Copy( d, periodAfter+1, periodlen);
        Ok:=true;
        for i:=periodAfter+1 to Length(d) do begin
           ch:=period[((i - periodAfter - 1) mod periodLen) + 1];
           if d[i] <>  ch then begin
              Ok:=false;
              Break;
           end;
        end;
        if (not Ok) and (periodLen >= maxPeriod) then begin
           periodAfter:=periodAfter + 1;
           periodLen:=0;
        end;
     until Ok or (periodAfter > maxAfter) or (periodLen >= maxPeriod);

     if not Ok then begin
        periodlen:=0;
        periodAfter:=0;
        period:='';
     end;
end;
```

Что вы думаете об это тексте? Почему в условии periodLen >= maxPeriod присутствует знак *больше*? Говорит ли это о моей повышенной тревожности и мнительности? Мучают ли меня кошмары или я ими наслаждаюсь? Демонстрирует ли финальный условный оператор мою занудность тоскливость или, наоборот, аккуратность и внимание к мелочам?

Задания для самостоятельной работы:

Оцените всё-таки сложность алгоритма – линейная или квадратичная, или что? Можно теоретически, можно опытным путём.

Сопутствующая задача – по периодической дроби найти соответствующую ей рациональную дробь - в этом уже есть некоторая польза.

Обратная величина – это просто. Составные дроби, то есть $7\frac{40}{60}$ и работа с ними. Оно нам надо?

Цепные дроби. Попутно узнайте, что это такое. Красиво и полезно. Подумайте о переводе натуральной дроби в другую систему счисления, то есть $\frac{2}{9_{10}} = \frac{2}{100_3}$. Если вы понимаете это по-другому, то обоснуйте. В любом случае, красиво, но бесполезно.

Обдумайте порождённый класс, в котором в числителе и знаменателе не целые числа, а многочлены. А ещё лучше – аналитические функции.

Очень длинные числа

Выполним незатейливое упражнение, это даже программой назвать трудно:

```
   var
      int1,int2,int3           : integer;
begin
     int1:=1000;  int2:=2;
     int3:=int1 div int2;
     ShowMessage( IntToStr(int3));
end;
```

Что будет, если вместо 1000 (тысячи) написать 100500000000 (что пятьсот миллиардов)? Правильно, компилятор на пропустит. Далее пример чуть сложнее:

```
     int1:=1000;  int2:=2;
     int3:=int1 * int2;
     ShowMessage( IntToStr(int3));
```

Теперь подрисуем три нуля к первому сомножителю и сколько надо – например пять – ко второму. Что будет? Понятно что, компилятор пропустит, но в результате выполнения получим ошибку. Что-то переполнилось через что-то.

Обратите внимание на главное – с точки зрения математики и даже, не побоюсь этого слова, арифметики. У нас всё чисто. Всё должно работать – но не хочет. Потому что числа длинные. В первом случае они просто слишком длинные, что бы поместиться в integer. Во втором случае сами по себе они хоть куда, но произведение их в integer уже не вмещается. Задача – написать класс или набор процедур, позволяющих задать числа хоть в тысячу или миллион знаков и произвести над ними четыре основные арифметические операции. Заметьте, при этом становится не очень важным, являются ли числа целыми, рациональными или действительными.

Задача не выдуманная, задача реальная. В ответ на вопрос – а зачем, собственно? – обычно отвечают – а если вам захочется вычислить число π с миллионом знаков? Ответить нечего. Называется это *вычисления с большим количеством знаков* или просто – *длинная арифметика*.

В уважающих себя языках программирования средств для работы с длинной арифметикой нет. Сторонних библиотек – в ассортименте. Я решил попробовать написать свою. Обязательное требование – чтобы работала не *очень* медленно. Сложение и вычитание – элементарно. Умножение – немного сложнее. На делении я задумался. Реализовать просто, но вот чтобы не очень медленно. Поскольку лимит незаполненных страниц в книге иссякает, я эту мысль оставил, заглянув напоследок в многократно упомянутую книгу

Ч.Уэззерел "Этюды для программистов", М, Мир, 1982

В главе *πэр Квадрат* нашёлся раздел *Что можно сказать относительно деления?*

Уэзерелл оценивает трудоёмкость написания библиотеки в 5 недель для одного программиста. Попробуйте, вам же делать нечего!

Глава 3
Математическая логика.
А надо? Пожалуй, всё-таки надо

Вступление в логику

Вот эта глава уже стопроцентно математическая. Она о математической логике. Однако логика бывает разная. Перед написанием этой главы я прочитал (за вас) все нужные книги и выяснил следующее. Логика бывает:

- Человеческая или бытовая
- Аристотелева
- Школьного учебника сталинских времён
- Из учебника для техникумов (теперь колледжей) по специальности «программист»
- Для студентов пединститутов
- Математическая логика для университетов, которой меня, собственно, и учили
- Ещё есть курс А.В.Гладкого, который показался мне очень занимательным, но он отсканирован настолько небрежно, что у меня глаза вылезли наружу на стебельках и собрались в кучку.

Далее обо всём понемногу и по порядку.

Человеческая логика полезна и необходима для каждого. Это умение рассуждать здраво. *Думать надо меньше, а соображать больше*, как сказано в популярном фильме. Особенно важно это как раз для программистов. Впрочем, всю программистскую логику можно свести к одной цитате:

У каждого чрезвычайного происшествия есть имя, отчество и фамилия
© Сталинский нарком М.Л.Каганович

Если программа не работает – значит в ней есть ошибка. Если есть ошибка – у неё есть автор. И почаще повторяйте – *Чудес не бывает!*

Логика Аристотеля. Основное сочинение Аристотеля называется *Органон*. Надо пояснить, что сочинения Аристотеля делятся на три категории – те, что написал вроде бы он, те, что написал вроде бы не он, и те, что написал

точно не он. *Органон* относится к первой категории. Что можно сказать об Аристотеле, если вас кто-то спросит, конечно. Смело отвечайте – Аристотель доказал, что у мухи восемь ног. Поймайте муху и отрывайте по одной – ног шесть. Потом положите тельце на стол и ударьте по столу кулаком – муха не улетает. Следовательно, муха слышит ногами. Это и есть Аристотелева логика. Не брат ты мне, Аристотель, философ греческий.

// Философическое размышление
Когда я был не то, чтобы маленьким, и даже совсем не юным, я сдавал кандидатский минимум по философии. Преподавательница наша была ещё юнее нас всех и в больших очках, хотя и с другими достоинствами. Взгляды у неё были самые либеральные и прогрессивные – это были забавные девяностые. В качестве обязательного чтения нам предписывались три книжки. Немецкий философ Ясперс (основной тезис - осевое время), грузинский философ Мордашвили (основной тезис - русское свинство), третьего забыл, но прочитал.

В результате на экзамен я принёс *Краткий философский словарь* пятьдесят четвёртого года издания, который мы все радостно и списывали, пока преподавательница ушла греться. Лично у меня первый вопрос был как раз по Аристотелю. Экзамен мы сдали на отлично. Не подкачал старый словарь!

Кстати, согласно преданию, в словаре написано, что *Кибернетика – продажная девка империализма.* Это не соответствует действительности. Не в словаре, а в журнале *Вопросы философии*. И написано там всего-навсего *Кибернетика – прислужница мракобесов.*
// конец Философического размышления

Сталинский школьный учебник. *С.Н.Виноградов, А.Ф.Кузьмин «Логика. Учебник для средней школы», издание восьмое, М., Государственное учебно-педагогическое издательство Министерства просвещения РСФСР, 1954*

Сталин умер в пятьдесят третьем, так что учебник с полным правом сталинский, и цитаты из И.В.С. там есть, в качестве образцов логических рассуждений. Не идеальных, но вполне приемлемых, в семинариях всё же

не груши учили околачивать. Отменили преподавание логики в школах вскоре после смерти Сталина.

Сначала надо отжать применение логики к актуальной политике, национальному вопросу и мировому рабочему движению. После этого учебник сводится к предыдущему пункту – Аристотелевой логике. С одной стороны в учебнике встречаются неизвестные (мне) слова – например, *энтимема*. Word этого слова не знает, я не знаю, а школьник через десять лет после войны должен был знать. Это, наверное, плохо. С другой стороны, отдельная глава выделена нечестным методам дискуссии, сохранившимся с аристотелевских времён, пережившим сталинские и процветающим теперь. Как-то, переход на личности – *твоя* бабушка *чекистка, а сам ты злобный жидоупырь*, подмена тезиса – *зато співаем гарно,* и так далее и тому подобное. Это очень полезное знание, во всех отношениях.

Проработав учебник, неплохо попытаться перевести его содержание на язык множеств. Кстати, я просмотрел современное ЕГЭ по математике и не нашёл там вопросов по логике. Я плохо искал?

Теперь учебник для техникумов. *И.Л.Никольская «Математическая логика», М. «Высшая школа». Допущено Министерством приборостроения, средств автоматизации и систем управления в качестве учебника для техникумов по специальности «Прикладная математика».*

Тональность добрая и ласковая. Дескать, не плачь, мой маленький дружок из ухрюпинского заборостроительного техникума. Сейчас я тебе всё-всё объясню, недоумку. И объясняет. В общем, неплохая книга, стоит прочитать.

Меня заинтересовал свежий подход – конструирование для логического выражения эквивалентной ему электрической схемы. У других авторов такой идеи я не помню. Основная проблема этого учебника та же, что и у всех других – совершенно непонятно, зачем эта логика вообще нужна и как это всё применять.

Из учебника для пединститутов, учебника для университетов и монографии я ничего более полезного уже не узнал. Рекомендую

остановиться на техникуме. Дальнейшее изложение от моего лица будет идти строго на уровне этого самого фанеростроительного учебно-лечебного заведения.

Исчисление предикатов, алгебра высказываний и прочее. Звучит страшно, а так нет

Начать можно по-разному. Есть булева алгебра, есть алгебра логики, есть логика высказываний. Надо четко запомнить, что всё это совсем разные вещи. И тут же забыть, потому что в программировании всё это превращается в одно и то же. Объясняю.

Берём в руки предмет - лягушку. У вас есть лягушка? У меня есть лягушки, домашние животные, узнают в лицо и руками показывают, что хотят жрать. Формулируем высказывание *Это лягушка*, обозначим его ***Л***, потому что какая разница, как его обозначить. Поглядев на предмет, замечаем, что он зелёный. Возражать не надо, я знаю, что лягушки не обязательно зелёные, конкретно мои розовые и с когтями. Формулируем высказывание *Это зелёное*, обозначим высказывание ***З***.

Поглядев на предмет в руках, мы можем определить истинность нашего высказывания. Если у нас в руках лягушка, то высказывание ***Л*** является истинным. Если мы крепко сжимаем в мозолистой руке степлер, то высказывании ***Л*** ложно. То же самое с высказыванием ***З***. Если лягушка розовая, ***З*** ложно. Если степлер зелёный, высказывание ***З*** истинно. Пока всё не просто, а очень просто.

Напишем вот такое: ***¬Л***. Кочерга спереди обозначает *отрицание*. В переводе на человеческий означает *Это не лягушка.* Выражение является истинным тогда и только тогда исходное выражение ложно. Если у нас лягушка, то ***¬Л*** ложно, если степлер – истинно.

На этом этапе неплохо задуматься, как обозначать *истина* и *ложь*. Часто используют цифры 1 и 0. В этом есть свой резон, но, по понятным причинам я предпочитаю true и false. То есть запись ***Л***=false эквивалентна *Это не лягушка*. Обратите внимание, если Л=false, то обязательно ***¬Л***=true. По другому можно это сформулировать $\neg(\neg Л) \Leftrightarrow Л$. Знак $\Leftrightarrow$

означает *эквивалентно*, или другие, более циничные соответствия. Бывает, что вместо $\neg Л$ пишут Ё.

Отрицание является унарной операцией – говоря по-русски, оно применимо только к одному высказыванию. Есть операции и для двух высказываний – при желании, их можно выдумать целых пять, но нам хватит двух – всё равно адекватные языки программирования больше не поддерживают.

Для начала немного расширим ассортимент нашего зоопарка. Высказывание *Это жаба* обозначим ***Ж***. Высказывание *Это серое* обозначим ***С***. На всякий случай, если вы не любите Родную Природу так, как люблю её я, поясню – типичная жаба серого цвета. Что написать, если вы хотим выразить мысль *Это лягушка и это зелёное*? Есть варианты. Можно так: $Л \wedge З$. Можно так $Л \& З$. Эта формула истинна тогда и только тогда, когда истинны оба её операнда – то есть у нас в руках зелёная лягушка. Если у нас голубая лягушка и степлер произвольного цвета – формула ложна, иначе говоря, имеет значение false.

По-другому это называется *логическое умножение* или *конъюнкция* или *логическое и*. Отрицание для конъюнкции выглядит так: $\neg(Л \& З) \Leftrightarrow (\neg Л) \vee (\neg З)$. Тут надо сделать два замечания. Во-первых, знак $\vee$ я объясню чуть дальше. Во-вторых, преобразование это советую запомнить, потому что в программировании оно встречается на каждом шагу. Ну не на каждом, но часто.

Теперь наша математическая логика заметно удаляется от логики бытовой. Вот такое с виду безобидное высказывание: *Это лягушка или это жаба.* Формула выглядит так: $Л \vee Ж$. Напоминаю, если вы забыли – лягушка и жаба совсем разные животные. С бытовой точки зрения это означает одно из двух – или лягушка или жаба. С математической к этому добавляется версия, что оно и то и другое одновременно. Такого не может быть? Ещё более бытовая версия: *Я приду во вторник или в среду.* Простой человек будет ждать незваного гостя только в один из двух дней. Специалист по математической логике припрётся два раза – и во вторник, и в среду.

Это называется *логические сложение* или *дизъюнкция* или *логическое и*. Формула ложна тогда и только тогда, когда ложны оба операнда. Во всех

других случаях формула истинна. Отрицание для дизъюнкции: $\neg(Л \vee Ж) \Leftrightarrow (\neg Л) \& (\neg Ж)$. Аналогично – советую запомнить.

Для расширения кругозора – вашего – добавлю, что существует ещё и строгая дизъюнкция, она же разделяющая. Мы это презираем и игнорируем, всё равно в популярных языках программирования этого нет. Всё, что осталось понять – что всё это может сочетаться меж собой до любого уровня сложности. Например, (Л&З)∨(Ж&С) характеризует вас как любителя классики, но человека при этом непривередливого – вам всё равно, что жаба, что лягушка, лишь бы цвета были строго каноническими.

Теперь временно отложим математику в сторону и поглядим, что всему этому соответствует в Delphi, В C++ всё то же самое, вот только обозначения другие. Объявляем две переменные булевского типа, одна для лягушки, другая для зелёного цвета, третья для результата:

```
var
   frog                    : boolean;
   green                   : boolean;
   rez                     : boolean;
```

Истинность. Лягушка – да, Зелёная – нет.

```
frog:=true;
green:=false;
```

Отрицание – не Лягушка:

```
rez:=not frog;
```

Конъюнкция – Лягушка и Зелёная

```
rez:=frog and green;
```

Дизъюнкция – Лягушка или Зелёная – то есть, на выбор: зелёная лягушка, розовая лягушка, зелёный степлер.

```
rez:=frog or green;
```

Хитрое выражение $(Л \& З) \vee (Ж \& С)$

```
rez:=(frog and green) or (toad and gray);
```

Теперь немного о таблицах истинности. Попросту говоря, таблица истинности – это такая шпаргалка для альтернативно одарённых. На вход один, два или больше логических (булевских) операндов. На выходе значение некоторой формулы. Для каждой формулы таблица истинности своя. Примеры.

Для отрицания:

A	¬A
true	false
false	true

Чуть сложнее, для дизъюнкции:

A	B	A∨B
true	true	true
true	false	true
false	true	true
false	false	false

И ещё сложнее, операндов теперь четыре, и формула уже наша собственная, нетривиальная - (***Л***&***З***)∨(***Ж***&***С***). Выпишу только первые шесть строк, дальше продолжите сами:

Л	***З***	***Ж***	***С***	(***Л***&***З***)∨(***Ж***&***С***)
true	true	true	true	true
true	true	true	false	true
true	true	false	true	true
true	true	false	false	true
true	false	true	true	true
true	false	true	false	false

Как несложно заметить, если число операндов N, то строк в таблице истинности будет 2^N.

А что такое *предикат*? Да вот это всё и есть предикаты. Если мы пишем true∨false=true, то это не предикат, а если пишем ***Л***∨***Ж*** = неведома зверушка, то это предикат, потому что мы не знаем заранее, чему равно ***Л*** и чему равно ***Ж***.

Теперь о том, о чём написать надо, так какая разница – где об этом написать? Я о кванторах. Формально это относится к математической логике, но, по сути, используется везде, но только для упрощения записи.

Квантор общности - $\forall$. Демонстрирую на примере: $\forall N \in \{$множество целых чисел$\}$, что $N^3 \geq N^2 \geq N$. В переводе это значит, что для любого целого, например 7, имеем $7^3 \geq 7^2 \geq 7$. Это высказывание является истинным, что вовсе не обязательно, поменяйте знак на противоположный и убедитесь.

Квантор существования - $\exists$. Пример: $\exists N$, что $2^N = 1024$, то есть где-то есть, безусловно есть, оно не может не есть, такое N, что если в него возвести двойку, то получим ровно килобайт.

Возвращается Чапаев из Москвы после сдачи экзаменов в Академию.
- Нет Петька, не сдал. Спрашивают, сколько будет ноль пять плюс ноль пять. Нутром чувствую, что литр, а доказать – не могу...
© Народное математическое

Обычно кванторы используются не по одному, а кучно, вот так нас учили на первом курсе определять предел функции:

$$\lim_{x \to x_0} f(x) = F \Leftrightarrow \forall \varepsilon \exists \delta > 0, что \forall x, что \mid x - x_0 \mid < \delta \Rightarrow \mid f(x) - F \mid < \varepsilon$$

Вы заметили присутствие вполне русского слова *что*?

- Что это вы пишите поручик?
- Гимн полка.
- Позвольте, да тут же один мат!
- Ну как же, вот во второй строчке, слово *знамя*. ©, просто ©

О пределах мы поговорим подробнее позже.

Множества. В астрале и в реале

Сначала теория. Теория будет здесь изложена в пределах реализованного в Delphi типа данных **set**. В других языках программирования с этим

немного лучше или немного хуже. Так что такое множество? Вопрос, конечно, интересный. Поскольку множество – одной из базовых понятий математики, имеется в виду математика вообще, не только математическая логика, понятие это относится к неопределяемым – линия, плоскость, число, множество. Или, другими словами, множество – это то, что удовлетворяет определённому набору аксиом.

Множество содержит элементы. Это аксиома. *Множество, содержит, элемент* – неопределяемые понятия. Примеры множеств:

{1,2,4,7}
{синий, красный, зелёный}
{все целые числа}
{1,2, π, зелёный, дядя Евгения Онегина, таракан с шестнадцатью ногами}

Все элементы множества *разные*. {синий, синий, иней} – это не множество. В смягчённой трактовке это множество, но элемент *синий* входит в него только один раз – {синий, иней}. Множество номер три, в отличие от остальных, является бесконечным. С точки зрения математики это абсолютно нормально. Множество номер четыре какое-то подозрительное. Порядок элементов в множестве безразличен – {1,2,4,7} и {7,1,4,2} это одно и то же множество.

В традиционных языках программирования реализованы множества однотипных элементов. Во всех, традиционных и нетрадиционных языках по понятным причинам присутствуют только конечные множества. Иными словами, о множестве номер три можно забыть. О множестве номер четыре придётся забыть по другим причинам.

Отношение к допустимости повторяющихся элементов разное. В Delphi множества трактуются в классическом смысле. Если добавить к {1,2,3,4} ещё один элемент 3, то ошибки не будет – просто ничего не произойдёт. В C++ можно объявить и неправильное множество, с двумя тройками. Мы это, разумеется, осуждаем. Великий Кнут тоже признаёт множества с одинаковыми элементами, но они для него, как бы разные. Если что, это называется *мультимножество*, но, напоминаю в математике при мне такого безобразия не было.

Существует пустое множество – {}. Оно, в математике, обозначается специальным знаком – Ø. Открытие пустого множества считается для математической логики равнозначным изобретению нуля для арифметики. Для удобства дальнейших упражнений определим три конечных множества, чьи элементы являются натуральными числами.

A = {1,2,3,4,5}
B = {4,5,6,7}
C = {2,4}

То, что единица содержится в первом множестве, обозначается так:
$1 \in A$. То, что семёрка *не* содержится в первом множестве, обозначается так: $7 \notin A$.

Подмножество. Множество C является подмножеством множества A, потому что (и если) все элементы множества C принадлежат также множеству A. Обозначается это знакомым уже способом $C \in A$. Любое множество является подмножеством самого себя - $A \in A$. Если подмножество не совпадает с множеством, оно называется собственным. Для целей программирования это бесполезно, но расширяет кругозор. Если $A \in B$ и $B \in A$, то множества равны, то есть A=B. Интересный момент - с точки зрения теории множеств равенство множеств обозначает что это, просто напросто, одно и то же множество. С точки зрения программирования, это означает, просто напросто, что значения двух переменных равны. Обдумайте. Это нетривиально.

Для одного множества определена операция дополнения – все элементы, которые могли бы быть в составе множества, но их там почему-то нет. Например, если к нашему множеству принадлежит натуральное число 5, то дополнением будут все остальные натуральные числа, не равные пяти. Для программиста смысла в этом определении мало. Другое дело в Delphi – всего допустимых элементов в дельфийском множестве, как вы скоро узнаете, не больше чем 256, так что дополнение имеет вполне конкретный и конечный смысл.

Бинарные операции – то, что можно сделать, имея два множества, или больше.

Множества можно сложить (построить объединение), обозначается по разному, например так, A+B или A∨B . Смысл: {1,2,3,4,5} + {4,5,6,7} = {1,2,3,4,5,6,7}. Сумма множеств – элементы, которые входят или только в первое множество, или только во второе множество, или в оба.

Множества можно перемножить (построить пересечение). Варианты обозначения A*B, A∧B. Смысл: 1,2,3,4,5} * {4,5,6,7} = {4,5}. Пересечение множеств – элементы, которые входят в оба множества.

Гораздо реже встречается вычитание множеств, A-B, A \ B. Смысл: {1,2,3,4,5} - {4,5,6,7} = {1,2,3}. Разность множеств – элементы, которые входят а первое множество, но не входят во второе. Как легко видеть, вычитание множеств – операция несимметричная, в отличие от двух предыдущих.

Декартово произведение множеств – вещь очень важная и часто встречающаяся. Другое дело, что встречается оно обычно под самыми разными названиями. Смысл –перебираются все возможные пары, где первый элемент из первого множества, а второй – из второго. Обозначается обычно как A×B. В нашем множестве A пять элементов, во множестве B четыре элемента, соответственно их декартово произведение будет иметь 5x4=20 элементов. В результате {1,2,3,4,5} x {4,5,6,7} = {1,4}, {1,5}, {1,6}, {1,7}, {2,4}, и так далее, {5,7}.

Мощность множества. В математической логике сложнейший и важнейший вопрос. Поинтересуйтесь, что такое алеф-нуль, кардинальное число и континуум-гипотеза. Проникнитесь. Затем немедленно забудьте. Мощность любого множества в любом языке программирования всегда является конечной, что для настоящей математики тривиально. Я, кажется, забыл сказать, что же такое *мощность* множества? Мощность конечного множества – это сколько множество содержит элементов.

Практика часто бывает богаче теории, то есть теория, никому не нужная в математике, оказывается очень востребованной в программировании. Великий Кнут (как звучит-то!), в качестве примера приводит гармонический ряд, $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+...+\frac{1}{n}$, мало интересный математикам, но имеющий массу приложений в программистских алгоритмах. Будем проще, мы не Кнуты. Мы, зато, знаем формулу определения расстояния

между двумя точками на плоскости, и, возможно, даже в пространстве. Что это значит в масштабах математики? Мошка в зенице Господней! (книжка такая есть, если что, будет время – прочитаю, дрянь, наверное). А для программиста эту, скажем прямо, убогую формулу знать необходимо. И не только для программиста, специализирующегося в машинной графике. Мест ещё много есть, нужны где они.

Множества предъявляют нам пример обратный. Сколь велика их роль в математике, столь ничтожна в программировании, сиречь Computer Science.

Хотя понимание суммы, пересечения, дополнения множеств должны быть интуитивно понятны программисту, знание это ограничивается объемом того самого сталинского учебника логики для средних школ. Большего не нужно. Тем не менее, в Delphi множества есть, как тип, поэтому, хоть немного, но сказать о них нужно. А вдруг вы найдёте им применение? Первое и основное, что надо знать о множествах в Delphi – тип этот во многом игрушечный. Никакое множество не может содержать больше чем 256 элементов. Это ещё полбеды. Беда в том, что так называемый *базовый тип* для множества тоже не имеет права содержать более чем 256 элементов. Что это означает для программиста?

Объявление множества выглядит так: **set of** чего-то. Можно объявить **set of** char, потому что всех символов никак не может быть больше, чем 256 – char кодируется одним байтом. Можно объявить **set of** byte, по той же причине. Но нельзя объявить **set of** integer, даже если в вашем множестве никогда не будет больше, чем 256 целых чисел одновременно. Зато (всегда есть *зато!*) можно объявить **set of** 1000..1200. Здесь количество допустимых элементов множества явно меньше рокового числа. А если бы множества в Delphi имели (практически) неограниченные размеры, то им немедленно нашлось бы разумное и полезное применении и они не оказались бы на задворках программистского (читай – моего) разума.

Однако, изложу всё, что знаю о множествах в Delphi, лучше на примерах. Это не займёт много времени. Объявляем множество. Причём очень аккуратно, с заданием констант:

```
const
   lowS  =   0;
   highS = 255;
```

```
type
   TSet255  = set of lowS..highS;
   TSetChar = set of char;
```

Точнее, мы объявили два множества. Второе пригодится потом. Первое, представляющее непосредственный интерес, может вмещать в себя числа от нуля до 255, то есть byte, то есть наше, дельфийское, множество большее число элементов просто в себя не поместит. Второй тип – множество символов, для него ограничение выполняется автоматически – вспомните, почему. Далее объявление и инициализация двух множеств:

```
var
   a,b,c                    : TSet255;

  a:=[0,2,4,5];
  b:=[1,4,5];
```

Элементы множества заключаются в квадратные скобки. Пустое множество, соответственно, выглядит так – []. Важно! Если мы хотим добавить к любому, пустому или непустому множеству ещё один элемент, то это выглядит вот так:

```
  a:=[];
  a:=a + [5];
  a:=a + 5;  // транслятор не пропустит!!!
```

Что вообще можно сделать с одним множеством? Немного. Для начала проверить, есть ли в нём конкретный элемент:

```
ShowMessage( BoolToStr( 2 in a, true));
```

Заметьте, вместо *целого* 2 нельзя написать [2] – *множество*, содержащее целое 2. Если нам захочется проверить является ли одно множество подмножеством другого, это придётся сделать другим способом. Но пока закончим с манипуляциями над *одним* множеством. Самое очевидное и напрашивающееся, что можно сделать с переменной неважно какого типа – вывести её значение. Функции, которая выводила бы всё множество, в смысле все его элементы, в Delphi нет, придётся делать это самим.

На пути к реализации нас встречают две проблемы. Первая – для перебора всех элементов массива очень подошёл бы оператор вида

for each elem **in** a **do** <что-то>. В некоторых других языках что-то похожее присутствует. Смысл конструкции интуитивно понятен – для каждого элемента elem, входящего в множество A, мы выполняем *что-то*. Вторая

проблема – элементы одного конкретного множества принадлежат к одному конкретному типу, но тип этот может быть любым. Ну, или почти любым. Ну, не любым, только некоторым, зато разным. Поэтому я плохо представляю, как можно в Delphi написать универсальную процедуру для вывода множества в удобочитаемом виде.

В итоге, я предлагаю *две* процедуры для вывода *двух* объявленных нами ранее типов множеств.

```
procedure ShowSet255(     s : TSet255);
   var
      stroka                  : string;
      i                       : integer;
begin
     stroka:='';
     for i:=lowS to highS do begin
        if i in s
           then stroka:=stroka + IntToStr(i) + '  ';
     end;

     ShowMessage(stroka);
end;

procedure ShowSetChar(     s : TSetChar);
   var
      stroka                  : string;
      i                       : integer;
begin
     stroka:='';
     for i:=0 to 255 do begin
        if Chr(i) in s
           then stroka:=stroka + Chr(i) + '  ';
     end;

     ShowMessage(stroka);
end;
```

Доработайте процедуры, чтобы они выводили в начале строки количество элементов. Добавьте директиву **overload**, слегка изменив интерфейс. Напоминаю, эта директива позволяет объявлять одноимённые процедуры с разными параметрами. Результат должен быть примерно таким, здесь только вторая процедура:

```
procedure ShowSet(     s : TSetChar);  overload;
   var
      howMany                 : integer;
      stroka                  : string;
      i                       : integer;
```

```
begin
     howMany:=0;
     stroka:='';
     for i:=0 to 255 do begin
        if Chr(i) in s then begin
           stroka:=stroka + Chr(i) + '  ';
           howMany:=howMany + 1;
        end;
     end;
     stroka:=IntToStr(howMany) + '  // ' + stroka;

     ShowMessage(stroka);
end;
```

Это всё относилось к одному множеству. Что можно сделать с двумя?

Одним фломастером можно разрисовать всё, кроме самого фломастера. Двумя фломастерами можно разрисовать всё! © Шутка

К сожалению, для двух множеств итог не столь оптимистичен, но кое-что сделать всё-таки можно. Напоминаю, a = [0,2,4,5], b = [1,4,5]. Далее допустимые операции для множеств и в комментариях результат и, собственно, комментарий:

```
c:=a + b;  //  [0,1,2,4,5] объединение a и b
c:=a - b   //  [0,2]       принадлежит a, не принадлежит b
c:=a * b;  //  [4,5]       пересечение a и b
a <= b     //   false      a является подмножеством b - subset
c:=a + b;  c >= b
           //   true       b является подмножеством c
           //              они называют это superset, Клара!
a=b        //   false
a<>b       //   true       смысл интуитивно понятен
```

Вывод – сами множества в Delphi игрушечные, а набор операций убогий. Теме не менее – с паршивой овцы хоть шерсти клок и я попытаюсь изобрести и продемонстрировать хоть какое-то разумное для множеств применение. В следующем разделе.

Сделать что-нибудь полезное

Бывают ситуации, когда знание теории множеств полезно. Как-то где-то я программировал что-то из области клеточных автоматов. Это не совсем идеальное применение множеств, но близко. Ситуации, когда без множеств (в смысле множеств Дельфийского типа) нельзя обойтись, я вообразить не могу. Иногда они приносят крошечную пользу, избавляя от

необходимости писать собственные булевского типа функции с циклами внутри. Вспомним и применим наше второе множество, символьного типа.

```
var
   chVow                    : TSetChar   //  гласный
   chCons                   : TSetChar;  //  согласный
   chExp                    : TSetChar;  //  взрывной согласный

     chVow:= [ 'а', 'е', 'и'];
     chCons:=[ 'б', 'в', 'г', 'р', 'т', 'к'];
     chExp:= [ 'р', 'т', 'к'];
```

Множества неполные, мне просто надоело их инициализировать, но идея понятна. Теперь, если мы захотим проверить, что символ принадлежит к гласным, то напишем вот такое:

```
if (ch in chVow) then <реагируем>;
```

Лучше, конечно, взять условие в скобки. Так, на всякий случай. А теперь мы убедимся, что символ принадлежит к согласным, но *не* принадлежит к взрывным согласным:

```
if (ch in (chCons - chExp)) then <что-то ужасное>
```

Повторюсь, если бы базовый тип для множества не ограничивался байтом, множества были бы очень полезным предметом. В качестве несложного математического упражнения, сосчитайте, сколько места в памяти и на диске займёт множество, базирующееся на двухбайтовом целом. А на четырёхбайтовом?

Теперь чуть более полезная программа, имеющая некоторое отношение к реальности. Сначала – взгляд с птичьего полёта орлиным взглядом. На вход программы поступают данные, принадлежащие некоторому множеству, или декартову произведению нескольких множеств. На выходе – аналогично. Каждое из этих множеств имеет свою мощность, которая не может превышать 256. Если мощности обоих множеств, входного и выходного, больше рокового числа, то роль множеств в смысле Delphi сводится к нулю. Однако, возможны варианты. Например, на вход поступает множество мощности 100, а на выходе что-то огромное. К сожалению, я не смог с ходу придумать удачный пример. Обратная ситуация – на входе что-то огромное, а на выходе небольшое, помещающееся в мощность 256. Немного подумав, я придумал.

Сейчас мы подсчитаем количество *разных* символов в тексте. Для чего это нужно? Унылых и практических приложений я знаю, их есть у меня. Разумеется. все эти полезные приложения относятся только к тому случаю, когда разных символов в тексте не очень много. Если мы хотим, к примеру, с помощью такой технологии определить авторство текста, то нам придётся применить немного более сложный аппарат. А почему бы и нет? Как сейчас *они* пытаются определять компьютерным способом автора? Анализируют длину слов, предложений, абзацев. Употребляемые падежи, количество местоимений, и предлогов, процент существительных, имеющих перед собой прилагательные… Я ещё много могу придумать. Пойду почитаю об этом в интернетах.

Upd. Почитал. Тот редкий случай, когда на нейтральную тему в русской Википедии написано больше, чем в английской. Но в целом, именно те критерии, что я и предложил, что называется, от балды. Это потому, что я такой умный. И красивый. И в белом пальто.

Впрочем, я напишу об этом в своей следующей книге, посвященной алгоритмам работы с текстом. Великий Кнут тоже что-то подобное обещал, насчёт десятой главы. Ждём-с.

А пока подсчитаем символы. Перед кодированием мы должны себе задать один вопрос – считаем ли мы все, абсолютно все символы – то есть буквы, цифры, запятые и неотображаемые символы тоже - или только одни буковки? Если наша программа не относится к лингвистике, то все. Если относится, то в дополнение надо завести множество символов, символами не являющихся и проверять текущий символ на то, что он не принадлежит к этому множеству (**not in**). Поскольку программа несложная и должна быть очевидной, привожу сразу результат, ранее определённая процедура пропускается:

```
   type
      TSetChar = set of char;

   var
      chInText                  : TSetChar;
      badChar                   : TSetChar;
      s                         : string;
      ch                        : char;
      i                         : integer;
begin
```

```
badChar:=[ ',', '.'];
chInText:=[];
s:='Мой дядя самых честных правил. Когда не в шутку занемог.';
//  пунктуация авторская, обдумайте
for i:=1 to Length(s) do begin
   ch:=s[i];
   if not (ch in badChar)
      then chInText:=chInText + [ch];
end;
ShowSet( chInText);
```

Комментировать особенно нечего, только обратите внимание на [ch], а не просто ch, это принципиально. И да, у Пушкина фраза действительно кончается на второй строке, это при последующих изданиях всё испортили.

Ещё предложение, возможно полезное. У нас есть логическая формула, говоря красивее – предикат. Подумайте о вычислении её/его значения при произвольных значениях операндов. Это не очень сложно, особенно, если не думать об эффективности.

Подумав, подумайте ещё раз, насколько хорошо для этого подходит язык LISP. Когда-нибудь я всё-таки напишу книгу о нетрадиционных языках программирования.

Разрозненные логические замечания

Логический калейдоскоп, если угодно. Разное о разном.

Чем из всего этого приходится заниматься программисту? Программисту, много и часто приходиться кодировать условия. Здесь программиста ждут две засады. Звучит это настолько мелко, что мне как-то стыдно об этом говорить. Первая проблемка, общего характера – не все и не всегда понимают где и как надо ставить скобки в логических выражениях.

