
Л.В. Городняя
НЕФОРМАЛЬНОЕ ВВЕДЕНИЕ
В ФУНКЦИОНАЛЬНОЕ
ПРОГРАММИРОВАНИЕ
(ОБРАЗЫ, АНАЛОГИИ, ПОДОБИЯ)
188

Российская академия наук
Сибирское отделение
Институт систем информатики
им. А.П. Ершова

Л.В. Городняя

НЕФОРМАЛЬНОЕ ВВЕДЕНИЕ В ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ
(ОБРАЗЫ, АНАЛОГИИ, ПОДОБИЯ)

]

Препринт
188

Новосибирск 2023

Рецензент — Т.А. Андреева

Препринт посвящен рассмотрению аналогий и подобий, знакомящих читателя с идеями функционального программирования, методикой символьной обработки данных и техникой программирования на уровне языка Pure Lisp, представляющего ядро многих языков функционального программирования. Символьная обработка информации является основой решения задач сверхточных вычислений, искусственного интеллекта, инженерного проектирования, лингвистической информатики, обработки больших объёмов данных и прочих задач. Она будет рассмотрена одновременно с техникой и базовыми методами функционального программирования. Упражнения можно выполнять на разных реализациях языка Lisp, включая Home Lisp, на котором, благодаря поддержке компьютерной графики и современному пользовательскому интерфейсу, можно выполнять и работу по иллюстративным моделям. Материал препринта рассчитан на интересующихся программированием методистов и преподавателей программирования, работающих со школьниками и студентами младших курсов.

Siberian Branch of the Russian Academy of Sciences
A.P. Ershov Institute of Informatics Systems

L.V. Gorodnyaya

INFORMAL INTRODUCTION TO FUNCTIONAL PROGRAMMING
(IMAGES, ANALOGIES, SIMILARITIES)

Preprint
188

Novosibirsk 2023

Reviewed by T.A. Andreyeva

The preprint is devoted to the consideration of analogies and similarities that present the ideas of functional programming, the methods of symbolic data processing and programming techniques at the core level of many functional programming languages represented by the Pure Lisp language. Symbolic information processing is the basis for solving the problems of ultraprecise computing, artificial intelligence, engineering design, linguistic computer science, big data processing, etc. It will be considered along with the technique and basic methods of functional programming. The exercises can be done on different implementations of the Lisp language, including Home Lisp, which can also be used to work on illustrative models thanks to its modern user interface and computer graphics support. The preprint is intended for the teachers of programming working with schoolchildren and undergraduate students interested in programming.

Л.В. Городняя Неформальное введение в функциональное программирование (образы, аналогии, подобия). Новосибирск, 2023. 90 с.

Аннотация. Препринт посвящен анализу и сопоставлению ряда аналогий, иллюстрирующих понятия и особенности языков и систем функционального программирования. Идеи и механизмы функционального программирования опираются на интуитивное понятие о ***функциях*** как о достаточно общем подходе к представлению и анализу решений сложных задач. Решения представляются в форме соответствий между аргументами и результатами функций. Механизм функций основательно изучен математиками, и это позволяет программистам наследовать выверенные построения, обладающие предельно высокой моделирующей и доказательной силой. Систематическое применение функционального программирования впервые достаточно ярко было продемонстрировано Джоном Маккарти и его учениками в методах реализации языка Lisp и программирования на этом языке. Наиболее очевидные из этих методов были успешно ассимилированы другими языками и системами современного программирования. Обычно про функциональное программирование вспоминают при смене технологий, когда возрастает роль аналитики, исследовательских задач, и происходят открытия новых приложений ИТ. Связь функционального программирования с математическими основами позволяет в тексте программы наследовать доказательность построения результата, если она достигнута, причем с использованием разных методов абстрагирования понятий решаемой задачи.

L.V. Gorodnyaya Informal introduction to functional programming (images, analogies, similarities). Novosibirsk, 2023. 90 p.