```
a:=10;  b:=3;  c:=12;
if a>b and a>c then чего-то там
```

Это выглядит просто и, главное, правдоподобно. Мы хотим убедиться, что A больше чем B и больше чем C. Что конкретно мы получим, зависит от компилятора, но наверняка получим мы совсем не то, что ожидали. Операция вычисления логического «И», то есть **and**, согласно стандарту языка Паскаль имеет приоритет выше, чем операции сравнения. Сначала

будет вычислено значение выражения (B **and** C), а уже потом прочая ерунда. Хотите жить долго и счастливо, пишите вот так:

```
if (a>b) and (a>c) then чего-то там
```

И вообще – евреи, не жалейте заварки! А вы, пацаны, не жалейте скобок.
Вторая проблема, частная, но куда более серьёзная – как инвертировать условие? Необходимость в этом возникает часто и по самым разным причинам. Простой пример, проверка нахождения числа в интервале:

```
if (x>=a) and (x<=b) then …
```

Всё правильно, мы проверили входимость числа в интервал [a.b]. А если надо наоборот, проверить нахождение числа вне интервала. Вы ведь, как я и просил, заучили формулы из главы о математической логике? Тогда ответ очевиден, а главное – не надо даже думать:

```
if (x<a) or (x>b) then …
```

Теперь поговорим о глупостях, например о доказательности правильности программ. Безусловно, тема эта неотъемлемо принадлежит и математической логике и программированию. Впервые я прочитал об этом у Дейкстры, *Дисциплина программирования*. Ничего не понял, но восхитился. Книгу я перечитывал много раз, но всё, касающееся доказательности программ, ровно столько же раз пропускал.

Много позже, уже во времена Интернета, я прочитал книгу не помню какого американского автора, не помню потому, что книгу я после прочтения немедленно удалил, чуть не написал *сжёг*. Книга эта была раза в три толще чем оригинал Дейкстры и посвящена полностью теме доказательства правильности программ. Автор относился к вопросу настолько серьёзно и благоговейно, что у меня закрались смутные подозрения – кто-то из нас дурак – или я, или Дейкстра, или забытый мною автор.

Или я, или шах или ишак © Ходжа Насреддин в переложении Соловьёва

Поскольку ни я, ни Дейкстра дураками, по определению, быть не могли, оставался только автор монографии. И я наконец понял, что Дейкстра просто шутил, рассуждая о доказательности программ, а некоторые восприняли его шутку всерьёз, да ещё и неплохо на этом заработали.

Теперь замечание, плавно вытекающее из предыдущего. Недавно я прочёл книгу, да, опять. Но, для разнообразия, не сталинских времён, а очень даже новую. Я на неё уже ссылался, но повторюсь:

Стюарт Й. "Величайшие математические задачи", М. Альпина, 2015

Книга хорошая. Книга очень хорошая. Там легко и понятно объясняются сложнейшие математические проблемы, над которыми бились и бьются лучшие математические умы. Я даже понял, *за что именно* Перельман не взял миллион. Заодно автор объяснил, почему, с американской точки зрения, Перельман очень неумный человек. Нет, не потому. что не взял миллион. Но сейчас я не об этом, а о другом, что удивило меня в этой книге.

Я, видимо, уже не в тренде, потому что не представлял, какую роль в доказательстве теорем сейчас играют компьютеры. Конечно, я читал об этом много-много лет назад, но тогда об этом рассказывалось с удивлением, умилением, восхищением и придыханием и просто, как о курьёзе. Сейчас, в последние десять лет, это – компьютерные доказательства - обычное дело. Разумеется, компьютер по определению конечен, он не может перебрать бесконечное количество вариантов. Но современные доказательства в математике стараются построить так, чтобы дело свелось к доказательству огромного, но вполне конечного, набора частных случаев. И вот тут-то железные мозги отличным образом справляются. И я верю, что компьютером действительно перебраны все возможные случаи и доказательства вполне исчерпывающие.

Но что меня удивило, так это то, что у автора книги и творцов доказательств не возникает и тени сомнения, что компьютер работает правильно. Да, сам по себе компьютер работает *обычно* правильно, хотя все, или не все, или теперь уже мало кто помнят печальную историю с первым Пентиумом – загляните в настройке компилятора Delphi и подумайте. Где искать?

Project/Options/Compiler, далее флажок, или, по-русски чекбокс, Pentium-safe FDIV.

В первом пентиуме, прямо в самом процессоре была ошибка в реализации целочисленного деления. Но такое бывает очень редко. А вот программисты ошибаются часто и очень часто, я знаю.

/ / Замечание, исполненное скорби и неверия в людей
Неверия не только в людей, но и в специалистов, и в них – особенно. Кто-то из коллег рассказал о своей родственнице. Девушка работает в аэропорту, на контроле, в звании полковника. Девушка никогда не летает самолётами – лучше паровозом. Вообще не пойму, как ещё стоматологи зубы лечат.
// конец Замечания

По-хорошему, каждый, применивший программу для доказательства теоремы, должен сначала озаботиться доказательством правильности этой программы по Дейкстре или по книге того автора, которого я стёр и забыл. И, само собой, доказательство адекватности программы должно быть составной и неотъемлемой частью доказательства самой теоремы. Но, понятное дело, об этом не может быть и речи.

Глава 4
Комбинаторика и… И всё

Сразу предупреждаю, глава будет короткой. Для чего это всё вообще надо? На самом деле я хотел поговорить об очень разных областях математики. Комбинаторика – нечто почти игрушечное. Теория чисел – нечто очень и очень серьёзное. Теория чисел - это алгоритмы шифрования. Как только вы изучите теорию чисел, за вами прилетят люди в сером на чёрном вертолете. И это в лучшем случае – могут прилететь и на голубом. Но теория чисел нам, в общем-то ни к чему, а вот комбинаторика временами требуется, особенно в начальных разделах теории вероятностей.

Перестановки. Туда-сюда обратно, тебе и мне приятно

Даже маленький ребёнок знает, что речь в этой загадке идёт о качелях. На досуге поразмышляйте, что бы это могло быть *Чтобы спереди похлопать, надо сзади полизать.* Легенда утверждает, что это из публикации в советском детском журнале *Мурзилка.* Сам не читал, но со временем начинаю верить. Сейчас мы будем этим заниматься – туда-сюда и обратно. Мы будем тасовать колоду. Потому что этот объект и пример понятнее, для меня, по крайней мере.

// Неполиткорректное замечание или наоборот?
В конце концов, у нас свободная страна или наоборот? Мне рассказали, что цыганские дети, наши русские цыганские дети, абсолютно не поддаются обучению математике и даже арифметике, но если сформулировать задачу в доступных и понятных терминах – например, пересчитать рубли в доллары с учётом текущего процента инфляции и налога на обмен валюты, задача, чудесным образом, мгновенно решается. Мы не цыгане, мы будем иллюстрировать на малой колоде из тридцати шести карт.
// конец Неполиткорректного замечания

Что надо знать изучающему комбинаторику? Знать надо совсем немного базовых понятий. И самой базовое из всех базовых понятий – это факториал. Просто потому, что он, как числа π и e, встречается везде. Разумеется, все прекрасно знают, что такое факториал. Сам я, в своих нетленных произведениях, писал об этом много раз. Но всё-таки, в очередной раз, напомню:

$N! = 1 \times 2 \times 3 \times \ldots \times N$

Просто, очевидно и легко запоминается. Чуть сложнее запомнить, что $0! = 1$, просто по определению. Так надо.

Теперь номер первый, первое понятие комбинаторики – *перестановка.* Если у меня на рабочем столе есть *кружка*, то говорить не о чем. Если у меня есть *кружка* и *степлер*, то есть варианты, с точки зрения комбинаторики, само собой. Я могу разместить их как *кружка, степлер* или как *степлер, кружка.* Если к этой коллекции добавляется рюмка, то наши возможности в перестановках растут просто стремительно: *кружка, степлер, рюмка и кружка, рюмка, степлер и степлер, кружка, рюмка…* Тут я экономлю место, всего вариантов шесть.

Теперь подумаем. Для выбора первого предмета у нас три варианта – ведь предметов всего три. После того, как мы выбрали первый предмет, предметов, что удивительно, осталось два. Так что второй предмет мы можем выбрать двумя способами. Для третьего выбора – выбора, по сути, уже нет. В результате, для трёх предметов имеем $3 \times 2 \times 1 = 3!$ вариантов перестановок. Для N предметов имеем N! Очень легко запомнить и очень легко запрограммировать.

Почитайте Перельмана-старшего. У него это гораздо лучше объясняется.

Размещение, переходящее в сочетание

Возможно, вас это удивит, но *сочетание* – то же самое, что *биномиальный коэффициент.* Вот видите, всё сразу стало понятно. О биномиальных коэффициентах много и вдохновенно пишет Кнут:

Это самые важные величины, использующиеся в анализе алгоритмов.

...

Соотношений, в которых используются биномиальные коэффициенты, так много, что открытие нового тождества не волнует практически никого, кроме самого исследователя.

...

Впервые упомянуты в трактате китайского математика Чжу Ши Цзе «Яшмовое зеркало четырёх элементов» 1303

Особенно хороша последняя ссылка. Сразу стал уважать Кнута в два с половиной раза больше.

Хочется, конечно, сразу написать формулу и дело с концом – не подумайте плохого. Однако сначала всё же покажем на пальцах. Трёх предметов будет не то, что бы мало, но недостаточно для демонстрации закономерностей. К кружке, степлеру, рюмке, добавим фломастер. Для краткости пора всё это закодировать – К,С,Р,Ф. Наша задача – выбрать из этой дребедени *два* предмета. Первый предмет можно выбрать четырьмя способами, второй – тремя. Итого имеем, извините мою занудность: (К,С), (К,Р), (К,Ф), (С,К), (С,Р), (С,Ф), (Р,К), (Р,С), (Р,Ф), (Ф,К), (Ф,С), (Ф,Р). Это называется размещение. Всего у нас N=4 элементов. Разместили мы из них K=2 элементов. Вариантов у нас $4 \times 3 = 12$. Если не очень долго подумать, то это можно записать как $\frac{4 \times 3 \times 2 \times 1}{2 \times 1} = \frac{4!}{(4-2)!} = \frac{n!}{(n-k)!}$. Запомнили? Хорошо, забудьте, потому что никому это не надо.

Размещения никому не нужны. Всем нужны *сочетания*. Посмотрите на список сверху. Какая разница, что у вас есть – (К,Ф) или (Ф,К). Главное, у вас есть и кружка и фломастер, и теперь можно одно разрисовать другим. Соответственно, список сокращается: (К,С), (К,Р), (К,Ф), (С,Р), (С,Ф), (Р,Ф). Остаётся только шесть вариантов. Говоря другими словами, все перестановки для нас на одно лицо. Вспоминая, что перестановок всего k!, получаем наконец ответ:

$$C_n^k = \frac{n!}{k!(n-k)!}$$

То, что слева, обозначение сочетания K элементов из N элементов. Другое обозначение:

$$\binom{n}{k}$$

Обратите внимание, что N и K поменялись местами. Это специально, чтобы всех запутать. Биномиальный коэффициент – это оно и есть. Ещё почитайте про Треугольник Паскаля. Это опять оно и есть. Где оно применяется – лично для меня и в первую очередь – в теории вероятностей.

Связанная с биномиальным коэффициентом тема – биномиальная теорема.

$$(x+y)^r = \sum_k C_r^k x^k y^{r-k}, r \geq 0$$

Цитата в тему:

Он происходит из хорошей семьи, получил блестящее образование и от природы наделён феноменальными математическими способностями. Когда ему исполнился 21 год, он написал трактат о биноме Ньютона, завоевавший ему европейскую известность. После этого он получил кафедру математики в одном из наших провинциальных университетов, и, вполне вероятно, его ждало блестящее будущее. Но в его жилах течёт кровь преступника. У него наследственная склонность к жестокости. И его необыкновенный ум не только не сдерживает, но даже усиливает эту склонность и делает её ещё более опасной. Тёмные слухи поползли о нём в том университетском городке, где он преподавал, и в конце концов он был вынужден оставить кафедру и перебраться в Лондон, где стал готовить молодых людей к экзамену на офицерский чин© Последнее дело Холмса, это о профессоре Мориарти.

Как страшно жить вообще, и как страшно жить математику в особенности. Могу добавить, когда я был ещё очень молодой, я тоже готовил молодых людей к экзамену на офицерский чин. Ну не совсем на офицерский чин, я готовил молодых людей к поступлению в Суворовское Училище. Но неплохо готовил, не хуже профессора Мориарти.

Кстати, читал я много, читал я часто, но ни разу не читал ни об одном маньяке-убийце с высшим образованием у них там, не считая врачей. Впрочем, к теме маньяков мы ещё вернёмся, в главе о теории вероятностей.

Пример из Кнута, пригодится далее:

Сколько существует вариантов выбора 5 карт из стандартной колоды, содержащей 52 карты? Поскольку порядок карт не имеет значения, речь идёт о выборе 5 объектов из 52, поэтому существует $C_{52}^2 = \frac{52!}{47!5!} = 2598960$ возможных комбинаций.

Композиция и разбиение

Сначала ответ на простой вопрос – чем *разбиение* отличается от *композиции*? Да тем же, чем сочетание отличается о размещения. Для разбиения порядок безразличен, для композиции, если части идут в другом порядке, то это другая композиция. В отличие от композиций, которые никому не нужны, композиции вполне востребованы. Теперь ответ на сложный вопрос – о чём речь вообще и зачем это надо?

Разбиения обычно применяются к целым положительным числам, то есть к натуральным. Есть натуральное число, наша задача – представить это число в виде суммы натуральных чисел, не пропустив ни одного варианта. Если у нас есть число 5, то имеем разбиения {1,1,1,1,1}, {2,1,1,1,1}, {2,2,1}, {3,1,1}, {3,2}, {4,1}, {5}. Обратите внимание, когда мы постигали перестановки, размещения и сочетания, сами они нас не интересовали. Мы только хотели узнать их количество. В случае с разбиениями, настоящих, очень чистых математиков интересует опять-таки количество вариантов. Поскольку дать точный ответ наука не может, она – наука – предлагает асимптотические приближения. Формулы для этих приближений настолько странные, что в Википедии помечены как сомнительные с требованием указать источник. Ничего там сомнительного нет, а в качестве источника может выступать тот же Литлвуд «Математическая смесь». Впрочем, и Литлвуд странность формулы признает. Формула настолько ужасна, что я с трудом удержался, чтобы тут же её не нарисовать. Важно другое.

Для разбиения, как я уже сказал, важно не только количество вариантов, но и сами разбиения. Где это применяется? В целочисленном программировании, например. На всякий случай, *целочисленное программирование* – один из разделов методов оптимизации. Где это применяется в реальной жизни? Имея степень кандидата технических наук по специальности *разведка и поиск полезных ископаемых*, со всей ответственностью могу заявить – именно там и применяется. Алгоритмы разбиения, к сожалению, рекурсивные. Не люблю рекурсию.

Глава 5
Теория вероятностей.
Очень полезная на самом деле вещь

Апология (слово красивое)

Любите ли вы теорию вероятностей так, как люблю её я? Тривиальная истина – человеку нравится заниматься тем, что у него получается и что он понимает. Я понимаю теорию вероятностей, хотя бы и на самом примитивно-интуитивном уровне, поэтому в книге будет глава о ней. Я не понимаю дифференциальные уравнения ни на каком уровне, поэтому в книге я их игнорирую, хотя не сомневаюсь, что это крайне полезные и самые необходимые знания для программиста-математика, а тем более программиста-физика.

В этой главе будет относительно много слов и относительно мало формул. В теории вероятностей главное понять, а не заучить формулы и алгоритмы. В заучивании нет ничего плохого. Вычислительная математика состоит из огромного количества формул, а методы оптимизации – из огромного числа алгоритмов, и очень желательно их заучить наизусть. А в теории вероятностей главное - понять.

Начнём с литературы – сначала математической, а потом и художественной. Вариантов нет. Поскольку нам даровано счастье владения русским языком, мы имеем возможность изучать интересующую нас науку по следующей книге, я её каждый раз рекомендую:

Е.С. Вентцель «Теория вероятностей», Государственное издательство физико-математической литературы, М, 1962

Автора вообще-то звали Елена Сергеевна, и Вентцель она не потому, что Вентцель, а потому, что фамилия её мужа была Вентцель, который был Вентцель потому, что был немец и сверх того генерал-майор Красной Армии во время войны с немцами, а также лауреат Сталинской премии. А сама Елена Сергеевна ещё сочиняла и художественную литературу, под вполне математическим псевдонимом И.Грекова.

То, что книге пятьдесят и даже больше лет, и я ссылаюсь на второе издание, которому уже много больше пятидесяти лет, никакого значения

не имеет. Какая разница, что там изобрели в математике за последние полвека, программисту бы освоить то, что изобрели раньше. Я не спорю, бывают и забавные новации. Вот были при мне экспертные системы или как-то так. И где эти экспертные системы? А ведь их реально пытались внедрять, я помню. Где нейронные сети? Как сейчас помню иллюстрацию к статье в солидном журнале – *Двухслойный персептрон распознаёт кошку.* Так всё и распознаёт с тех пор. По наивности поверил в вейвлет-анализ, но и тут, похоже, обманули. Хорошо работает только то, что является дубовым и кондовым.

Возвращаясь к рекомендуемой книге, нельзя не отметить, что она содержит определённые приметы времени. Так, вероятность попадания во вражеский самолёт, согласно монографии, можно определить двумя способами – или банально, вычисляя двойной интеграл, или нетрадиционно – с помощью остроумного механического приспособления, сконструированного слушателем ВВИА Н.М.Ивановым (звание не указано) в 1950 г. *При пользовании прибором Н.М.Иванова изображение цели в произвольном масштабе обводится штифтом, после чего на специальном барабане читается вероятность попадания.* (Op.cit*).*

Рекомендую также совсем другого рода книгу:

В.Китайгородский «Невероятно – не факт», М, Молодая Гвардия, 1972

Для меня с этой книги и началось знакомство с теорией вероятностей. У автора есть ещё и другая книга, под загадочным названием *Реникса.* В детстве мне эта книга понравилась, а сейчас перечитал и понял – не во всём автор прав и во многом компостировал юношеские умы.

Ещё интересная и очень тонкая книга. Тонкая не в смысле тонкости чувств-с автора, а в смысле количества страниц.

Ф.Мостселлер «Пятьдесят занимательных вероятностных задач», М., Наука, 1975

В заголовке присутствует слово *занимательный*, что означает – для неспециалиста. Вроде бы обоснованно – только в решении одной из задач нарисован интеграл, при этом автор заранее предупреждает в формулировке задачи о необходимости знания высшей математики. Все

остальные задачи решаются без привлечения методов высшей математики, исключительно методами элементарной. Автору, конечно, приходится сильно извращаться и чесать левой задней ногой правое ухо, стараясь увернуться от сложных, но эффективных методов. Или, как несколько изящнее сформулировал это великий математик Литлвуд в отношении другого великого математика – он *оперирует с этим простейшим сырым материалом с виртуозностью, недоступной для любителя.* Несколько задач из этой книги я ещё приведу, без решений, конечно. Решения приводить не буду, решения в той самой книге, будет интересно узнать ответ – найдёте, скачаете и узнаете. А может, даже, и книгу прочитаете.

Того же уровня, но с присутствием высшей математики, хотя сами рассматриваемые задачи попроще:

Г.Секей «Парадоксы в теории вероятностей и математической статистике»

Как сообщает автор вышеназванной книги – *Самой ранней книгой по теории вероятностей является «Книга об игре в кости», («De Ludo Alcaae») Джероламо Кардано (15011-1576гг), которая в основном посвящена игре в кости.* Кардано, если что, это который придумал карданный вал. Я эту книгу не читал, поэтому рекомендовать не могу.

Г.Кимбл «Как правильно пользоваться статистикой», М. Финансы и статистика, 1982

То, что книга не новая, никакой роли не играет – поверьте, ничего с тех пор не изменилось. Мне нравится подход автора:

Читая эту книгу, вы обнаружите, что автор не проявляет ни малейшего интереса к объяснению процедур, которые обычно применяются при осуществлении статистических вычислений

Это значит, что автор предъявляет результат и даёт честное слово, что результат правильный. Я хотя бы иногда пытаюсь применить какую-нибудь формулу.

И ещё, не о главном. Прочёл я рассуждения одного гражданина. Рассуждал он не о математике, рассуждал он о теории относительности. Смысл его

рассуждений сводился к тому, что раз эта штуковина называется *теория*, стало быть, всё это предположительно и вообще *вилами на воде писано*. У гражданина было своё мнение об устройстве вселенной, если что. Не знаю, как там у них в физике -

Не знаю, как там в Лондоне, я не была. Может, там собака — друг человека. А у нас управдом — друг человека! © Бриллиантовая рука

Но у нас в математике теория ничего такого предположительного не предполагает. В математике всё достоверно и бесспорно.

В целом. Начало. Предельные теоремы

С чего начинается теория вероятностей? Начинается она с основных теорем теории вероятностей и с предельных теорем теории вероятностей. На мой взгляд это не совсем правильно. Не совсем правильно то, что основные теоремы излагаются раньше предельных и не совсем правда то, что это настоящие математические теоремы. Это нельзя доказать, в это надо просто верить. Тем не менее, предельные теоремы надо понимать, а основные теоремы – знать. Не перепутайте.

Начинается всё с теоремы Бернулли, которая гласит:

При большом числе опытов частота события приближается (сходится по вероятности) к вероятности этого события

Больше похоже на заклинание, но какая разница – для программиста главное, чтобы работало. На примере. Берём колоду карт, ту, что называется *36 листов*. Тасуем, извлекаем одну карту. Какова вероятность, что это пиковый туз? Поскольку все исходы равновероятны, то, интуитивно понятно, вероятность 1/36. Если мы ограничились извлечением только одной карты, то это число кажется странным. Карту, или это пиковый туз, или это не пиковый туз, нельзя вытащить из колоды одну тридцать шестую раза. Чтобы постичь дзен, вы должны представить хлопок одной ладонью. Чтобы постичь теорию вероятностей, вы должны представить одну тридцать шестую пикового туза.

Однако если мы вытащим карту 1000 раз, каждый раз возвращая её в колоду, то можем надеяться, что искомый туз, можно я далее буду

называть для краткости его ТП, не подумайте плохого, встретится 1000/36 ≈ 28 раз. Нет, на самом деле не 28, в точности, но где-то рядом.

Урок математики в грузинской школе. Учитель:
- Гиви, сколка будэт дважды два...?
- Адынадцат...
- Нэт, нэ вэрно...! Вано, сколка будэт дважды два...?
- Восэм...
- Нэт, нэ правильно...! Гоги, сколка будэт дважды два...?
- Пять...
- Маладэц, Гоги... Да, гдэ-то пять-шэсть, но ныкак нэ адынацат
© Старый народный анекдот. Все совпадения с реальностью случайны

Попробуем проверить. Для опытов применим объект из приложения, найдите сами. Тестовая программа выглядит так – или выглядела, если я не успел внести в ней изменения до выхода книги.

```
   var
      D                          : TDeck;
      SA                         : TCard;
      skoka                      : integer;
      numOfSpadeAces             : integer;
      i                          : integer;
begin
     D:=TDeck.Create;
     SA.suit:=spades;
     SA.rank:=Ace;

     skoka:=100000;
     numOfSpadeAces:=0;

     for i:=1 to skoka do begin
        if  SameCard(D.CardNumToCard(D.RandomCard), SA) then begin
           numOfSpadeAces:=numOfSpadeAces + 1;
        end;
     end;

     ShowMessage( 'Пиковых тузов ' +
                  IntToStr(numOfSpadeAces) + '   ' +
                  FloatToStr((100*numOfSpadeAces)/skoka) + '%');

     D.Free;
end;
```

Не могу оставить ни одной программы в непрокомментированном состоянии. По-русски Туз Пик сокращается до ТП. По-английски Ace of

Spades ожидаемо сокращается до AS, я же написал SA. Причина, как вы конечно уже догадались – слово **AS**, в любом регистре, является зарезервированным идентификатором в Delphi. Кстати, если вам не нравится мой английский, напишите письмо моему *Редактору* и моему *Корректору*. А пока мы посмотрим на результаты работы. В столбцах, соответственно, - количество попыток, число выпавших пиковых тузов, оно же в процентах. Напоминаю, предсказанный результат равен 1/36 ≈ 2.78%.

1	0	0.00%
10	0	0.00%
100	4	4.00%
1000	29	2.90%
10000	293	2.93%
100000	2737	2.73%
1000000	27684	2.77%

Теперь то же самое, но для числа испытаний, кратных 36. Идея понятна – для 36000 испытаний ожидаемый ответ ровно 36000/36 = 1000. Мне кажется, это нагляднее:

36	2 (сюрприз!)	5.63%
360	9	2.59%
3600	94	2.62%
36000	1002	2.78%
360000	9805	2.72%
3600000	99855	2.77%
36000000	1000840	2.78%

Вот так, через какую-то хитрую загогулину, и работает теорема Бернулли. Если мы вытащим карту не тысячу, а сто тысяч раз, то приближение к идеалу (2800) будет ближе, не в абсолютных числах, разумеется, а в процентах отклонения. Хотя мне интуитивно казалось, что сходимость к теоретическому значению должна быть намного быстрее.

Дальнейшее развитие эта отрасль теории вероятностей получила в теореме Чебышева, иначе именующейся Законом Больших Чисел:

При достаточно большом числе независимых опытов среднее арифметическое наблюденных значений случайной величины сходятся по вероятности к её математическому ожиданию

Теорема имеет некоторый оттенок шаманских заклинаний.

О математическом ожидании в следующем разделе, но интуитивно и так понятно – для ТП математическое ожидание его извлечения равно 1/36. Математическая наука, насколько я вижу со своей, программистской, точки зрения, разделилась на два течения – одно считает эти теоремы вполне разумными и достойными доказывания, а другое – полной чепухой и, не побоюсь этого слова, тавтологией. Интересно то, что в любом случае это работает. Вернёмся к нашей колоде карт.

Колода карт номер один из тридцати шести содержится в приложениях. Не в смысле, что там подклеена натуральная колода - в смысле там подклеена её программная эмуляция.

// А вот раньше было
А раньше было не так. Не так всё было. Я очень любознательный. Мало того, я готов (был готов) потратить на это своё хобби несколько бывших советских рублей. Однажды, примерно за десять советских рублей я приобрёл роскошное издание *Истории Гражданской Войны*. Год издания 37-й. Между страниц была вклеена натуральная матерчатая повязка дружинника Выборгской части. Вот так бы надел бы, да и пошёл.
// конец А вот раньше было

В тридцать шестой раз повторяю - вероятность извлечения ТП равна 1/36. Мы сделали 36 попыток. В среднем - а что такое *в среднем*? - ТП должен был встретиться ровно один раз. Удивит ли нас, если мы не вытащим ни одного? Нет, нисколько. Жизненный опыт подсказывает, что ничего удивительного в этом нет. А чему равна вероятность *не* вытащить ни одного ТП при 36 попытках? А за это отвечают уже основные теоремы теории вероятностей, и этим мы займёмся в следующем разделе. Продолжаем развлечения с колодой.

Мы извлекли карту из колоды 3600 раз. Предельные теоремы как бы намекают, что ТП должен был попасться 100 (сто) раз. Разумеется, такого идеального совпадения странно было бы ожидать, но что если наша

любимая карта не выпадет ни разу? Удивит ли это нас? Очень! Численное значение того, насколько нас это удивит, мы рассчитаем позже.

Всё предыдущее понятно всем и возражений не вызывает. Теперь о том, что далеко не всем понятно, но что непосредственно следует из этих, то ли математически доказанных, то ли принятых на веру предельных теорем. Повторяю – мы извлекли карту 36000 раз – и ни одного пикового туза. Напоминаю – это маловероятно, хотя и вполне возможно. Если мы извлечём карту ещё тысячу раз, сколько там будет ТП? Вот на этом вопросе ссылающиеся на теорию вероятностей делятся на два направления – правильное и неправильное.

Их будет 28, в среднем. Тузов – в среднем - не будет ни насколько больше их математического ожидания. Обычно далее следует аргумент о том, что, согласно предельной теореме, математическое ожидание для конкретной карты равно 1/36, а мы, вместо этого, получили аккуратный ноль, следовательно природа должна компенсировать свою ошибку и восстановить норму, увеличив выпадение пиковых тузов.

Природа своё дело делает.

Божьи мельницы мелют медленно, но верно © Автор неизвестно кто, даже век точно неизвестен

Это так, но происходит это немного по-другому. Мы вытащили карту 36000 раз и не получили ни одного ТП, то есть 0% от общего количества, просто ноль, хотя должны были теоретически иметь 2.77% и семь в периоде на сдачу. Что будет, когда мы извлечем ещё 36 тысяч карт? Будет ли среди них больше ТП, чисто ради компенсации ошибки природы? Нет, если природа захочет компенсировать недобор, тузов будет ровно столько же, сколько и положено по теории – 1/36. Когда мы посчитаем итоговый процент по 72000 попыток, то получим 1.39%. Не идеал, но уже лучше. При следующих тысячах извлечений соответствие теории будет всё лучше и лучше, хотя приближение к идеалу будет всё медленнее и медленнее. Напоминает второй закон термодинамики – всё везде остынет и везде будет одинаково холодно. Будет. Но остывать пространство будет всё медленнее и медленнее с каждым миллионом лет.

Вот так они и работают, предельные теоремы теории вероятностей. Только не говорите, что для вас это было очевидно с самого начала. Может да, а может и нет. Я наблюдал людей, аккуратно записывающих и компьютерным образом анализирующих результаты лотерей, с целью выяснить у какого номера шансы больше, потому что он раньше не выпадал.

В целом. Продолжение. Основные теоремы

Основные теоремы теории вероятностей намного приятней предельных теорем её же. Они практичны, полезны и, возможно, их доказательства даже имеют смысл. Но главное, повторюсь, они не относятся к разделу философии – они понятны и доступны, даже мне, но главное – не обязательно заставлять себя в *это* верить. *Это* очевидно и так. Все дальнейшие опыты будут производиться над той же самой колодой из тридцати шести карт.

Какова вероятность извлечения ТП – $1/36 \approx 2.8\%$. Это недоказуемо, это аксиома. Какова вероятность извлечения двух ТП подряд? Это как-то неуверенно доказывается, но мне кажется, что надёжнее здесь положиться на интуицию. Чтобы вытащить два туза, надо, сюрприз, чтобы тузом была первая карта и вторая тоже. Вероятность для первой карты мы знаем – 1/36. Вероятность для второй та же самая. Итоговая вероятность вычисляется перемножением этих двух вероятностей $1/36 * 1/36 = 1/1296 \approx 0.077\%$. Мы их перемножили. По научному это так и называется – умножение вероятностей.

Важное замечание. В той форме, в которой я это изложил, это является применимым только для *независимых* событий. Независимые события – тот случай, когда исход второго события не зависит от исхода первого события. Для развлечений с колодой это означает, что карта после извлечения возвращается в колоду. Иначе понятно – если в первую попытку мы вытащили ТП и не вернули его в колоду, то вытащить его при второй попытке крайне маловероятно. Заметьте, я не говорю невозможно – случаи, они, как известно, разные бывают. Я сам такие случаи видел.

Если этот пример кажется вам совершенно очевидным и даже местами глуповатым, рассмотрим задачу немного сложнее – какова вероятность извлечь два произвольных туза любой масти – с возвращением и без? С

возвращением: 4/36 * 4/36 = 1/81 ≈ 1.2%. Без возвращения: 4/36 * 3/36 = 1/108 ≈ 0.093%. Разница заметна.

Второе замечание скорее не замечание, а из области вопросов на сообразительность. После того, как клиент трудолюбиво (и правильно) вычислит вероятность выпадения двух пиковых тузов (1/1296), надо внезапно и грубо спросить его, чему равна вероятность выпадения двух одинаковых карт. А вот вторая карта уже должна совпасть с первой, что даёт нам всё то же число 1/36.

Есть немалая вероятность, что клиент даст тот же ответ - 1/126. Это неверно. Поскольку карта не задана, вероятность извлечения для первой карты тождественно равна единице – потому что нам все равно! А вот для второй карты вероятность, что она совпадёт с первой всё та же – вечное 1/36, чему и равняется результат.

Следующая задача взята из книги Мартина Гарднера «Математические головоломки и развлечения». Обдумайте:

Мистер Смит сообщает, что у него двое детей и по крайней мере один из них мальчик. Какова вероятность того, что второй ребёнок мистера Смита тоже мальчик? <далее у Гарднера следует ответ>. *Ситуация резко изменилась бы, если бы Смит сказал, что мальчиком является старший из его детей.*

Мы в совершенстве освоили вероятность событий, разумеется при условии, что они получаются в результате умножения вероятностей, сложение мы ещё не учили. А чему равна вероятность того, что событие *не* случится? *Элементарно, Ватсон* - вы знаете, что оригинальный ШХ никогда не произносил эту фразу? Если P вероятность того, что оно случится, то вероятность того, что оно не случится, равна 1 – P. Если вероятность выпадения ТП равна 1/36, то вероятность того, что это не случится равна 1 – 1/36 = 35/36. Для двух ТП подряд 1 – 1/1296 = 1295/1296. Это очень простой раздел теории вероятностей. Он должен был называться теоремой о вычитании вероятностей, но у авторов учебников всё же есть совесть.

Немного сложнее другой вопрос – какова вероятность, что из двух последовательно извлечённых из колоды карт не будет ни одного ТП?

Сначала напрашивающийся, но неверный ответ из предыдущего абзаца: P = 1 – 1/36*1/36 = 99.13%. Это легко, но неверно. Мы применили обе теоремы: об умножении и о вычитании. Теоремы правильные, но только применили мы их неправильно. Решим задачу с самого начала. Чему равна вероятность, что первая карта *не* будет пиковым тузом? Правильно – 35/36. Для второй карты, естественно, то же самое. Вероятность двух событий одновременно 35/36 * 35/36 = 1225/1296 ≈ 94.5%. Разница заметна.

Теперь можно ответить на вопросы, невзначай заданные в предыдущем разделе. Чему равна вероятность, что из 36 случайных карт не будет ни одного туза? Правильно, $(35/36)^{36} \approx 36.2\%$. Я ожидал меньшего значения, получается, что событие это не то что реальное, а даже совершенно заурядное. Разумеется, это при условии, что карта каждый раз возвращается в колоду. На языке теории вероятностей это означает, что испытания *независимы*.

И второй вопрос - чему равна эта вероятность для 36000 попыток? Первый результат я получил на калькуляторе. Для второй задачи доверия к БГ у меня не было и я использовал логарифмы. $(35/36)^{36000} \approx 0.97^{36000}$. 36000 * lg(0.97) ≈ 36000 * -0.0132 * 36000 ≈ -476.21. Теперь надо бы по хорошему возвести 10 в эту степень, но при таких размерах можно смело писать просто: 10^{-476}. Спите спокойно, дорогой товарищ, один туз точно будет. Что интересно, калькулятор от БГ дал практически такое же значение – 10^{-477}. Вывод – тупые там не все, а Задорнов не всегда прав.

Теперь займёмся сложением вероятностей. Надо заметить, что разделение на сложение и умножением вероятностей несколько условно – одно можно выразить через другое, возможно, с применением вычитания. Уже пройденное умножение применяется в тех случаях, когда события не взаимоисключающие (1), а сложение – наоборот – когда события взаимоисключающие (2). Слова очень длинные, поясню. В поисках пикового туза мы два раза вытаскиваем из колоды карту и каждый раз возвращаем её. ТП может выпасть и в первый раз, и во второй. Это слово №1. Если мы *не* возвращаем карту в колоду, то только в первый или только во второй, ТП ведь только один. Это слово №2. Если приглядеться, то заметно различие – во втором случае мы могли бы взять две карты сразу, результат очевидным образом был тот же. В первом случае вероятность другая.

Итак, мы берём карту, прячем её в рукав, берём другую. Какова вероятность, что одна из двух ТП? 1/36 + 1/36 = 1/18. Очень просто. Рассмотрим предельный случай – мы берём все 36 карт, по одной или сразу – уже не важно. Интуитивно ясно, что у нас 100% вероятности успеха. Математика, или даже арифметика интуицию подтверждает: 1/36 * 36 = 1.

Сложение вероятностей с успехом заменяется умножением с вычитанием. Для нашего случая – вероятность, что первая карта не ТП 35/36. Вероятность, что вторая не ТП, при условии, что первая не ТП – 34/35. Вероятность того, что хотя бы одна из двух нас удовлетворит, равняется 1 – 35/36 * 34/35 = 1/18.

Сложение применяется не только последовательно, но и параллельно. Какова вероятность, что вытащенная карта или пиковый туз или пиковая дама или бубновый туз? Карту мы тащим только одну. Поскольку выпадение одной карты полностью исключает другую, то применимо сложение вероятностей: 1/36 + 1/36 + 1/36 = 1/12.

В целом. Окончание. Байес и компания

А теперь вершина теории вероятностей – если, конечно ограничиться первой главой учебника. Формула Байеса. Все вы слышали старую заезженную шутку о блондинке, которую спрашивают – какова вероятность, выйдя из дома, встретить динозавра? Одна вторая, отвечает блондинка – или встречу, или не встречу. Блондинка, разумеется, ошибается – но не так, как вы думаете. Байес с вами и блондинкой не согласен.

Однако вернёмся к нашей колоде, точнее уже не совсем к нашей. Наша колода попала в руки к жалким, ничтожным людишкам. Людишки выкинули из колоды не глядя половину карту, взяли ещё три колоды – и не факт, что на 36 листов, а может на 54, с джокерами. Отсыпали оттуда в нашу колоду неведомое количество неведомо каких карт. И тщательно перетасовали. Как результат, мы не знаем ни сколько карт в нашей колоде, ни какие там вообще карты. Вопрос всё тот же – какова вероятность извлечения пикового туза? На помощь спешит Байес со совей потрясающе простой формулой. Формул у Байеса, если честно, две: для умных и для программистов. Нам, конечно, лучше для программистов.

А вот и знаменитая формула: $P = \frac{m+1}{n+2}$

Слева, понятное дело, искомая – нами - вероятность. N – число успешных попыток. Для нас успешная попытка та, которая принесла ТП. M – число попыток вообще. Вначале, как нетрудно заметить, и M и N равны нулю и формула возвращает P = ½. Тот же анекдот про блондинку, вид сбоку. А если в первый раз туза пикового не будет? Вероятность его выпадения при второй попытке $^{1}/_{3}$. Соответственно, при десяти неудачных попытках, вероятность будет $^{1}/_{12}$. Вернёмся к блондинке и динозавру и решим задачу правильно. Сколько раз блондинка вообще выходила из дома? Пусть будет 100000, нам не жалко, блондинки они такие, входят и выходят. По формуле Байеса получаем $^{100001}/_{100002} \approx 0.001\%$. Если вам кажется что этот Байес просто жулик, а его формула просто мухлёж, то вы не один такой. В математическом вероятностном мире идет война сторонников байесовского подхода - это мы - и кого-то ещё. В наше время всё было проще – подход бы только один. А главное – этот подход работал – и до сих пор работает хорошо.

А я Байеса люблю, я за Байеса пойду © студенческая перефразировка советской эротической песни. Да, при Советской Власти тоже были эротические песни.

Глава 6
Теория вероятностей. Некоторые бесполезные применения

То, что было рассказано о теории вероятностей выше, примерно соответствует первой главе любого учебника по этому предмету. А за первой главой в этом любом учебнике следует ещё много, много глав. Но, с точки зрения понимания о чём она вообще, эта теория вероятностей, эта первая глава содержит где-то половину того, что надо понять. Ещё сильнее – если вы поняли первую главу – вы поняли главное.

А теперь, чтобы вы поняли первую главу ещё лучше, применим понятое к некоторым не очень практическим и не очень серьёзным примерам. Впрочем, если вы возьмёте в руки любую популярную книгу по теории вероятностей, то увидите там не точно те же, но очень похожие примеры. Преимущество моей книги в том, что я ещё и перевожу эти примеры на язык программирования.

Несколько простых упражнений. Ельцин, Титаник, Луна и мёртвые президенты

Любите ли вы теории заговора злобного сияния порочного разума так, как люблю их я? Или вы вообще не знаете, что такое теория заговора? Из старенького – американцы не были на Луне, а Титаник и Ельцина подменили. Нет, не взаимно, не посудину на человека, а внутри элементов соответствующего множества однородных элементов. Египетские пирамиды и Сфинкса смастерил Наполеон и компания в начале девятнадцатого века методом некачественной заливки бетона с кривой арматурой и незатёртой опалубкой. А внутри пирамида Хеопса пустая – в смысле, там песок, и хорошо, если кварцевый. Я очень люблю такие идеи.

Возьмёмся за что-то относительно свежее – за катастрофу *Челленджера.* Это, если вы не в курсе, такой американский нелетающий космический корабль, бывают ведь нелетающие птицы, пингвины, к примеру. Так вот, этот корабль взорвался при взлете в январе 1986 года. Все семь членов экипажа погибли. Альтернативная версия, придуманная самими американцами же, предполагает, что никто не погиб (я уже забыл, почему так было надо), а все семь членов экипажа живут и поныне, под другими именами, а кто и под своими. В доказательство предъявляются

фотографии портретов лиц – очень похожие, согласен, и имена почти те же. У двух членов экипажа внезапно обнаружились братья-близнецы – алиби, что ни говори, идеальное. Мысль в том, что оригинальные герои не умерли, а продолжили свою жизнь под видом своих как бы близнецов.

О заговорах мы поговорим в другом месте, надеюсь, а пока разрешим несколько простых вопросов. Какова вероятность что из семи случайно выбранных человек *двое* имеют близнецов? Какова вероятность, что *хотя бы один* имеет близнеца? Какова вероятность, что ни один не имеет близнеца? Упражнение несложное, но очень способствует закреплению пройденного.

Начнём. Мы не математики, мы прикладные математики – а кто конкретно вы, я не знаю.

Або жид, або москаль – © украинский народный юмор.

Прикладному математику не нужно точно, пусть приблизительно, лишь бы думать меньше и считать быстрее. Разумеется, когда мы интересуемся вероятностью выпадения двух близнецов, нам не нужно ровно два, можно и больше, лишь бы формула была проще.

Второй и третий вопрос – один и тот же вопрос, причём очень простой. Или хотя бы один из астронавтов имел брата-близнеца или ни один не имел, события взаимоисключающие. Если одна вероятность P, то вторая P-1. Вероятность рождения близнеца примерно 1%, это вне теории вероятностей, это просто медицинский факт. Для наших целей точность достаточная, а число очень удобное.

Как легко догадаться, или вспомнить, вероятность того, что ни у одного из семи астронавтов близнецов нет равна $0.99^7 \approx 93\%$. Вероятность наличия хотя бы одного близнеца соответственно $\approx 7\%$., очень даже вероятная величина. Впрочем, можно было не заморачиваться с возведением в степень а сосчитать просто: 1% * семь астронавтов = 7%. Можно и так, но надо чётко понимать, что равенство здесь не точное, а приблизительное – более точное значение 6.8%. Такая нехитрая манипуляция возможна только потому, что 0.01 малое значение, а седьмая степень для такого значения тоже является малой. Это как с классической формулой для

синуса – Sin(x) ≈ x, но это верно только для малых значений x. И только в радианах.

Но нас интересует не один близнец, а два - можно больше. Сначала решим задачу интуитивно. Это полезно, потому что если полученный на пальцах ответ хотя бы одного порядка с рассчитанным по науке, значит мы всё понимаем правильно – и я, и пальцы, и наука. Какова вероятность что 1-й и 2-й астронавты одновременно имеют близнецов? Правильно, P = 1% * 1% = 0.01%. Но ведь близнецов могут иметь и (1-й и 3-й), (1-й и 4-й), далее (2-й и 3-й), (2-й и 4-й) и т.д. Первого из двух мы можем выбрать семью способами, второго только шестью. Результат = 7*6 = 42. Подумайте, не следует ли разделить результат на два, ведь нас не интересует порядок выбора. Итого 0.01%*42 где-то 0.42%. Такая неуверенность оттого, что вероятность рождения близнецов не равна точно 1% и оттого, что рассуждения и вычисления наши носят очень уж эмпирический (≈ на пальцах) характер. Что это число может значить применительно к теории заговора, обсудим позже, а пока получим искомую вероятность способом из учебника (Вентцель, конечно).

Кстати, обдумайте, что при увеличении кратности близнецов мы неизбежно приходим к факториалу.

То, что нам нужно, называется *Частная теорема о повторении опытов.* Вначале идут обязательные оговорки о независимости опытов, наш случай этим условиям удовлетворяет. Обозначения:

p – вероятность успеха в одном опыте (1%)
q = 1 – p – вероятность неуспеха в одном опыте (99%)
n – количество опытов (7)
m – требуемое нами количество успешных результатов (2)
C_n^m – число сочетаний из n элементов по m – смотри главу по дискретной математике и комбинаторике

В результате имеем искомую вероятность:

$$P_{m,n} = C_n^m \, p^m q^{n-m} = C_7^2 * 0.01^2 * 0.99^{7-2} \approx 0.00399$$

Это очень хорошо совпадает с предварительной оценкой (0.4%), но надо учесть один момент. Формула даёт вероятность *ровно* для двух пар близнецов, нас же вполне устроят и три, и больше. Но, опять-таки интуитивно, кажется, что вносимая поправка будет пренебрежимо мала. Проверьте, если не лень. Другой, даже более важный момент – ведь обычные разнояйцовые близнецы не очень-то и похожи, да и вообще, в большинстве случаев бывают разного пола. Так в интернетах написано, а интернеты никогда не врут. Для настоящего заговора с подменой они не очень подходят. Для настоящего заговора нужны идентичные, однояйцовые близнецы. Я полез искать вероятность их рождения и запутался в разночтениях и умолчаниях, но ясно, что она намного ниже вероятности рождения неидентичных близнецов. Разберитесь сами и посчитайте всё заново. Обдумайте, так был ли заговор?

А вас ждут давно обещанные мёртвые президенты - ожидают в следующем разделе – по четыре штуки в ряд.

Те самые президенты

Продолжим математически развлекаться на тельце американской демократии. Только что на каком-то заборе я прочёл, что три американских президента умерли в День Независимости (четвертое июля). Что это – совпадение или заговор? Посчитаем. Сколько всего есть *умерших* президентов? Всего президентов было и есть 45. Живых 5 + 1 *нынешний*. Умерло 45 – 5 – 1 = 39. Вероятность, что один из них умер в День Независимости равна $39/365 \approx 0.107$. Поправку на то, что кто-то из них мог умереть в високосный год, мы игнорируем и презираем. Возводим это число в куб и получаем $0.0012 = 0.12\%$. Может, и вправду заговор? А может нет?

Расчёт несколько грубоватый, я сам вижу недостатки и даже имею какие-то подозрения в правильности метода, посчитаем аккуратнее. Но прежде чем считать аккуратнее, сначала посчитаем просто по-другому, вдруг поможет.

Сначала рассчитаем вероятность того, что три президента умерли в один день, а потом умножим полученную величину на 1/365. Мне эта методика кажется разумной. Первый президент из 39 имеет право умереть когда ему хочется. Для второго вероятность умереть в тот же день равна 38/365, а для

третьего 37/365. Перемножаем и получаем ≈ 0.01 = 1%. Делим на 365, имеем ≈ 0.003%. Расхождение результатов без малого на два порядка. Кто-то не прав. Надо проверить третьим способом и перепроверить четвёртым. Займёмся проверкой, а потом обдумаем.

Сначала попробуем получить результат даром – может быть, кто-то в интернетах уже посчитал? Если бы нас интересовала вероятность совпадения *двух* дней рождения, то ссылок гигантское число – запрос *Парадокс дней рождения*. В нашем контексте, что день рождения, что день смерти – эквивалентно.

Без особых затруднений находим *Парадокс дней рождения для трёх человек* на habrahabr.ru. Вот результат, прочтите и немедленно забудьте, тем более, что запись формулы неаккуратная, неэстетичная и неэротичная:

P(A) = 1 — P(B) = 1 — k! (Cn0 Cn-0k-0 / 20 + Cn1 Cn-1k-2 / 21 + … + Cnm Cn-mk-2m / 2m + … + Cn[k/2] Cn-[k/2]k-2[k/2] / 2[k/2]) / nk .

Трудолюбивые люди трудолюбиво выводят длиннейшую формулу, которую я даже и анализировать и понимать не буду, не то, что использовать в расчётах. К счастью, трудолюбивые люди уже трудолюбиво посчитали ответы для некоторых значений числа людей в группе. Наших мёртвых президентов, напоминаю, 39. Такого числа там, понятное дело, нет, но есть 40. Вероятность для него 0.067. Делим на 365 и получаем 0.00018 = 0.018%. Расхождение почти на порядок, причём расхождение с обоими ранее полученными результатами – в разные стороны, так не бывает, кто-то не прав.

Теперь посчитаем строго канонически, по формуле приведённой чуть выше – *Частная теорема о повторении опытов.*

Обозначения те же:

p – вероятность успеха в одном опыте 1/365 ≈ 0.0027
q = 1 – p – вероятность неуспеха в одном опыте 364/365 ≈ 0.9973
n – количество опытов (мёртвых президентов) 39
m – требуемое нами количество успешных результатов 3
C_n^m – число сочетаний из n элементов по m 9139

Итоговая вероятность

$$P = 9139 * 0.0027^3 * 0.9973^{36} \approx 0.00016 = 0.016\%$$

Подозрительно похоже на результат с Хабра, с учётом того, что там испытаний на одно больше. Обратите внимание, что версия теории заговора стремительно набирает очки. Ещё обратите внимание что более общая формула из учебника является несравненно более простой по сравнению с формулой трудолюбиво выпиленной лобзиком умельцами-рукодельниками. А результат – то же. Прочитайте, наконец, учебник!

Теперь занудное уточнение – мы посчитали вероятность *ровно* трёх успехов. Когда в день независимости умирают ровно четыре или ровно пять президентов – такой расклад не учитывается. Требуется некоторое умственное усилие для осознания этого – ведь если в один день умерли четыре, значит умерли и три тем более! На самом деле с точки зрения применяемой формулы это взаимоисключающие события – или три, или четыре, извините, без хлеба нельзя (© ≈ Кролик и Вини-Пух). Интуитивно кажется, что вызываемая этим погрешность пренебрежимо мала.

Теперь о главном – похоже, первые две попытки были безнадёжно ошибочными, хотя и выглядели весьма правдоподобно. В реальной программистской жизни времени всегда – или, что то же, никогда – какой забавный русский язык – нет. Берём два последних ответа и забываем про два первых. У нас время есть, и мы попытаемся разобраться, в чём наша ошибка. Ладно. В данном случае не наша, а моя.

В качестве несложного упражнения предлагаю вам самостоятельно посчитать сравнительную вероятность достоверности гипотез верности первого, второго и третьего-четвёртого ответов. Оценка достоверности гипотез не очень простое занятие, но у нас случай тривиальный, почти вырожденный.

Посчитав, вы убедитесь, что очень возможен вариант, что убедительное единогласие третьего и четвёртого ответов ничего не значит, а прав кто-то из двух первых. Как решить, кто прав?

В нашем распоряжении чисто программистский метод – смоделировать ситуацию, программно, разумеется – где нам взять столько президентов? К тому же, все они какие-то одноразовые. Сейчас мы всё смоделируем и узнаем *Правду с большой буквы*. А потом мы сделаем выводы. А потом узнаем почему некоторые уже полученные мной ответы входят в категорию того, что кратко характеризуется русским народным фразеологизмом *Не пришей кобыле хвост*.

Но пока…

Интерлюдия или Coitus Interruptus

Ничего не могу с собой поделать, люблю красивые слова.

Но пока мне кажется, что сейчас самое удобное время, чтобы поговорить о важном.

Вызывают еврея в КГБ:
- Мы узнали, что у вас есть брат в Америке.
- Да, есть, но я не поддерживаю с ним никаких отношений.
- Вы не поняли, ваш брат видный деятель Коммунистической партии США. Напишите ему письмо!
«Дорогой Абрам, наконец-то я нашёл время и место тебе написать… »
© типичный советский антисоветский анекдот

Далее я смоделировал проблему трёх президентов программно, получил абсолютно странный результат, и потерял веру вообще во всё, а конкретно – в генератор случайных чисел. В программе я чуть позже нашёл глупую ошибку, точнее, даже не в программе – с программой всё было в порядке. Вместо того, чтобы смоделировать вероятность смерти трех президентов в день независимости, я смоделировал вероятность смерти трех президентов в один день, любой. Это правильно, но только позже надо было поделить на 365. Я всегда готов признать свою ошибку, особенно, если мне за это ничего не будет.

Тем не менее, я, ненадолго, усомнился в качестве работы генератора случайных чисел, реализованного в Delphi. Справедливости ради, некоторые основания у меня для этого были.

Напомню, как организовано получение случайных чисел в Delphi, Turbo Pascal, Pascal ABC и многих, многих других средствах разработки. Каждый раз, когда нам требуется случайное число, мы обращаемся к функции Random. Но в самом начале, один раз, мы вызываем процедуру Randomize. Вот её реализация:

```
procedure Randomize;
var
  Counter: Int64;
begin
  if QueryPerformanceCounter(Counter) then
    RandSeed := Counter
  else
    RandSeed := GetTickCount;
end;
```

Функция Random почти проста и очевидна. Почти – потому что, когда мы пишем r:=Random(100), то получаем равномерно распределённое число в диапазоне 0..99. Все, почему-то, ожидают 1..100. Для этого надо написать r:=Random(100) +1. Если написать r:=Random, то нас ожидает сюрприз. Переменная слева немедленно должна стать плавающей и мы получаем случайное число в интервале (0,1). Исходные тексты этих функций от пользователей Delphi скрыты.

Вызов Randomize не то, чтобы гарантирует, но, по крайней мере, обещает нам, что так называемые случайные числа будут каждый раз другие – это самое скромное требование, предъявляемое к случайным числам. Мне так кажется. Усомнившись в генераторе, я тупо вставил вызов Randomize перед *каждым* вызовом Random. После этого генератор перестал работать вообще. Тут я усомнился во всём, но чуть позже нашел где я был неправ. Кстати, интерактивная справка: Delphi категорически не рекомендует вызывать в одном цикле Randomize и Random. Видимо они что-то подозревают.

Ещё есть функция RandG, но для её использования надо хорошо познакомиться с математическим ожиданием и дисперсией. Возможно, мы доберёмся до них в следующей главе.

Тем не менее, сейчас самое время и место разобраться, откуда в компьютере берутся случайные числа. Для начала – все говорят *Генератор случайных чисел (ГСЧ)* . Это неверно. То, к чему мы обращаемся в программе – *Генератор псевдослучайных чисел (ГПСЧ)*. Простая разница в том, что ГСЧ берёт своё случайное число откуда-то из реального внешнего

его мира, а, поскольку реальным миром правит бардак, числа получаются действительно случайными. ГПСЧ берёт из реального мира только начальное, стартовое число (seed), а каждое следующее вычисляет, исходя из предыдущего. Да и реальный мир в этом случае ограничен пределами компьютера, обычно для запуска генератора используется таймер.

Важный вопрос – для чего вообще нам нужны случайные числа? Девяносто девять процентов текстов, написанных на эту тему, предполагают, что нам нужно что-то зашифровать. Соответственно, где-то за углом сидят сотрудники ЦРУ, КГБ, АНБ, ФСБ, ФБР, далее по списку – кстати, а почему они все трёхбуквенные? Соответственно, максимум усилий прилагается к тому, чтобы никаким усилием мысли невозможно было бы предсказать следующее как бы случайное число в последовательности как бы случайных чисел. А если мы программируем игру в подкидного дурака, и нам только нужно выбрать, с какой карты ходить, если не с чего ходить? Известная карточная мудрость гласит *хода нет — ходи с бубей*. Иногда добавляют – *нет бубей, ходи с* червей. Примерно такому же ходу мысли следовал я, программируя в своей первой книге искусственный интеллект для игры крестики-нолики. Для крестиков-ноликов это было безразлично – игра элементарная, легко анализируемая и разлагаемая на молекулы. В сущности, ещё до первого хода кристально ясно, чем дело кончится. Подкидной дурак на этом фоне – как галактика по сравнению с глобусом.

Разумеется можно дать простой совет – используй стандартный Random. Можно и так, но всегда интересно попробовать что-то своё. Повторю в очередной раз – запрограммировав нечто, его надо протестировать. Более того – сначала надо программировать тестовую программу, а только потом уже нечто. Ещё конкретнее – сначала интерфейс нечта, потом тестовая программа, и, наконец, искомый долгожданный модуль. Интерфейс будет почти такой же, как и у дельфийских функций, только слегка поменяем имена, что бы не было конфликта. Вот пустой модуль:

```
unit SiRand;  //  10.06.2017
              //  11.06.2017
{-------------------------------------------------------------}
interface
{-------------------------------------------------------------}
procedure SiRandomize;
function SiRandom(     range : integer) : integer;
{-------------------------------------------------------------}
```

```
implementation
{-------------------------------------------------------------}
procedure SiRandomize;
begin
end;
{-------------------------------------------------------------}
function SiRandom(     range : integer) : integer;
begin
end;
{-------------------------------------------------------------}
end.
```

Теперь о том, как мы это будем тестировать. Существует масса критериев оценки качества работы ГСЧ. Нам оно надо? У нас самый простой генератор для самого простого подкидного дурака. Генерировать он будет числа от 1 до 36 – как легко догадаться, по числу карт в колоде.

Тест номер раз – просто посмотреть на несколько сгенерированных чисел, например, на те же самые 36, и визуально и интуитивно оценить, насколько они случайные. Это просто.

Тест номер два. Получить, к примеру, 36 миллионов случайных чисел в том же диапазоне 1-36 и оценить равномерность распределения. Получить результат просто. Оценить равномерность распределения гораздо сложнее, потому что мы этого ещё не проходили.

Тест номер три. Любой генератор псевдослучайных чисел неизбежно, рано или поздно, просто по своей подлой сущности, начинает повторяться. То есть, он выдаёт что-то типа 3, 17, 25, 39, 3,17,25,39, 3, 17. Ну, вы поняли. Хорошие, правильные генераторы, начинают повторяться через сто пятьсот миллионов раз, не меньше. Интересно поглядеть, когда зациклится наш. Идея понятна, но в процессе программирования придётся немного задуматься.

Первый, примитивный, интуитивный тест:

```
   const
      num = 36;
   var
      T                          : TextFile;
      rand                       : integer;
      i                          : integer;
begin
     AssignFile( T, 'rand36.txt');
     ReWrite( T);
```

```
     siRandomize;
     for i:=1 to num do begin
        rand:=siRandom(num) + 1;
        WriteLn( T, rand:3);
     end;

     CloseFile(T);
```

Комментарий - записывать данные в текстовый файл гораздо лучше, чем не записывать данные в текстовый файл, а тупо считывать их с экрана.

Второй тест, на равномерное распределение немного сложнее:

```
   const
      num   =      36;
      skoka = 1000000;
   var
      T                          : TextFile;
      rand                       : integer;
      arand                      : array[1..num] of integer;
      nom,min,max                : integer;
      rasbr                      : single;
      average                    : double;
      i,k                        : integer;
begin
     AssignFile( T, 'rand36x000000.txt');
     ReWrite( T);

     FillChar( arand, SizeOf(arand), #0);
     siRandomize;
     average:=0;

     for i:=1 to num*skoka do begin
        rand:=siRandom(num) + 1;
        average:=average + rand;
        arand[rand]:=arand[rand] + 1;
     end;
     average:=average/(num/skoka);

     for k:=1 to num do begin
        Writeln( T, IntToStr(arand[k]));
     end;

     nom:=skoka;
     min:=arand[1];  max:=arand[1];
     for i:=1 to num do begin
        if arand[i] < min then min:=arand[i];
        if arand[i] > max then max:=arand[i];
     end;
```

```
rasbr:=((max-min)/nom) * 100;

Writeln( T, 'nom = ', nom:7);
Writeln( T, 'min = ', min:7);
Writeln( T, 'max = ', max:7);
Writeln( T, 'rasbr = ', rasbr:7:2, '%');
Writeln( T, 'average = ', average:7:2);

CloseFile(T);
```

Мы считаем сколько раз выпало каждое из наших чисел, или каждая из наших карт – как кому ближе и больше нравится. Кроме того мы считаем в процентах отличие максимального числа выпадений карты от минимального. Честно говоря, это вообще ни о чём не говорит, лучше бы мы посчитали дисперсию, но о дисперсии позже. И ещё мы считаем среднее. Чему равно среднее число от единицы до тридцати шести? Если вы ответите *восемнадцать*, то вы глубоко неправы. Подумайте ещё раз.

Теперь по существу. Как устроен самый простой генератор псевдослучайных чисел? Я этим никогда практически не интересовался, но некоторое интуитивно представление у меня было. Когда я решил, наконец, это проверить, так оно всё и оказалось. Это потому, что я самый умный. Ну и самый скромный. И ещё, я никогда не вру. Точнее, те алгоритмы простейшего генератора, что я нашёл, работали, но как-то не очень убедительно, пришлось дорабатывать интуитивным способом, то есть – напильником.

Алгоритм простейшего генератора такой – берём число, вообще говоря, не предъявляя к нему особых требований, в нашем простейшем случае, и для наглядности, пусть это будет ноль. Желая получить некоторое случайное число, мы прибавляем к нему некоторое желательно большое простое число, умножаем результат на, опять-таки, желательно большое простое число. Затем мы берем от всего этого остаток о деления по модулю на очень большое простое число – это чтобы не было переполнения. И только затем получаем наше долгожданное случайное число. Если мы написали r:=Random(100), то берём остаток по модулю 100. Работает ли это? Врать не буду, работает, но очень, очень плохо. При всём при том, что это общеизвестная официально рекомендуемая версия получения из воздуха случайных чисел. На этом и закончим.

Моделирование, выводы и поиск виноватых

Теперь, обогащённые интересными, расширяющими наш кругозор, но в целом бесполезными знаниями, приступим к моделированию. Сама идея проста до идиотизма – перебираем в цикле 39 президентов, случайно выбираем для каждого день смерти и записываем в массив. Выясняем есть ли три совпадающих даты. Повторяем всё это во внешнем цикле много-много раз, считаем сколько раз имело место совпадение. После завершения внешнего цикла выводим результат. Не забыть поделить на 365, ведь мы посчитали только вероятность того, что три дня смерти совпали, а теперь надо внести поправку на день независимости.

Это было очевидное, над чем думать практически не надо. Теперь о том, о чём надо хоть немного, но подумать. Как мы будем выбирать, запоминать и потом с ним работать? Очевидное решение - номер дня и номер месяца, оба целые. Ведь у нас конкретная дата – во-первых четвёртое, во-вторых июля. После недолгого размышления видны незначительные, но ощутимые трудности, вызванные таким подходом. Проще задавать день его порядковым номером в году – от 1 до 365. Даже если мы бы пошли другим путём, непосредственно ориентируясь на американский народный праздник имени одноимённого дебильного кинофильма, – дебильного и с программистской точки зрения, в том числе, мы бы просто задали константу 185. Это номер для четвёртого июля, если год не високосный.
Почему фильм *День независимости* с точки зрения программиста дебильный? Потому что в фильме присутствуют программисты. И даже *чем-то* занимаются. Лично я фильм не смотрел, мне рассказали. Что характерно, это зрелище вполне коррелирует с уровнем отечественной кинопродукции той же эпохи, я про компьютеры.

Имеем заготовку программы:

```
const
  numOfPres =  39;
  daysYear  = 365;
type
  TPresDeadArray = array [1..numOfPres] of integer;
var
  PD                      : TPresDeadArray;
  presNumber              : integer;
  allThreeAreDead         : boolean;
  sortOk                  : boolean;
  tmp                     : integer;
  skoka                   : integer;
```

```
      numOf                    : integer;
      i,k                      : integer;
{.....................................................}
procedure ShowPD;
   var
      stroka                   : string;
      i                        : integer;
begin
     stroka:='';

     for i:=1 to numOfPres do begin
        stroka:=stroka + IntToStr(PD[i]) + #13#10;
     end;

     ShowMessage(stroka);
end;
{.....................................................}
begin
     Randomize;

     skoka:=10000;
     numOf:=0;

     for i:=1 to skoka do begin
        for presNumber:=1 to numOfPres do begin
           PD[presNumber]:=Random( daysYear) + 1;
        end;

        //  Здесь мы ищем три совпадающие даты

        if allThreeAreDead
           then numOf:=numOf + 1;
     end;

     ShowMessage( 'numOf = ' + IntToStr(numOf) + #13#10 +
                  'skoka = ' + IntToStr(skoka));
     ShowMessage( 'result = '+ FloatToStrF( ((numOf/skoka)*100),
                  fFixed, 8,4) + '%');
     ShowMessage( 'exact result = ' +
                  FloatToStrF((((numOf/skoka)/100)/365)),
                  ffFixed, 8,4) +'%');
end;
```

Избыточно-подробный вывод имеет целью увидеть не только финальный результат, но и промежуточные совпадения и вероятности. Небольшая процедурка ShowPD предназначена для отладки и анализа данных в процессе. Вызовы этой процедуры я уже удалил, но они были. Подумайте, куда бы вы их воткнули.

Теперь немного подумаем, как проще найти совпадающие даты. Первая пришедшая мне мысль – написать три вложенных цикла. Первый перебирает все элементы массива, второй цикл от первого индекса до конца массива. При совпадении двух дат запускается третий цикл – теперь уже от второго индекса опять-таки до конца массива – и ищет ещё одно совпадение. Но эта конструкция мне как-то сразу не понравилась. Во-первых, надо, хотя и немного, но думать головой. Во-вторых, всегда лучше вместо одной умеренно сложной задачи получить две простых.

Моё предложение такое – отсортировать даты по возрастанию, что, впрочем, непринципиально. Затем без затей пробежаться по массиву в поиске трёх подряд стоящих одинаковых элементов.

Сортировка, разумеется, пузырьковая. Элементов массива 39, забудьте об эффективности. Даже если нам придётся в цикле повторять это 100000 раз – а нам придётся.

```
repeat
   sortOk:=true;
   for k:=1 to numOfPres-1 do begin
      if PD[k] > PD[k+1] then begin
         tmp:=PD[k];
         PD[k]:=PD[k+1];
         PD[k+1]:=tmp;
         sortOk:=false;
      end;
   end;
until sortOk;
```

Ну что сказать? Сортировка, как сортировка. Поиск повторов так же незатейлив:

```
allThreeAreDead:=false;
for k:=1 to numOfPres-2 do begin
   if (PD[k] = PD[k+1]) and (PD[k+1]=PD[k+2]) then begin
      allThreeAreDead:=true;
      Break;
   end;
end;
```

Разумеется, мне можно справедливо поставить в упрёк, что я пренебрёг счастливой возможностью задать и число удачно померших в Светлый Денёчек президентов как константу. Где-то раньше я написал, что

единственно допустимые в программе числа, не заданные константами – это 0 и 1. Ладно, был не прав, исправьте.

Теперь, наконец, будем пользоваться. Что мы, собственно, хотим узнать? Теоретически мы имеем три ответа:

0.12%	на пальцах номер раз
0.003%	на пальцах номер два
0.016	Хабр
0.018%	теория

Сколько раз надо повторить наш опыт, чтобы получить результат, которому можно доверять? И что значит доверять? Наверное, можно доверять, если при нескольких запусках получатся сходные результаты. Осталось уточнить термины. *Несколько* - это три. *Сходные* – считаем термин интуитивно понятным. Разумеется, мы могли бы рассчитать дисперсию величины, чтобы оценить заранее требуемое число опытов, но при нынешнем быстродействии компьютеров проще подобрать опытным путём.

Начнём с 1000. Интуитивно ясно, что этого недостаточно. Теоретически предсказуемый ответ для первого случая 0.12% от 1000 ≈ 1 успех, и это самый большой ответ, остальные варианты практический ноль – совпадение трёх мёртвых президентов и дня независимости невероятно. То есть, ничего мы не увидим. Вообще. Запускаем три раза и получаем:

0.018%
0.014%
0.013%

С одной стороны, разброс значений слишком велик, и видно, что тысячи испытаний недостаточно. Это было понятно с самого начала. Непонятно, откуда такие мелкие значения вообще взялись, ведь мы должны были получить только ноль или единицу? Немного поразмыслив, я вспомнил, что случайным образом мы моделируем число трёх смертей в один день, и всё. А это число даже при 1000 попыток гораздо больше нуля. Вот те же данные в расширенном составе – процент и количество совпадений:

0.018% 64

0.014% 54
0.013% 46

Числа очень даже ощутимые, можно успокоиться и расслабиться. Увеличим количество испытаний, чтобы два раза не вставать, сразу до 100000. Результаты:

0.0171% 6226
0.0168% 6138
0.0169% 6184

Я на этом не успокоился, задал 1000000 и провёл три замера. Результаты совпали: 0.0171%.

Выводы. Правильный результат дала формула из учебника. Поймите меня правильно, я нисколько не сомневался в правильности формулы из учебника. Я сомневался, правильно ли я её, формулу, понимаю и применяю. Стало быть, правильно. Теперь осталось разобраться, что не так в первыми двумя попытками, с теми, которые на пальцах и, вроде бы, всё интуитивно очевидно. Чтобы всё-таки разобраться, сильно упростим задачу – 39 мёртвых президентов – это перебор, в голове не помещается. Пусть президентов только 4 (четыре), а нас интересует вероятность того, что 2 (два) умерли в пятницу.

Сначала применим логику из первого подхода. Вероятность умереть конкретно в пятницу $\frac{1}{7} \approx 0.149$. Вероятность для двух клиентов скончаться в пятницу $\left(\frac{1}{7}\right)^2 \approx 0.0204 \approx 2.04\%$. Теперь смоделируем программно и пожалеем, что мы – ладно, я – схалтурил на тупой проверке совпадений и не написал их в цикле. Кроме того, моё чувство прекрасного оскорбляет имя переменной allThreeAreDead – ведь теперь не все три померли, а только два. Мораль – когда пишете программу, помните, что переписывать её обязательно придётся.

Программное моделирование даёт 10.05%. Расчет по формуле из учебника – 9.00%. Напоминаю, что два нуля в конце записи числа отбросить нельзя – они обозначают нашу уверенность в том, что после девятки действительно

нули. Если мы их откусим, это будет значить, что кроме девятки мы вообще ни в чём не уверены. А теперь о главном. Что всё это значит? То, что первый грубый метод дал неверный ответ, не очень расстраивает, мы уже знаем, что он неверен. Расстраивает, что учебник и модель разошлись – ведь они так хорошо совпадали!

А как дысал, как дысал… © грустный анекдот

Проблема в ошибке. В очевидной, глупой ошибке. Та самая волшебная формула дает нам вероятность не для полного случая – трёх совпадений из 39, а для упрощенного случая – двух из четырёх. Если, в первом случае, совпали четыре президента, это вероятность не учитывается. Если нам нужна вероятность того, что совпали три или четыре – надо посчитать по формуле отдельно для трех и для четырёх и сложить. Потому что это *взаимоисключающие* события. Считаем для упрощенного варианта вероятности:

ровно 2 из 4 9.00%
ровно 3 из 4 1.00%
ровно 4 из 4 0.05%

Складываем всё и получаем то, что и должны были получить с самого начала. А почему всё отлично сработало в первом случае? Считаем и пересчитываем. Найдите ошибку. Или ошибки?

А теперь - Главный Вывод о вероятности Заговора;

Это ЖЖЖ – неспроста © Вини-Пух. Был заговор, был!

Ещё в копилку заговоров. Как вы много раз слышали, американцы на Луне не были, чего там делать-то. И американцам такое мнение обидно. И недавно кто-то очень американский применил теорию вероятностей к вычислению вероятности сокрытия Лунной Аферы. По его подсчётам, в Лунной Миссии™ участвовало 410 тысяч сотрудников. Оцените правдоподобие числа. Далее автор даёт оценку вероятности, что в течение года кто-то один проболтается – дайте вашу оценку. Посчитайте результат.

Автор каким-то образом получает результат, что продержаться это безобразие могло максимум десять лет. Пересчитайте. Перечитайте Свод

законов Соединённых Штатов Америки – как сейчас принято говорить, СГА – Раздел 18. Оцените сроки за разглашение государственной тайны. Скорректируйте ответ.

Ещё упражнения.
Болконский, Потрошитель и все-все-все

Эти примеры могут показаться несерьёзными – в лучшем случае, или подозрительными – в худшем. В любом случае – они безусловно полезны, потому что учат главному. А самое главное отнюдь не применение формул и тем более не вычисления по ним. Главное – извлечение фактов в математическом смысле из бесформенного и загадочного потока событий, да ещё и данных в чужом пересказе. Ошибка в фактах губит всё. Действует принцип правильной программы – если на входе мусор, значит и на выходе мусор, и не важно, правильна ли программа.

В вычислениях ошибаются только двоечники, в применении формул – троечники. В отборе существенных фактов ошибаются все. Даже, к примеру, профессор Китайгородский.
Верить в наше время нельзя никому, даже себе. Мне – можно
© Папаша Мюллер

Вот здесь профессор анализирует с точки зрения математики Льва Николаевича Толстого:
«Но ведь и в шедеврах литературы случайности играют важную роль», – скажет читатель. Несомненно. Но это случайности, которые могут произойти; события, вероятность которых вполне значима. Скажем, у Л. Толстого раненый Болконский оказывается в хирургической палате рядом с Курагиным. Толстому нужна была эта встреча, чтобы показать душевный перелом князя Андрея. Вероятно ли это событие? Без сомнения. Офицерских палат вблизи поля боя было немного, а может быть, даже и одна. Вероятность очутиться в одной палате двум офицерам, грубо говоря, равняется вероятности быть раненными в один день. Если раненых офицеров в этот день был один процент, то вероятность попасть в один процент для каждого из них равняется 0,01, а обоих сразу – 0,0001, вполне разумное число, с которым надо считаться.
© А.И.Китайгородский «Невероятно – не факт».

Есть ли возражения к методу? Профессор сначала оценивает вероятность того, что Болконский и Курагин попали *в одно* военно-лечебное заведение (как равную 100%), а затем вероятность того, что один из них вообще туда попал (как 1%). К методу возражений нет, разве что логичнее было бы задавать вопросы в обратном порядке. Тем более нет вопросов к использованным формулам и точности расчетов.

К пуговицам вопросов нет. Пришиты насмерть, не оторвёшь
© Аркадий Райкин

К методу и формулам вопросов нет, вопросы есть к фактам. Берём в жадные ручонки первоисточники сведений. Сначала, ясен пень, самый длинный роман в литературе. На самом деле роман только десятый по длине, но первые девять написаны иероглифами, это не считается.

Сначала перечитываем эпизод из романа. Много думаем.

Князя Андрея, как полкового командира, шагая через неперевязанных раненых, пронесли ближе к одной из палаток и остановились, ожидая приказания. Князь Андрей открыл глаза и долго не мог понять того, что делалось вокруг него. Луг, полынь, пашня, черный крутящийся мячик и его страстный порыв любви к жизни вспомнились ему. В двух шагах от него, громко говоря и обращая на себя общее внимание, стоял, опершись на сук и с обвязанной головой, высокий, красивый, черноволосый унтер-офицер.
……
Ну, сейчас, - сказал он на слова фельдшера, указывавшего ему на князя Андрея, и велел нести его в палатку. В толпе ожидавших раненых поднялся ропот. - Видно, и на том свете господам одним жить, - проговорил один.
© Том третий

Складывается чёткое, но однозначное впечатление, что и нижние чины, и офицеры попадали в один и тот же госпиталь и никакого особенного офицерского госпиталя не было. Что логично – странная привилегия – тащить офицера через всё поле сражения в какой-то далёкий загадочный офицерский госпиталь.

Впрочем, гениальный теоретик литературы Шкловский давным-давно доказал, что все знания о войне двенадцатого года Лев Николаевич

почерпнул, прочитав не то десять, не то пятнадцать книжек, причём, прочитав, немедленно вернул их книгопродавцу, со скидкой, разумеется. Потому что очень деньги нужны были. Так что классик мог и ошибаться. Поглядим, что пишут умные люди в электрических интернетах.

Толстой называет место, куда принесли князя Болконского *перевязочный пункт,* однако путается в терминологии. Все сходятся на том, что, после оказания первичной помощи в полку, раненого доставляли в *развозной госпиталь*, а оттуда в *подвижной госпиталь*, что уже несущественно. Перевязочный пункт – это полк Болконского, и никакого Курагина там не могло быть просто по определению. И доктор там ровно один. У Толстого описан именно что развозной госпиталь. Вопрос в том, сколько их было? Есть только информация, что перед Смоленским сражением их было три. Меньше их стать никак не могло, так что оставляем число три.

Какова вероятность для двух раненых попасть в один госпиталь? Правильно – 1/3. А почему не 1/9? А потому что 1/9 это вероятность для двух раненых попасть в один *заранее* выбранный госпиталь, например, первый. В результате видим, что по первому пункту профессор ошибся, но не сильно, даже не на порядок.

Займёмся вторым пунктом – какова для офицера вероятность быть раненым в Бородинском сражении? По хорошему сначала надо бы узнать число офицеров, участвовавших в сражении и число раненых офицеров. Но поскольку мы занимаемся расчетами исключительно для личного удовольствия, предположим для простоты, что процент раненых офицеров совпадает с процентов раненых военнослужащих вообще. Лев Николаевич с нами согласен:

Все их набегания и наскакивания друг на друга почти не производили им вреда, а вред, смерть и увечья наносили ядра и пули, летавшие везде по тому пространству, по которому метались эти люди. © Том третий. Ядрам всё равно, в кого лететь.

Сначала находим число участников сражения с русской стороны – 103 тысячи регулярных войск и 7 тысяч казаков, всего 110 тысяч. Нули на конце намекают на точность величины.

Далее берём старую, но авторитетную книгу

В.Ц. Урланис "История военных потерь", СПб, Полигон, 1994

Ранено – 19226, но, возможно, учтены не все. Поскольку число участников у нас с точностью до тысяч, то и число раненых разумно будет взять для расчета с точностью до тысяч, чтобы не создавать иллюзии множества верных десятичных знаков. Таким образом, вероятность (для офицера) быть раненым 19/110 $\approx$ 0.17. Вероятность быть ранеными для двух конкретных офицеров 0.17 $*$ 0.17 $\approx$ 0.03. Общая вероятность – быть ранеными и попасть в один госпиталь –

$$P_{\text{Болконский/Курагин}} = 0.33 * 0.03 \approx 0.01$$

Применив абсолютно верную методику к абсолютно неверным исходным данным, уважаемый профессор ошибся в сто раз. Впрочем, профессору уже всё равно, по понятным причинам. Обратите внимание, что первая ошибка отчасти компенсирует вторую – очень нередкое явление.

Напоследок ещё одно не очень традиционное упражнение на освоение базовых понятий теории вероятностей. Задача нечёткая – не только входные данные нечёткие, но и поставленный вопрос расплывчатый. Приступим.

Все слышали о Джеке Потрошителе. Это такой маньяк, орудовавший в лондонском районе Уайтчепел в 1885-м году. Он убил не менее пяти проституток, что по нынешним меркам убого, но его считают первым в мире серийным убийцей. Это неверно, но мы сейчас не об этом. Подозреваемых десятки, включая наследника английского престола и Льюиса Кэрролла. Основных подозреваемых четверо – английский джентльмен и три польских еврея. Польша на тот момент была не страной, а географическим понятием, так что технически один еврей приехал из Австро-Венгрии, а два из России.

Знаменитый профилёр из ФБР Джон Дуглас, послуживший прототипом героя *Молчания ягнят*, не Ганнибала Лектора, конечно, но тоже неплохо, считал наиболее вероятным подозреваемым Северина Клозовского. Почему у польского еврея имя и фамилия польского поляка, я не знаю. Главным подозреваемым его сделало не то, что он жил в Уайтчепеле, и не

то, что он изучал медицину в Варшавском университете, а его дальнейшие свершения на жизненном пути.

После того, как убийства Джека Потрошителя прекратились, Клозовский уехал в Америку. Там он начал жениться. Женившись, он убивал жену, получал наследство и страховку и женился снова. Всего он убил то ли две, то ли три жены, одну тёщу, и сколько-то жён пытался убить, но как-то недоубил. В конце концов его таки повесили.

А теперь возвращаемся к теории вероятностей. Вторую убитую Джеком проститутку звала *Анна Чапмен*. Одну из жён Северина Клозовского тоже звали *Анна Чапмен*. Вы должны

Не мы, а вы! © Операция Ы

оценить правдоподобие двух гипотез.

Первая гипотеза – совпадение чисто случайное. Оцените вероятность. Соберите информацию о распространённости в те времена имени Анна, это несложно. Соберите информацию о распространённости фамилии Чапмен, это посложнее. Посчитайте вероятность.

Вторая гипотеза – это был такой английский чёрный юмор, проявленный Северином Клозовским. Тут, понятное дело, ничего считать не надо, надо посмотреть на вероятность первой гипотезы и сделать вывод. И это будет исключительно *ваш* вывод. Кстати, после брака он взял фамилию жены, и повесили его уже как Северина Чапмена.

И ещё. Вспомните самую обаятельную и привлекательную шпионку всех времён и народов по имени – сюрприз! – *Анна Чапмен*. Посчитайте вероятность гипотезы. Какой? Даже и не знаю. Например, того, что шпионка с таким кармическим погонялом окажется на грани провала. Или вероятность гипотезы, что это был такой тонкий русский троллинг, который никто не оценил. Кроме меня.

Съездил мужик в Японию, рассказывает:
- Вот это жизнь! Саке! Гейши!! Харакири!!! А у нас только водка, б@№;%и и поножовщина... © народное

К чему это я? Вот отечественный вариант. Взято отсюда: *http://lost-kritik.livejournal.com*.

С такой фамилией скрываться, все равно что явку с повинной написать

Кулак из дер. Битак Евпатор[ийского] р-на КИДЕНКО Герасим Петрович в июне м-це 1931 г. бежал из Урала в Крым. По приезде в Крым КИДЕНКО занялся спекуляцией. При продаже КИДЕНКО смушек подошел неизвестный, который купил последние, но в связи с отсутствием при себе полностью необходимой суммы направился для получения последней вместе с КИДЕНКО к своему знакомому. В процессе беседы неизвестный заявил, что он служит в одном учреждении и что он может достать КИДЕНКО необходимый документ за 50 рубл. Сделка была совершена, и на другой день КИДЕНКО получил удостоверение ПИДОРЦЕВА Василия, по каковому проживал до ареста.

Теория вероятностей и азартные игры.

Теперь обсудим несколько азартных игр. *Азартные* игры – те, в которых успех или неуспех зависят исключительно от игры случая, а никак не от умения игрока. Шахматы – игра *не азартная*, случай в них не участвует, если не рассматривать тот случай, что у претендента в чемпионы мира внезапно случился понос. Шашки и крестики-нолики относятся к той же категории. Между азартными и не азартными есть нейтральная полоса, где разлеглись игры *коммерческие*. То есть такие, где случай роль играет, но ещё и думать надо – *преферанс, бридж*. Туда же относится и *покер*, и любимый мною *храп*, что ужё вызывает лёгкие сомнения. С формальной точки зрения коммерческой игрой является и *очко*, что уже ни в какие ворота. Но что есть, то есть.

Важное понятие – *справедливая игра*, то есть такая, в которой ни у одного игрока нет преимущества. Под преимуществом подразумевается не то, что один из игроков умнее другого. Преимущество означает, что игроки неравноправны – сами правила игры дают преимущество одному из игроков. Такая игра не справедливая.

Фараон или Штос

Самая простая и чисто вероятностная игра – штос, он же фараон. Точнее, с процедурной точки зрения игра не самая простая, а с математической – да,

проще уже некуда, кости просто неимоверно сложнее. Была, в чистом, натуральном виде, популярна в восемнадцатом и девятнадцатом веках, особенно, что называется, в *пушкинскую эпоху*. Упоминается у Лермонтова, у Гоголя, у более некрупных писателей. Лев Николаевич не мог пройти мимо, в романе *Война и мир*.

Вот справедливое замечание профессора Китайгородского о методе графа Толстого:

В «Войне и мире» Долохов обыгрывает Ростова вполне планомерно. Долохов решил продолжать игру до тех пор, пока запись за Ростовым не возрастёт до 43 тысяч. Число это было им выбрано потому, что 43 составляло сумму сложенных его годов с годами Сони.

Читатель верит, что смелый, резкий и решительный Долохов, которому удаётся все, хорошо играет в карты. А мягкий, добрый, неопытный Ростов, кажется, не умеет играть и не может выиграть. Великолепная сцена заставляет нас верить, что результат карточной борьбы предопределён.

И Александр Сергеевич, тоже о том же, само собой, часто в философски-метафорическом стиле:

А перед ним воображенье свой пёстрый мечет фараон
© Евгений Онегин

Но чаще в совершенно реальном – самый известный пример – *Пиковая дама*. Ближе к финишу Германну с двумя буквами *Н* является неживая графиня и объявляет:

- Тройка, семерка и туз выиграют тебе сряду, но с тем, чтобы ты в сутки более одной карты не ставил и чтоб во всю жизнь уже после не играл.

Кончается она сценой игры, игра описана так:

Чекалинский стал метать, руки его тряслись. Направо легла дама, налево туз.
— Туз выиграл! — сказал Германн и открыл свою карту.
— Дама ваша убита, — сказал ласково Чекалинский.

Германн вздрогнул: в самом деле, вместо туза у него стояла пиковая дама. Он не верил своим глазам, не понимая, как мог он обдернуться.

Что всё это означает, кроме того, что Германн с двумя *Н* проигрался и сошёл с ума? И что он нарушил правило – не ставить более одной карты в сутки? В чём смысл игры? Смысл прост как огурец, или, как говорили в советское время – *простой, как батон за тринадцать копеек*.

Есть *банкомёт*, у него колода карт. Есть *понтировщик*, у него колода карт. Являются ли слова *понтировщик* и *понты* однокоренными, я не уверен. Понтировщик выбирает из своей колоды одну карту и кладёт её перед собой. Банкомёт начинает раскладывать свою колоду налево и направо от себя. Если карта, выбранная понтировщиком, легла налево – понтировщик выиграл, если направо – понтировщик проиграл. Просто до идиотизма, за это его, фараон, штос, и любили. За А.С.Пушкина после смерти карточные долги заплатил Государь Император Николай Первый.

Игра *справедливая*, ни одна из сторон не имеет преимущества, в отличии от, например, рулетки. Вероятность выигрыша ½. На этом теория игры исчерпана.

Однако, что происходит в повести Пушкина? Германн ставит на кон все свои деньги – сорок семь тысяч рублей. Деньги бешеные, по тем временам, хотя не ясно – серебром или ассигнациями, разница большая. Александр Сергеевич Пушкин экономистом был самым хреновым. Это не я – это мнение современников. Ставит он их на тройку. Он – в смысле Германн. Выигрывает. Ставит на семёрку – выигрывает. Ставит на туз – выиграл бы, но по ошибке поставил на даму. Как пишут в этих ваших интернетах – *Пр@#$%ли все полимеры*! Кстати, имение Михайловское приносило поэту в среднем 5000 рублей в год, а за все издания Онегина он, по разным расчётам, получил 10-15 тысяч.

А какова вероятность выигрыша на тройку, семерку и туза? Точно такая же, как и на любые три заранее выбранные карты. ½×½×½ = 1/8. Не так уж и мало. Это всё, что можно сказать о фараоне, он же штос.

Моя «Пиковая дама» в большой моде. Игроки понтируют на тройку, семерку и туза © А.С.Пушкин

Кости

Тоже простая игра, но гораздо сложнее, чем штос – кости. Игральная кость – куб с шестью сторонами, промаркированными цифрами от единицы до шести. Есть много разных игр с разным количеством костей, но если в простейшем варианте – два игрока, каждый бросает одновременно две игральные кости. Кто выбросил в сумме больше очков – на верхней грани костей – тот и выиграл. Игра *справедливая*.

Бросая две кости, как нетрудно заметить, можно выбросить от двух до двенадцати очков. Подумайте, все ли из результатов равновероятны. С первого взгляда кажется, что да. А если ещё раз подумать и нарисовать таблицу? По горизонтали вверху и по вертикали слева отложены выпавшие очки на одной и на другой кости. Порядок абсолютно неважен, в этом отношении игра симметрична. На пересечении – сумма выпавших очков.

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Теперь смастерим таблицу попроще – четыре столбца – количество очков, количество сочетаний, которыми они могут выпасть и вероятность – в натуральных дробях и в десятичных, кому что больше нравится.

Если просуммировать последний столбец, то получим в сумме 1.001, это нормально для чисел с плавающеё точкой. Обе таблицы абсолютно симметричны, потому что игральных костей две и они совершенно равноправны. Совершенно очевидно, что от двух очков до шести вероятность возрастает, далее убывает. Составьте таблицы для трёх костей. Подумайте.

Очки	Число сочетаний	Вероятность	Тоже вероятность
2	1	1/36	0.028
3	2	1/18	0.056
4	3	1/12	0.083
5	4	1/9	0.111
6	5	5/36	0.139
7	6	1/6	0.167
8	5	5/36	0.139
9	4	1/9	0.111
10	3	1/12	0.083
11	2	1/18	0.056
12	1	1/36	0.028

Теперь рассмотрим эпизод из повести Соловьёва *Возмутитель спокойствия (Повесть о Ходже Насреддине)*. Эпизод так же абсолютно симметричен, как и таблицы. Ходжа Насреддин наблюдает за игрой в кости:

Как везет, однако, этому рыжему игроку: он выигрывает четвертый раз подряд... Смотри, смотри - он в пятый раз выиграл! О безумец! Он обольщен ложным призраком богатства, между тем нищета уже вырыла яму на его пути. Что?... Он в шестой раз выиграл!.. Я никогда еще не видел, чтобы человеку так везло. Смотри, он ставит опять!

Рыжий игрок бросил кости и в седьмой раз выиграл.

Ходжа Насреддин ставит почти все свои деньги. Рыжий бросает кости:

На костях было всего три очка - верный проигрыш, ибо двойка выбрасывается так же редко, как и двенадцать, а все остальное годилось Ходже Насреддину.

Но Ходжа Насреддин выбрасывает два очка и проигрывает. Он играет на оставшиеся деньги – 25 таньга - и выигрывает семь раз, каждый раз удваивая ставку.

- Ты не можешь выигрывать подряд, если сам шайтан не помогает тебе! - вскричал рыжий.

Рыжий предлагает сыграть на все. Автор повести считает, что на кону теперь 1600 таньга. Прав ли автор?

Первым бросил кости рыжий и сразу зажмурился, - он боялся взглянуть.
- Одиннадцать! - закричали все хором. Ходжа Насреддин понял, что погиб: спасти его могли только двенадцать.
- Одиннадцать! Одиннадцать! - твердил в неистовой радости рыжий игрок. - Ты видишь - у меня одиннадцать! Ты проиграл! Ты проиграл!

Разумеется, Ходжа Насреддин выбрасывает двенадцать и выигрывает. Теперь будем задавать вопросы:

Какова вероятность семи выигрышей подряд? Или, что то же самое, семи проигрышей, игра симметрична. Ответ очевиден - $\left(\frac{1}{2}\right)^7 = \frac{1}{128} \approx 0.78\%$. Не так уж и мало. Удивляться и вспоминать шайтана рановато. Вероятность *шести* выигрышей подряд в два раза больше $\frac{1}{64} \approx 1.56\ \%$. Странно, что Ходжа Насреддин утверждает, что никогда такого не видел. Врёт, наверное.

Для проверки гипотезы о патологической лживости как Ходжи Насреддина, так и Рыжего, определим вероятность семи и шести выигрышей подряд при ста играх – не так уж и много, для людей которые играют или, хотя бы, наблюдают за игрой. Сначала теоретически, а потом смоделируем программно.

Сначала сделаю глубокомысленное замечание – нам всё равно, было ли это шесть выигрышей или шесть проигрышей. Ходжа Насреддин не говорит ведь, что это он шесть раз подряд не выигрывал ни разу, он говорит, что никогда такого не видел. А выигрыш одного, напомню, это проигрыш

другого. Непонятно, о чём я? $\left(\frac{1}{2}\right)^7$ - это вероятность семи подряд выигрышей конкретного, заранее выбранного игрока. Если играют рыжий и лысый, мы ставим, к примеру, на лысого, что он выиграет семь раз подряд – тогда вероятность рассчитана правильно. Но если после первой игры мы утверждаем, что следующие шесть закончатся с тем же победителем, то нам требуются только шесть совпадений. Формула слегка меняется - $\left(\frac{1}{2}\right)^{7-1} = \frac{1}{64} \approx 1.56\%$. Для шести повторений результата, соответственно - $\left(\frac{1}{2}\right)^{6-1} = \frac{1}{32} \approx 2.86\%$. Это, более-менее, понятно и даже очевидно.

Теперь внесём предположение – Ходжа Насреддин сидит в чайхане весь день – а чем ему, бездельнику, заняться – и наблюдает за игрой. Разумно предположить, что за день сыграют в кости, для удобного счёта, пусть 100 (сто) раз. Это означает, что возможностей получить 6 одинаковых результатов подряд у нас 100 – 6 + 1 = 95, то есть серии из шести могут начаться с 1-й игры по 95-ю. Для семи соответственно.

Далее используем стандартный метод. Сначала для шести повторений. Вероятность, что при 95 попытках событие с вероятностью 2.86% *не произойдет* равняется $(1-0.0286)^{95}$. Итоговая вероятность, что событие *произойдет*:

$$P_7 \approx 1-0.9714^{95} \approx 1-0.064 \approx 0.936 \approx 93.6\%.$$

Для семи повторений имеем где-то 77.2%. Результат явно показывает, что событие достаточно частое и почти заурядное. Формулы я взял правильные, но остаётся риск, что я их неправильно применил. Неплохо бы проверить опытным путём. Генератор случайных чисел у нас есть, так что вперёд.

Программное моделирование. Естественно, оно очень похоже на предыдущий вариант, с неживыми президентами. Приведу сразу весь код, а затем прокомментирую.