Abstract. The preprint is devoted to the analysis and comparison of a number of analogies illustrating the concepts and features of functional programming languages and systems. The ideas and mechanisms of functional programming are based on the intuitive concept of ***functions*** as a fairly general approach to the representation and analysis of complex problems solutions. They are presented in the form of a correspondence between arguments and function results. The mechanism of functions has been thoroughly studied by mathematicians, which allows programmers to inherit the verified constructions of an exceptionally high modeling and demonstrative power. John McCarthy and his students were the first to demonstrate vividly the systematic application of functional programming in the methods of Lisp implementation and programming. The most obvious of these methods have been successfully assimilated into other modern programming languages and systems. As a rule, functional programming is referred to when technologies change, the role of analytics and research tasks increases, and new IT applications are discovered. The connection of functional programming with mathematical foundations allows inheriting, within the program text, the argumentation of the result construction (if achieved) involving different methods of abstracting the concepts of the problem being solved.

Введение. Бум языкотворчества в области проблемно-ориентированных языков программирования (ЯП), знаменующий переход практики программирования от накопления опыта на уровне эффективных библиотечных модулей к накоплению удобных подъязыков, показывает важность измерения различий в определениях новых ЯП, улучшаемых версий программ и выборе инструментов для практических работ. Функциональное программирование (ФП) объединяет разные подходы к определению процессов вычисления на основе достаточно строгих абстрактных понятий и методов символьной обработки данных [1—7].

Изложение начинается с краткого описания особенностей ФП, далее приведены неформальные иллюстрации его принципов, следствий, практичных компромиссов и конкретизации к задачам параллельных вычислений (Таблица 1). В Приложении 1 для методистов и преподавателей приведена схема изучения ФП на основе таких иллюстраций, а в Приложении 2 — комплекты вопросов для контроля понимания изложенного материала.

Упражнения можно выполнять на разных реализациях языка Lisp, включая Home Lisp, на котором, благодаря поддержке компьютерной графики и современному пользовательскому интерфейсу, можно выполнять и работу по иллюстративным моделям [8].

1. Особенности и принципы

Особенности функционального программирования можно описать с помощью небольшого числа семантических **принципов**, которым следует программист на уровне языков программирования, и прагматической их **поддержки**, обеспеченной системами программирования [9]. На практике выделяются вытекающие из них **следствия**, позволяющие повышать продуктивность программирования, **расширения**,

дающие возможность повышать производительность программ, и **конкретизации**, обеспечивающие удобство программирования в отдельных областях приложения. Здесь в качестве области приложения рассматривается организация параллельных процессов (Таблица 1).

Таблица 1. Основные особенности функционального программирования

Особенности	Семантика	Прагматика
Принципы и их поддержка	Универсальность Самоприменимость Равноправие параметров функции	Гибкость границ памяти Неизменяемость данных в памяти Строгость результата функции
Следствия и удобство мысли	Мета-программирование Верификация программ Автономное развитие модулей программы	Бесконечность данных Обратимость процессов Унарность функций или сведение к функциям одного аргумента
Расширение и практичность	Типы данных Планы на будущее Схемы циклов — практичность	Мемоизация Восстановление и обработка состояний памяти Псевдофункции
Конкретизация и специфика параллелизма	Ленивые вычисления — частичность Пространство итераций Сравнение прогонов — повторы и идентичность	Распараллеливание программ — автоматизация Балансировка нагрузки Многоконтактные узлы

Многие особенности функционального программирования незаметно присутствуют в обычной жизни. Аналогия с ними или их подобие полезны как база для понимания семантики и прагматики языков и систем функционального программирования, что и является целью данного препринта [10]. При переходе к языкам программирования такие особенности называют идеями, свойствами или принципами.

2. Неформальные иллюстрации: принципы

Универсальность функций в программировании подобна универсальным устройствам в обычной жизни. Выигрыш от универсальных устройств можно оценить, сравнивая их пользу со специализированными приборами. Например, кухонный комбайн удобнее, чем отдельно мясорубка, миксер, блендер, соковыжималка, и т. п. Он меньше занимает места, чем набор таких приборов. Эта оценка не абсолютна. Если места достаточно, то удобнее иметь отдельные специальные приборы. Кроме того, применение комбайнов обычно сложнее, чем отдельных приборов. Не всегда удаётся разместить все насадки к комбайну так, чтобы в любой момент можно было взять любую, обеспечить к ним произвольный доступ.

Другой пример — универсальный дорожный складной нож, незаменимый инструмент в походных условиях. В дороге удобно, что в одном корпусе ножа имеется не только одно-два лезвия, но ещё и ножницы, шило, штопор, консервооткрыватель, а может и ещё что-нибудь, что может пригодиться совершенно неожиданно. Тем не менее, это выгодно не всегда. При большом числе вариантов нож становится хрупким, может неожиданно сломаться, в таких ножах бывает труднее воспользоваться отдельным нужным лезвием, чем в простом ноже с одним лезвием, а некоторые

прилады никогда не находят применения и просто остаются лишним грузом.

Ещё один пример — стиральный автомат. Нет ничего удобнее при дефиците свободного времени, отсутствии возможности контролировать процесс стирки по шагам. При этом отдельно простая машина и центрифуга дешевле. Примем во внимание, что можно более гибко управлять ходом стирки, в то время как общий процесс стирки автоматом довольно длинный, он выполняется полностью от начала до конца, и приходится ждать, пока он завершится.

Самоприменимость функций чаще всего использует рекурсию в представлении как программ, так и данных. Этот принцип ясно виден на популярном сувенире матрёшка (см. Приложение 1). Начнём с задачи изучения новой матрёшки. Мы смотрим на матрёшку и не знаем, сколько матрёшек внутри. Можно ли описать оптимальный процесс выяснения числа матрёшек? Конечно, — разнимаем внешнюю матрёшку, тогда внутренняя становится внешней. В ней будет на одну матрёшку меньше. Значит, можно 1 прибавить к числу матрёшек, выясняемому для этой внутренней матрёшки, и так далее. Если матрёшку разнять невозможно или внутри ничего нет, то число внутренних 0. Здесь мы сводим решение задачи к её решению на уменьшенной матрёшке, что и является сущностью рекурсии или самоприменимости. Тем не менее, можно поступить иначе – разнять матрёшку, а затем разбирать внутренние до тех пор, пока не обнаружится целостная матрёшка. Попутно при разборе можно считать число разобранных матрёшек. Разница в том, что переход к внутренней матрёшке не объявлен как решение той же самой задачи, не считая того, что случай с отсутствием внутренних матрёшек остался за бортом. Можно и по-другому — сначала всех матрёшек разобрать, потом полученные детали сосчитать, вычесть из полученного числа 1 и результат поделить пополам. А можно и просто поискать упаковку, на

ней возможно написано число внутренних матрёшек. Все эти методы являются решениями исходной задачи, но только первое решение соответствует принципу самоприменимости в функциональном программировании.

Можно рассмотреть другой пример — задачу с коробками и размещаемыми в них карточками. Пусть в двух коробках размещены карточки. Как узнать, в какой из них карточек больше? Первое решение: если обе коробочки пусты, то можно считать, что число карточек одинаково. Если одна коробочка пуста, а в другой карточки имеются, то в ней карточек больше. Если обе коробки заполнены, то из каждой вынимаем по одной карточке и сравниваем то, что осталось, как в начале решения этой задачи. А можно и по-другому — вынимаем карточки из коробок в отдельные стопки и сравниваем их по высоте. В более высокой стопке карточек больше, если они одинаковы по толщине. Можно и просто подсчитать число карточек в каждой коробке и сравнить полученные числа. Ещё одно решение: вынимаем из каждой коробки по одной карточке до тех пор, пока одна из коробок окажется пустой. Тогда во второй коробке карточек больше. Может оказаться, что и во второй коробке карточек не оставалось — значит, карточек одинаковое количество. Все решения правильны, но только первое использует принцип самоприменимости.

Похожее разнообразие методов можно видеть на задаче выяснения общего веса произвольного числа мешков с урожаем. Пусть при сборе урожая заполнено несколько мешков разного размера, и потому они разного веса. Требуется узнать, каков вес собранного урожая. Действуем без особых хитростей. Берём и взвешиваем первый попавшийся мешок, записываем полученное число. Это число прибавим к результату взвешивания оставшихся мешков. Их число конечно, значит процесс вскоре завершится. Типичный прием самоприменения. А можно поочерёдно взвешивать

мешки, потом суммировать полученные веса. Можно поступить и сложнее. Планируя предстоящую перевозку, упорядочиваем мешки по объёму, затем приступаем к взвешиванию, записываем веса и их суммируем — получили вес урожая. Возможен и радикально другой метод: находим большие весы и на них загружаем все мешки сразу, получаем суммарный вес.