```
   const
      numOfGames = 100;
      numOfWins  =   7;
   type
      TResults = array[1..numOfGames] of boolean;
   var
      ress                    : TResults;
      numOf                   : integer;
      WinsOk                  : boolean;
      i,j,k                   : integer;
begin
     Randomize;
     skoka:=1000000;
     numOf:=0;

     for i:=1 to skoka do begin
        for k:=1 to numOfGames do begin
           ress[k]:=(Random(2)=1);
        end;

        for k:=1 to numOfGames-numOfWins+1 do begin
           winsOk:=true;
           for j:=k+1 to k+numOfWins-1 do begin
              winsOk:=winsOk and (ress[k]=ress[j]);
           end;

           if winsOk then Break;
        end;


        if winsOk then numOf:=numOf + 1;
     end;

     ShowMessage( 'numOf = ' + IntToStr(numOf) + #13#10 + 'skoka = ' +
                  IntToStr(skoka));
     ShowMessage( 'result = '+ FloatToStrF( ((numOf/skoka)*100),
                  ffFixed, 8,4) + '%');
end;
```

На что обратить внимание? С одной стороны, код проще – нет необходимости что-то сортировать, возможных результатов у нас всего два – чей-то выигрыш и чей-то проигрыш. Чей именно, нас не интересует – ведь Ходжа Насреддин утверждает, что никогда такого не видел *вообще*, он не говорит конкретно о себе, к примеру. Причём, напомню, говорит это уже при шести повторениях результата игры. С другой стороны, упокоившихся президентов было всего трое, и три проверки я счёл допустимым написать явно, тупо и в лоб. Для семи повторений это как-то уже говорит о профессиональных качествах программиста – в

отрицательном смысле. Будет время, поищите в Гугле по словам *индусский код*. В смысле, вот это он и есть. То есть мог бы быть, но его здесь не будет – заменяем циклом.

Число испытаний	Для семи повторений	Для шести повторений
1000	53.8%	81.7%
10000	54.7%	80.3%
100000	54.3%	80.8%
1000000	54.3%	80.6%
Теория	77.2%	93.6%

Что-то пошло не так. Практика нагло, мерзко и возмутительно отклоняется от теории. Придётся разбираться. Формулы, как всегда, вне подозрений. Подозрения вызывает наша способность эти формулы правильно применить. Программа вроде бы выглядит правильной, но, опять-таки, случаи разные бывают. Что делать, кому верить? Только себе, то есть посчитать самому, на пальцах. Поскольку посчитать на пальцах вероятность одинакового исхода шесть раз подряд для ста игр никак невозможно, резко упростим задачу.

Пусть игр будет только пять, а повторений потребуем три. Всего вариантов завершения для пяти игр будет только $2^5 = 32$, что вполне посильно нашему неокрепшему коллективному разуму. Рисуем таблицу возможных исходов. Жирная чёрная точка обозначает выигрыш рыжего игрока – извините, книга издаётся в чёрно-белом варианте, пустая клетка – отсутствие волос, выиграл лысый.

1	●	●	●	●	●
2	●	●	●	●	
3	●	●	●		●
4	●	●	●		
5	●	●		●	●
6	●	●		●	
7	●	●			●
8	●	●			
9	●		●	●	●
10	●		●	●	
11	●		●		●
12	●		●		
13	●			●	●
14	●			●	
15	●				●
16	●				
17		●	●	●	●
18		●	●	●	
19		●	●		●
20		●	●		
21		●		●	●
22		●		●	
23		●			●
24		●			
25			●	●	●
26			●	●	
27			●		●
28			●		
29				●	●
30				●	
31					●
32					

Начинаем думать, но сначала считаем по формулам и моделируем программно.

По формулам имеем:

$P_3 = 1-(1-(\frac{1}{2})^2)^3 = 1-(1-0.25)^3 = 1-0.75^3 \approx 58\%$. Моделирование уверенно демонстрирует нечто, уверенно стремящееся к 50%. Кто прав? Тупо и в лоб, на пальцах, посчитаем в скольких строках – сериях игр – встречаются три повторения. Это несложно, но если заметить, что нижняя половина таблицы является зеркальным отражением верхней половины, то будет ещё проще – достаточно посчитать для первых шестнадцати строк и умножить на два. Вы, само собой, считать не будете, поэтому сразу выдаю ответ – 16. Те самые 50%. Похоже, программа права, а формулы ошибаются. Но так ведь не бывает?
А что у нас в формуле, собственно? И что эта формула нам обещает? При каких условиях она верна?

Поскольку мы не можем, или нам трудно, посчитать вероятность события, мы считаем вероятность того, что событие не произошло, а в качестве ответа берём дополнение до единицы. Вероятность трёх событий (в отрицательном смысле – не совпало) из трёх испытаний равна произведению трёх отдельных событий, при условии, что они независимы.

Первый вопрос – являются ли события независимыми? То есть, если случился выигрыш рыжего – жирная чёрная точка – меняется ли от этого вероятность его следующего выигрыша? Математическая, не бытовая интуиция говорит, что нет. Классика теории вероятностей хором подтверждают – *у рулетки памяти нет*! Если хотите, проверьте по таблице, это несколько трудоёмко, но результат тот же – события абсолютно независимы.

Что мы предполагаем далее – что в каждой серии из пяти игр три последовательно одинаковых результата могут начинаться только с трёх позиций – 1-й, 2-й и 3-й. Может быть, эти случаи не равновероятны? Проверяем – всех трёх вариантов встречается ровно по восемь раз. Но, может быть, эти случаи зависимы? То есть, если у нас есть три повторения с первой позиции, то вероятность трёх повторений со второй позиции как-то меняется, в бóльшую или мéньшую сторону. В нашем конкретном случае, нас не интересует повторение трёх повторений после первого повторения – вы не запутались? Но нас очень интересует, не зависит ли вероятность трёх выигрышей/проигрышей со второй позиции от того, что *не было* трёх выигрышей/проигрышей начиная с первой позиции. Сначала

показалось, что написал невнятную фигню, потом перечитал, подумал и успокоился – всё изложено правильно, чётко и ясно.

А когда совсем успокоился, посчитал по таблице. Искомая последовательность начинается с 1-й позиции 8 раз. После этого со 2-й позиции начинается такая же последовательность 4 раза, и 4 раза не начинается. Всё чисто.

Всё хорошо, события кристально независимы, формулы идеальны. Не складывается слово *СЧАСТЬЕ* из букв *А,Ж,П,О*. Придётся ещё немного подумать.

Еще десять тысяч вёдер воды, синьор, — и золотой ключик у нас в кармане! © кино про Буратино

Ну почему всё я, да я? Подумайте сами, своей собственной головой!

Кстати в тему, Мостселлер задаёт вопрос:

Сколько в среднем раз надо бросить кость до появления шестёрки?

Разбирается эта задача также в книге Секейя. Что удивительно, интуитивно ожидаемый ответ оказывается правильным.

Рулетка

С одной стороны, о рулетке написано неимоверно много. С другой стороны, написано как-то уныло – всё больше об азарте игрока, и совсем мало об игре. Гений А.С.Пушкина рулетку не зацепил – есть такое мнение, что рулетка появилась несколько позже его времён и классик в этом безобразии не участвовал. Зато очень даже имел участие другой классик – Ф.М.Достоевский, и отразил в своем творчестве, сначала проиграв всё, что было. Потом, говорят, целых десять лет не играл, пока не помер.

Если вы каким-то образом не в курсе, что такое рулетка, напомню. В физическом смысле рулетка – это колесо, на котором присутствуют 36 ячеек. Колесо крутится., туда запускают шарик. Шарик, в конце концов, оседает в одной из ячеек. Каждая ячейка имеет номер – это ясно, и цвет – чёрный или красный. Игрок ставит на номер, или на цвет, или на чёт-нечёт,

дальнейшие извращения неважны. Если выигрывает его номер, игрок получает свою ставку умноженную на 36, если выигрывает цвет или чёт-нечёт – ставку, умноженную на два.

В чём подвох? На колесе есть ещё одна ячейка, обозначенная цифрой ноль – зеро. Если выигрывает зеро, все деньги уходят к хозяину рулетки. Википедия утверждает, что жадные американцы добавили ещё один сектор – два нуля – с тем же эффектом. Википедия слишком хорошего мнения об американцах – из книги Мостселлера следует, что у американской рулетки целых *три* грабительских сектора!

Игра, что очевидно, несправедлива. Игрок всегда неправ. На каждой ставке он теряет в среднем $\frac{1}{37} \approx 2.70\%$.

Возвращаясь к Достоевскому, свой опыт Ф.М. отразил в гениальной повести или в гениальном романе *Игрок* (1866). Ради этого случая, я это знаменитое произведение наконец прочитал. Про Раскольникова интереснее. Вообще-то я хотел уже проанализировать Достоевского с точки зрения математики, но, неожиданно купил – за деньги – следующую книгу:

Н.М.Карпушина Любимые книги глазами математика, 2011

Короче, читал зря. Всё уже проанализировано до нас. Придётся заняться другими произведениями. Далее слегка сокращённый фрагмент из очень известного романа Оноре Бальзака *Шагреневая кожа*. С математической точки зрения говорить вообще не о чем, но в финале возникает интересный вопрос. Только задал этот вопрос не я, а Мостселлер в книге *50 задач*.

В конце октября 1829 года один молодой человек вошел в Пале-Руаяль, как раз к тому времени, когда открываются игорные дома, согласно закону, охраняющему права страсти, подлежащей обложению по самой своей сущности. Не колеблясь, он поднялся по лестнице притона.

…

Он прошел прямо к столу, остановился, не задумываясь, бросил на сукно золотую монету, и она покатилась на черное. Ход этот возбудил такой интерес, что старики ставки не сделали; однако итальянец с

фанатизмом страсти ухватился за увлекавшую его мысль и поставил все свое золото против ставки незнакомца.

…

- Красная; черная, пасс, - официальным тоном объявил банкомет.

Что-то вроде глухого хрипа вырвалось из груди итальянца, когда он увидел, как один за другим падают на сукно сложенные банковые билеты, которые ему бросал кассир. А молодой человек только тогда постиг свою гибель, когда лопаточка протянулась за его последним наполеондором.

…

Это был, конечно, последний его заряд, - сказал, улыбнувшись, крупье после минутного молчания и, держа золотую монету двумя пальцами, показал ее присутствующим.

- Разве это игрок? - вставил кассир. - Игрок разделил бы свои деньги на три ставки, чтобы увеличить шансы.

Запоминается последняя фраза © Штирлиц и кассир

Теперь вернёмся к последней фразе. Молодой человек поставил на чёрное, выиграло красное – он проиграл. Это понятно и математически неинтересно. Поставил он наполеондор – золотую монету в двадцать франков. Кассир – задним числом – советует молодому человеку разменять золото на двадцать серебряных монет по одному франку и ставить их в три приёма. Как математик и программист, я обязан требовать продолжения банкета, то есть рекурсии. Если три ставки – семь, семь и шесть – лучше одной ставки в двадцать франков, то и три ставки – три, два и два – лучше одной ставки в семь франков. Быстро доходим до вывода, что ставить надо двадцать раз по одному франку – если кассир прав, конечно.

Задача из Мостселлера:

37. Смелая игра и осторожная игра

Человеку, находящемуся в Лас-Вегасе, нужны 40 долларов, в то время как он располагает лишь 20 долларами. Он не хочет телефонировать жене о переводе денег и решает играть в рулетку (отрицательно относясь к этой игре) согласно одной из двух стратегий: либо поставить все свои 20 долларов на «чёт» и закончить игру сразу же, если он выиграет или проиграет, либо ставить на «чёт» по одному доллару до тех пор, пока он

не выиграет или не проиграет 20 долларов. Какая из этих двух стратегий лучше?

Хотя я очень рекомендовал прочитать упомянутую книгу своими собственными руками, в этом случае я перескажу ответ. Автор ответ даёт два раза – математически и на пальцах. Математически автор доказывает, что вероятность выигрыша выше, если играть одноразово, а не оттягивать свой конец. Причём разница заметна. При одноразовой игре шанс выигрыша 47.4%, при многоразовой – 11%. На пальцах, цитата – *интуитивное объяснение этого явления состоит в том, что быстрая игра сокращает время игры против казино, которая не является справедливой.*

Кстати, только сейчас заметил – книга Мостселлера *Пятьдесят занимательных вероятностных задач* была издана в 1974-м году тиражом 250000 экземпляров. Прописью – 250 тысяч, Карл!

Теперь произведение сильно проще и вообще малоизвестное. Впрочем, автор известен всем – Жюль Верн. Малоизвестный его роман называется *Матиас Шандор*. Верн совершенно честно и аккуратно передрал роман Дюма-отца *Граф Монтекристо*, и даже ему, Дюма-отцу, свой роман и посвятил.

- Этак и я сумею, ты Мурку сыграй!
© Место встречи изменить нельзя

Дюма-отцу было уже все равно, он помер. Далее цитата из романа Жюля Верна:

– Семнадцать раз?..
– Семнадцать!
– Да, да! Красное вышло семнадцать раз подряд!
– Возможно ли?
– Возможно или нет, но это так!
...
– Семнадцать раз!.. Семнадцать раз!..
– В рулетку или в "тридцать и сорок"?
– В "тридцать и сорок"!
– Этого не бывало целых пятнадцать лет!

– Пятнадцать лет, три месяца и две недели! – холодно заметил завсегдатай рулетки, принадлежавший к почтенной категории дотла разорившихся игроков. – И вот что любопытно, сударь, то же самое случилось в тысяча восемьсот шестьдесят седьмом году… в разгар лета. Мне ли не помнить этого дня!
Такие восклицания раздавались в холле и даже под колоннадой Клуба иностранцев в Монте-Карло вечером третьего октября, ровно через неделю после исчезновения Карпены с сеутской каторги.
…
Выигрыш «красного» при совершенно непонятном упрямстве игроков, по-видимому, многим принёс разорение, ибо сумма, полученная банком, была весьма значительна. "Около миллиона", – перешёптывались в толпе. Это объяснялось тем, что почти все игроки упорствовали в борьбе с неудачей, продолжая ставить на "чёрное".
…
– Эта проклятая серия обошлась нам более четырёхсот тысяч франков! – воскликнул старший из игроков.
– Точнее четыреста тринадцать тысяч! – поправил младший тоном кассира, подсчитывающего крупную сумму.

По памяти мне казалось, что речь идёт о рулетке, почему этот текст и попал в этот раздел. Оказалось, что казалось – хотя рулетка где-то рядом присутствует, играют они в *Тридцать и сорок*. В интернетах нашлось следующее описание:

«Тридцать и сорок» играется колодами по 52 карты. Эта игра допускает четыре вида ставок: на черный, красный, лицевой и оборотный. Карты учитываются в соответствии с их достоинством: туз стоит 1 очко, двойка – 2, дамы и короли – 10 очков.

Крупье распределяет карты на два ряда, переходя ко второму ряду, как только сумма очков в первом ряду превысит 30. Первый ряд – для ставок на красные масти - красный ряд, второй – для ставок на черные масти - черный ряд. Выигрывает ряд, в котором сумма очков наиболее близка к 30.

Если у двух рядов сумма очков одинакова, то общий итог оказывается нулевым, и игра переигрывается. Причем, в случаях, когда обе суммы

превышают 31, ставки теряют половину своей стоимости. Это и определяет доход заведения.

С первого взгляда игра топологически эквивалентна рулетке. Из описания непонятно, сколько стоит валет, путь тоже 10. Игра явно несправедливая. Отсюда первый вопрос – какая игра является более справедливой – или несправедливой – рулетка или *Тридцать и сорок*. Потерю игрока при одной игре в рулетку мы уже знаем, это было очень просто определить. Здесь, по крайней мере с первого взгляда, всё не так очевидно. Какова вероятность, что в обоих рядах суммы превысят 31?

Посчитать по всем теоремам теории вероятности это, разумеется, можно. Но сложно. По крайней мере, для меня. Поэтому опять применим компьютерное моделирование, иначе говоря – метод Монте-Карло. Вы уже поняли, что это вполне законный метод решения любой вероятностной задачи.

В последний момент я задумался – а действительно ли игра эквивалентна рулетке? В рулетке очевидно, что шансы на выигрыш красного и чёрного одинаковы, здесь такой очевидности нет. Заодно проверим и это. Эмулятор колоды содержится в приложении. Приступим. Сначала маленькая занудная функция для конвертации достоинства карты в очки:

```
function Price(     card : TCard) : integer;
begin
     result:=0;
     if card.rank = two   then result:=2  else
     if card.rank = three then result:=3  else
     if card.rank = four  then result:=4  else
     if card.rank = five  then result:=5  else
     if card.rank = six   then result:=6  else
     if card.rank = seven then result:=7  else
     if card.rank = eight then result:=8  else
     if card.rank = nine  then result:=9  else
     if card.rank = ten   then result:=10 else
     if card.rank = Jack  then result:=10 else
     if card.rank = Queen then result:=10 else
     if card.rank = King  then result:=10 else
     if card.rank = Ace   then result:=1;
end;
```

Далее основное тело:

```
   var
```

```
      D                          : TDeck;
      card                       : TCard;
      winRed,winBlack,winBank    : integer;
      redRow,blackRow            : integer;
      gameOver                   : boolean;
      i                          : integer;
begin
     winRed:=0;  winBlack:=0; winBank:=0;

     for i:=1 to 1000 do begin
        D:=TDeck.Create;
        gameOver:=false;
        redRow:=0;  blackRow:=0;

        repeat
           D.RandomCardWithoutReturn(card);
           if redRow   < 30  then redRow:=  redRow   + Price(card) else
           if blackRow < 30 then blackRow:=blackRow + Price(card)
           else gameOver:=true;
        until gameOver;

        if (redRow > 31) and (blackRow > 31) then winBank:= winBank+1
        else begin
           if redRow-30 < blackRow-30 then begin
              winRed:=winRed + 1;
           end else
           if redRow-30 > blackRow-30 then begin
              winBlack:=  winBlack + 1
           end
           else begin
              //  replay
           end;
        end;
        D.Free;
     end;

     ShowMessage('Red/Black/Bank = ' + IntToStr(winRed) + '/' +
                                       IntToStr(winBlack) + '/' +
                                       IntToStr(winBank) + '/');
```

Обратите внимание на тот факт, что создание и уничтожение объекта находится *внутри* основного цикла. Сначала я поместил это всё *снаружи* цикла и изумлялся тонким неожиданным эффектам в процессе выполнения программы. Как вам кажется, есть ли ещё ошибки в программе? Мне кажется, что нет. Это я спрашиваю к тому, что результат меня очень удивил. Вот он, результат, оформленный в таблицу.

Число испытаний	Красное	Чёрное	Банк	Ч-%	К-%	Б-%
1000	228	250	485	22.8%	25.0%	48.5%
10000	2296	2350	4887	23.0%	23.5%	48.9%
100000	23571	23216	48689	23.6%	23.2%	48.7%
1000000	233035	231899	489478	23.3%	23.2%	48.9%

Результат, мягко говоря, поражает. Шансы красного и чёрного можно считать равными, что естественно – если бы шансы красного были выше, кто поставил бы на чёрное? Но вот прибыль банка… Если при игре в рулетку выигрыш банка скромные 2.70%, то здесь имеем $\frac{48.9\%}{2} \approx 24.5\%$. Деление на два необходимо потому, что банк забирает не всю ставку, а только половину. Всё равно - грабёж среди бела дня!

Будучи в тупике, я полез в англоязычный интернет, там эта игра называлась чисто по-европейски *Trente et Quarante*, и, буквально во второй строке, отмечалось, что игра эта очень даже честная, игрокам уходит более 98% остальное – менее 2% - банку . Я удивился. Далее выяснилось, что играют сразу шестью колодами по 52 карты, что, мне кажется, не вносит ощутимых изменений. Есть ещё нюансы, столь же незначительные. Главное – банк выигрывает только в случае, если оба ряда *равны* 31, а это уже существенно. Меняем строку

```
if (redRow > 31) and (blackRow > 31) then winBank:= winBank+1
```

на

```
if (redRow = 31) and (blackRow = 31) then winBank:= winBank+1
```

Запускаем, последняя строка таблицы трансформируется:

Число испытаний	Красное	Чёрное	Банк	Ч-%	К-%	Б-%
1000000	445285	443332	18332	44.5%	44.3%	1.83%

Это очень хорошо совпадает с только что прочитанным в английской Википедии, про выигрыш банка меньше двух процентов. К сожалению,

далее авторы текста путаются в показаниях и утверждают, что банк в среднем выигрывает один раз из тридцати восьми. $\frac{1}{38} \approx 2.63\%$. Кто-то где-то неправ.

Теперь второй вопрос, точнее два вторых вопроса. Сначала простой вопрос - какова вероятность выпадения красного 17 раз подряд?

$$p_{17} = \frac{1}{2^{17}} = \frac{1}{131072} \approx 0.00000763 \approx 0.000763\%$$

И так было ясно, что вероятность не маленькая, а очень маленькая. Поэтому второй вопрос, тот же, что был задан Ходже Насреддину – насколько правдоподобно утверждение, что такого события не было пятнадцать лет?

Посчитайте сами, мне надоели эти однообразные механические вычисления – см. анекдот *Опять эти нелепые телодвижения*.

Вопрос номер три, чисто арифметический. Персонажи проиграли – вдвоём, но на одой руке - ≈413000. В те времена в моде была стратегия удвоения ставок – считалось, что при этой стратегии проиграть невозможно, если, конечно, хватит денег. В чём идея? Игрок ставит франк на красное, проигрывает, ставит два франка на красно, проигрывает, ставит четыре франка…И вдруг выигрывает! Теперь считаем, поставил игрок 1+2+4=7, а выиграл целых 8! Профит – целый франк, или 27 копеек серебром! Дела пошли на лад! Скорее всего, персонажи играли по этой схеме, поскольку утверждают, что их погубила серия. Можно ли узнать, с какой начальной ставки они начали, или Верн запутался в расчетах? Общий проигрыш за N игр $2^N - 1$, единицу игнорируем, персонажи считают в тысячах, та что для начальной ставки в один франк имеем $2^{17} = 131072$, для ставки два франка 262144, для трёх франков 393216, для четырёх выходим за пределы указанной суммы. Или играли они как-то не так, или автор обсчитался.

Задача из Мостселлера:

Браун всегда ставит один доллар на номер 13 в американской рулетке, вопреки совету своего благожелательного друга. Чтобы отучить Брауна

от игры в рулетку, этот друг спорит с ним на 20 долларов, утверждая, что Браун останется в проигрыше после 336 игр. Имеет ли смысл Брауну принять такое пари?

В американской рулетке 38 номеров. Последние три в пользу заведения.

Очко

Сначала, чисто на всякий случай, напомню правила. Играют малой колодой в 36 карт. Каждая карта оценивается во сколько-то очков. О шестерки до десятки – по номиналу, валет – два, дама – три, король – четыре, туз – одиннадцать. Если игрок набирает 21, то однозначно выигрывает. Если больше, однозначно проигрывает. Если меньше, выигрывает тот, у кого больше, ну вы поняли, да?

Вы видели в ютубе, как кубинские негры поют *Владимирский централ*? Я это слышал в натуре. Я люблю Кубу. Теперь о главном. Как поётся в песне – *не очко нас губит, а к одиннадцати туз*. Вопрос элементарный – какова вероятность поднять к одиннадцати туза? Несмотря на всю кажущуюся элементарность игры, вопрос совсем не элементарен.

Примем для рассмотрения традиционный вариант правил, как играли в моём детстве и что мы видим в фильме *Операция Ы* – к сожалению, вставить в текст видео пока невозможно. Это означает, что в нашей ситуации игрок берёт карты из *полной* колоды, у него набирается карт на одиннадцать очков. Вопрос тот же – какова вероятность поднять туза? Вопрос не так прост, поскольку вероятность зависит от того, каким образом набраны одиннадцать очков. 11 очков – это может быть один туз, тогда в колоде три туза, а может и не быть туз, тогда все четыре туза в колоде. Других карт для искомой суммы требуется от двух – семь, король, например – до пяти – четыре валета и дама. От числа взятых карт, как легко догадаться, зависит вероятность получения туза. Но, опять-таки, разные, формирующие одиннадцать, комбинации, выпадают совсем не с равной частотой. Короче, *Чума на оба ваши дома!* © Шекспир, вроде бы.
Используем метод Монте-Карло.

Применяем модифицированный объект – колода на тридцать шесть карт. Смастерите сами, если по другому не получится, разрешаю просто размножить и поменять константы и. главное, имя класса.

```
   var
      D                          : TDeck36;
      card                       : TCard;
      sum                        : integer;
      numEleven,numElevenAndAce: integer;
      gamover                    : boolean;
      i                          : integer;
{.....................................................}
function Price(     card : TCard) : integer;
begin
     result:=0;
     if card.rank = six   then result:=6  else
     if card.rank = seven then result:=7  else
     if card.rank = eight then result:=8  else
     if card.rank = nine  then result:=9  else
     if card.rank = ten   then result:=10 else
     if card.rank = Jack  then result:=2  else
     if card.rank = Queen then result:=3  else
     if card.rank = King  then result:=4  else
     if card.rank = Ace   then result:=11;
end;
{.....................................................}
begin
     numEleven:=0;
     numElevenAndAce:=0;

     for i:=1 to 1000000 do begin
        D:=TDeck36.Create;
        sum:=0;
        gamover:=false;

        repeat
           D.RandomCardWithoutReturn(card);
           sum:=sum + Price(card);
           if sum > 11 then begin
              Break;
           end else
           if sum = 11 then begin
              numEleven:=numEleven + 1;
              D.RandomCardWithoutReturn(card);
              if card.rank = Ace
                 then numElevenAndAce:=numelevenAndAce + 1;
              Break;
           end;
        until gamover;

        D.Free;
     end;

     ShowMessage( '  n11 = '      + IntToStr(numEleven) +
                  '  n11Ace = '   + IntToStr(numElevenAndAce) +
                  '  % = '        + FloatToStrF(
```

```
(numElevenAndAce/numEleven)*100, ffGeneral, 5,2));
```

Комментарии. Ценовая функция почти та же. Из цикла **repeat-u**ntil мы никогда не выйдем законным способом. Хорошо ли это? Плохо. Но цикл простой, а *один раз – не водолаз*. На выходе результат.

Число испытаний	Одиннадцать	К одиннадцати туз	Вероятность туза к 11	Вероятность ситуации в игре
1000000	234408	24981	10.66%	2.50%

Как видим, вероятность прихода туза к одиннадцати достаточно велика – при условии что одиннадцать *уже* есть. Вероятность того, что это вообще случится в игре намного ниже, но вполне существенная.

Здесь должен быть разбор эпизода из фильма *Чрезвычайное поручение*, 1965, про героя-революционера Камо, где он героически обыгрывает батьку Махно или кого-то вроде него в очко. Пересмотрите и проанализируйте математически. Попутно прочтите книгу *Грач птица весенняя*, о революционере Баумане, названным так в честь одноимённого высшего учебного заведения. Обдумайте эпизод игры в карты, он близко от начала.

Покер

В покер играют карточной колодой. Есть ещё какой-то позорный *покер на костях*, я в это не играл, но презираю. Обычно карт 52, но бывает и 54 – тогда в колоде есть ещё два джокера. Джокер это такая хитрая карта, которую игрок может объявить любой нормальной картой по своему желанию. Игрок получает пять карт. Карты могут образовать какую-то комбинацию, комбинаций много. Например *двойка* – две восьмёрки *или* два туза. *Две двойки* – две восьмёрки *и* два туза. *Тройка* – три восьмерки *или* три туза. Одни комбинации лучше других. Тройка лучше двух двоек, а две двойки лучше просто двойки. Выигрывает тот, у кого самая лучшая комбинация. Есть ещё и ещё дополнительные правила, но для нас они неважны.

Покер игра американская, потому и воспет в американской литературе. Профессор Китайгородский анализирует покер на фрагменте романа Джека Лондона *Время не ждёт*. Я, для разнообразия, использую

полусоветского писателя Александра Грина. Полу – потому что первую половину своего творческого пути он провёл при царизме, а вторую половину – при Советской власти. Оба государственных строя он очень не любил, и они отвечали ему тем же. Только царский режим Грина сажал и отправлял в ссылку, а советский бил самым больным способом – не давал денег. Грин был романтик – то есть пил очень много. Лучше всего у него получались романы – *Алые паруса, Золотая цепь, Бегущая по волнам*. Рассказы не так известны, я возьму для рассмотрения совсем который не очень – *Серый автомобиль*. Смысл первой половины – главный герой приходит в казино и играет в покер на бешенные деньги. Его противник – мулат, что добавляет экзотики в сюжет. Ставка – миллион долларов, это в 1925-м году! Мне столько не выпить. Далее следует высокохудожественная сцена:

Ронкур смотрел на меня взглядом птички, зрящей змею. Наступил момент открыть карты. Игроки, заключавшие пари, перестали дышать.
— Ну, — сказал я, смеясь, — Гриньо, выкладывайте ваше каре! Он перевернул карты, пристукнув кистью руки, так что туз отлетел в сторону. Но там их было еще три — каре из тузов, вот что было в его руках! Бешеный рев покрыл это движение
— Вы ошиблись, — сказал я, показывая свои пять с улыбающимся чертом и раскладывая их одна к одной. — Гриньо, нравится вам этот джентльмен?
Момент не поддается изображению. Я не слышал криков и воплей, так как наслаждался бесконечно выражением лица опешившего мулата.
— Ваша... — сказал он сквозь звуки, напоминающие вой. Затем он откинулся, глаза его закатились... он был в обмороке.

Переводим на человеческий язык. Мулат Гриньо выложил четыре туза и был уверен в выигрыше. Главный герой выкладывает четыре одинаковых карты – конкретно четыре семёрки, но это неважно – и джокера, что даёт пять семёрок и уходит с миллионом.

Вопрос номер один – какова вероятность такого расклада вообще? Вопрос номер два, на самом деле номер один, потому что он проще – какие были шансы проиграть у мулата Гриньо?

В процессе игры игрок может поменять несколько своих карт. Рассказчик бросает все пять карт и берёт вместо них пять из колоды, так что

интуитивно кажется, что обмен почти никакого влияния на вероятность не имеет. На том и порешим. Мулат меняет четыре карты из пяти. Легко догадаться, что у него был один туз и он поднял к нему ещё три. Напрашивается вопрос номер три – какова вероятность события?

Вопрос номер последний – стоит ли эмулировать эту фигню программно, или мы никогда не дождёмся ответа – вследствие крайней маловероятности события? Приступим, однако. Надо заметить, что Грин, похоже, сам в покер не играл, а прочитал о нём в какой-то книжке о красивой заграничной жизни, так что некоторые детали приходится домысливать и корректировать за писателя. Грин, к примеру, полагал, что играют колодой в 53 карты – 52 обычных и джокер. Джокеров в колоде два, так что для нашего игрока шансы заметно повышаются.

Сколькими способами можно сдать пять карт из 54-х?

$$C_n^k = \frac{n!}{k!(n-k)!} \Rightarrow C_{54}^5 = \frac{54!}{5!(54-5)!} = 3162510$$

Сколько существует комбинаций из пяти карт таких, что четыре из них тузы и ещё одна всё равно какая, но не джокер? Четыре туза набираются только одним способом, ещё в колоде остаются кроме джокеров 47 карт, значит 47 комбинаций, поскольку порядок карт нас не волнует. $P = \frac{47}{3162510} \approx 0.0000148$. Или, по другому, приблизительно один раз на 67000. Это без учёта обмена карт.

Какова вероятность получить четыре одинаковых карты (иначе говоря, *каре*), любые, и один из двух джокеров? Каре набирается тринадцатью способами, но тузы на руках у другого игрока, так что остаётся двенадцать. Джокеров два, в итоге $12 \times 2 = 24$. Искомая вероятность $P = \frac{24}{3162510} \approx 0.00000759$, или один раз на 131 тысячу случаев, всего в два раза реже, чем каре конкретно тузов. С одной стороны, мулату конечно было очень обидно. С другой стороны, по ходу игры рассказчик демонстрирует фантастическую самоуверенность - *Имея на руках четыре одинаковых карты с "джокером" в придачу, вы можете обобрать противника до последней копейки* – скромнее надо быть, девушки.

Вероятность обеих комбинаций в одной игре посчитайте сами. Кстати, у Грина в рассказе никто не верещит, что не припомнит *такого* и никогда такого не было. Ну такое, ну подумаешь…

Теперь вероятность поднять трёх тузов к одному, или, что то же самое, к любой карте три других того же достоинства. Метод тот же. В колоде осталось 49 карт, берём четыре. Обратите внимание, то, что физически в колоде не 49, а гораздо меньше, ведь часть карт на руках игроков, что совершенно неважно.

$$C_{49}^{4} = \frac{49!}{4!(49-4)!} = 211876$$

Из четырёх три туза выбираются единственным способом, и ещё 46 карт остаётся, следовательно $P = \frac{46}{211876} \approx 0.000217$

Подкидной дурак

Игра никем не уважаемая. Никогда не видел, чтобы в дурака играли на деньги. Краткое напоминание о правилах в объёме, необходимом для понимания контекста. Играют малой колодой в 36 карт. Открывают козыря. Козырь бьёт любую не козырную карту и мѐньшего козыря. Сдают по шесть карт. Игрок ходит под соседа. Если сосед не может побить карту, то берёт обе. Проигрывает тот, у кого на руках в конце остались карты.

Насколько известно мне, подкидной дурак воспет только в одном произведении, впрочем, гениальном – *Пропавшая Грамота* Гоголя. Краткое содержание предыдущих серий. Нечистая сила утащила у казака грамоту, которую он вёз к царице. Точнее – шапку, в которую была зашита грамота, которой, шапкой, казак поменялся с другим казаком, которого утащила ведьма. Где логика? *Зато спиваемо гарно!* А царица у Гоголя всегда одна - Екатерина Вторая. Казак с трудом, но добрался до нечистой силы, которая диктует свои условия.

– Ладно! – провизжала одна из ведьм, которую дед почел за старшую над всеми потому, что личина у ней была чуть ли не красивее всех. – Шапку отдадим тебе, только не прежде, пока сыграешь с нами три раза в дурня!

Играть в дурака – дело позорное.

Что прикажешь делать? Козаку сесть с бабами в дурня! Дед отпираться, отпираться, наконец сел. Принесли карты, замасленные, какими только у нас поповны гадают про женихов.

Казак два раза проигрывает, остался последний шанс.

Стал набирать карты из колоды, только мочи нет: дрянь такая лезет, что дед и руки опустил. В колоде ни одной карты. Пошел уже так, не глядя, простою шестеркою; ведьма приняла. «Вот тебе на! это что? Э-э, верно, что-нибудь да не так!» Вот дед карты потихоньку под стол – и перекрестил: глядь – у него на руках туз, король, валет козырей; а он вместо шестерки спустил кралю

Вопрос, собственно, один – какова вероятность набрав шесть карт из колоды, получить на руки четыре старших козыря? Вопрос вероятности события во всей жизни казака не изучаем – какая теории вероятностей, кругом ведьмы! Применить напрямую любимую формулу не получится, придётся сначала подумать головой, а потом, само собой, применить метод Монте-Карло.
Сначала всё же попробуем применить чистую формульную математику. Сколькими способами можно извлечь четыре карты из колоды, что все они были четырьмя старшими козырями? Правильно, одним единственным, поскольку порядок карт нас не интересует. Сколькими способами можно добавить к нм ещё две любых карты из оставшихся 32-х? Правильно, $\frac{32\times 31}{2}=496$. Деление на два символизирует безразличность порядка этих двух карт.

Второй вопрос – сколькими способами можно получить шесть карт из колоды в 36 карт? Имеем любимый биномиальный коэффициент :

$$C_n^k=\frac{n!}{k!(n-k)!}\Rightarrow C_{36}^6=\frac{36!}{6!(36-6)!}=1947792$$

Осталось поделить первое на второе и выразить в промилле: $P\approx 0.000255\approx 0.0255\%\approx 0.255‰$.

Промилле – чтобы выглядело солиднее. А теперь программируем и проверяем. Все программы для проверки карточных раскладов похожи

друг на друга, но есть и специфика – никакой цены у карт нет, и вопрос наш очень простой – есть валет, дама, туз, король одновременно, или нет.

```
   var
      D                          : TDeck36;
      trump                      : TSuit;
      numOfHighestTrumps         : integer;
      numOfSuxx                  : integer;
      card                       : TCard;
      i,k                        : integer;
begin
     trump:=diamonds;
     numOfSuxx:=0;

     for i:=1 to 100000000 do begin
        D:=TDeck36.Create;
        numOfHighestTrumps:=0;

        for k:=1 to 6 do begin
           D.RandomCardWithoutReturn(card);
           if (card.suit = trump) and
              ( (card.rank = Jack) or (card.rank = Queen) or
                (card.rank = King) or (card.rank = Ace) )
              then numOfHighestTrumps:=numOfHighestTrumps + 1;
        end;

        if numOfHighestTrumps = 4
           then numOfSuxx:=numOfSuxx + 1;

        D.Free;
     end;

     ShowMessage('numOfSuxx = ' + IntToStr(numOfSuxx));
end;
```

В общем, вопросов нет. Единственное, обратите внимание на количество выполнений цикла. Всё так и было, так я программу и запустил. В связи с этим, настоятельно рекомендую прикрутить к форме компонент типа TLabel - просто текст и по условию `(i mod 1000000) = 0` выводить значение I. А то возникает некоторая тревога – как там программа, жива ли.

Понятно, что запускать для тысячи итераций смысла никакого нет, да и для десяти тысяч немного. Начнём со ста тысяч.

Число испытаний	Козырные валет, дама, король, туз	‰
100000	23	0.230‰
1000000	252	0.252‰
10000000	2509	0.251‰
100000000	25909	0.259‰

Сходится медленно, но всё-таки сходится, и сходится туда, куда надо. Что касается вероятности события, то, заменяя проценты и промилле на понятный язык, такая карта случается один раз на примерно 4000 раздач, или, очень грубо, один раз на 200-500 игр. Мне так кажется. Ничего особенного, даже ведьму привлекать не надо.

Глава 7
Теория вероятностей.
Скучные и важные понятия в виде конспекта

Пояснение

Я сначала хотел, чтобы эта глава была очень длинной. Потом решил перенести её в виде конспекта в Приложение. Потом решил написать коротко, но здесь. Но конспективно.

Распределение. Что это такое

В этой главе мы займёмся программированием. Но сначала снова возьмём в руки нашу любимую колоду из 36 карт. Вытащим карту. Потом другую. Третью. В руки попался любимый пиковый туз, пиковая дама и туз бубён. Какова вероятность их извлечения, точнее извлечения одной из трёх? Правильно, 1/36. А почему мы не спрашиваем, о какой именно карте идет речь? А потому, что вероятности для всех карт равны. Соответственно, такое распределение называется *равномерным*. Заметьте, я не дал определения тому, что такое распределение, мне кажется, что это понятие скорее относится к интуитивно понятным - или интуитивно непонятным.

А что может быть понятнее картинки? Так нарисуем картинку! Точнее, график. По горизонтали отложим карты – от шестёрки пик до туза бубен, я не знаю, почему они именно в таком порядке идут в колоде. По вертикали, сколько именно раз каждая карта выпал при 36 опытах (вытаскивании из колоды) с возвращением в колоду, само собой. Потом повторим опыт при 36^2 опытах и при 36^3. Нарисуйте графики, желательно на одном поле. Обдумайте.

Процедура для рисования графика находится в приложении. В основном она была медленно, по шагам, обоснована и написана в моей предыдущей книге *Полезное программирование*. В качестве домашнего задания я попросил внести в программу ряд улучшений, но, поскольку домашние задания никто не делает, я сделал всё сам. Кроме того, я добавил возможность рисовать на одном поле несколько графиков одновременно, что особенно полезно для вероятностных задач. Ну и, конечно, оформил всё это богатство в виде класса.

Ещё теория.
Несколько несложных, но скучных формул.
Поговорим о среднем

Сначала о грустном. Странные слова – среднее, средневзвешенное, медиана, мода. Некоторые их этих слов вам прекрасно известны, некоторые нет. А некоторые вам известны, но вы их неправильно понимаете. Или не понимаете вообще. Или думаете, что не понимаете, но на самом деле понимаете… Ну то такое – как говорят вна Украине.

Возьмём десять небольших – чтобы легче считать -случайных чисел. Для разнообразия, программу писать не будем. Сделаем это в Excel. Чтобы получить одно случайное число надо написать в клетку следующее:

СЛЧИС()

Число будет в диапазоне [0,1],

Чтобы число стало в диапазоне [0,10] напишем

СЛЧИС()*10

Можно, конечно и по-другому, скобки, они не просто так. И, наконец, чтобы случайное число стало целым, завершим композицию:

ОКРУГЛ(СЛЧИС()*10;0)

Теперь, ухватив клетку за правый нижний угол, растянем одноклеточное на десять клеток в ряд. Если вы думаете, что я выучил Excel Специально для этой книги, то ошибаетесь, я его два года преподавал. И что? А ничего.

Наша кошечка сначала не любила пылесос. А потом ничего, втянулась. © А вы знаете, что пылесос на украино-галицийском диалекте будет *порохотяг*?

При каждой новой загрузке страницы мы получим десять новых чисел. Вот типичный, специально не подобранный, результат:

8,10,7,6,5, 10, 3, 6,9,2

Что с этими числами можно сделать? Посчитать *среднее*. Точнее – *среднее арифметическое*. Сам термин как бы намекает, что бывает и другое среднее, не арифметическое. Действительно, есть ещё *среднее квадратичное*. Но это мы поползли по дереву эволюции вверх. А могли бы и вверх. А могли бы и вниз. По мнению некоторых, среднее, просто среднее считается так:

$$\text{среднее} = \frac{8+10+7+6+5+3+9+2}{8} = 6.25$$

Идея понятна - взять все разные числа и усреднить – и она имеет право на жизнь. Но на результат потом часто обижаются.

Гораздо чаще применяется *среднее арифметическое*, оно же – *средневзвешенное*. Отличается оно тем, что для него учитывается частота появления отдельных значений в выборке. Да, на научном языке эти десять чисел называются *выборка*.

$$\text{среднее арифметическое} = \frac{8+10+7+6+5+10+3+6+9+2}{10} = 6.6$$

Разница есть. Следующим номером в программе, пока чисто для кругозора, следует *среднее квадратичное*, оно же *среднее квадратическое*:

$$\text{среднее квадратичное} = \sqrt{\frac{8^2+10^2+\ldots+9^2+2^2}{10}} \approx 7.09$$

В быту, то есть в той, повседневной, статистике, которую употребляют в СМИ, среднее квадратичное не употребляется. Но в теории вероятностей и в математической статистике одним из важнейших понятий является производное от него – среднее квадратичное отклонение. Об этом чуть позже. И ещё – среднее квадратичное всегда больше или равно среднего арифметического. Вдруг это знание вам пригодится.

Пошли дальше. Если вас заинтересует исконно американская статистика, не в пересказе, а именно в оригинале, вы заметите почти полное отсутствие понятия среднего. Никто не говорит о средней – average – зарплате.

Говорят о медианном – medium – доходе. Тонкая разница между зарплатой и доходом нас тут не интересует, а вот об отличиях среднего от медианного мы и поговорим.

Как посчитать *медиану*? Собственно, уметь считать для этого не обязательно. Выпишем числа в столбик по возрастанию. Затем проведём горизонтальную линию ровно посередине. Выглядит примерно вот так, нарисовал как смог.

2
3
5
6
6
———————
7
8
9
10
10

У нас чисел ровно десять, то есть число чётное, линия прошла между пятым и шестым по порядку числом. Если бы чисел было одиннадцать, линия перечеркнула бы шестое число сверху. Оно бы и стало медианой. В нашем случае можете выбрать любое из двух – то, что сверху, или то, что снизу. Если вы считаете себя математиком, можете взять среднее из этих двух чисел, но это уже лишнее, и так неплохо получилось.

Заметьте – медиана может быть и больше среднего арифметического, и меньше. Обдумайте.

Мода – это очень просто. Даже чересчур. Мода это то значение, которое в выборке встречается чаще всего. В нашей выборке число 6 и число 10 встречаются по два раза, все остальные по одному. Они, то есть 6 и 10, и есть моды. Две моды. Ну, бывает.

Прежде чем перейти от школьной арифметики к практически высшей математике, я расскажу вам быль. Быль – значит правда, не враньё. Надо заметить, что согласно договору, заключённому мной с Главным

Издателем™, я отвечаю за нарушения копирайта – то есть, воровать у других авторов низзя. Всё остальное можно. Но я добрый и трусливый, поэтому заранее отмазываюсь сам и отмазываю Главного Издателя™. Всё далее изложенное я почерпнул из энциклопедии Microsoft Cinemania, так что все претензии к Билу – или Биллу? – Гейтсу. Более того, далее следующий текст представляет собой вольный пересказ матюгов Роджера Эберта из этой самой энциклопедии. Как, вы не знаете кто это? Роджер Эберт – величайший кинокритик всех времен и народов. Единственный критик, которому *заложили* звезду на Аллее Славы в Голливуде, или кто там где и кого закладывает.

Это был Disclaimer, если чё.

// Опускание Михалкова Биллом Гейтсом. Или наоборот
Гениальный исполнитель роли сэра Генри Баскервиля – *овсянка, сэр* – получил ещё и американского Оскара за неведомую хрень под названием *Опалённые солнцем*. Дело было в Америке, где, как всем известно, *среднее арифметическое* изучают в университетах в курсе математической статистики.

Фильм номинировался в категории *Лучший фильм на туземном языке*, то есть не на английском. Как определяют победителя в этой категории? То есть, как определяли до Никиты, нашего, Михалкова? Есть члены Киноакадемии с правом голоса, пусть тысяча штук академиков. Они смотрят фильм и выставляют оценку – от единицы до десяти баллов, ноль нельзя, ноль – это очень обидно, поэтому ноль нельзя. Все оценки складывают, у кого больше, тот и победил. Просто и логично – до Никиты, нашего, Михалкова, который таки, изучал в советской школе арифметику.

Схема эта замечательно работает с чисто американскими фильмами, которые смотрят все или почти все академики. С фильмами от дикарей возникают мелкие проблемы – их не смотрят. Гениальный фильм *Мумба-юмба* какого-нибудь гениального режиссёра из Нигера или Нигерии или Либерии, или, не побоюсь этого слова из Гондураса, смотрят пятьдесят рыл и выставляют ему почти высшую оценку – 9. Итого он имеет $50 \times 9 = 450$.

Что происходит дальше – со слов Гейтса и Эберта? Михалков устраивает просмотр своего фильма, с икрой и шампанским. Икра – это caviar, в

смысле чёрная икра. Вся прочая икра – fish eggs, вот вы, персонально, будете жрать рыбьи яйца? Так что икра была только чёрная. Насчет блекджека и шлюх, они, Билл с Роджером, что-то не договаривают. Являются триста голодных академиков, жрут и пьют. Разумеется, это честные академики – в американском смысле слова. Сожрав и выпив, они ставят фильму по три балла. Итого имеем 300×3 = 900. Напрягаем мозг – 900 > 450, правда? Гондурас идёт лесом, обалдевшая Академия чешет репу. Спартак чемпион!

Академия приглашает для консультаций профессоров из Беркли, Гарварда и Йеля и те рассказывают им сакральное знание о среднем арифметическом. Правила немедленно меняются. Все, в общем-то, счастливы
// конец Взаимного Опускания

Очень коротко, потому что всё ясно
То же самое, вид сбоку и немного сложнее

Важнейшие, нет не так, ВАЖНЕЙШИЕ! понятия теории вероятностей – математическое ожидание и дисперсия. Покажем конкретно, то есть на программном коде. В программном коде этом сначала заполним массив случайными числами, потом посчитаем для них математическое ожидание. Выглядит это, в простейшем варианте, так:

```
   const
      N = 10;
   var
      a                          : array[1..N] of single;
      av                         : single;
      stroka                     : string;
      i                          : integer;
begin
     RandSeed:=0;

     for i:=1 to N do begin
        a[i]:=Random*100;
     end;

     av:=0;
     for i:=1 to N do begin
        av:=av + a[i];
     end;
     av:=av/N;

     stroka:='';
```

```
    for i:=1 to N do
       stroka:=stroka + FloatToStrF( a[i], ffFixed, 7,2) + '  ';
    ShowMessage(stroka);

    ShowMessage( 'av = ' + FloatToStrF( av, ffFixed, 7,2));
end;
```

Технические замечания. Функция Random без параметров возвращает случайное число в диапазоне [0,1], будучи умноженным на 100, имеем число в диапазоне [0,100]. Хорошее число ноль, на что его ни умножай, всё равно ноль и останется. RandSeed отвечает за то, что при каждом запуске программы случайные числа у нас будут совершенно не случайными, а совершенно одними и теми же. Программы, обрабатывающие случайные числа, гораздо проще отлаживать, если число не случайные, а наоборот – постоянные. Вот такой парадокс.

В общем и целом, не кажется ли вам, что вы стали жертвой вульгарного обмана? Вам пообещали некое романтическое *математическое ожидание*, а взамен выдали убогое *среднее арифметическое*? Увы, всё так оно и есть.

Скучно, девочки © ВВП

Впрочем, вас должна утешить дисперсия. Начать с того, что есть собственно *дисперсия* и есть *среднее квадратичное отклонение*. В принципе, это абсолютно одно и то же, но их положено тонко различать. Но сначала поглядим на результаты работы. Вот случайные числа:

0.00 3.14 86.10 20.26 27.29 67.17 31.87 16.18 37.22 42.57

Вот математическое ожидание:

Av = 33.18

Какие мысли нас посещают? Сначала мысль абсолютно левая – какова вероятность встретить среди десяти случайных чисел число π с точность в три знака? Посчитайте сами. Далее, мы заказывали числа в диапазоне (0,100), логично было бы думать, что математическое ожидание найдётся в районе пятидесяти, однако у нас оно оказалось сильно сдвинутым влево. А почему? А потому, что чисел всего десять, а на такой маленькой выборке

может иметь место ну совершенно любое чудо. Поглядим, что происходит при увеличении N.

Число испытаний	Математическое ожидание
10	33.18
100	48.79
1000	49.40
10000	49.81

Как видим, математическое ожидание быстро сходится, и, причём, сходится туда, куда надо. Теперь ввернёмся к обещанным дисперсии и средне-квадратичному отклонению. Тут нас ждёт определенный парадокс. Ср.-кв.-отклонение всегда определяется как корень квадратный из дисперсии, то есть – величина заведомо вторичная и более сложная. Между тем, если приглядятся, всё очень и очень просто – среднее квадратичное это математическое ожидание отклонения величины от её математического ожидания. Дисперсия, в таком случае, это квадрат отклонения величины от её математического ожидания. На пальцах:

Есть три случайных значения, пусть это будут [1,3.11]. Считаем их математическое ожидание:

$M[X] = m_x = \sum_{i=1}^{n} x_i p_i$. Это каноническая формула из учебника Вентцель. Поскольку у нас простейший случай – System.Random – то вся эта фигня сводится к $\frac{1+3+11}{3} = 5$. Это мы вычислили математическое ожидание. Если немного задуматься, то нетрудно догадаться, что математическое ожидание отклонения случайной величины от её математического ожидания неизбежно и неуклонно стремится к нулю, что банально и бесполезно. Поэтому *для начала* вводится такое понятие, как *дисперсия* – математическое ожидание квадрата отклонения случайной величины от её математического ожидания:

$$D[X] = \mu_2 = \sum_{i=1}^{n} (x_i - m_x)^2 p_i$$

Для нашего случая $\frac{(1-5)^2+(3-5)^2+(11-5)^2}{3} \approx 18.66$. Это дисперсия. Далее сама мадам ~~Вонг~~ Вентцель признаётся, что дисперсия – это ни о чём, потому что размерность её отличается от размерности случайной величины, и чтобы хоть какой-нибудь смысл вернулся, из дисперсии надо выдернуть квадратный корень, что мы с удовольствием и делаем:

$$\sigma[X] = \sqrt{D[X]} \approx \sqrt{18.66} \approx 4.319$$

Далее то же самое, но программно и для общего случая.

```
disp:=0;
for i:=1 to N do begin
   disp:=disp + Sqr(a[i]-av);
end;
disp:=disp/N;

msq:=Sqrt(disp);
```

Нарисуйте таблицу, аналогичную предыдущей , но с добавлением дисперсии и среднего квадратичного. Посмотрите, куда сходится математическое ожидание – ведь это очевидно? Почему дисперсия не сходится к очевидному нулю? Почему она *вообще* никуда не сходится? Задумайтесь.

А теперь – и так далее. Если среднее квадратичное отклонение является математическим ожиданием отклонения случайной величины от её математического ожидания, то мерещится ли вам здесь рекурсия? Нельзя ли и дальше продолжить эту цепочку? Можно. Общее название этого безобразия – центральный момент. Дисперсия – *второй центральный момент*, а далее моменты следуют по возрастающей. И нет им конца…

Для чего всё это нужно? Пока что – запомните это всё.

Теперь два вопроса с ответами – ожидаемых от вас. Первый – каков физический/повседневный смысл математического ожидания и дисперсии. Второй – а нужно ли нам это?

Великий и ужасный метод Монте-Карло

Здесь об этом не будет. Об этом будет в моей следующей книге по методам оптимизации. Для начала почитайте у Я.Перельмана о вычислении значения π методом бросания иглы. Если думаете, что это банально - лучше из Кнута (том 2 страница 420) разложение на простые множители случайным образом.

Немного о разном вероятностном

Далее идет то, что обычно называется *Математическая Смесь*, в данном случае – *Смесь Вероятностная*.

Сначала о зависимости и корреляции.

Что такое зависимость случайных величин? Пусть мы случайным образом определили радиус. Потом мы вычислили площадь окружности с этим радиусом, причем число π для формулы взяли со случайной помехой в третьем знаке, да хотя бы даже и во втором. Здесь мы имеем классический случай зависимых величин – случайная сама по себе площадь зависит от случайного самого по себе радиуса.

Теперь возьмём опять-таки классическое исследование британских ученых. Так вот, британские учёные однозначно доказали, что способность к решению математических задач пропорциональна размеру ботинок. Сюрприз? Не совсем, попросту говоря, старшеклассники решают задачи лучше и *ботинки* у них побольше будут. Здесь мы имеем *корреляцию* – две величины – математическое образовании и размер обуви – зависят от третьей – возраста.

Ещё пример. Я стараюсь не писать – и не пишу, да – о том, что относится к узкой сфере моих узко профессиональных узко-интересов. Однако, иногда журналисты достают. Когда они несут бред первый раз, я переношу это спокойно. Когда сто первый, меня это раздражает. Вот только что, пятнадцать минут назад, какой-то деятель, в очередной раз объявил, что своим успехам в добыче сланцевой нефти США – или, как сейчас можно писать, СГА – обязаны двум технологиям – гидроразрыву пласта (ГРП), причём нетрадиционному ГРП, и горизонтальному бурению. А Россия, типа, всё выдумала уже, но отстаёт, потому что *Нет пророка в своем отечестве*.

ДБь! © Лавров

Технологии эти между собой независимы. Они коррелируемы. Они зависят от другой величины, от самой важной на свете – от времени. Так вот, чисто случайно получилось, что нетрадиционный ГРП и полный цикл технологий горизонтального бурения появились в одно десятилетие, потому и пошли иногда в одной связке. По факту и ГРП и горизонтальное бурение чудесно обходятся друг без друга.

Понимаю, что вам это совершенно неинтересно.

Немного разных штучек.
Вот такая загогулина, понимаешь! © Б.Н.Ельцин

Первый кандидат на загогулину от Литлвуда. Стержень в вагоне.

Стержень шарнирно прикреплен к полу железнодорожного вагона и предоставлен самому себе. Тогда существует малая, но отличная от нуля вероятность, что он не упадет в течение двух недель: вероятность равна примерно $1:10^{10^5}$. Я припоминаю, что мне сообщили, что гений, впервые задавший этот вопрос, не был в состоянии ответить на него
© Литлвуд

Далее у Литлвуда следует обсуждение самых удивительных совпадений, какие могут случиться с человеком в течение всей его жизни. При первом чтении книги Литлвуда – в восьмом классе – мне это казалось интересным. Сейчас всё это нисколько не удивляет и кажется банальным и унылым.

Из Мостселлера *Пятьдесят вероятностных задач*:

Мы все в школьные годы забавлялись, проводя на страницах книги извилистую линию, проходящую между словами.

Вы забавлялись?
Они тупые! © Голосом Задорнова

Допустим, что мы взяли страницу энциклопедии, набранную убористым шрифтом, содержащую 100 строк, и наугад черкнули по ней линию. Принимая 1:5 за вероятность того, что эта линия пройдёт между словами

одной строки, мы найдём, что вероятность успешного прохождения линии через всю страницу равна $1:10^{70}$

К этой же категории вопросов относится нахождение вероятности того, что обезьяны, барабанящие по пишущей машинке, отпечатают Гамлета. Считая, что текст состоит из 27000 букв и пропусков и что имеется 35 клавиш, найдём, что эта вероятность равна $1:33^{27000}$.
© Мостселлер

Найдите разумное объяснение, как 35 превратилось в 33 © Я

Опять из *Пятьдесят задач* Мостселлера. Метазадача:

При бросании 100 монет какова вероятность выпадения ровно 50 гербов?

А вот это встречается часто, везде и в разных вариациях:

Молодой человек идёт в метро и садится на поезд в любом направлении – налево или направо. Поезд налево – к девушке. Поезд направо – к матери. Мать жалуется на то, что он редко у неё бывает, но юноша утверждает, что его шансы равны. Молодой человек обедал с матерью дважды в течение 20 дней. Объясните это явление.

А здесь главное – формулировка задачи, или – понимание этой формулировки:

Если хорда выбирается наудачу в заданном круге, то какова вероятность того, что её длина больше радиуса круга?

Ответ – вероятность плавает в широчайших пределах в зависимости от того, как вы определите термин *наудачу*.

Вот ещё глубочайшая мысль в простой формулировке, причём предназначенная для самых что ни на есть – не то что бы маленьких – но явно невзрослых:

Одиночные случайные события не имеют вероятности, они случайны
© Ю.М.Отряшенков «Юный кибернетик», М. Детская литература, 1978

Долго думайте. Вам надо примеров? Их есть у меня! Помните, не, конечно не помните, замечательный стих:

Вот мост через тихую местную реку,
С которого сбросить нельзя человека,
Поскольку, по данным замеров, река
Под этим мостом чрезвычайно мелка.

А вот и Борис, что с моста сброшен в реку,
С которого сбросить нельзя человека,
Поскольку, по данным замеров, река
Под этим мостом чрезвычайно мелка.

А вот и мешок от вьетнамского риса,
Который, по слухам, надет на Бориса,
Который был сброшен с моста прямо в реку…

Это, собственно, о чём?

It was many and many a year ago,
In a kingdom by the sea…

Извините, попутал. Так и хочется иногда образованность показать. Второй стих их Эдгара По об Аннабель Ли, первый стих из неизвестного – для меня автора о Б.Н.Ельцине, который не был тогда ещё президентом.

Какова вероятность для Председателя Президиума Верховного Совета РСФСР оказаться быть скинутым в реку? Да никакая! Не в смысле, что таких людей в речку Вонючку не кидают – в смысле мы не можем её вычислить. А то, что мы не можем вычислить, для нас не существует. Для априорного вероятностного подхода мы не в состоянии задать исходные вероятности величин, от которых зависит результирующие событие. Для байесовского подхода преград нет – поскольку никогда такого не было, то вероятность $\frac{1}{2}$ - или кинут, или нет. Обратите внимание, что автор шедевра с нами согласен - *С которого сбросить нельзя человека*, то есть, в глубине души, он с нами согласен – вероятность события равна нулю, однако – вот оно.

Никогда такого не было – и вот опять! © Черномырдин

Глава 8
Настоящая Математика – математический анализ и что с ним делать

Нужен ли вообще программисту математический анализ? В плане общего развития он нужен всем, без исключения. Другой вопрос, найдутся ли ему конкретные, сиюминутные применения в повседневной практике программиста. Скорее всего – нет. Девять шансов из десяти, что нет. Но так и со всеми разделами математики, некоторым персонажам и арифметика не понадобится. Теперь, вооружённые знанием теории вероятностей, оцените вероятность того, что *хоть что-то* из высшей математики пригодится. Так что приступаем.

Сначала о том, что почитать. Вот это считается классикой:

Г.М.Фихтенгольц, Курс дифференциального и интегрального исчисления, в трёх томах

Мы по *этому* не учились, *это* уже тогда было чем-то легендарным и ужасным. Некоторое время назад я всё же прочел. Фихтенгольц мне показался нестрогим, простым и занудным – сначала глава об определённых интегралах, затем глава о двойных интегралах, потом – сюрприз! – глава о тройных, и, наконец, об интегралах произвольной размерности. Зато уже никогда не забудешь Если прочитаешь

А у нас в университетской библиотеке учебники выдавали случайным образом. Мне достался вот такой:

С.М.Никольский, Курс математического анализа, в двух томах

Мне понравилось и пробудило интерес. Автор, кстати, только недавно помер, в сто с чем-то лет. Высшая математика, она, вообще, того – продляет жизнь

А вот это древность даже на фоне Фихтенгольца, но чем древнее – тем понятнее:

Акад. Н.Н.Лузин, Дифференциальное исчисление

Академик Н.Н.Лузин, Интегральное исчисление

Всё изложено кратко, понятно, бодро и весело. Немного отвлекает, что по оси абсцисс у автора всегда отложено время.

Основная проблема с математическим анализом, как и со всей прочей математикой, это – посмотрев на омерзительную реальность, догадаться, какой формулой её можно описать, и обратно – вычитав в книжке красивую идею найти для неё подходящий объект.

Теперь о том, что из математического анализа надо знать и понимать. Обратите внимание, я не говорю – пригодится в жизни. Возможно, то что вы будете понимать, вам не придётся применять напрямую, но благодаря этому вы, внезапно, окажитесь в состоянии понять нечто гораздо более сложное и полезное. Так бывает, я знаю.

Окинув взором убегающий волна за волной по оси абсцисс график синуса, я сделал вывод. Нам надо – для начала – три вещи: ряды, дифференцирование, определённые интегралы. А для самого начала надо вспомнить, что такое предел.

А для самого начала – в этой главе повсеместно переменные типа single заменяются на переменные типа double. В математическом анализе очень любят точность. Ещё раз напоминаю, что не ставлю никаких проверок деления на ноль – в математическом анализе это сплошь и рядом. Я имею в виду, сплошь и рядом формулы, которые незаметно приводят к делению на ноль.

Пределы

Предел – это очень просто. Предел бывает очень много у чего, как в математике, так и вне её. К примеру, предел есть у функции. Это очень просто. Определение, самое простое, потому что нас так учили, теперь в ходу определения позамысловатее:

$$\lim_{x \to x_0} f(x) = a \Leftrightarrow \forall \varepsilon > 0 \exists \delta > 0 \forall x : 0 < |\, x - x_0 \,| < \delta \Rightarrow |\, f(x) - a \,| < \varepsilon$$

Правда ведь, очень просто? На всякий случай, переведу на человеческий с математического. Есть функция f(x) от одного переменного, от одного –

разумеется, только для начала. К примеру, $f(x) = x^2 + 3$. Мы задаёмся вопросом, чему равно $\lim_{x \to 2}(x^2 + 3)$. У меня есть интуитивное, ни на чём не основанное предположение, что значение этого предела равно в точности семи. Если у вас есть интуитивное предположение, что автор глуповат, то слова ваши обидные. Я понимаю, что это само собой очевидно, но всё-таки, применим определение и нарисуем таблицу:

ε	2-ε	f(2-ε)	δ
1	1	4	3
0.1	1.9	6.61	0.39
0.01	1.99	6.96	0.04
0.001	1.999	6.996	0.004
0.0001	1.9999	6.9996	0.0004

По определению предела, мы должны задавать δ и, каким-то волшебным способом, определять для него ε. Мы поступаем наоборот, то есть – задаём ε и получаем для него δ, но какая разница? Результат тот же. Но зачем же мы вообще этим занимаемся, ведь с первого взгляда ответ очевиден? Кстати, всё у нас так идеально потому, что функция наша непрерывная, гладкая, без разрывов и, вроде бы, ещё и кусочно-монотонная. Совсем забыл, она ещё и аналитическая.

Теперь рассмотрим два других примера – один с ответом мало очевидным, второй с ответом чуть менее очевидным, чем в первом примере.

Чему равен $\lim_{x \to 0} \frac{\sin(x)}{x}$? Это любимая задача всех авторов учебников, замечательный предел, и вообще – правило Лопиталя. Эту замысловатую задачу мы решим чуть позже, но не аналитически, а чисто программным способом, как если бы мы этой высшей математики и вообще не знали.

А пока заметим, что вот это вот $x \to x_0$ может выглядеть и вот так: $x \to \infty$. То есть наш x стремится к бесконечности, в этом случае к положительной, а с тем же успехом мог бы и к отрицательной. Надеюсь, вы знаете точный математический смысл слов *стремится к бесконечности*. А если не знаете, то интуитивно догадываетесь. В математике это почти одно и то же, хотя она и самая строгая наука. Рассмотрим простенький предел

$\lim_{x\to\infty}\frac{3}{x^2}+1$. Если вам опять всё интуитивно понятно, то попрошу немного помолчать. Тройка наверху и квадрат в показателе внизу добавлены, чтобы вас запутать. Единица служит той же цели.

В этом случае, в отличие от предыдущего предела, который $\lim_{x\to 2}(x^2+3)$, и для которого можно просто взять значение в точке предела, эта самая точка бесконечно удаляется и всегда за горизонтом. Нарисуем таблицу, только немного другую, потому что ε у нас нет, и в последнем столбце, который δ, будем считать что это отклонение от интуитивно ожидаемого значения – а вы какое значение ожидаете?

x	f(x)	δ
1	4	3
10	1.03	0.03
100	1.003	0.003
1000	1.0003	0.0003
10000	1.00003	0.00003

Идея понятна, то есть понятно – к какому пределу стремится значение нашей функции по мере того, как её аргумент стремится к своему пределу. Обратите внимание, что предел совершенно не зависит от того, какая константа написана у дроби в числителе. Ещё обратите внимание, что предел точно так же не зависит от того, что написано у икса в показателе, если показатель больше единицы. А если меньше? Обдумайте.

Ещё один элементарный пример, но другого рода: $\lim_{x\to 3}\frac{1}{x-3}$. Проблема здесь в том, что функция имеет разрыв при x=3. Это плохой разрыв – *бесконечный*, он же *неустранимый*. Потому что функция по мере приближения к точке разрыва стремится к бесконечности и починить это невозможно:

ε	x-3+ε	f(x-3-ε)	δ
1	1	1	как-то
0.1	0.1	10	всё
0.01	0.01	100	пошло
0.001	0.001	1000	вразнос

Раньше у нас был конечный предел в бесконечности, теперь бесконечное значение предела при конечном значении x.

Тоже простой предел, но с хорошим концом: $\lim\limits_{x\to -2}\dfrac{x^2-4}{x+2}$. Опять функция имеет разрыв, теперь при x=-2. Это хороший, безобидный разрыв – *устранимый*. То есть, мы можем просто для себя решить, что f(-2) *чему-то* равно, и разрыв исчезнет. Чему именно равно? Пределу в точке. Рисуем и заполняем таблицу:

ε	-2-ε	f(-2-ε)	δ
1	-3	-5	1
0.1	-2.1	-4.1	0.1
0.01	-2.01	-4.01	0.01
0.001	-2.001	-4.001	0.001
0.0001	-2.0001	и так ясно	

Предел явно стремится к минус четырём. Возможно, у вас возникла мысль провести следующую нехитрую алгебраическую операцию, требующую начальных познаний в школьной алгебре:

$$\lim_{x\to -2}\frac{x^2-4}{x+2}=\lim_{x\to -2}\frac{(x+2)(x-2)}{x+2}=\lim_{x\to -2}(x-2)=-4$$

Если такая мысль возникла, то академик Н.Н.Лузин, умерший в 1950-м году с вами совершенно согласен, но с тех пор наука шагнула далеко вперёд. Теперь та функция, что до сокращения, и та, что после – это совсем разные функции.

Лузин по-простому, рабоче-крестьянским способом классифицирует функции и их пределы так – по отношению к пределу в конкретной точке:

- хорошие, гладкие, непрерывные $\lim\limits_{x\to 3}3x^2-5x+1$
- стремящиеся в бесконечность $\lim\limits_{x\to 3}\dfrac{1}{x-3}$
- бестолково колеблющиеся $\lim\limits_{x\to 3}\sin\dfrac{1}{x-3}$

- с разрывом и предел с двух сторон разный – примером, согласно академику, является тот же самый предел $\lim_{x\to 3}\frac{1}{x-3}$, с одной стороны от точки $x=3$ он стремится в положительную бесконечность, а с другой – в отрицательную. Часто встречается в физике и в инженерном деле

- с устранимым разрывом – то, что интересует академика больше всего. Пример уже был выше. Лузин ласково называет это явление *испорченная формула* и настоятельно советует формулу немедленно починить.

Однако, рисовать таблицы и считать на калькуляторе грустно и печально.

Но работать без подручных,
Может грустно, а может скучно. © Высоцкий

Пусть, наконец, компьютер займется свое прямой работой – не набором моих текстов, а расчётами. Сначала, сами себе, сформулируем техническое задание. Оно мне видится примерно так:

На вход нашей процедуры поступает функция. Функция в смысле как математическом, так и в программистском. Так же поступает величина x_0. Наша процедура должна попытаться выяснить, есть ли у функции предел в этой точке. Возможные ответы – в стиле акад. Лузина - кроме самого значения предела – предел есть, предел бесконечен, предел есть, но функция сходится абсолютно. Процедура должна обработать правильно и особый случай, когда существует $\lim_{x\leftarrow\bullet} f(x)$. Разумеется, нельзя обойтись без объявления процедурного типа.

Вот интерфейсная часть:

```
unit siLimit;  //  23.06.2017
               //  24.06.2017
{-------------------------------------------------------}
interface
   type
      TFunction = function(     x : double) : double;
      TLimType = ( ltOk, ltInf, ltOsc, ltNone);
      TLimSide = ( lsLeft, lsRight);

procedure FindLimit(     F       : TFunction;
                         x0      : double;
```

```
        limSide : TLimSide;
    var limType : TLimType;
    var limit   : double);
```

Комментарии. Объявление функционального типа вам должно быть понятно. TLimType – тип предела или тип его отсутствия – надо ещё немного обдумать и, возможно, расширить. TLimSide - безусловно, правильная, настоящая программа поиска предела должна его искать везде. Но за деньги. У нас программа бесплатная и игрушечная. Поскольку есть совершенно строгие математические понятии *предел слева* и *предел справа*, то мы и попросим пользователя нашего продукта их указать. В чём смысл? Если мы задаём $x_0 = 10$ и limSide = lsLeft, то поиск предела начнётся от $x \approx 9$ и будет идти слева направо, при x возрастающем. А если наоборот – то наоборот, от ≈ 11.

Обратите внимание – наша процедура не ищет предел, для этого у неё мозгов нет. Она всего лишь проверяет, есть предел в точке x_0, и если есть – чему он равен.

Первым делом пишем тест. А перед тем, как написать тест, напишем незатейливую функция. На практике таких незатейливых функций приходится писать часто и много.

```
function LimTypeToStr(     sLimType : TLimType) : string;
begin
     if sLimType = ltOk  then result:='Ok'  else
     if sLimType = ltInf then result:='Inf' else
     if sLimType = ltOsc then result:='Osc'
     else result:='None';
end;
```

Назначение функции просто и очевидно – она переводит в строковый тип имя переменной нашего специфического типа. Это нужно чаще, чем кажется.

Подумайте о применении вышесказанного к разрывным функциям разных типов. Я сам не могу, объем книги конечен.

Вы следите за моими мыслями? Вы следите, я сам не могу
© Не помню откуда

Ряды в теории

Что такое ряд? Очень просто. Ряд, это последовательность чисел. Последовательность может быть конечной – 1, 0, 9, 13, 3.11419, 7.40 - но такая последовательность никому не интересна. Правильные пацаны исследуют только бесконечные ряды. А ещё более правильные только сходящиеся, но об этом позже.

Числа могут быть натуральными:
1,2,3,4,5…

Числа могут быть рациональными:

$$1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\frac{1}{5},...$$

Этот ряд называется *гармонический*. Или лучше вот так, но числа тоже рациональные:

$$1,\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{16}...$$

Это частный случай *степенного* ряда. Иногда его называют *геометрический ряд*. А чем он лучше? Я скоро объясню. Числа могут, само собой, быть иррациональными:

$$1,\frac{1}{\sqrt{2}},\frac{1}{\sqrt{3}},\frac{1}{\sqrt{4}},\frac{1}{\sqrt{5}}...$$

$$1,\frac{1}{(\sqrt{2})^2},\frac{1}{(\sqrt{3})^3},\frac{1}{(\sqrt{4})^4},\frac{1}{(\sqrt{5})^5}...$$

Ряды могут состоять и из трансцендентных чисел, но нам это не интересно. С высоты нашего орлиного программистского полёта, что рациональные, что иррациональные, что трансцендентные – одна фигня. Есть размер машинного слова и за его пределы никакая математика не вылезет. Занудства ради, ряд может состоять из комплексных чисел, но об этом я вам ничего не скажу. Не потому, что это не важно – это очень важно. Не скажу по той простой причине, что у нас не было курса Теории Функций Комплексного Переменного. Ну не было, ну и всё…

А вот такие ряды называются знакопеременными, почему – понятно каждому и сразу:

$$+1,-1,+1,-1,+1\ldots$$
$$1,-\frac{1}{2},\frac{1}{4},-\frac{1}{8},\frac{1}{16}\ldots$$

Ряды обычно записывают не по-простому, то есть как мы записали, а, задавая член ряда номер N как функцию от N. Ранние примеры рядов будут выглядеть так:

$$1,2,3,4,5\ldots \Leftrightarrow a_n = n$$
$$1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\frac{1}{5},\ldots \Leftrightarrow a_n = \frac{1}{n}$$
$$1,\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{16}\ldots \Leftrightarrow a_n = \frac{1}{2^{n-1}}$$

$$1,\frac{1}{\sqrt{2}},\frac{1}{\sqrt{3}},\frac{1}{\sqrt{4}},\frac{1}{\sqrt{5}}\ldots \Leftrightarrow a_n = \frac{1}{\sqrt{n}}$$
$$1,\frac{1}{(\sqrt{2})^2},\frac{1}{(\sqrt{3})^3},\frac{1}{(\sqrt{4})^4},\frac{1}{(\sqrt{5})^5}\ldots \Leftrightarrow a_n = \frac{1}{(\sqrt{n})^n} = \frac{1}{n^{\frac{n}{2}}}$$
$$+1,-1,+1,-1,+1\ldots \Leftrightarrow a_n = (-1)^{n-1}$$
$$1,-\frac{1}{2},\frac{1}{4},-\frac{1}{8},\frac{1}{16}\ldots \Leftrightarrow a_n = \frac{(-1)^{n-1}}{2^n}$$

Что, вообще, интересного может случиться с рядом? Функция интересна сама по себе, как объективная реальность, как записанный буковками физический процесс. То, что у функции есть предел – мелкая деталь, добавляющая некоторый дополнительный интерес. А ряд?

Разумеется, мы можем рассматривать каждый член ряда как функцию от целочисленного аргумента – его номера. Немедленно после такой формулировки возникает вполне естественный вопрос – а есть ли у этой

функции предел? Можно применить этот вопрос к ранее приведённым в качестве примера рядам:

1,2,3,4,5… - предел бесконечен, что равносильно тому, что предела нет.

$1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\frac{1}{5}\cdots$ - предел равен нулю.

$1,\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{16}\cdots$ - аналогично.

Для двух примеров иррациональных рядов ответ тот же. Со знакопеременными рядам сложнее и разнообразнее.

$+1,-1,+1,-1,+1\ldots$ - предела нет, даже бесконечного

$1,-\frac{1}{2},\frac{1}{4},-\frac{1}{8},\frac{1}{16}\cdots$ - предел равен нулю.

По большому счёту, никому это всё не интересно, почти. По-настоящему интересно, сходится ли сумма ряда. Почему – увидите позже, а пока объясню, что это такое – сумма ряда. Интуитивно это понятно, попытаюсь рассказать чуть более формально.

Если есть ряд a_n, то сумма первых, к примеру, пяти членов ряда обозначается так $\sum_{n-1}^{5} a_n$. Для конкретного ряда, например гармонического, это запишется $\sum_{n-1}^{5} \frac{1}{n}$ и будет равно $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}=\frac{137}{60}=2\frac{17}{60}$.

Это называется *частичная сумма* ряда и, опять-таки, никому это не интересно. А интересна только просто *сумма* ряда:

$S=\lim_{k\to\infty}\sum_{n=1}^{k} a_n$. Другое, краткое, обозначение $\sum_{n=1}^{\infty} a_n$. Для многократно помянутого добрым словом гармонического ряда это будет $\sum_{n=1}^{\infty} \frac{1}{n}$.

Поскольку в записи присутствует предел, немедленно возникает вопрос – существует он, предел, или нет. В первом случае говорят, что ряд

сходится, во втором – ряд *расходится*. Занудства ради, бывают ряды, которые сходятся абсолютно, то есть сходится ряд, состоящий из абсолютных величин членов ряда – существует $S = \lim_{k\to\infty}\sum_{n=1}^{k} | a_n |$. Впрочем, про это можно немедленно забыть.

Чуть выше мы смотрели на ряд и пытались догадаться, к чему стремится его член в бесконечности – звучит-то как! Так вот, *необходимым* условием сходимости ряда является стремление его члена к нулю. Необходимым, но не *достаточным*, если вы понимаете, о чём я говорю. По простому. Вот у этих рядов член явно к нулю не стремится:

$1,2,3,4,5\ldots$
$+1,-1,+1,-1,+1\ldots$

Простой и однозначный вывод – эти ряды расходятся. А у следующих рядов, наоборот, стремится:

$1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\frac{1}{5}\ldots$
$1,\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{16}\ldots$

Вывод? Никакого. Может сходятся, а может и нет.

В предыдущем разделе мы написали не очень красивую программу для работы с пределами функции. Некрасивость заключается в том, что на вход программы надо задать точку, в которой мы предполагаем предел, а программа проверяет, есть ли предел и чему он равен. С рядами проще – в какой точке искать предел мы точно знаем – в бесконечности.

Поэтому сейчас самое время написать программу которая определяет, сходится ли ряд, и чему равна его сумма. В отличие от программы про предел функции, которая была чисто учебной, эта – при необходимости – может иметь вполне практическое применение.

Вот интерфейс:

```
unit siSeries;  //  27.06.2017
                //  27.06.2017
{-------------------------------------------------------------}
interface
   type
      TTermOfSeries = function(     num : integer) : double;
      TSeriesType = ( stConvergent, stDivergent);

procedure SumOfSeries(     F         : TTermOfSeries;
                       var sType     : TSeriesType;
                       var sum       : double;
                       var howMany   : integer);
function SeriesTypeToStr(     sType : TSeriesType) : string;
```

Параметр HowMany не является жизненно необходимым, но служит лишь удовлетворению моего любопытства – он сообщает, сколько членов ряда мы успели просуммировать. Функцию SeriesTypeToStr любезный читатель, без сомнения, способен написать собственными умелыми ручонками.

Алгоритм работы процедуры мне видится простым, незатейливым и общепонятным. Мы вычисляем значение частичной суммы ряда для первых шестнадцати членов. Почему шестнадцати ? – мне так кажется. Одновременно вычисляем разности между двумя частичными суммами. Если разности не убывают – ряд расходится, всё пропало. Если убывают – дожидаемся, пока разность между двумя соседними суммами станет меньше 0.001. Почему 0.001? Потому что я живу в реальном мире, а в реальном мире, если две какие-то величины, измеренные в любых единицах измерения, отличаются между собой меньше чем на 0.001, то величины считаются равными.

Прежде чем что-то программировать, надо написать тест. А прежде чем написать тест, надо знать правильный ответ. Причём правильный ответ надо знать *до* написания теста. В нашем случае мы должны взять для опытов как минимум один ряд, который сходится и один, который не сходится.

В качестве сходящегося берём степенной ряд $1,\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{16}\cdots$. А почему он сходится? А это с первого взгляда видно. А чему равна сумма? А сумма равна двум. В роли плохого, расходящегося ряда выступает гармонический ряд $1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\frac{1}{5}\cdots$. Тут с первого взгляда всё не так

очевидно, но тоже всё очень просто. Утверждение, что ряд расходится, эквивалентно тому, что сумма ряда превышает любое конечное число. Кстати, многократно помянутый академик Лузин строго замечает, что говорить *конечное* число совершенно неправильно – все числа конечные.

Далее ход мысли такой. Если есть ряд a_n и есть ряд b_n и для каждого n $b_n<=a_n$, то элементарно доказывается и, главное, совершенно очевидно следующее – если расходится ряд b_n, то расходится и ряд a_n. Для наглядности выпишем гармонический ряд, а под ним напишем ряд, каждый член которого меньше или равен соответствующему члену гармонического ряда:

$$1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\frac{1}{5},\frac{1}{6},\frac{1}{7},\frac{1}{8}\ldots$$
$$1,\frac{1}{2},\frac{1}{4},\frac{1}{4},\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8}\ldots$$

Далее обращаем внимание на забавный факт:

$$\frac{1}{4}+\frac{1}{4}=\frac{1}{2}$$
$$\frac{1}{8}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}=\frac{1}{2}$$ и так далее. Очевидно, что таким образом можно набрать любую сумму. Да, я знаю, что так теоремы не доказывают.

Возвращаемся к написанию теста. Сначала кодируем функции, возвращающие значение члена ряда по его номеру.

```
function HarmonicS(     num : integer) : double;
begin
     result:=1/num;
end;
{-------------------------------------------------------------}
function PowerS(     num : integer) : double;
begin
     result:=1/IntPower( 2, num);
end;
```

Обращаем внимание на то, чего здесь нет – а нет здесь проверки на корректность номера члена. Буква S на конце идентификатора напоминает

о том, что *ряд* в математическом смысле слова по-английски вовсе не *row*, а *series*.

А вот и сам тест – простой, но незатейливый.

```
   var
      sType                    : TSeriesType;
      sum                      : double;
      HowMany                  : integer;
begin
//      SumOfSeries( HarmonicS, sType, sum, howMany);
     SumOfSeries( PowerS, sType, sum, howMany);

     ShowMessage( 'sType = ' + SeriesTypeToStr(sType));
     ShowMessage( 'sum = ' + FloatToStrF( sum, ffFixed, 9,5));
     ShowMessage( 'howMany = ' + IntToStr(howMany));
end;
```

А почему закомментирован вызов для гармонического ряда, но оставлен вызов для степенного? А потому, что начинать всегда надо с доброго, хорошего случая. А теперь осталась сущая мелочь – запрограммировать всё, что надо. Случай для сходящегося ряда трудностей не представляет, являясь классическим упражнением для начинающего программиста.

Цикл **for** явно не походит, ведь мы заранее не знаем, сколько раз выполнится цикл. Интуитивно более удобным кажется цикл **repeat-until**, его и применим. Промежуточных переменных много, это для наглядности.

```
   const
      nobodyCares = 0.001;
   var
      term                     : double;
      oldSum                   : double;
      epsSum                   : double;
      ready                    : boolean;
      num                      : integer;
begin
     sum:=0;
     oldSum:=0;
     num:=0;
     sType:=stConvergent;

     repeat
        num:=num + 1;
        term:=F(num);
        sum:=sum + term;
        epsSum:=Abs(sum-oldSum);
        oldSum:=sum;
```

```
        if epsSum < nobodyCares
           then ready:=true
           else ready:=false;
     until ready;

     howMany:=num;
end;
```

Лично мне здесь не нравится использование параметра в качестве сумматора. Как-то это неправильно.

Запустили тест? И как? Убедились? Нет, не запустили. Я очень честный и всегда признаю свои глупости. В тестовой функции ошибка. Написано

```
result:=1/IntPower( 2, num);
```

что соответствует ряду $a_n = \frac{1}{2^n}$, то есть ряду $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16} \cdots$ Надо поправить:

```
result:=1/IntPower( 2, num-1);
```

После этого результат сходится с ожиданиями. Сумма равна – приблизительно, само собой – 1.999. Собственно, задавая точность 0.001, именно этого мы и должны были ожидать. Число итераций – 11. Эта величина на самом деле не есть предмет праздного любопытства. Она характеризует *скорость сходимости* ряда. Хороший ряд сходится быстро, плохой – медленно. Для чего это является важным, увидите чуть позже.

А теперь о грустном – пора взяться за работу и добавить к нашей процедуре обнаружение расходящихся рядов. Вы уже пробовали запустить её, процедуру, на гармоническом ряде? И не пробуйте. Распознавать нехороший ряд мы будем *эмпирически*. Эмпирически в переводе с эмпирического на русский – может работает, а может и нет. Но обычно работает.

Признаков мы будем иметь и проверять два. Первый – если член ряда не убывает, то ряд плохой. Обратите внимание – именно *не убывает*, а не *не возрастает*. Иначе прошел бы ряд 7.40, 7.40, 7.40… Второй признак – убывает приращение частичной суммы ряда, то, что выше обозначено как epsSum.

Сначала лёгкая доработка напильником. В константы добавляем

```
  timeToThink = 16;
```

В объявления переменных

```
manyTerms                : array[1..1024] of double;
manyEpss                 : array[1..1024] of double;
```

Зачем 1024? На всякий случай. Случаи, как известно, разные бывают. А теперь добавляем главное, в самый конец основного цикла

```
if num <= timeToThink then begin
   manyTerms[num]:=term;
   manyEpss[num]:=epsSum;
end else
if num = timeToThink then begin
   ready:=CheckDivergency;
   if ready then sType:=stDivergent;
end;
```

Всё очевидно, но, как всегда, осталось ненаписанным основное – функция, проверяющая предположительную расходимость. Установка признака расходимости ряда находится здесь из принципиальных соображений – не может признак устанавливаться в одно значение в одной процедуре, а в противоположное – в другой. Сама функция выглядит вот так:

```
function CheckDivergency : boolean;
begin
     result:=false;

     if Abs(manyTerms[1]) <= Abs(manyTerms[timeToThink])
     then begin
        result:=true;
     end
     else begin
        if manyEpss[1] <= manyEpss[timeToThink]
           then result:=true;
     end;
end;
```

Для гармонического ряда это работает – расходится, да. Есть ли куда это улучшать – безусловно, да. Легко придумать что-то извращённо синусообразное, которое болтается в пространстве как синус, то есть вверх-вниз, время от времени совершая неадекватные скачки прочь от оси

абсцисс, при этом всё-таки к ней, оси абсцисс, приближаясь. Возможен и обратный случай – когда нехороший ряд случайно проходит критерий

Но случаи такие очень редки © Гилберт Кит Честертон

Если однажды вам предъявят претензии подобного рода, вы должны ответить, что виновата не программа, виновата неправильная функция. Главное, при этом вы должны смотреть прямо в глаза, и взгляд у вас должен быть честный-честный. О важности уверенности в себе и честных-честных глазах – читайте приложение с именем *Баллада о Синусе*.

Я совсем забыл о двух наших очень иррациональных рядах. Первый из них очевиден – кладём рядом гармонический ряд и сравниваем соответствующие члены:

$$1, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{4}}, \frac{1}{\sqrt{5}} \ldots$$

$$1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5} \ldots$$

$\frac{1}{\sqrt{2}} \approx \frac{1}{1.41} \approx 0.707 > \frac{1}{2} = 0.5$. А гармонический ряд, мы знаем, и все знают, расходится. Значит и этот тоже. С другим рядом интереснее, потому что с первого взгляда не очевидно. Придётся программировать.

$$1, \frac{1}{(\sqrt{2})^2}, \frac{1}{(\sqrt{3})^3}, \frac{1}{(\sqrt{4})^4}, \frac{1}{(\sqrt{5})^5} \ldots$$

Ряд трансформируется в вот это:

```
function SomethingS(     num : integer) : double;
begin
     result:=1 / Power( num, num/2);
end;
```

И удивительно быстро сходится – за 10 итераций до ≈1.779. Подумайте, чему бы это могло быть равно – в понятиях чистой математики и аналитических функций.

Ряды в живой природе

Теперь о том, для чего вообще нужны ряды. Ряды нужны для разложения в них функций. Вы не задумывались о том, откуда вообще взялись значения синуса. Sin(90°) = 1. Sin(0°) =0/. Это очевидно. Чуть сложнее, но легко доказуемо Sin(30°)=1/2. А вот синус от 60° равен 0.866 и какая-то хрень дальше, даже не в периоде, потому что число не рациональное. Откуда они все эти числа узнали?

А они разложили синус в ряд. Как разложили – другой вопрос, а результат – он вот:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Идея понятна, функция программируется легко.

```
function SinusS(     num : integer) : double;
begin
     result:=IntPower( -1, num+1) *
             IntPower( DegToRad(60), 2*num-1) / Fuct(2*num-1);
end;
```

Факториал рассчитывается как

```
     result:=1;
     for i:=2 to n do
        result:=result*i;
```

Обратите внимание, что запрограммировали мы именно и конкретно вычисление синуса угла α=60°. Разумеется, вам будет совсем нетрудно расширить процедуру и функцию для любого угла.

Результат вычисления совпадает с таковым результатом от калькулятора имени Билла Гейтса, что хорошо. Количество итераций всего четыре, что очень хорошо.

Теперь применим самый популярный ряд в мире – для числа π.Точнее, один из самых популярных. Рядов для числа π много и очень много. Мы возьмём, разумеется, худший, чтобы потом было куда улучшать. Зато ряд этот не сказать, чтобы очень красивый, но очень простой:

$$\pi / 4 = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \ldots$$

Вам не кажется, что все ряды немного похожи? *Ряд довольно прост для понимания, но он ужасно медленно сходится* – меланхолически замечает Ч.Уэзерелл, автор книги *Этюды для программистов*. Кстати, официальное название это ряда – *ряд Лейбница*.

Кодируем функцию:

```
function PiSimpleS(     num : integer) : double;
begin
     result:=4 * IntPower( -1, num+1) / (2*num-1);
end;
```

Вы уже запустили программу на выполнение? Получили ответ? Я запустил несколько раз, вручную установив разное количество итераций. Первые четыре дали вот такие значения:

16	3.200	1.86%
50	3.161	0.62%
100	3.151	0.30%
500	3.144	0.07%

В последнем столбце ошибка в процентах. Напоминаю, что $\pi \approx 3.14159\ldots$ И только при числе итераций более 20000 мы получаем на что-то похожее значение 3.14155. Последний знак всё равно неверен. К этому времени возникает желание сделать процедуру суммирования ряда более универсальной. Конкретно, поменять её интерфейс на вот такой:

```
procedure SumOfSeries(     F           : TTermOfSeries;
                       var sType       : TSeriesType;
                       var sum         : double;
                       var howMany     : integer;
                           nobodyCares : double = 0.001;
                           timeToThink : integer = 16;
                           forced      : integer =  0;
```

Напоминаю, что такая форма записи означает, что последние три параметра являются параметрами *по умолчанию*. То есть, вызов

`SumOfSeries( F, sType, sum)` эквивалентен `SumOfSeries( F, sType, sum, 0.001, 16).`

`SumOfSeries( F, sType, sum, 0.0001)` то же самое, что `( SumOfSeries( F, sType, sum, 0.0001, 16)`

С третьим параметром аналогично. В чём его смысл и назначение? Назначение его заключается в том, чтобы сделать нашу процедуру более приспособленной к потребностям реальной жизни. Иначе говоря, сейчас у нас только одна возможность ограничить количество итераций – задать минимальную величину изменения частичной суммы. Если сумма изменилась менее, чем на заданную величину, то всё – приплыли. К сожалению, нетрудно убедиться, что величина это, хотя и безусловно связана с точностью вычисления суммы ряда, но не напрямую. Иногда проще задать явным образом число итераций. Вот за это число итераций и отвечает параметр forced. По умолчанию его значение равно нулю – то есть, он вообще ни за что не отвечает.

Для реализации этих смелых идей не забудьте удалить объявления соответствующих констант – для первых двух добавленных параметров по умолчанию. Для третьего параметра надо добавить только один оператор.

```
repeat
   //  всё, как было…
   if num < forced then ready:=false;
until ready;
```

Плохой ряд оказался действительно плохим.

Не обманул фашист © Брат-2

Теперь опробуем ряд получше. Рядов, как я уже упоминал – много. Начнём с того, который лучше, но на лицо не очень ужасный. С авторством некоторая путаница – то ли Мадхава из индийского города с непроизносимым названием изобрёл его в XV-м веке, то ли он изобрёл вместо Лейбница предыдущий, плохой ряд, то ли что. Короче, вот, встречайте:

$$\sqrt{12}(1-\frac{1}{3\times 3}+\frac{1}{5\times 3^2}-\frac{1}{7\times 3^3}+...)$$

Ряд этот мне не очень нравится, из-за присутствия иррациональности в виде $\sqrt{12}$. Оскорбляет, понимаете ли, эстетическое чувство. Программный код выглядит так:

```
function PiNotSoSimpleS(     num : integer) : doble;
begin
     result:=IntPower(-1,num+1)*Sqrt(12)*
             (1/((2*num-1)*IntPower(3,num-1)));
end;
```

Задаём для сходимости разности частичных сумм $\varepsilon < 0.001$. Имеем 7 итераций и результат 3.14167 – три верных знака. Задаем 1000 итераций, то есть

`SumOfSeries( PiNotSoSimpleS, sType, sum, howMany, 0.001, 16, 1000)`. Имеем приближение 3.1415925 – шесть верных знаков. Результаты налицо. Однако хочется чего-нибудь эдакого, с подвывертом.

Лично мне понравился вот этот ряд. Главное, он прост для понимания и легко запоминается, а изобрёл его Сринивас Рамануджан аж сто лет тому назад:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801}\sum_{k=0}^{\infty}\frac{(4k)!(1103+26390k)}{(k!)^4 396^{4k}}$$

Уж если я чего решил, то выпью обязательно © Высоцкий

Короче, эту формулу я точно запрограммирую. Чтобы не запутаться, пришлось объявить промежуточные константы и переменные:

```
function PiNotSimpleAtAllS(     num : integer) : double;
   const
      mult = (2*1.41421356)/9801;
      c1   =  1103;
      c2   = 26390;
      c3   =   396;
   var
      up,down                  : double;
      k                        : integer;
begin
     k:=num-1;
     up:=Fuct(4*k) * (c1 + c2*k);
     down:=IntPower( Fuct(k), 4) * IntPower( c3, 4*k);
     result:=mult * (up / down);
```

```
end;
```

Разумеется, культурнее было бы написать `mult = (2*Sqrt(2) ) / 9801;` но транслятор почему-то не пропускает. Впрочем, результат того стóит. При двух итерациях имеем результат 3.1415927 – шесть верных знаков после запятой.

Однако, как учит нас *Опыт, сын ошибок трудных* © А.С.Пушкин, градус должен идти на повышение. Вот вам повышение градуса от 1987-го года (братья Чудновские), и запрограммируйте это сами:

$$\frac{426880\sqrt{10005}}{\pi} = \sum_{k=0}^{\infty} \frac{(6k)!(13591409 + 545140134k)}{(3k)!(k!)^3(-640320)^{3k}}$$

Но мы же не звери. Вот совсем не уродливая и очень даже эффективная формула, впрочем, сам я не проверял:

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}\right)$$

Если запрограммированный ряд не захочет сходиться, не спешите винить себя и искать в программе ошибку – возможно, я просто ошибся в процессе набора.

А теперь задумаемся о том, откуда эти забавные ряды вообще берутся. С Раманудженом всё ясно, он был гений по части формул. Для не гениев сначала надо освоить производные или, иначе говоря, дифференцирование, чем сейчас и займёмся.

Дифференцирование

Что такое производная? Нет, не по определению – определение будет, не сомневайтесь. Что такое производная, чтобы сразу понять. Чаще всего дают физическую формулировку – производная есть мера изменения скорости. Ещё более физически – есть скорость, а есть ускорение. Так вот, ускорение и есть производная скорости, скорость – функция, математическим языком. Параметр x – время.. То есть, мы держим в руке кирпич силикатный облегчённый. Скорость кирпича равна нулю,

ускорение тоже. Отпускаем кирпич. За одну секунду скорость кирпича увеличится на 9.8 м/сек. Это не математика, это физика. Говорят, Галилей измерил на Эйфелевой башне.

Теперь посчитаем производную самостоятельно, вот этими собственными руками. Возьмём для опытов очень хорошую во всех отношениях функцию $f(x) = x^2 - 2x + 4 = (x-1)^2 + 3$. Это парабола, сдвинутая по оси абсцисс на единицу вправо и приподнятая по оси ординат на три единицы. Вычислим её производную в точке $x_0=3$.

Ещё раз – производная есть скорость возрастания функции в конкретной точке. Мы можем взять точку слева от точки x_0, обозначить её x_{-1}, вычислить значения $f(x_0)$ и $f(x_{-1})$, вычесть второе из первого и поделить на $x_0 - x_{-1}$. Это и будет в точности то, что называется производной. Не обязательно брать точку слева, разумеется. Можно и справа – результат *в конечном итоге* будет тем же. Кстати, чтобы два раза не вставать, договоримся, что символом d мы обозначим, насколько именно мы отступили влево – или вправо – от точки, в которой ищем производную.

С конкретными цифрами, $f(3) = 7$, $d = 1$, $f(3-d) = f(2) = 4$. Разность между ними $7 - 4 = 3$. Далее - $x_0 - x_{-1} = 3 - 2 = 1$. Делим первое на второе, получаем $3/1 = 3$. Это искомая переменная в первом, грубом приближении. То, что это неточное значение, понятно. Но *насколько* оно неточное? Мы программисты, или кто? Пишем несложную функцию для вычисления приблизительного значения производной и применяем её при разных d. Вот функция дифференцирования, чрезвычайно простая функция, которую дифференцируют, и вызов первой функции для второй:

```
function Diff(     F  : TFunction;
                   x0 : double;
                   d  : double) : double;
begin
     result:=(F(x0+d) - F(x0)) / d;
end;

function Parab(     x : double) : double;
begin
     result:=x*x - 2*x + 4;
end;

Diff(Parab,x0,d);
```

А вот результат:

x0	*d*	*x0+d*	f(*x0*)	*f(x0+d)*	*f(x0)-f(x0-d)*	*≈производная*
3	1	2	7	4	1	3
3	0.1	2.9	7	6.61	0.39	3.9
3	0.01	2.99	7	6.96	0.04	3.99

То, чем мы сейчас занимались, подозрительно напоминает вычисление $\lim_{x \to a} f(x)$. Технически – да, по существу – нет. В первом случае, если функция в точке *a* ведёт себя хорошо, ответ очевиден. Предел в точке равен значению функции в точке и зачем мы этой ерундой вообще занимались? В случае производной ответ никогда не тривиален. Ну или почти никогда.

Как видим, значение производной стремительно приближается к четырём. Что бы это значило? Посчитаем значение производной в точке *x=5*. Кстати, пора напомнить, что если наша функция обозначена как $f(x)$, то её производная обозначается $f'(x)$. То есть, в нашем случае, $f'(3) = 4$, а $f'(5) = 6$. Через эти две точки можно прямую, и она задаётся формулой $y = 2x - 2$. О чём это говорит? Только о том, что через любые две точки можно провести прямую, причём только одну, а это мы и без того знали. Однако, определим программно $f'(10)$. Правильно, $f'(10) = 18$, что в точности ложится на уравнение прямой. Третья точка на прямой это уже не случайность. Можно ли отсюда сделать вывод, что уравнение прямой и есть уравнение искомой нами производной? Да, это хороший, правильный вывод. Можно ли отсюда сделать вывод, что производная любой функции – прямая? Нет, нельзя.

Дадим, наконец, определение производной, оно очень простое, если помнить, что такое предел:

$f'(x_0) = \lim_{\delta \to 0} \frac{f(x_0 + \delta) - f(x_0)}{\delta}, \delta > 0$. Далее обязательно должны следовать оговорки на тему, существует ли производная в точке x_0 и тому подобное. Для нас важно одно – если производная существует, то она задаётся именно этим пределом. Теперь одно мелкое замечание – если есть

функция и есть её производная, которая, ясное дело, тоже функция, то можно взять производную и от производной. Это обозначается $f''(x)$ и называется *второй производной*. Третья производная $f'''(x)$, ну и так далее.

Теперь замечание очень простое, но совсем не мелкое. Поскольку $\delta > 0$, то ясно, что при $f(x_0+\delta)-f(x_0)>0$ производная положительна. Если разность меньше нуля, производная отрицательная. Обратные утверждения тоже являются справедливыми. Если $f'(x_0)>0$ то $f(x_0+\delta)-f(x_0)>0$. Звучит банально, но отсюда следует, что если производная в точке x_0 положительна, то функция в этой точке возрастает. А если наоборот – то наоборот. А если производная равна нулю?

// Анекдот
Здесь мог бы быть известный анекдот от Никулина о попугае в зоомагазине, у которого к обеим ногам привязано по верёвочке, и если дёрнуть за левую, то попугай поёт, а если за правую, то матом ругается, и покупатель спрашивает – *А если дёрнуть за обе?* – но, поскольку анекдот очень хорошо всем известен, а место в книге надо экономить, то анекдота здесь не будет.
// конец Анекдота

А будет самое интересное – это я уже о производной. Но об этом в другой раз.

Разумеется, не надо каждый раз запускать программу для получения численного значения производной в точке. В старые времена вместо этого вычисляли производную по определению через предел. Так вот, ещё в старые времена все полезные производные уже вычислили. Вы их найдёте написанными на любом заборе. А здесь микро-справочник в виде микро-таблицы. Для начала заучите заклинание – производная суммы равна сумме производных! Я потом это ещё раз повторю.

функция	*первая производная*	*вторая производная*	*пример первой производной*
$f = c$ (константа)	$f' = 0$	$f'' = 0$	$f'(3.141) = 0$
$f = x^n$	$f' = nx^{n-1}$	$f'' = n(n-1)x^{n-2}$	$f'(3x^2 + 2) = 6x$
$f = \sin x$	$f' = \cos x$	$f'' = -\sin x$	
$f = \cos x$	$f' = -\sin x$	$f'' = -\cos x$	
$f = e^x$	$f' = e^x$	$f'' = e^x$	
$f = \ln x$	$f' = \frac{1}{x}$	$f'' = -\frac{1}{x^2}$	
$f = f_1 + f_2$	$f' = f_1' + f_2'$	$f'' = f_1'' + f_2''$	

Производные и программирование

Сначала надо делать то, что легко и просто. Легко и просто программировать символьное вычисление производных от многочленов.

А на экзаменах в Военную Академию велели Василию Ивановичу извлечь квадратный трёхчлен. И вот он плачет, а саблю-то точит...
© Просто анекдот

Пользы практической от этого почти никакой, но и усилий программиста тоже почти никаких. Что обычно понимается под символьным дифференцированием? Напоминаю, что у настоящих математиков взятие производной называется *дифференцирование*, хотя, конечно, есть тонкие различия. Понимается, что на вход задаётся строка, например $\cos x$, а на выходе мы имеем строку $-\sin x$. И тому подобное. На входе формула – на выходе формула. В частном случае многочлена мы пойдём другим путём. Имеем:

$$f(x) = a_1 x^n + a_2 x^{n-1} + \ldots a_n x + a_{n+1}$$
$$f'(x) = a_1 n x^{n-1} + a_2 (n-1) x^{n-2} + \ldots a_n$$
$$f''(x) = a_1 n(n-1) x^{n-2} + a_2 (n-1)(n-2) x^{n-3} + \ldots$$

и так далее…

На примере:

$$f(x) = 2x^3 - 5.3x^2 - 9x + 1$$
$$f'(x) = 6x^2 - 10.6x - 9$$
$$f''(x) = 12x - 10.6$$
$$f'''(x) = 12$$
$$f^{IV} = 0$$

на этом всё кончается.

В качестве нехитрого упражнения смастерим процедуру – на вход степень многочлена, массив коэффициентов многочлена и порядок производной, на выходе - массив коэффициентов производной. Использовать открытые массивы в качестве параметров или динамический массив мне почему-то не хочется. Разумным кажется просматривать массив коэффициентов справа налево до тех пор, пока идут сплошные нули и так определить степень многочлена, но мне и этого не хочется. Предлагаю вот такой проект интерфейса;

```
  const
     maxPoly = 16;
   type
     TPolyKoef = array[1..maxPoly] of double;

procedure PolyDiff(     pk        : TPolyKoef;
                        pOrder    : integer;
                        diffOrder : integer;
                    var pkDiff    : TPolyKoef);
```

Но сначала напишем совершенно служебную процедуру для вывода коэффициентов многочлена, без неё нам никак не обойтись – проверять работу процедуры ведь как-то надо?

```
procedure ShowPolyKoef(     pk     : TPolyKoef;
                            pOrder : integer);
   var
      stroka                   : string;
      i                        : integer;
begin
     stroka:='';
     for i:=1 to pOrder+1 do begin
        stroka:=stroka + FloatToStrF( pk[i], ffGeneral, 5,2) +
                         'x^' + IntToStr(pOrder-i+1) + '  ';
     end;
```

```
     ShowMessage(stroka);
end;
```

Доработайте напильником – например, затем степень при свободном члене? Теперь собственно процедура дифференцирования, она несложная, труднее было написать тест, тут я пожалел, что не реализовал автоматическое определение степени многочлена. Вот процедура:

```
procedure PolyDiff(     pk        : TPolyKoef;
                        pOrder    : integer;
                        diffOrder : integer;
                    var pkDiff    : TPolyKoef);
   var
      i,k                       : integer;
begin
     pkDiff:=pk;

     for i:=1 to diffOrder do begin
        for k:=1 to pOrder+1 do begin
           pkDiff[k]:=pkDiff[k] * (pOrder-k+1-i+1);
        end;
     end;
end;
```

А тест написан вот в таком примерно стиле:

```
     //  2x^3 - 5.3x^2 - 9x + 1
     pk[1]:=2;  pk[2]:=-5.3;  pk[3]:=-9;  pk[4]:=1;
     pOrder:=3;

     diffOrder:=1;
     PolyDiff( pk, pOrder, diffOrder, pkDiff);
     ShowPolyKoef( pkDiff, pOrder-diffOrder);
```

Результат удовлетворителен, всё работает. Вечный вопрос – а зачем и где это нужно? Наверное, нужно это много где, но единственное доступное мне применение относится к методу Штурма – фамилия такая, причём французская – нахождения корней многочлена. Об этом в следующей книге.

Такими вещами приходится заниматься редко. Чаще приходится заниматься – для того, для кого вообще приходится – заниматься численным дифференцированием. Мы этим вообще-то уже занимались несколько раньше, неявно применяя и программируя формулу

$f'(x_0) \approx \frac{f(x_0+\delta)-f(x_0)}{\delta}, \delta > 0$. Главное, чтобы δ было ну *очень* маленькое. Можно считать, что эту задачу мы уже решили.

Для чего это надо – отвечу позже, после расширения постановки вопроса. Функции бывают не только от одной переменной. Вот хорошая непрерывная функция от двух переменных

$$F(x,y) = 3x^2 + 2y^2 - 18x - 8y + 36$$

Если переменных много, они обычно обозначаются $x_1, x_2, x_3, x_4 \ldots$ Но если переменных только две, то традиционно употребляют обозначения *x,y*. От этой функции тоже можно взять производную, но не одну, а целых две – по *x* и по *y*. Когда мы берём производную по x, мы считаем символ *y* просто константой и соответственным образом с ним обращаемся. А когда по *y* – наоборот. Простое следствие – если мы берём производную от суммы по *x* и в одном из слагаемых *x* отсутствует, то слагаемое, согласно правилам дифференцирования, аннигилируется, даже если оно равно $y^{1024}y^{y^y}$. Беспокоюсь – вы разглядели *y* в третьем верхнем ряду? Байка от Литлвуда:

В докладной записке, которую я написал (около 1917 года) для Баллистического управления, в конце была фраза «Таким образом, δ следует сделать сколь возможно малым». В печатном тексте записки этой фразы не было. Но П.Дж.Григг сказал: «Что это такое?» Едва заметное пятнышко на пустом месте в конце оказалось миниатюрнейшим δ, которое я когда-либо видел (наборщики, вероятно обыскали весь Лондон).

Кстати немцам о рентгене – то же самое относится и к $\delta \to 0$. Теперь практически. Производная от функции двух – или больше – переменных по одной из них обозначается $\frac{df}{dx}$. Производная по *y* - $\frac{df}{dy}$. Для нашей хорошей функции имеем:

$$\frac{df}{dx} = 6x - 18 \quad \frac{df}{dy} = 4y - 8$$

Само собой, есть и вторые – далее - производные, берущиеся по тем же правилам:

$$\frac{d^2 f}{dx^2} = 6 \quad \frac{d^2 f}{dy^2} = 4$$

Занудства ради, можно поиметь и смешанные производные, то есть $\frac{d^2 f}{dxdy}$ и $\frac{d^2 f}{dydx}$. Наука быстро доказывает, что два последних выражения равны друг другу и, в нашем случае ещё и тождественно равны нулю. Забудьте.

Следующая программная задача – численное определение производной от двух функций. Здесь требуется некоторое умственное усилие. Только что мы вычисли две первых производных от хорошей функции. Эти производные, в свою очередь, тоже являются функциями. Но численная производная не может быть функцией, она может быть только числом. В нашем случае – двумя числами, то есть вектором. Далее, когда я буду объяснять, зачем это всё надо, слово *вектор* приобретёт глубокий смысл. А пока, запряжём калькулятор и посчитаем ручками частные производные в точке, например, (1,1). Это самая что ни на есть заурядная точка для этой функции. Почему? Функцию можно путём школьных алгебраических преобразований записать так:

$$F(x,y) = 3x^2 + 2y^2 - 18x - 8y + 36 = 3(x-3)^2 + 2(y-2)^2 + 1$$

Очевидно, что в точке (3,2) два первых слагаемых обращаются одновременно в ноль. Это точка экстремума. Об экстремумах в следующей книге, она уже более готова, чем нет, честное слово. Это всё к тому, что банальная точка (1,1) действительно является банальной, поскольку страшно далека от точки экстремума.

Сначала словами. Есть функция $f(x,y)$. Нам нужны первые производные по x и y в точке (x_0, y_0). Для вычисления производной по x мы это самое x не трогаем, а меняем y на малую величину – и смотрим, что произошло со

значением функции. Затем делим приращение функции на приращение аргумента. Почти так же, как и раньше.

$$\frac{df}{dx}(x_0, y_0) \approx \frac{f(x_0+\delta, y_0) - f(x_0, y_0)}{\delta}$$
$$\frac{df}{dy}(x_0, y_0) \approx \frac{f(x_0, y_0+\delta) - f(x_0, y_0)}{\delta}$$

Само собой, δ>0. Возвращаемся к калькулятору. Сначала считаем значение функции в точке: $f(1,1) = 15$. Теперь значения частных производных, опять же, по формулам.

$$\frac{df}{dx} = 6x - 18 = -12 \qquad \frac{df}{dy} = 4y - 8 = -4$$

А теперь приблизительные значения частных переменных, для разных δ, отдельно по *x* и по *y*:

δx	δy	$f(x_0+\delta x, y_0+\delta y)$	$\approx \frac{df}{dx}$	$\approx \frac{df}{dy}$	Отклонение от теории
1	0	6	-9		3
0.1	0	13.83	-11.7		0.3
0.01	0	14.8803	-11.97		0.03
0	1	13		-2	2
0	0.1	14.62		-3.8	0.2
0	0.01			-3.98	0.02

С первого взгляда видно, что всё работает как надо. Оно и понятно – не нами придумано. Со второго взгляда заметно, что первая производная по второй переменной приближается к цели быстрее. Как нетруд–но догадаться, связано это с тем, что у неё и коэффициент при старшем члене поменьше. С точки зрения математического анализа, никаких проблем здесь нет, а с точки зрения программного применения это–го самого математического анализа – немного есть.

Но займёмся, наконец вопросом – зачем нужно численное дифференцирование. С моей точки зрения, оно нужно для применения

методов оптимизации. Разумеется, есть и другие разумные приложения, но я встречался только с этой областью. Вдаваться в подробности не буду, почти в двух словах. Заранее прошу прощения, если это всё вам давно известно.

Методы оптимизации ищут экстремум функции – то есть минимум, или максимум, какая разница. Чисто математически это идеально, просто и красиво решается с помощью производных. Вот, к примеру, всеми любимая двести без малого лет задача о заборе – есть возможность построить забор длиной 1000 метров, огородив им прямоугольник – так надо. Задача – захапать побольше землицы. Имеем – если периметр всего забора 1000 метров, то две смежные стороны имеют в сумме длину 500. Если одну сторону обозначить через x, то смежная сторона будет 500-x. Площадь, как нетрудно догадаться, будет $S(x) = x(500 - x) = 500x - x^2$. Великая и страшная тайна Древних Математиков скрыта в том, что функция принимает экстремальное значение тогда, когда её производная равна нулю – если, конечно, это хорошая функция. Имеем:

$$S(x) = 500x - x^2$$
$$S'(x) = 500 - 2x$$
$$500 - 2x = 0$$
$$x = 250$$

Результат слегка ожидаем – забор должен быть квадратным. А ещё лучше – круглым. Я не поленился и посчитал – круг даёт выигрыш ≈27.3%. Только никому не говорите.

В реальной жизни настолько красиво не бывает Минимизируемая функция всегда от нескольких переменных – от пяти-шести обычно. Переменные не разделены. О чём речь? Поглядим ещё раз на частные производные нашей хорошей функции:

$$\frac{df}{dx} = 6x - 18 \qquad \frac{df}{dy} = 4y - 8$$

Если мы приравняем их нулю, то мгновенно найдём точку экстремума - (3,2). Но если бы функция выглядела чуть иначе, например вот так:

$$F(x,y) = 3x^2 + 2y^2 + 2xy - 18x - 8y + 36$$

то всё чуть усложнилось. Частные производные были бы такими:

$$\frac{df}{dx} = 6x + 2y - 18 \qquad \frac{df}{dy} = 4y + 2x - 8$$

Теперь для нахождения экстремума нам надо решить очень простую систему линейных уравнений:

$$6x + 2y - 18 = 0$$
$$4y + 2x - 8 = 0$$

Это не просто, а очень просто, но что, если переменных пять-шесть и выражения совсем не многочлены? И, наконец, что если формул-то у нас нет, а есть численные значения в некоторых точках, полученные неведомым способом? Можно попросить ещё значение, в какой хочется точке, но получено оно опять-таки будет неведомым путём. Вот здесь и приходит на помощь численное дифференцирование.

Ещё раз повторю, в подробности я вдаваться не буду, подробности в другой книге. Здесь очень кратко. Если нам надо найти экстремум функции одной переменной, от производной мало пользы. Если она больше нуля – функция возрастает, если меньше нуля – функция убывает. Разумеется, если глядеть слева направо. Для нахождения минимума функции одной переменной в таких извращениях нет никакой надобности. Есть много простых и быстрых способов, и все они никаким образом не пользуются производными.

Другое дело – многомерная оптимизация. Производная в этом случае – вектор. Вектор – это направление. Вектор этот указывает куда-то туда – туда, где функция быстрее всего возрастает. Взяв обратный вектор, мы получим указание – в каком направлении функция быстрее всего убывает. Это может быть непосредственно минимум функции, но так бывает редко. В обычном случае производная задаёт направление, вдоль которого производится одномерная оптимизация. Дойдя то точки минимума по оси, программа оглядывается по сторонам, смотрит, куда указывает вновь

вычисленная производная и снова минимизирует вдоль оси. И так до тех пор, пока не надоест, или, говоря по ученому, пока $|F_n - F_{n+1}| > \varepsilon$. В завершение этого раздела расширение процедуры вычисления численного приближения производной для случая функции двух переменных.

Сначала необходимо объявить функцию от двух переменных как тип, и нашу конкретную хорошую функции в соответствии с этим объявлением:

```
TFunction2 = function(     x,y : double) : double;

function GoodFunc(     x,y : double) : double;
begin
     result:=3*x*x + y*y - 18*x - 8*y + 36;
end;
```

А вот и сама процедура численного дифференцирования:

```
procedure Diff2(     F2     : TFunction2;
                     x0,y0  : double;
                     d      : double;
                var dx,dy   : double);
begin
     dx:=(F2(x0+d,y0) - F2(x0,y0)) / d;
     dy:=(F2(x0,y0+d) - F2(x0,y0)) / d;
end;
```

Что с ней делать дальше? Очевидным образом, расширить на функцию от произвольного числа переменных. Или почти произвольного числа. Следующим шагом – сейчас мы задаём точность по переменным – то есть $F(x+\delta) - F(x)$. Хотелось бы задавать точность по значению частной производной, то есть считать не сразу, а в цикле, постепенными приближениями и останавливаться когда $\left| \left(\frac{df}{dx}\right)_n - \left(\frac{df}{dx}\right)_{n+1} \right| < \varepsilon$. Точность должна подбираться по каждой переменной независимо. И, само собой, необходимо расширить процедуру на производные высших порядков, в том числе и смешанные.

Ряды. Откуда всё-таки они берутся

Сначала байка с разоблачением.

/ / Байка про махновцев и ряд Макларена с разоблачением
Каноническая версия байки
Во время гражданской войны будущий лауреат Нобелевской премии по физике Игорь Тамм попал в плен к одной из банд Махно. Его отвели к атаману – «бородатому мужику в высокой меховой шапке, у которого на груди сходились крест-накрест пулеметные ленты, а на поясе болталась пара ручных гранат».
— Сукин ты сын, коммунистический агитатор, ты зачем подрываешь мать-Украину? Будем тебя убивать.
— Вовсе нет, — ответил Тамм. – Я профессор Одесского университета и приехал сюда добыть хоть немного еды.
— Брехня! – воскликнул атаман. – Какой ты профессор?
— Я преподаю математику.
— Математику? – переспросил атаман. – Тогда найди мне оценку приближения ряда Маклорена первыми n членами. Решишь – выйдешь на свободу, нет – расстреляю.
Тамм не мог поверить своим ушам: задача относилась к довольно узкой области высшей математики. С дрожащими руками и под дулом винтовки он сумел-таки вывести решение и показал его атаману.
— Верно! – произнес атаман. – Теперь я вижу, что ты и в правду профессор. Ну что ж, ступай домой.
Тамм так никогда и не узнал фамилию атамана
Уолтер Гратцер "Эврики и эйфории"

В чём разоблачение? Ряд Макларена изучается на первом году не то, что математического факультета в курсе математического анализа, но и любого технического факультета в курсе высшей математики. Кстати, вы помните – на случай встречи с махновцами –что у математиков не бывает курса высшей математики?

В альтернативной версии байки Тамм ничего вывести не смог, на что махновец ему сказал – *да ладно мужик, я и сам эти формулы давно забыл.*

И вообще, всё было совсем не так.
// конец Байки с разоблачением

А теперь – возьмёмся за дело. Сначала – *функциональный ряд*. Это очень просто, это такой же ряд, как и ряды раньше, только членами его являются не числа, а функции. В самом общем виде – вот так:

$f_1(x), f_2(x), f_3(x), f_4(x), f_5(x)...$

Выглядит совершенно неконкретно и ни о чём, но надо сразу понять главное. А главное здесь то, что f_1, f_2 и f_3 это совсем разные функции, а вот x как раз наоборот – везде один и тот же. Теперь добавим конкретики. Конкретика добавляется в два этапа. Сначала надо задать функции $f_n(x)$, например, так:

$$\frac{1}{x^0}, \frac{1}{x^1}, \frac{1}{x^2}, \frac{1}{x^3}, \frac{1}{x^4}... \Leftrightarrow f_n(x) = \frac{1}{x^{n-1}}$$

Вторым шагом надо задать конкретный x и посмотреть, что получится. Задаём *x=2* и получаем хорошо уже знакомый геометрически ряд:

$$1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}...$$

Задаём *x=0.*5 и получаем вот такое:

$$\frac{1}{0.5^0}, \frac{1}{0.5^1}, \frac{1}{0.5^2}, \frac{1}{0.5^3}, \frac{1}{0.5^4}... = 1, \frac{1}{0.5}, \frac{1}{0.25}, \frac{1}{0.125}, \frac{1}{0.0625}... = 1,2,4,8,16...$$

Первый ряд, как мы уже знаем, сходится. Второй ряд, очевидным образом, расходится. Как меланхолично замечает в связи с этим академик Лузин – *расходящиеся ряды вообще никому не нужны*. Особенно программистам. А теперь о главном, сосредоточьтесь. Вот это *степенной ряд*:

$$a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + ... \Leftrightarrow f_n(x) = a_{n-1}x^{n-1}$$

На первый взгляд, ряд как ряд. На второй тоже. Но Макларен, или Маклорен доказал, что любую функцию можно представить в виде такого ряда. Тут я загнул немного, не совсем *любую* функцию – но любую *хорошую* функцию. Причём делается это просто, понятно и доступно. Даже я всё понял с почти первого раза.

Итак, у нас есть *хорошая* функция $f(x)$. Соответственно, $f'(x)$ её первая производная, $f''(x)$ вторая производная, а $f^{(n)}$ производная порядка *n*. Тогда имеем

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \ldots$$

Как это выглядит на практике – берем синус. Для синуса имеем следующие известные из предыдущего раздела соотношения:

$$f(x) = \sin(x) \Rightarrow f(0) = 0$$
$$f'(x) = \sin'(x) = \cos(x) \Rightarrow \cos(0) = 1$$
$$f''(x) = \cos'(x) = -\sin(x) \Rightarrow -\sin(0) = 0$$
$$f'''(x) = (-\sin x)'(x) = -\cos(x) \Rightarrow -\cos(0) = -1$$

И так далее, я устал. Видим, что члены с нечётными номерами просто исчезают, а остаётся вот это:

$$\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Как видите, ряд Макларена – это очень просто. Очень просто это ещё и потому, что ряд Макларена является частным случаем ряда Тейлора, который, впрочем, - тоже очень простой. Действительно сложным здесь является решение вопроса – мы действительно получили ценное разложение функции в ряд? По этой технологии можно получить и расходящийся ряд и, что ещё хуже, ряд сходящийся, но не туда.

Здесь должен быть рассказ о ряде Тейлора, ряде Фурье, и, само собой, о быстром преобразовании Фурье, но я решил оставить это до следующей книги.

Неопределённые интегралы

А что такое интеграл вообще? Впрочем, это не главное. Только что мы изучили дифференцирование. Интегрирование – это то же самое, только наоборот. Вычитание – это сложение наоборот. Деление – это умножение наоборот. Интегрирование – дифференцирование наоборот.

$$(x^3 - x + 99)' = 3x^2 - 1$$

$$\int (3x^2 - 1)dx = x^3 - x + C$$

Загадочная буква C как бы намекает, что конкретное число 99 сгорело в пламени дифференцирования. И восстановить его оттуда никаким способом невозможно. Впрочем, буковка эта имеет сугубо ритуальный характер и далее сгорает снова. Называется эта буковка *постоянная интегрирования*. Забудьте. Далее о важном, важного много.

Если у нас есть функция, её всегда можно продифференцировать, то есть – взять от неё производную. Да, я в курсе, что почётный гражданин Украины, уроженец исконно-украинского города Лемберг, он же Львов, по имени Захер Мазох однажды изобрёл мазохизм. Его математические последователи не отставали и придумали много интересных и разных функций, от которых производные не очень берутся. Но вот вы сами попробуйте придумать функцию без производной. Причём, сто очень важно, взятие производной не требует гигантских умственных усилий – оно требует хорошей памяти и аккуратности – то есть, легко становится алгоритмом.

С интегралами всё не так. Взятие интеграла – это *Искусство*, именно так, с большой буквы. Вот что говорит академик Лузин:

Интегральное исчисление предлагает ряд целесообразных приёмов, достаточных для довольно многих случаев. *Но учащийся не должен заблуждаться относительно силы этих приёмов: приёмы эти лишь систематизируют и приводят в некоторый порядок первоначальный подход при помощи непосредственного нащупывания и догадки, и ничего более.*

По простому – бывают интегралы, которые берутся легко. Пример вы видели выше. Бывают интегралы, которые берутся только через очень сложно:

$$\int \frac{x^{3/2}dx}{(a^2 - b^2x)^2} = \frac{3a^2x^{1/2}2b^2x^{3/2}}{b^4(a^2 - b^2x)} - \frac{3a}{2b^5}\ln\left|\frac{a + bx^{1/2}}{a - bx^{1/2}}\right|$$

Как меланхолично замечает Литлвуд:

Математический текст такого стиля (вдохновлённый, несомненно, дьяволом) не может не содержать опечаток.

Если вы думаете, что я взял этот интеграл сам, то мне даже и странно, что вы могли обо мне так подумать. То есть, взять-то я его взял, но взял из книги, в которой он числится под номером 187.23, а книга называется

Г.Б.Двайт «Таблицы интегралов и другие математические формулы» М.Наука, 1978

Ещё бывают интегралы, которые не берутся вообще, при этом выглядят вполне безобидно:

$$\int \frac{dx}{\ln x}$$

Что важно, если интеграл не берётся, это, конечно печаль - для математиков. Программисты этого даже и не заметят.

Так вот, всё что только что было выше написано об интегралах, написано о тех интегралах, которые называются *неопределённые интегралы*. Запомните вот такое обозначение:

$$\int f(x)dx = F(x) + C$$

Это не теорема, не формула, не определение – это именно обозначение. Если есть функция $f(x)$, но неопределённый интеграл от неё - тоже функция, но совсем-совсем другая, и мы обозначим её $F(x)$. Обратите внимание на вечную постоянную интегрирования C и немедленно забудьте.

Никаких программ в этом разделе нет. Какие программы? Там академики с профессорами не могут интеграл взять. А мы тут со своими программами…

Определённые интегралы

Определённые интегралы – это вовсе не неопределенные интегралы, которые вдруг определили. Начнём с то го, что неопределённый интеграл – функция, определённый интеграл – число. Хотя выглядят они очень похоже – вот, слева неопределённый, справа – наоборот:

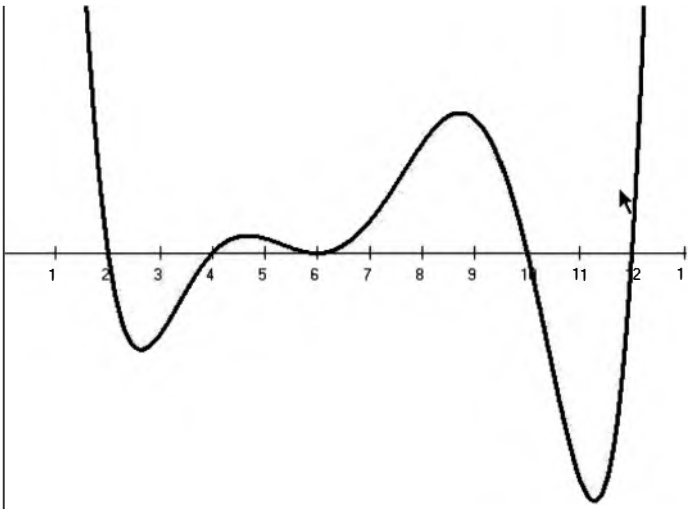
$$\int x^2 dx \quad \int_0^1 x^2 dx$$

Что такое определённый интеграл? Сначала с точки зрения физического смысла, он же геометрический. Для опытов возьмём функцию, которая не очень хорошо выглядит, но от которой, однако, очень легко взять интеграл:

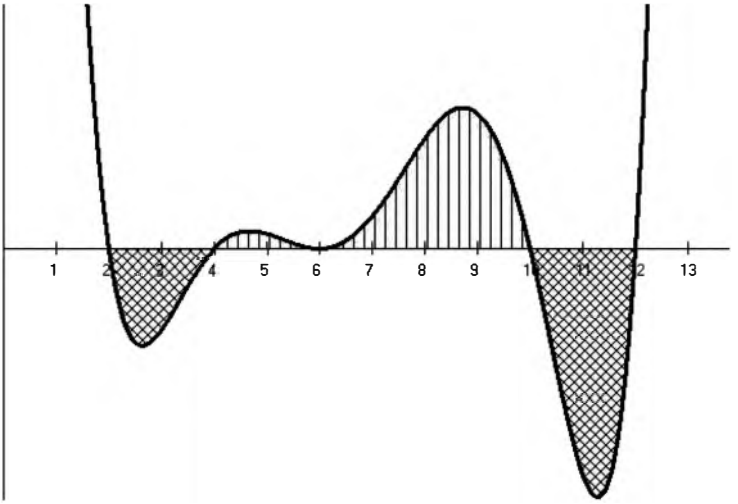
$$f(x) = x^6 - 40x^5 + 632x^4 - 5024x^3 + 21072x^2 - 43776 + 34560$$

$$\int f(x)dx = \frac{1}{7}x^7 - \frac{20}{3}x^6 + \frac{632}{5}x^5 - 1256x^4 + 7024x^3 - 21888x^2 + 34560x$$

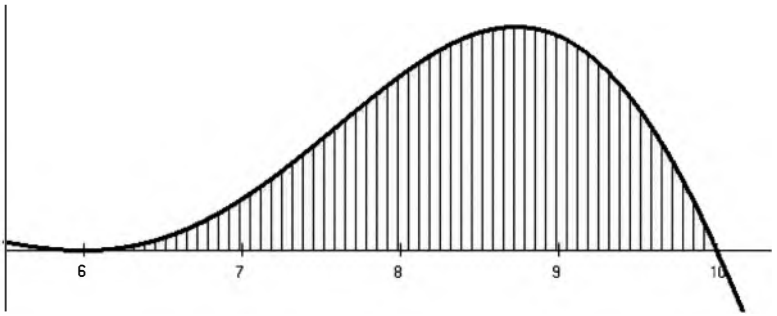
Вы проверили? Я нигде не ошибся? Обозначим это безобразие, как и положено, через $F(x)$. График исходной функции не так уж и ужасен.



Очень может быть, что функция эта имеет некоторый физический смысл – или будет иметь, если мы захотим. Обычно предполагается, что по оси *x* отложено время, а по оси *y* зависимая от времени величина, обычно скорость. А что тогда означает площадь фигуры между осью абсцисс и кривой, как сверху, так и снизу? Вот картинка:

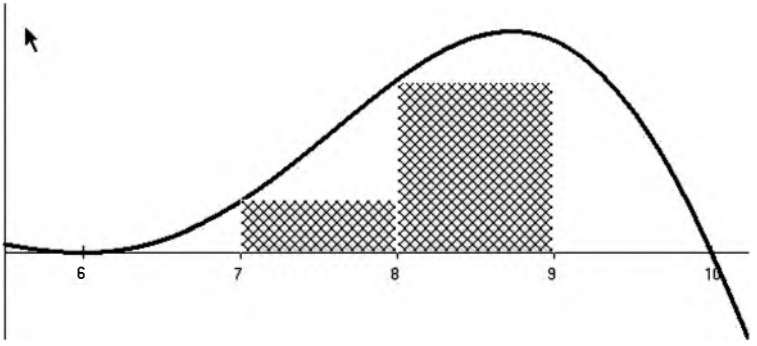


Это просто понять, если представить себе этот график для скорости, неизменной от времени, то есть $f(x) = const$. Тогда заштрихованная область принимает форму прямоугольника, по горизонтали – время, по вертикали – скорость, произведение - время×скорость = пройденный путь. В нашем случае всё точно так же, только скорость переменная. А какой физический смысл у областей снизу оси абсцисс? Если скорость ниже оси, значит она отрицательная, значит едем назад. Суммарная площадь областей сверху – сколько проехали вперёд, снизу – сколько проехали назад. Если вычесть из первого второе, получим итоговое перемещение, опять-таки со знаком. Простой вопрос – а как посчитать площадь? Сначала почти строгое математическое обоснование. Для наглядности считать будем на интервале [6,10] который весь сверху, и для той же наглядности немного укрупним масштаб. Обратите внимание – пока что программистам и математикам по пути.

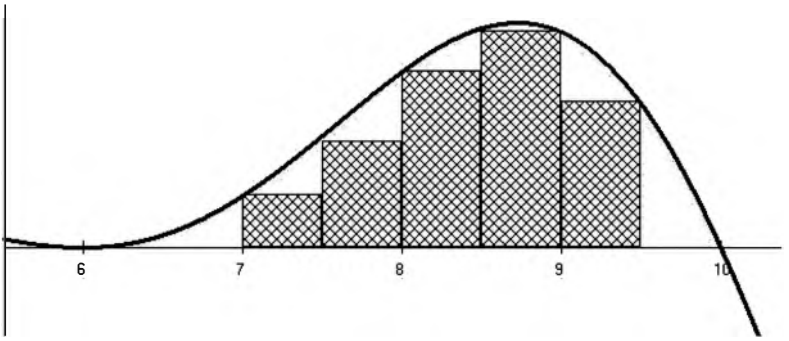


Теперь по шагам, как программисты, но, в конкретном данном случае, и как математики тоже. Что будет, если мы проведём из точки (6,0) вертикальную линию до пересечения с кривой? Ничего не будет, кривая уже с нами. Проведём вертикальную линию из точки (7,0). Это уже имеет смысл.

Из той точки где эта *вертикальная* прямая пересеклась с кривой, проведём *горизонтальную* прямую, до пересечения с вертикальной линией, проведённой из точки (8,0) до пересечения с кривой. И так далее. Непонятно? Сейчас нарисую.



Можно ли считать, что площадь заштрихованных прямоугольников близка площади под кривой? Вряд ли. Уменьшаем шаг по оси x вдвое.



Уже правдоподобнее. *Продолжаем разговор* © Карлсон. Что произойдёт с общей площадью прямоугольников, когда они будут становиться всё уже и уже, и их будет всё больше и больше? Интуитивно кажется, что площадь их в конце концов сольётся с площадью под кривой.

Вот как-то так, через хитрую задницу, оно и работает © Билл Гейтс о Windows 7

Интуиция нас не обманывает. Математики в любом учебнике легко и просто доказывают, что $\lim\limits_{n\to\infty}\sum\limits_{x_i=b}^{x_i=b} f(x_i)\Delta x_i = S_{ab}$. Здесь S_{ab} та самая площадь, которую мы ищем, она же $\int\limits_a^b f(x)dx$, Δx_i - основание прямоугольник, n – количество прямоугольников. С этого места дороги программистов и математиков расходятся. Сначала о том, как поступают настоящие математики.

Определённые интегралы. Что делают математики

Вы ещё помните, что взятие неопределённого интеграла – это искусство. Так вот, определение предела суммы, которая вверху, - тоже искусство. А теперь длинная, но умная цитата из академика Лузина:

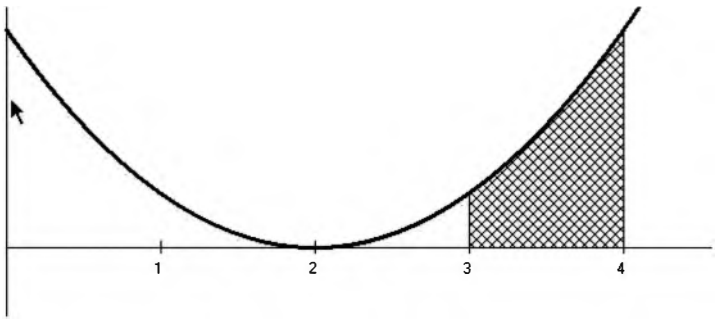
До сих пор мы рассматривали интегрирование, как процесс, обратный дифференцированию. Но это является лишь только одной стороной дела. Другой стороной и притом несравненно более важной для бесчисленных применений интегрального исчисления к геометрии, механике и, вообще,

ко всему естествознанию, является то обстоятельство, что интегральное исчисление дает нам в руки могущественный способ фактически находить пределы сумм бесконечно увеличивающегося числа бесконечно умаляющихся слагаемых. Стремление сводить свои проблемы к отыскиванию пределов таких сумм естествознание имело еще до эпохи Ньютона, и затем это стремление окончательно оформилось, окрепло и систематизировалось.

Действительно, этот процесс суммирования есть процесс не прямой, но косвенный. Прямым образом производить отыскивание пределов таких сумм мы до сих пор не умеем. Кроме двух-трех редчайших случаев, когда мы умеем найти величину определённого интеграла с численными пределами, не зная соответствующего неопределенного интеграла.

Далее настоящие математики переобуваются в воздухе и легко доказывают, что если у нас есть $f(x)$ и $\int f(x)dx = F(x) + C$, то $\int_a^b f(x)dx = F(b) - F(a)$. Шутки, смех, аплодисменты. Обратите внимание, что в процессе вычитания постоянные интегрирования взаимно уничтожаются.

Всё очень просто. Для иллюстрации берём не только хорошую, но и простую функцию. Вот функция - $f(x) = \sqrt{2}(x-2)^2$, вот её график, заштриховано то, что мы хотим посчитать:



Имеем функцию: $f(x) = \sqrt{2}(x-2)^2 = \sqrt{2}(x^2 - 4x + 4)$, далее получаем первообразную $\int f(x)dx = \sqrt{2}(\frac{x^3}{3} - 2x^2 + 4x) = F(x)$. Ничего, что я больше не пишу постоянную интегрирования *C*? Теперь итог: $\int_3^4 f(x)dx = F(4) - F(3) = \sqrt{2}\frac{7}{3} \approx 3.300$

Как видите, всё легко и просто . Проблема в том, что не всегда легко и просто производить вычисления, даже с помощь калькулятора. Помните нашу зигзагообразную, не помню какой степени функцию вверху. Для которой я трудолюбиво рисовал прямоугольники? Пересмотрите картинку. Интеграл от этой функции легко берётся, но считать его пределы вручную немного трудно. Посчитаем всё-таки и его, для дальнейших опытов. Обозначим обе функции (хорошую и не очень хорошую) соответственно:

$$goodf(x) = \sqrt{2}(x-2)^2$$
$$badf(x) = x^6 - 40x^5 + 632x^4 - 5024x^3 + 21072x^2 - 43776 + 34560$$

Хорошую – *goodf* – мы только что посчитали своими собственными руками. На плохую – *badf* – меня не хватило, я запустил *Maple*. Результат $\int_6^{10} badf(x)dx = \frac{217088}{105} \approx 2067.50$. Запомним этот результат, чуть дальше мы получим его безо всякой высшей математики, чисто программным способом.

А ещё там вверху был какой-то противный интеграл из справочника, мне он сразу не понравился – взять бы я его не смог. А ещё, как напоминает нам академик Лузин, бывают интегралы, которые *вообще* не берутся. При этом, как заботливо разъясняет академик, если неопределённый интеграл не берётся, определённый интеграл никуда не делся – вот же она, площадь над кривой, или под. А у программистов всё просто.

Здесь я, как математик не чистый, а как чисто прикладной, должен заметить, что в реальности нашей вселенной входные подинтегральные данные проходят скорее по ведомству теории вероятностей.

Определённые интегралы. А программисты делают так

Программисты начинают с того места, где остановился академик Лузин, потому что у него не было программистов. Программисты тупо считают сумму площадей прямоугольников. Вот именно тех, что на картинке и в пределе, который соответствует значению определённого интеграла. Чисто для расширения кругозора – в вычислительной математике это так и называется – метод прямоугольников – the Method of Rectangles. Над интерфейсом я долго думал:

```
function MOfRectangles(     F     : TFunction;
                            a,b   : double;
                            delta : double;
                            eps   : double = 0) : double;
```

Комментирую. *Delta* – ширина прямоугольника. Обычно предполагают, что *количество* прямоугольников стремится к бесконечности, но тогда ширина одного прямоугольника будет зависеть от интервала (a,b), что безразлично для математического анализа, но не совсем хорошо для программирования. *Eps* – параметр по умолчанию, если он не ноль, то *delta* автоматически уменьшается до тех пор, пока разность между двумя последовательными приближениями площади не станет меньше *eps*. Все проверки на корректность входных данных вы добавите самостоятельно. И ещё, математики для нас уже доказали, что совершенно безразлично, вычислять ли значение функции по левой границе прямоугольника, по правой, или где угодно внутри него. Оцените, насколько тупо и в лоб реализована мною реализация:

```
   var
      numOf                    : integer;
      oldResult                : double;
      i                        : integer;
begin
     result:=0;

     if eps = 0 then begin
        numOf:=Floor((b-a)/delta);
        for i:=1 to numOf do
           result:=result + delta*F(a+i*delta);
     end
     else begin
        repeat
           oldResult:=result;
           numOf:=Floor((b-a)/delta);
```

```
            result:=0;
            for i:=1 to numOf do
               result:=result + delta*F(a+i*delta);
            delta:=delta/10;
         until Abs(oldResult-result) < Eps;
      end;
end;
```

А вот результаты применения метода для очень хорошей функции $f(x) = \sqrt{2}(x-2)^2$:

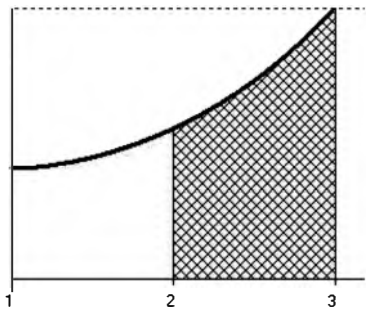
delta	*сумма*	*отклонение %*
0.1	2.949	10.64
0.01	3.264	1.10
0.001	3.296	0.12
0.0001	3.299	0.03

Если задействовать параметр по умолчанию и задать *eps=0.001*, то результат будет 3.300. Это не значит, что мы достигли абсолютной точности, значение конкретно этого интеграла вообще иррационально - $\sqrt{2}\frac{7}{3}$, это значит, что погрешность лежит за третьим знаком после точки. Если мы поинтересуемся, чему к этому моменту равно delta, то обнаружим, что его значение равно $10^{-6} = 0.000001$. Здесь может возникнуть некоторое недопонимание, ведь из таблицы видно, что при $delta = 10^{-6}$ отклонение от теоретического результата в пределах 0.001. Но обратите внимание, что мы требуем несколько бо̀льшего – мы требуем чтобы разность двух последовательных приближений была меньше 0.001.

Теперь применим программу на нашей плохой функции. Я применил. Примените вы. Сходится, и даже быстрее. Объясните кажущийся парадокс.

Теперь пара слов о неберущихся интегралах. Напоминаю, интеграл который не берётся – такой интеграл, который нельзя взять, оставаясь в рамках начальной высшей математики. Или, говоря по другому, он невыразим в аналитических функциях. Примеры неберущихся интегралов

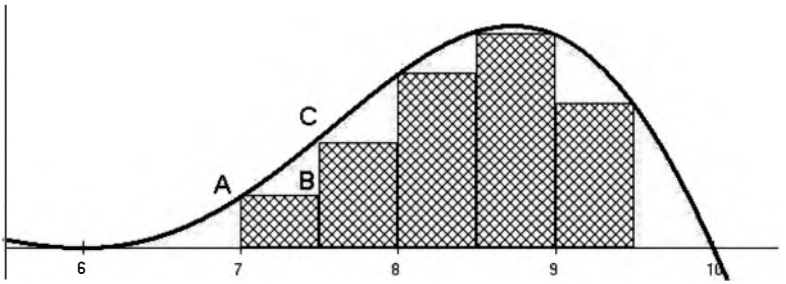
- $\int \frac{dx}{\ln[} \cdot \int \frac{e^x}{x} \cdot \int \frac{\sin x}{x}$. Что здесь надо понять – неопределённого интеграла нет, а определённый есть. Он не может не есть! То есть, его не может не быть, ведь определённый интеграл – всего-навсего площадь над или под кривой. Возьмём функцию $f(x) = \frac{e^x}{x}$, называется это интегральной показательной функцией, видимо потому, что интеграл от неё не берётся. График выглядит заурядно и заунывно.



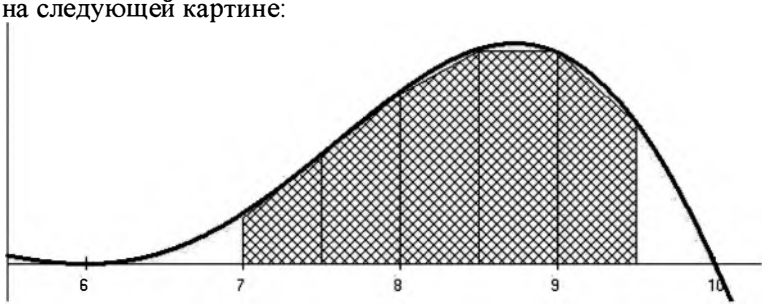
Обратите внимание, что график начинается с единицы. Объясните себе и окружающим причину. Запишите причину на бумажку и прилепите её магнитиком к системному блоку или холодильнику.

Запускаем программу. Поскольку программа не знает, что интеграл не берётся, то быстро получаем ответ: $\int_2^3 \frac{e^x}{x} dx \approx 4.980$. А говорили не берётся, не берётся…

Теперь любимая мною тема – стоит ли улучшать программу? Повторю ещё раз картинку с методом прямоугольников:



А нельзя ли улучшить? Мы проводим вертикальную линию из точки $x=7$ до пересечения её с кривой. Затем проводим горизонтальную линию до пересечения её с вертикальной, восстановленной из точки $x=7.5$. Иначе говоря, мы провели линию из точки *A* до точки *B*. Немного подумав, задаем вопрос – а не лучше ли провести линию сразу в точку *C*? Результат на следующей картине:



С первого взгляда видно, что фигура заполнена гораздо лучше, отчасти потому, что функция у нас *очень* хорошая. Что случилось с геометрической точки зрения? Прямоугольники мы заменили на прямоугольные трапеции. Выглядят они не вполне канонически, потому что поставлены боком. Алгоритм расчёта суммы нисколько не меняется. Меняется только формула расчёта элементарной фигуры. Для прямоугольника $S = ah$, где, в нашей терминологии, $a = f(x_n)$, а $h = delta$. Для прямоугольной трапеции $S = (a+b)h/2$, где *a, h = то же самое*, $b = f(x_{n+1})$. Программа слегка меняется:

```
var
   numOf                    : integer;
   fi,fip1                  : double;
   oldResult                : double;
```

```
      i                          : integer;
begin
     result:=0;

     if eps = 0 then begin
        numOf:=Floor((b-a)/delta);
        for i:=1 to numOf do begin
           fi:=F(a + i*delta);
           fip1:=F(a + (i+1)*delta);
           result:=result + delta*((fi+fip1)/2);
        end;
     end
     else begin
        repeat
           oldResult:=result;
           numOf:=Floor((b-a)/delta);
           result:=0;
           for i:=1 to numOf do begin
              fi:=F(a + i*delta);
              fip1:=F(a + (i+1)*delta);
              result:=result + delta*((fi+fip1)/2);
           end;
           delta:=delta/10;
        until Abs(oldResult-result) < Eps;
     end;
end;
```

Если код кажется вам скучным и затянутым – оптимизируйте. Лично мне всё нравится, за исключением вот этого delta:=delta/10; Нельзя ли задать вместо десятки параметр, или, что лучше, подбирать делитель динамически? Когда вы отработаете подбор делителя до совершенства, сравните метод прямоугольников и метод трапеций. В качестве критерия используйте количество вызовов функции. Однако, мы отвлеклись – вот сравнительная таблица эффективности обоих методов при равных шагах:

delta	*сумма м. прямоугольников*	*отклонение %*	*сумма м. трапеций*	*отклонение %*
0.1	2.949	10.64	3.146	4.7
0.01	3.264	1.10	3.286	0.42
0.001	3.296	0.12	3.298	0.06
0.0001	3.299	0.03		

Эффект особенно ощутим для приближений с грубым шагом. Оно и понятно, при основании 0.0001, какая разница, что на этом основании возвышается – прямоугольник или трапеция?

Разумеется, разум человеческий на формуле трапеций не остановился и выдумал формулу Симпсона – к одноимённому мультфильму персонаж отношения не имеет. Это было давно, в восемнадцатом веке, компьютеров не было, всё приходилось считать вот этими самыми руками, при свечах и гусиным пером. На всякий случай запомните, это вычисление элементарной площади, то, что было прямоугольником в методе прямоугольников и трапецией в методе трапеций:

$$S_i = \frac{delta}{6}(f(x_i) + 4f(\frac{x_i + x_{i+1}}{2}) + f(x_{i+1}))$$

К сожалению, математическое искусство стремительно обесценивается. Всё побеждает грубая вычислительная сила в лице тупого метода прямоугольников.

Определённые интегралы. Не только это

Разумеется, на этом интегралы не кончаются, не кончаются даже в смысле способов их программно вычислять. Потому что внезапно выясняется, что интегралы бывают очень разные.

Сначала двойные, тройные и вообще кратные интегралы. Двойные интегралы в мозг простого человека помещаются легко, тройные с некоторым усилием, далее труднее. Двойной интеграл – число. И найти это число программно часто проще, чем аналитически. В чём смысл двойного интеграла? Сначала вспомним, в чём смысл простого определённого интеграла. Есть кривая, мы движемся по оси X от a до b и вычисляем площадь над осью. Занудства ради, если кривая уходит вниз, под ось, то мы из площади сверху вычитаем площадь снизу. А двойной интеграл, это то же самое, но в пространстве.

Перечитайте Фихтенгольца, он занудный но понятный. Переслушайте любимую песню моей юности *Раскинулся вектор по модулю пять*.

Приложение А
Простая процедура для рисования графиков с уместными комментариями. Длинное приложение, но полезное. Но длинное

Далее вашему вниманию предлагается действительно очень полезный класс для рисования графиков. Простой, но незатейливый. Примитивный, но несложный. Короче, вам обязательно понравится. Комментарии после программного кода. Их, комментарии, никто не читает, но вы прочитайте, пожалуйста. А здесь предварительные вводные замечания общего характера.

Кстати, рекомендую, свеженькая книжечка:

Г.Е.Шилов «Как строить графики», М. Государственное издательство физико-математической литературы, 1959

Этот класс предназначен для отображения традиционных, даже, не побоюсь этого слова, *аналитических* функций, не в каноническом смысле, конечно. Имеется в виду, что на вход функции подаётся действительный аргумент и результатом функции является действительное число. Иными словами, для отображения сугубо целочисленных функций класс не вполне походит. Он их нарисует, конечно, но не вполне идеально и заказчику картинка не понравится. Это хорошо видно в главе о теории вероятностей, на примере карточной колоды.

Но вы можете всё исправить, это же класс – просто перепишите виртуальный метод. Ещё обратите внимание на свойства. Там, где метод, реализующий запись (директива **write**), обязательно должны быть проверки на корректность параметров. Но мы же пишем для себя, можно и так?

Ведь мы играем не из денег,
А чтобы вечность проводить. © Ас Пушкин

А вообще, должен вам признаться, главное в написании любого класса – определить его интерфейс (**interface**). Реализация (**implementation**) гораздо проще, и ошибки там гораздо проще исправлять.

Интерфейс

```
unit SiGraphic;  //  18.09.2016
                 //  27.09.2016
{-----------------------------------------------------------------}
interface
   uses
      Types, Graphics;
{-----------------------------------------------------------------}
   const
      maxNum = 16;

   type
      TSiMathFunc = function(     x : single) : single;

   type
      TSiGraphic = class
       private
         fNumOf                : integer;
         fC                    : TCanvas;
         fBegX                 : single;
         fMsmX                 : single;
         fBegY                 : single;
         fMsmY                 : single;
         fRect                 : TRect;
         fColor                : array[1..maxNum] of TColor;
         fWidth                : array[1..maxNum] of integer;
         fF                    : array[1..maxNum] of TSiMathFunc;

         function GetF(     ind : integer) : TSiMathFunc;
         procedure SetF(     ind   : integer;
                            value : TSiMathFunc);
         function GetColor(     ind : integer) : TColor;
         procedure SetColor(     ind   : integer;
                                 value : TColor);
         function GetWidth(     ind : integer) : integer;
         procedure SetWidth(     ind   : integer;
                                 value : integer);

       public
         property numOf : integer read fNumOf write fNumOf;
         property F[ind:integer] : TSiMathFunc read GetF write SetF;

         property C : TCanvas read fC write fC;
         property begX : single read fBegX write fBegX;
         property MsmX : single read fMsmX write fMsmX;
         property begY : single read fBegY write fBegY;
         property MsmY : single read fMsmY write fMsmY;
         property rect : TRect  read fRect write fRect;
         property color[ind:integer] : TColor read GetColor write
SetColor;
```

```
        property width[ind:integer] : integer read GetWidth write
SetWidth;

        constructor Create;
        destructor Destroy;  override;
        procedure AddF   (     wF     : TSiMathFunc);
        procedure AddFExt(     wF     : TSiMathFunc;
                               wColor : TColor;
                               wWidth : integer);

        procedure SetAll(     wBegX,wMsmX, wBegY,wmsmY : single;
                              wRect                    : TRect);
        procedure Field;                                    virtual;
        procedure AxisOnly;                                 virtual;
        procedure LegendVer(     marks : array of single;
                                 isOn  : boolean = true);  virtual;
        procedure LegendHor(     marks : array of single;
                                 isOn  : boolean = true);   virtual;
        procedure OneFunction(     num : integer);          virtual;
        procedure AllFunctions;                             virtual;

      private
        MXpix,MYpix           : single;
        x0,y0                 : integer;
      end;
```

Комментирую. Константа maxNum задаёт максимальное количество одновременно рисуемых на одном поле разных графиков разных функций. В реализации никаких ощутимых ограничений не заложено, но мне показалось, что шестнадцати кривых более, чем достаточно – а какое-то ограничение должно быть, в нашей реализации. Функции не обязаны быть разными, это может быть и одна и та же функция, не в смысле математическом – это само собой разумеется, а в смысле программном. В этой конкретной версии класса в этой возможности смысла нет - масштабы и начала отсчёта для всех кривых совпадают. Нельзя отобразить одну и ту же кривую в разных масштабах, а иногда хочется.

TSiMathFunc – это объявление нашей функции как процедурного типа. Сразу после секции реализации будет простой пример, как этим пользоваться. Все методы, для которых это имеет смысл, объявлены как виртуальные. Простой пример на эту тему тоже чуть дальше.

Свойства. C: TCanvas – на чём рисовать. Color, width – цвет и ширина линий, свойства индексированные. BegX, BegY – начала отсчётов в физических единицах. MsmX, MsmY – масштабы в физических единицах на сантиметр экрана (разумеется, очень приблизительно).

Винтовки Мосина, я думаю, вас не заинтересуют © Брат-2

В смысле, конструктор с деструктором вас вряд ли заинтересуют, поэтому переходим к методам. AddF просто добавляет функцию для отображения. AddFExt тоже добавляет, но не просто, а вместе с характеристиками линии, которой она будет нарисована. Метод SetAll задаёт размеры поля, начала отсчётов по осям и масштабы, как уже замечено, для всех функция сразу. И, как легко подметить, все эти методы не производят никакого видимого эффекта.

Field рисует поле вывода, просто поле. А что там вообще рисовать? Не знаю, главное – метод виртуальный, то есть его можно переписать в порождённом классе под свои извращённые потребности. AxisOnly, как следует из названия, только рисует оси координат. LegendVer и LegendHor наносят числовую разметку на оси, стандартно – линейную, соответственно параметрам метода SetAll. Оставшиеся два метода, OneFunction и AllFunctions, как нетрудно догадаться, рисуют графики.

Реализация

Теперь реализация, она относительно большая, а мне не нравятся в книгах программные коды на несколько страниц. Поэтому реализацию разобьём на части и будем перебивать комментариями.

```
implementation
   uses
      SysUtils;
   const
      pSm = 30;
{=======================   TSiGraphic   =======================}
constructor TSiGraphic.Create;
begin
     inherited;

     fNumOf:=0;
     C:=nil;

     fBegX:=0;
     fMsmX:=0;
     fBegY:=0;
     fMsmY:=0;

     fRect.Left:=0; fRect.Right:=0; fRect.Top:=0; fRect.Bottom:=0;
```

```
     FillChar( fColor, SizeOf(fColor), #0);
     FillChar( fWidth, SizeOf(fWidth), #0);
end;
{-----------------------------------------------------------------}
destructor TSiGraphic.Destroy;
begin
     inherited;
end;
```

Всё, что здесь заслуживает доброго слова – константа pSm. Это предполагаемое количество пикселов на сантиметр. *Предполагаемым* оно является втом смысле, что наверняка мы никогда знать его не можем – обдумайте. Если очень хочется, можно её/его перенести в секцию **interface** и даже оформить в виде свойства.

```
procedure TSiGraphic.AddF(     wF     : TSiMathFunc);
begin
     if numOf < maxNum then begin
        numOf:=numOf + 1;
        fF[numOf]:=wF;
     end;
end;
{-----------------------------------------------------------------}
procedure TSiGraphic.AddFExt(    wF     : TSiMathFunc;
                                 wColor : TColor;
                                 wWidth : integer);
begin
     AddF( wF);
     color[numOf]:=wColor;
     width[numOf]:=wWidth;
end;
{-----------------------------------------------------------------}
procedure TSiGraphic.SetAll(     wBegX,wMsmX, wBegY,wMsmY : single;
                                 wRect                    : TRect);
begin
     begX:=wBegX;
     MsmX:=wMsmX;
     begY:=wBegY;
     MsmY:=wMsmY;
     rect:=wRect;
end;
```

Здесь, как ни странно, всё должно быть понятно и так.

```
procedure TSiGraphic.Field;
begin
     if C = nil then Exit;

     MXpix:=MsmX / Psm;
     MYpix:=MsmY / Psm;
```

```
     C.Brush.Color:=clWhite;
     C.FillRect(C.ClipRect);

     C.Pen.Width:=1;
     C.Pen.Style:=psDot;
     C.Rectangle(rect);

     x0:=rect.Left;
     y0:=(rect.Top+((rect.bottom-rect.Top) div 2))+ Round(begY/MYpix);
end;
```

Да, применение Exit идеологически невыдержанно, однако оно уменьшает вложенность на один уровень.

```
procedure TSiGraphic.AxisOnly;
begin
     C.Pen.Style:=psSolid;
     C.Pen.Width:=1;

     x0:=rect.Left;
     y0:=(rect.Top+((rect.bottom-rect.Top)div 2)) + Round(begY/MYpix);

     C.MoveTo(x0, y0);
     C.LineTo(x0+rect.Right, y0);

     C.MoveTo(rect.Left, rect.Top);
     C.LineTo(rect.Left, rect.Bottom);
end;
{-------------------------------------------------------------}
procedure TSiGraphic.LegendVer(     marks : array of single;
                                    isOn  : boolean = true);
   const
      dx = -20;
      dy =  -5;
   var
      whY                      : integer;
      i                        : integer;
begin
     if not isOn then Exit;

     C.TextOut( x0-dx, y0-dy, FloatToStrF(begY, ffGeneral, 3,1));

     for i:=Low(marks) to High(marks) do begin
        whY:=y0-Round((marks[i]-BegY)/MYpix);
        C.TextOut(x0+dx,whY+dy,FloatToStrF(marks[i],ffGeneral, 3,1));
        C.MoveTo(x0-5,whY);  C.LineTo(x0+5,whY);
     end;
end;
```

Обратите внимание – отметки по оси задаются явным способом, то есть если мы хотим надписи в точках 0, 3.14, 10, 99, то так их ручками и задаём.

Я долго думал, и решил, что так будет лучше. Константы dx и dy нужны для того, чтобы числа не налезали на оси и риски на них. А локальные они потому, что никому вне этой процедуры они не нужны.

```
procedure TSiGraphic.LegendHor(     marks : array of single;
                                    isOn  : boolean = true);
   const
      dx =  -5;
      dy =  10;
   var
      whX                       : integer;
      i                         : integer;
begin
     if not isOn then Exit;

     for i:=Low(marks) to High(marks) do begin
        whX:=x0+Round((marks[i]-BegX)/MXpix);
        C.TextOut(WhX+dx,y0+dy,FloatToStrF(marks[i],ffGeneral, 3,1));
        C.MoveTo(whX,y0-5);  C.LineTo(whX,y0+5);
     end;
end;
```

Константы слегка мутировали. Подобраны опытным путём. Если не трудно, выведите их математически и обоснуйте.

Обоснуй!
От обоснуя слышу! © Шутка юмора

```
procedure TSiGraphic.OneFunction(     num : integer);
   var
      oneF                      : TSiMathFunc;
      x,y                       : single;
      xP,yP                     : integer;
      xPlast, yPlast            : integer;
      i                         : integer;
begin
     C.Pen.Width:=width[num];
     C.Pen.Style:=psSolid;
     C.Pen.Color:=color[num];
     xPlast:=0; yPlast:=0;

     for i:=1 to (rect.Right - rect.Left) do begin
        x:=begX + (i-1)*MXpix;
        oneF:=F[num];
        y:=oneF(x) - begY;

        xP:=Round((x-begX)/MXpix);
        yP:=-Round(y/MYpix);

        if i = 1 then begin
```

```
            C.Pixels[x0+xP, y0+yP]:=color[num];
         end else
         if i >= 2 then begin
            C.MoveTo(x0+xPlast, y0+yPlast);
            C.LineTo(x0+xP, y0+yP);
         end;

         xPlast:=xP;
         yPlast:=yP;
      end;
end;
{-------------------------------------------------------------}
procedure TSiGraphic.AllFunctions;
   var
      i                          : integer;
begin
      for i:=1 to numOf do begin
          OneFunction(i);
      end;
end;
```

А здесь всё самое главное. Зачем нужна промежуточная переменная oneF? А затем, что без неё транслятор не пропускает. Это объяснению не подлежит, а остальное понять можно, при желании. А подробно я это объяснял в моей предыдущей книге.

```
function TSiGraphic.GetF(     ind : integer) : TSiMathFunc;
begin
      if (ind>=1) and (ind<=numOf)
         then result:=fF[ind]
         else result:=nil;
end;
{.............................................................}
procedure TSiGraphic.SetF(     ind   : integer;
                               value : TSiMathFunc);
begin
      if (ind>=1) and (ind<=numOf) then begin
         fF[ind]:=value;
      end;
end;
{-------------------------------------------------------------}
function TSiGraphic.GetColor(     ind : integer) : TColor;
begin
      if (ind>=1) and (ind<=numOf)
         then result:=fColor[ind]
         else result:=clBlack;
end;
{.............................................................}
procedure TSiGraphic.SetColor(     ind   : integer;
                                   value : TColor);
begin
      if (ind>=1) and (ind<=numOf) then begin
```

```
        fColor[ind]:=value;
     end;
end;
{-------------------------------------------------------------}
function TSiGraphic.GetWidth(     ind : integer) : integer;
begin
     if (ind>=1) and (ind<=numOf)
        then result:=fWidth[ind]
        else result:=1;
end;
{.............................................................}
procedure TSiGraphic.SetWidth(     ind   : integer;
                                   value : integer);
begin
     if (ind>=1) and (ind<=numOf) then begin
        fWidth[ind]:=value;
     end;
end;
```

Здесь скучное, но обязательное – реализация **read** и **write** clause для свойств.

Демонстрация применения и перспективы развития

Теперь простой пример, как этим богатством пользоваться. Сначала определим функции, пусть это будут sin(x), sin(2x), sin(3x), соответственно реализованные:

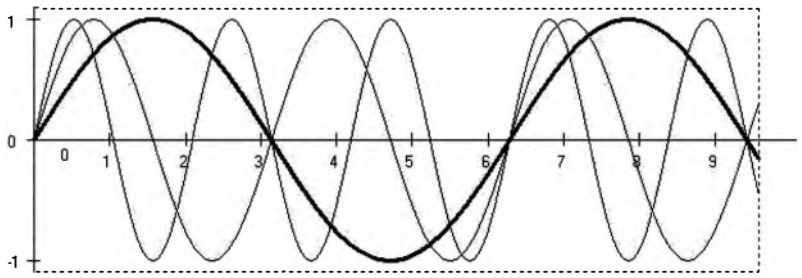
```
function MathSin(     x : single) : single;
begin
     result:=Sin(x);
end;
function MathSin2(     x : single) : single;
begin
     result:=Sin(2*x);
end;
function MathSin3(     x : single) : single;
begin
     result:=Sin(3*x);
end;
```

А вот так всё это мы применяем:

```
     G:=TSiGraphic.Create;
     G.C:=Canvas;

     skoka:=36;
     GetRandomDeckArray( DA, skoka);
```

```
G.AddF(DeckRand);
G.width[1]:=1;
G.SetAll( 0, 2, 0, skoka/36, Rect(50,50, 600,400));

G.Field;
G.AxisOnly;
G.LegendVer( [1,2,3,3,4,5,6], false);
G.LegendHor( [9,18,27,36], false);
G.AllFunctions;

G.Free;
```

А вот что на выходе:

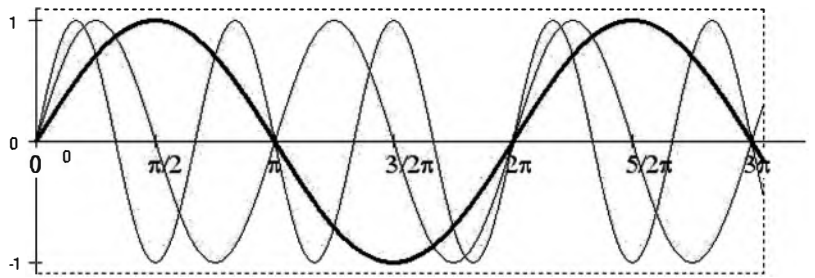


Хорошо ли это? В целом неплохо, но смущает разметка по горизонтали (оси абсцисс). Хотелось бы, как математически принято что-то вроде π, 2π, 3π… Легко. Напишем порождённый класс с переписанным методом LegendHor.

```
   type
      TSiGraphicTrig = class(TSiGraphic)
       public
          procedure LegendHor(     marks : array of single;
                                   isOn   : boolean = true);   override;
      end;

procedure TSiGraphicTrig.LegendHor(     marks : array of single;
                                        isOn  : boolean = true);
   var
      whX                      : integer;
      pinadva                  : integer;
      stroka                   : string;
begin
     pinadva:=Round( begX/(pi/2));

     C.Font.Name:='Symbol';
     C.Font.Size:=14;
```

```
    repeat
       whX:=x0 + Round((-begX+pinadva*pi/2)/MXpix);
       C.MoveTo( whX, y0-5);
       C.LineTo( whX, y0+5);

       if (pinadva mod 2) = 0 then begin
          if pinadva = 0 then stroka:='0' else
          if pinadva = 2 then stroka:=#112
          else stroka:=IntToStr(pinadva div 2) + #112;
       end
       else begin
          if pinadva = 1
             then stroka:=#112'/2'
             else stroka:=IntToStr(pinadva) + '/2'#112;
       end;
       C.TextOut( whX-5, y0+5, stroka);
       pinadva:=pinadva + 1;
    until (whX >= x0 + rect.Right);
end;
```

Если надолго задуматься, то всё будет сразу понятно. А теперь результат:



Нули в начале координат – атавизм. Самостоятельно найдите, что надо поправить и какой метод переписать, чтобы от них избавиться.

Что с этим можно и нужно делать дальше? Первое, на что я уже намекал. Для каждой функции должна быть возможность задать свои начала отсчёта и свои масштабы. Даже если это одна и та же функция. Второе, менее очевидное. Наш класс плохо подходит для рисования функций от целочисленного аргумента. То есть, он их, безусловно отобразит, но как-то неубедительно, неэротично. Подумайте о возможных способах улучшения ситуации.

Приложение В
Просто колода карт.
Полезна для простых опытов из теории вероятностей

Если мы вытащим случайную карту из колоды, какой она будет? Легко ли написать программу, эмулирующую (красивое слово!) эту операцию. Причём аккуратно, соблюдая все постулаты Теории Вероятности? Легко, причём очень легко. Наша программа может каждый раз возвращать Бубновый Туз (БТ), при каждом её, программы, запуске. Почему бы и нет? Если мы извлекаем из колоды одну, случайную, карту, почему бы ей не оказаться БТ? При каждом отдельно взятом запуске программы, само собой. Реализация функциональности обещает быть несложной и, что важнее, очень надёжной:

```
function ReallySimpleAsolutelyRandomManyCardsGeneretor
         (     skoka : integer) : TCard;
   var
      i                          : integer;
begin
     for i:=1 to skoka do begin
        result.suit:=diamonds;
        result.rank:=Ace;

        ShowMessage(    rankNames[result.rank]    +    '    of    '    +
     suitNames[result.suit]);
     end;
end;
```

А это заранее подготовленный тест:

```
ReallySimpleAsolutelyRandomManyCardsGeneretor( 5);
```

Разумеется, у внимательного читателя возникли два вопроса – один очевидный, другой – нетривиальный.

Вопрос номер один, лёгкий – а где, собственно объявления многочисленных используемых типов и констант – TCard, diamonds, Ace, rankNames, suitNames? Не надо тревожиться, ситуация под контролем, всё будет чуть позже, в полнофункциональной версии модуля, всего за 229.95 руб., а для покупателей этой книги, само собой, бесплатно.

Вопрос номер две, сложный – а можно ли рассматривать результаты нашей процедуры как *действительно* случайные карты? Буду бить козырем, то есть сошлюсь на непререкаемый авторитет:

Любая конкретная последовательность, содержащая миллион цифр, так же вероятна, как и любая другая. Если мы выберем миллион цифр наудачу и если окажется, что первые 999999 из них – нули, то вероятность того, что последняя цифра в этой последовательности – также нуль, всё ещё останется точно равной одной десятой, в истинно случайной ситуации. Это утверждение большинству кажется парадоксальным, однако оно не противоречит реальности.
© Кнут, том второй

Если от нашей программы требуется изобразить результат извлечения двух случайных карт, то два БТ будут выглядеть странно, даже при условии возвращения извлеченной карты в колоду. Говоря по другому, наша программа приобретает нетривиальный вид.

А теперь поставим вопрос – чего мы от нашей программы ждём? Какие манипуляции с колодой она должна выполнять, не нарушая законов вселенной и формулировки Е.С. Вентцель , чисто немецкая фамилия, а вы что подумали, если что? – хотя я уже разъяснял.

От нашей колоды требуется немного – если мы попросим её, колоду, извлечь несколько случайных карт, то они, карты, должны быть случайными, или, по крайней мере, быть псевдослучайными – то есть выглядеть как случайные. Это уже неплохо. Дальше есть варианты. Если мы попросим выдать две карты, то при этом первая карта может быть возвращена в колоду. А может и не быть возвращена. По простому – если мы получили ТП в первый раз, то, при возврате карты в колоду, он может к нам вернуться и во второй раз. Если карту в колоду вернуть уже нельзя то ТП нас не ожидает, максимум КП – король пик.

Ещё очень неплохо иметь возможность выводить результаты не в виде удобных для программиста, но никому не понятных внутренних кодировок, а как-то проще, по-человечески, что ли. То есть не «1-9» и даже не «ТП», а «Туз Пик». Значит, надо реализовать несложную, но в целом занудную, процедуру конвертации внутренних типов в человеческие.

Поскольку мы находимся Нев основной части книги, а в приложении, я не буду отображать на бумаге процесс формирования в моем разуме кода программы, а представлю сразу готовый результат, в два приёма.

Сначала интерфейс. Сейчас мы имеем тот случай, когда интерфейс едва ли не длиннее реализации.

```
unit SiDeck;  //  18.09.2016
              //  09.07.2017
{-------------------------------------------------------------}
interface
   uses
      Classes;
{-------------------------------------------------------------}
   const
      numOfSuits   = 4;
      numOfRanks   = 13;
      numOfCards   = numOfSuits * numOfRanks;

   type
      TSuit = (noSuit, spades, clubs, diamonds, hearts);
      TRank = (noRank,two,three,four,five,six,seven,eight,nine,ten,
Jack,Queen,King,Ace);
      TCard = packed record
         suit                 : TSuit;
         rank                 : TRank;
         inDeck               : boolean;
      end;

   const
      noneCard : TCard = (suit:noSuit; rank:noRank; inDeck:false);

   type
      TDeckArray = array[1..numOfCards] of integer;

   const
      suitNames               : array[noSuit..hearts] of string =
                                ( 'no',
                                  'spades  ',
                                  'clubs   ',
                                  'diamonds',
                                  'hearts  ');
      rankNames               : array[noRank..Ace] of string =
                                ('no',
'two','three','four','five','six','seven','eight','nine','ten',
'Jack','Queen','King','Ace');
      suitNamesShort          : array[noSuit..hearts] of string =
                                ( 'no',
                                  'S',
                                  'C',
```

```
                                    'D',
                                    'H');
      rankNamesShort           : array[noRank..Ace] of string =
                                   ('no',
'2','3','4','5','6','7','8','9','10', 'J', 'Q', 'K', 'A');

   type
      TDeck = class
       private
         function GetNumNominal : integer;
         function GetNumOf : integer;

       public
         constructor Create;
         destructor Destroy;  override;

         property numNominal : integer read GetNumNominal;
         property numOf      : integer read GetNumOf;

         procedure RandomCardWithReturn( var card     : TCard);
         procedure RandomCardWithoutReturn( var card     : TCard);

       private
         c                   : array[spades..hearts, two..Ace] of TCard;
      end;

function SameCard(     card1,card2 : TCard) : boolean;
function CardNumToCard(     num : integer) : TCard;
function CardToCardName(     card : TCard) : string;
function CardNumToCardName(     num : integer) : string;
```

Комментарии. Колода предполагается большая, 52 карты. Если нам понадобится 36, можно написать порождённый класс. Хотя лично мне кажется более удобным вариант добавить параметр в конструктор – тип колоды. В типах и константах разобраться несложно, если помнить как это всё называется по-английски. По-английски оно всё не потому, что я выпендриваюсь, а потому, что моё обострённое эстетическое чувство не допускает писать латиницей что-то вроде *semerka, Tuz*, а только *seven*, *Ace*.

// Ностальгия
Когда я программировал на языке Pascal для отечественного клона машины DEC PDP-1, там можно было писать идентификаторы по-русски – *семёрка, Туз.* Только потом препроцессор всё равно переводил это в латиницу – *semerka, Tuz*…
// конец Ностальгии

Функции в хвосте, не относящиеся к классу, обеспечивают ненавязчивый

сервис – позволяют обращаться к карте по номеру, предполагая некоторые соглашения по нумерации. Для чего это надо? Так бывает проще, внутри класса. Увидите, изучая код реализации. Кроме того, нельзя так просто взять, и сравнить на равенство две карты – пришлось написать функцию.

О самом классе – в нём нет почти ничего, но кое-что всё-таки есть. numNominal всегда возвращает то количество карт, которое в колоде было изначально – 52 в нашем случае. numOf – количество карт, которое в колоде осталось, если сколько-то из них извлекли и в колоду не вернули. Соответственно, имеем методы:

```
procedure RandomCardWithReturn( var card     : TCard);
procedure RandomCardWithoutReturn( var card     : TCard);
```

Первый метод извлекает карту из колоды с возвратом, второй с возвратом. То есть, для первого метода, если извлечён Туз Пик, то и при следующей попытке есть шанс, что это событие повторится. Для второго метода это исключено.

Теперь реализация класса:

```
constructor TDeck.Create;
   var
      card                    : TCard;
      i                       : TSuit;
      k                       : TRank;
begin
     inherited;

     Randomize;

     for i:=spades to hearts do begin
        for k:=two to Ace do begin
           card.suit:=TSuit(i);
           card.rank:=TRank(k);
           card.inDeck:=true;
           c[i,k]:=card;
        end;
     end;
end;
{-------------------------------------------------------------}
destructor TDeck.Destroy;
begin
     inherited;
end;
{-------------------------------------------------------------}
```

```
procedure TDeck.RandomCardWithReturn( var card     : TCard);
   var
      nCard                    : integer;
begin
     nCard:=Random(numNominal) + 1;
     card:=CardNumToCard(nCard);
end;
{-------------------------------------------------------------}
procedure TDeck.RandomCardWithoutReturn( var card     : TCard);
   var
      nCard                    : integer;
      nomer                    : integer;
      i                        : TSuit;
      k                        : TRank;
begin
     card:=noneCard;
     nCard:=Random(numOf) + 1;
     nomer:=0;

     for i:=spades to hearts do begin
        for k:=two to Ace do begin
           if c[i,k].inDeck then begin
              nomer:=nomer + 1;
              if nomer = nCard then begin
                 card:=c[i,k];
                 c[i,k].inDeck:=false;
              end;
           end;
        end;
     end;
end;
{------------   Set&Get procedures for prroperties   --------}
function TDeck.GetNumNominal : integer;
begin
     result:=numOfSuits*numOfRanks;
end;
{-------------------------------------------------------------}
function TDeck.GetNumOf : integer;
   var
      i                        : TSuit;
      k                        : TRank;
begin
     result:=0;

     for i:=spades to hearts do begin
        for k:=two to Ace do begin
           if c[i,k].inDeck
              then result:=result + 1;
        end;
     end;
end;
```

Некоторых умственных усилий потребовала только реализация извлечения карты без возврата. Ещё больших усилий понадобилось на тестирование этого продукта. Как убедиться, что карты не повторяются? Сравнивать во вложенном цикле? Сначала отсортировать? Можно, но сначала надо написать функцию сравнения двух карт – какая больше, какая меньше. Я пошел простейшим, хотя и халтурным путём – затребовал из колоды поочерёдно 53 карты, вот так:

```
D:=TDeck.Create;

stroka:='';
for i:=1 to 53 do begin
   D.RandomCardWithoutReturn(card);
   stroka:=stroka + CardToCardName(card) + #13#10;
end;

ShowMessage(stroka);

D.Free;
```

Последняя, пятьдесят третья карта, оказалась специально предусмотренной для этого случая инициализированной константой noneCard. Перевожу – это означает, что других карт в колоде не нашлось. С этого момента я счёл класс годным для применения. А теперь, для полного счастья, обещанные сервисные функции:

```
function SameCard(     card1,card2 : TCard) : boolean;
begin
     result:=(card1.suit = card2.suit) and (card1.rank = card2.rank);
end;
{-------------------------------------------------------------}
function CardNumToCard(     num : integer) : TCard;
   var
      suit                      : TSuit;
      rank                      : TRank;
begin
     suit:=TSuit( ((num - 1) div numOfRanks) + 1);
     rank:=TRank( ((num - 1) mod numOfRanks) + 1);

     result.suit:=suit;
     result.rank:=rank;
end;
{-------------------------------------------------------------}
function CardToCardName(     card : TCard) : string;
begin
     result:=rankNames[card.rank] + ' ' + suitNames[card.suit];
end;
{-------------------------------------------------------------}
```

```
function CardNumToCardName(     num : integer) : string;
begin
     result:=CardToCardName( CardNumToCard( num));
end;
```

Приложение С
Та Самая Гравюра из Невского Альманаха
И бонус

И обещанный бонус. Картинок было две, по крайней мере Александр Сергеич откликнулся на две:

Вот, перешед чрез мост Кокушкин,
Опёршись жопой о гранит,
Сам Александр Сергеич Пушкин
С мосье Онегиным стоит.
Не удостаивая взглядом
Твердыню власти роковой,
Он к крепости стал гордо задом –
Не плюй в колодец, милый мой.

Вторая гравюра намного слабее, на мой утончённый вкус.

Приложение D
Баллада о синусе

Мораль басни обычно приводится в конце, сейчас будет наоборот. Во-первых, это не басня, а быль – жанр другой. Во-вторых, мораль такова, что уместна в любом контексте, хоть спереди, хоть сзади. Мораль в том, что если вы в чём-то уверены, то должны отстаивать свою точку зрения до конца, особенно если это для вас выгодно. Если мы обнаруживаем, что наша точка зрения заводит нас в тупик и в более плохие места, наша задача от точки зрения отказаться и переобуться в воздухе. Основной принцип программиста – программист всегда прав.

Было это давно. Из участников, активных и пассивных, никого уже нет. Не в смысле померли, хотя, возможно и померли. Нет, просто сменили сферу деятельности или страну, или ориентацию, или всё.
Мы разрабатывали сложный агрегат. Кто-то паял железо, а мы программировали программы. Всё было впервые. Дисплей был плазменный, впервые. Кстати, вертикального расположения, то есть по оси Y точек было больше, чем по оси X, что несколько необычно и сейчас.

Генератор синуса генерировал синус, провода от генератора шли на АЦП (алфавитно-цифровой преобразователь). Провода от АЦП шли на управляющий *всем* компьютер. Компьютер был советским клоном PDP-1, в отличие от монитора, АЦП, плоттера и прочего – всё это было абсолютно отечественной разработки. Особенно отечественность чувствовалась в электростатическом плоттере – для работы в него требовалось залить авиационный керосин. В комплекте с плазменным монитором, наоборот, прибыли два бандеровца для наладки. Это я сейчас осознаю, что они были натуральные бандеровцы, тогда нам что-то парили про дружбу народов на всех трёх телевизионных каналах, хотя смутные сомнения уже терзали. Разговаривали эти смуглые хлопы меж себя на какой-то *гваре*, которой не понимали даже местные, вполне натурализовавшиеся украинцы. Что такое *гвара*, посмотрите статью в Википедии, и задумайтесь, троллинг это или как?

Но, отдадим должное, дисплей под их руководством работал исправно. Вот что Советская Власть животворящая делала!

Дисплей работал – как один из элементов системы. Система в целом не работала. При первом же запуске по экрану (оранжевому) вместо легко узнаваемого синуса пополз снизу вверх *трудно* узнаваемый синус. То есть понятно, что это *был* график какой-то периодической функции, отдалённо напоминавшеё синус. Но, к сожалению, именно, что был и именно, что напоминавшей. Кривая с примерно равными интервалами по высоте то достигала максимума, то уходила в минимум. При этом кривую не то, чтобы плющило и колбасило, а, скорее что-то подвергало лечению электрошоком.

Я тогда был ещё молодой, хотя и опытный. Я отвечал не за всё, а только за визуализацию данных, поэтому именно меня и призвали к ответу – что за фигня? Хотя я был и неопытным программистом, но кое-что я понимал. Я понимал, что надо переводить стрелки. Если нельзя перевести на конкретное лицо, то хотя бы на законы природы.

Я убедительно изложил причины, по которым синус в принципе не мог отобразиться не то, чтобы идеально, но хотя бы правильно. Причинами были объявлены потеря точности при аналого-цифровых преобразованиях, ограниченная длина машинного слова, особенности округления в используемом языке программирования, недостаточное разрешение дисплея (800x600 между прочим), а также недостатки применяемого алгоритма рисования прямых линий – у монитора по сути была только одна прошитая команда – нарисовать/стереть точку в заданной позиции. Ещё можно было стереть всё. Вот это *всё,* выделенное курсивом, и губило просто всё и без курсива. Повторюсь, я излагал очень убедительно.

В разгар моей речи кто-то заслушавшийся сделал неловкое движение и выдрал провод, шедший от генератора синуса на АЦП. Ничего не произошло – по экрану ползла та же злобная пародия на синус, страдающий Паркинсоном – хотя связи уже не было. Элементарное расследование обнаружило, что связи не было и раньше – точно так же и ещё раньше был оторван провод, идущий от АЦА к дисплею. Как только оба провода вернули на их законные места, по монитору плавно пополз идеально гладкий синус оранжевого цвета.

Говоря по-старому, в чём мораль? Или, формулируя по-другому, какие выводы я сделал?

Я обладаю значительным даром внушения и убеждения.
Если ты кому-то что-то впариваешь, желательно сохранять критичность к своим высказываниям.
Переобуваться на лету – высокое искусство.
Синус ни одна программа не испортит.

Приложение Е, печальное. Чего в книге нет, но могло быть

В самом начале я написал план книги. План был незатейлив – это были двенадцать строчек. Каждая строка – глава. Каждая глава – раздел математики. Через некоторое время обнаружилось, что одни главы пишутся быстро, другие медленно, а совсем другие вообще не пишутся. Так вот, далее список того, что здесь могло бы быть, но его нет. Но сначала маленькое вступление и провокационный вопрос - как вы относитесь к Дарвину и его теории естественного отбора?

Вопрос совсем не пустой и не тривиальный. Одни считают, что это полный бред и всеми раскопками опровергнуто – всё было создано сразу и вдруг – кем именно, не важно. Другие, которые как бы за Дарвина, тоже поделились. Есть сторонники Дарвина в натуре, то есть теории индивидуального отбора. А есть сторонник великого советского генетика академика Лысенко, то есть теории группового отбора. Поскольку ни то, ни другой, не американцы, шансов у них не было. И тогда на поверхность выполз совершенно американский профессор Докинз и предъявил отбор на уровне гена, в своей книге *Эгоистичный ген*, рекомендую к прочтению. К сожалению, не все слова в этой книге лично для меня оказались понятными. Даже и не все предложения и не все абзацы. Профессор Докинз это телепатически почуял и пошёл читателю-математику навстречу.

Следующая его книга называлась *Слепой часовщик*. Там слова были понятны все. Профессор взял учебник по теории вероятностей, учебник по теории игр и учебник по теории графов. Затем он их прочёл – не целиком, конечно, но в пределах первой главы. У профессора наступило просветление. После чего он прочёл учебник для начинающих по языку программирования Basic. Прочёл, разумеется, целиком – в этом случае от первой главы толку ноль. И сразу стал всё это применять к естественному отбору. Через некоторое время он даже понял, что программа должна куда-то сохранять результаты своей работы, а исходные тексты программ неплохо резервировать.

Я никаким образом не иронизирую. Генетик прочел несколько популярных книжек по математике и внезапно обнаружил сколько изо всей этой математики имеет явное и полезное применение в генетике. А почему программисты этих книжек не читают? Вопрос риторический, в смысле – ответ и дураку понятен.

Так вот, теперь о нескольких темах, о чём программисту прочитать неплохо бы. Их того, что перечислил профессор, с теорией вероятностей вы уже знакомы. Об остальном и поговорим.

Матрицы и Высшая Алгебра

Высшая алгебра вообще не о том и программисту не нужна. Не нужна совершенно. Но часто под видом высшей *алгебры* пытаются продать банальные *матрицы*. В этом есть *свой* резон - матрица один из объектов высшей алгебры. Матрицы подозрительно похожи на массивы. Применимы ли операции с матрицами к массивам и когда?

Системы линейных уравнений – что это такое и как их решать. Это отдельная отрасль древа математической науки. Называется это – *линейное программирование*. Программирование это, напоминаю, к программированию в человеческом смысле отношения не имеет. Для чего нужно? Для всего, в основном для планирования. Раньше составляли планы для предприятий. На вход сырьё, есть станки с известной производительностью, есть трудящиеся для станков. Как из всего этого произвести продукцию наибольшей стоимости? Такого рода программы крутились на каждом советском предприятии, толку было очень мало.

Тем не менее, знать это надо. Буквально под рукой валяется простая книжка. Но старая. Но простая. Потому и простая, что старая:

А.С. Солодовников, Введение в линейную алгебру и линейное программирование, М, Просвещение, 1966

Предназначено для педагогических институтов. Тогда это звучало почти как оскорбление. Ещё раз – это надо знать. Но об этом я уже пишу книгу.

Ещё напрашивается мысль – написать класс для матрицы. Сначала для матрицы квадратной, а затем класс порождённый – матрица прямоугольная. Или для частного случая – матрица-столбец-строка – предпочтительнее свой класс?

Графы и около

Тот же вопрос - а нужно ли это программисту вообще? В каждой второй книге чисто по программированию проектируют деревья в виде списков. Если язык программирования поддерживает ООП – Объектно Ориентированное Программирование – то пишут соответствующий класс.

Иногда надо. Даже часто надо, но только в форме банальных деревьев. Чем дерево отличается от графа, разберитесь сами.

Теория игр

Я уже рассказывал об играх в главах о теории вероятностей. Так вот, это всё не о том. Большинство тех игр совершенно не интересуют теорию игр.
// Воспоминания о тревожной молодости
как-то нашу группу занесло на специализацию по кафедре теории игр. Через полгода нас перебросили на кафедру вычислительной математики. Бывает. Это я всё к тому, что заведующий кафедрой теории игр был самый тоскливый, унылый и занудный мужик, который мне встречался. Это как в мемуарах Юрия Никулина – были у него два знакомых клоуна, по фамилиям – Преступляк и Кровопущенко.
// конец Воспоминаний
Для теории игр нужно как минимум два игрока, каждый из которых принимает *решение*. Понятно, что рулетка или кости сюда не относятся – второй игрок механическая железяка или унылый геометрический предмет. Очко тоже не сюда – второй игрок исключительно пассивен.

Вычислительная математика. Всё то же, вид сбоку. Почему всё то же, и почему сбоку?

Вообще-то, эта книга могла быть и книгой только о вычислительной математике, но мне показалось, что это будет очень скучно. Когда мы расстались с кафедрой теории игр, во главе с неописуемо скучным мужиком, нас перебросили на кафедру вычислительной математики. Люди там были хорошие, веселые, но вот сам предмет невообразимо скучен – на мой взгляд, по крайней мере.

Тот же вопрос - а нужно ли это программисту вообще? Нужно! Просто потому, что все остальные разделы прикладной математики сводятся к тому, чтобы что-то такое в конце концов посчитать. И вот здесь и выползает на сцену вычислительная математика. Важное замечание – составной и неотъемлемой часть вычислительно математики является – или, по крайней мере, являлось в годы моей учёбы – *математическое программирование*, то есть *методы оптимизации*. Этот раздел мне очень симпатичен и приятен, и об этом будет моя следующая книга.

Что ещё надо знать из вычислительной математики? Что такое интерполяция и что такое экстраполяция. И чем они отличаются между собой. Это важно. Это нужно. Это применяется часто. Другое дело, что знать это надо, а вот различать – совершенно не обязательно. Вот такой вот парадокс.

Аналитическая геометрия и немного человеческой

Сначала геометрия. Просто геометрия. Какая бывает геометрия вообще и в частности?

Бывает геометрия школьная. Сначала что-то о треугольниках, потом трапеции и круги. Даже не могу вспомнить, присутствовали ли в нашей школьной геометрии эллипсы. Потом, в самых старших классах, начиналась стереометрия – то, что не на плоскости, а в пространстве – куб, пирамида, призма. Ещё была тригонометрия. Но её никто за геометрию не считал, потому что там были только и сплошь формулы. Назовем это всё, до тригонометрии, *обычной* геометрией.

А нужна ли вообще программисту обычная геометрия? В поисках ответа я пошарил в ящиках стола и нашёл голубенький пластиковый стаканчик. Применив недорогую пластмассовую Линзу Френеля - а нужна ли программисту физика и знание о Линзе Френеля - я прочитал на её днище ёмкость – 200 мл. Наши люди, как всем известно, миллилитрами не пьют, наши люди пьют граммами, что смущает зарубежных барменов. Тем нем менее, объёмные величины русскому человеку в целом понятны и доступны, и нормальной единицей дискретизации кажутся сто грамм, они же миллилитры, привет Менделееву.

Разумеется, сто миллилитров равны в точности ста граммам, только если речь идёт о воде. После этой оговорки ставим несложную геометрическую задачу. Нет, не пересчёт на плотность C_2H_5OH Вспоминая об арифметике, когда я хотел проверить навыки своих учеников, то просил вычислить шестьдесят процентов от двадцати. Большинство было в шоке. Справедливости ради, не все из них уже успели поступить в университеты.

Наша задача гораздо хитрее. Объём стаканчика, как я уже сказал, 200 кубических сантиметров. Визуально посудина представляет собой усечённый конус. Верхний диаметр – 62 мм, нижний – 42. Высота – 85 миллиметров. До какой высоты следует набулькать туда жидкости, чтобы получились сакральные сто грамм, они же миллилитры.

Если вы рассчитали результат в уме – решив пропорцию, составив несложную систему уравнений, взяв определённый интеграл – вам будут рады в любой компании – во всех смыслах слова *компания*.

Ванечка на секунду благоговейно замер… Он всегда разливал водку, среди пьющих мужиков славился тем, что умел разливать на глаз любое количество спиртного с такой точностью, что промеры спичкой показывали абсолютную равность, и пьющие уважительно шептали: «Глаз-алмаз». Был случай, когда Ванечка разлил три бутылки «Столичной» в одиннадцать стаканов так, что в последней бутылке не осталось ни капельки, а стаканы содержали ровно по сто тридцать шесть граммов. © Виль Липатов «Серая мышь»

Я специально проверил, Липатов окончил *исторический* факультет, но здесь мыслит как математик – хотя бы как и средневековый.

Теперь забудьте всё это – *глаз-алмаз* - и решите задачу как программист, написав процедуру для тупого подбора ответа. Расширьте функциональность на любой сосуд, заданный аналитической кривой.

Да, работать со сложными стереометрическими фигурами, программисту приходится не каждый день. Хотя, вспомним Кеплера. Когда ему очень захотелось кушать, он написал реферат под названием *Стереометрия винных бочек*. Для программиста стереометрия слишком сложна, он живёт или в одной плоскости или сразу в N-мерном пространстве. Там, в одной плоскости, то есть в двумерном пространстве, мы и останемся.

Как именно отобразить геометрическую фигуру относится скорее к ведению аналитической геометрии. К геометрии школьной, если можно так её назвать, относится важный вопрос, достаточно ли у нас информации, чтобы однозначно нарисовать фигуру. Обратите внимание, в евклидовой геометрии для треугольника достаточно задать три его вершины. В программистской или аналитической геометрии, для вершин треугольника требуется задать их координаты. Нет координат – нет треугольника.

Теперь подумайте об этом с точки зрения программиста – как задать геометрическую фигуру- треугольник, четырёхугольник, круг, эллипс. Четырехугольник может быть конкретно параллелограммом, квадратом, ромбом и прочей неведомой зверушкой. А раз вы таки программист, подумайте о проверке – если вам велели нарисовать ромб, являются ли задающие его данные действительно ромбом. И, разумеется, слово *нарисовать* имеет здесь очень широкий неоднозначный смысл.

Ещё раз, программист имеет дело только с аналитической геометрией, не потому, что он может выбирать, а потому, что всё давно уже выбрано за него. Изобретатели всех возможных устройств отображения неизменно имели перед собой как идеал Декартову систему координат.

Когда я немного преподавал программирование старшим школьникам и младшим студентам, одной из задач, на вид простой, но вызывающей множество затруднений при тщательном рассмотрении, было решение обычного квадратного уравнения. А у нас для тренировки - простая с виду задача. Нарисуем четырёхугольник. Нет, если это слишком трудно, нарисуем хотя бы треугольник.

Только злобное требование – не рисуйте треугольник в Дельфи. Так всё немного упрощается – есть метод пера MoveTo, который ничего не рисует, но перемещает перо в требуемую точку, есть метод LineTo, который рисует линию и перемещает перо в конец линии. Здесь всё просто, запутаться трудно.

// сейчас Запла́чу
Когда я был маленький – нет, не как вы сейчас, а реально маленький – у мня были реальные бумажные книжки. И там были здоровенные тома под названием *Хочу всё знать!* И в одном из них была статья о роботах-черепашках, которые по командам свыше или по информации от своих датчиков-фотоэлементов перемещались по плоскости. Банально по нашему времени. Кроме того, у черепашки в заду було перо, которое она по команде могла опускать или подымать и, соответственно, рисовать или нет линию – прямую или не очень.

Никогда и не думал, что через много-много лет эта черепашка приползёт ко мне – хотя и слегка виртуальная.
// перестал Плакать

Аналитическая геометрия – два слова о ней позже - указывает нам, как именно рисовать. Геометрия простая объясняет, стоит ли вообще тратить силы на попытку рисования. Достаточно ли нам данных, не противоречат ли они между собой. Фигура, безусловно законная с точки зрения геометрии, может на конкретном физическом устройстве отображения выродиться в точку. Да, рассчитать многоугольник и нарисовать точку

стоит недорого, но таких неполноценных треугольников могут быть тысячи и десятки тысяч.

И ещё, обычная и заурядна задача – есть две фигуры, пусть это окружность и треугольник, простой случай. Найдите точки, в которых они пересекаются.

Теперь аналитическая геометрия. Просто для программистов

Начнем с того, с чего кончили в предыдущем разделе. Есть окружность и треугольник, найдите точки их пересечения. Это бывает надо, не то, чтобы каждый день надо, но бывает надо. Такие задачи проще решаются методами аналитической геометрии. А методы аналитической геометрии проще реализуются чисто компьютерными способами. То есть – компьютер есть – ОК, нет компьютера – сиди кукуй.

А что такое вообще аналитическая геометрия? Самое простое – это графики. О графиках я уже написал. Вообще говоря, аналитическая геометрия – это когда геометрическая фигура задаётся не умелыми жестами рук, а формулой. В подробностях я напишу об этом в своей следующей после следующей книге. Просто *следующая* моя книга будет о методах оптимизации.

В ожидании книги познакомьтесь системами координат – декартова, полярная, ещё какие-то наверное есть? Расскажите мне. И, разумеется, я не могу бросить вас вот так, на ровном месте, не рассказав чего-то супер полезного. Самое, кроме шуток, полезное для программиста из аналитической геометрии – это расстояние между точками. Пусть у нас есть n-мерное пространство. В нём заданы две точки. Проблема в том, что нет очевидного и простого способа эти точки задать. Очевидного – это чтобы даже мне сразу было всё понятно. Ладно, пусть первая точка будет $(a_a, a_2 \ldots a_n)$, а вторая $(b_a, b_2 \ldots b_n)$. Тогда расстояние между точками, которое, заметьте, величина уже безразмерная $s = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots (a_n - b_n)^2}$.

Приложение F, радостное.
Чем заняться на досуге, по главам

Просто вообще

Великий русский поэт Александр Сергеевич Пушкин поступил в Царскосельский Лицей. Принимали туда мальчиков дворянского сословия в возрасте 10-12 лет. Обучения продолжалось шесть лет, окончание Лицея приравнивалось к окончанию университета. А.С. учился слабовато и закончил по второму разряду. Особенно слабовато А.С. учился по математике. Ознакомьтесь с программой Лицея:

Начальный курс:
- *Математика:*
  - *арифметика, начиная с тройного правила;*
  - *простая геометрия;*
  - *алгебра до кубических уравнений;*
  - *тригонометрия прямолинейная.*
- *Прикладная математика:*
  - *основания механики;*
  - *математическая география.*

Окончательный курс:
- *Математика:*
  - *сферическая тригонометрия;*
  - *конические сечения;*
- *Прикладная математика:*
  - *статика;*
  - *гидравлика;*
  - *артиллерия;*
  - *фортификация.*

Оцените свои знания. Поставьте себе оценку, особливо по артиллерии и фортификации. Обдумайте пункт из программы по словесности: *Продолжение переводов с изъяснением идиотизмов.* Как у нас с этим?

Математическая логика.
Задания из Учебника логики для средней школы 1953-го года издания

1. *Страусы не летают, страусы — птицы. Какой следует вывод? Какая фигура?*

Фигура четвёртая, печальная © к/ф Любовь и голуби. Нет, фигура в другом смысле

2. Составьте суждения, равнозначные по содержанию следующим суждениям:

Защита отечества есть священный долг каждого гражданина СССР.

3. Рассмотрите следующие деления, и если в них есть ошибки, то укажите их [*выбраны математические примеры*]:

б) Числа делятся на целые, дробные, смешанные, именованные и отвлечённые.
в) Углы — прямые, тупые, острые, смежные, вертикальные.

3. В каком отношении находятся между собой следующие понятия:

строение, клуб, дом, изба, дворец, Дворец Советов,
Зимний дворец, беседка, хата?

Рик Гаско

Простая Математика для Простых Программистов

Ответственный за выпуск: **В. Митин**
Под редакцией: **Н. Комлева**
Обложка: **СОЛОН-Пресс**

По вопросам приобретения обращаться:
ООО «СОЛОН-Пресс»
123001, г. Москва, а/я 82
Телефоны: (495) 617-39-64, (495) 617-39-65
E-mail: kniga@solon-press.ru, www.solon-press.ru

ООО «СОЛОН-Пресс»
115487, г. Москва,
пр-кт Андропова, дом 38, помещение № 8, комната № 2.
Формат 60×88/16. Объем 16,25 п. л. Тираж 100 экз.