Равноправие параметров можно увидеть на бытовых примерах, а не только в арифметике: «От перемены мест слагаемых сумма не меняется». Попробуем этот принцип распространить на рецептуру блюд, меню в ресторане, репертуар театра.

Насколько математическое правило перестановочности слагаемых можно распространить на репертуар театра? Конечно, не так уж важно, в каком порядке перечислены спектакли, были бы указаны даты и время. Бывает, что порядок удобнее, например, когда репертуар упорядочен по датам. А кому-то нравится, когда репертуар начинается с самых интересных спектаклей. Можно упорядочить и по алфавиту, легче искать. Так что равноправие параметров приходится обосновывать условиями их применения.

Другой пример — кулинарный рецепт. Почему от перестановки составляющих кулинарного рецепта его сущность не меняется? Всегда ли это справедливо? Да, это так при записи рецепта, но при изготовлении технология диктует порядок применения составляющих. Порядок записи и вычисления параметров функции можно изменять, но порядок их использования в процессе вычислений может иметь значение. Кроме того, не исключено, что удобнее упорядочить по весу или цене. А бывает, что хорошо, когда выделена группа близких по смыслу ингредиентов. И конечно, приходится порой учитывать порядок обязательности или дефицита составляющих.

В качестве ещё одного примера рассмотрим ресторанное меню. Меню обычно структурируют по категориям блюд. Насколько важен порядок блюд внутри категории? Обычно внутри категории блюд немного, поэтому их хорошо видно в любом порядке. Тем не менее, всё-таки удобно упорядочивать по ценам или по времени готовки. Можно не упорядочивать, но поддерживать стандартное расположение.

Прагматическая поддержка функционального программирования — это гибкость границ памяти, неизменяемость данных и строгость результата функции.

Гибкость границ памяти похожа на методы освобождения холодильника для разморозки, перепланировки расстановки мебели в комнате или поддержания свободной раковины для мытья посуды. Границы бесспорно нужны и важны, хотя иногда их приходится пересматривать в самый неожиданный момент. Именно по этой причине следует заранее позаботиться о подходящих в таком случае решениях.

Представим, что на кухне стоят два холодильника, хотя почему-то вся провизия хранится в одном, второй выключен и стоит пустым. Иногда работающий холодильник требуется быстро разморозить. Может ли быть полезным второй холодильник? Конечно! Быстро всё перемещаем в пустой холодильник и включаем его, а первый выключаем до полной разморозки. А если есть время, то можно и попутно выбросить устаревшее, а нужное рассортировать и упорядочить, сразу компактно размещая в резервном холодильнике. Хотя бывает, что лучше после разморозки вернуть всё как было на прежнее место, чтобы не искать на новых местах. При быстрой разморозке возможен возврат в первый холодильник с уплотнением, если надо.

Вспомним, насколько бывает досадно, если срочно надо вымыть фрукты, а раковина переполнена грязной посудой, накопившейся во время интенсивной готовки.

Можно ли пользоваться раковиной так, чтобы фрукты можно было ополоснуть в любой момент? Да, опытные хозяйки при интенсивной готовке моют посуду при каждом удобном случае, не ждут завершения процесса — важнее, чтобы раковина почти всегда была свободна. Хотя можно не слишком усердствовать, мыть время от времени понемногу, не допуская переполнения, половина не так уж мешает. Ещё проще при необходимости просто всё выставить на стол, помыть фрукты и посуду вернуть обратно или помыть не всё, а часть, лишь бы места хватило для мытья фруктов.

Ещё один пример даёт проблема перепланировки помещений. Допустим, предстоит многолюдный домашний праздник. Можно ли квартиру превратить в банкетный зал? Да, если заранее предусмотреть такую вероятность. В Болгарии часто в квартирах внутренние стены делают как раздвижные двери-купе. Достаточно убрать всё от стен и раздвинуть двери — большой зал готов. Хотя может и проще сделать несколько столов в соседних комнатах, но уже не будет такого чувства общности. Впрочем, отчего бы не разделить гостей на группы и праздновать по очереди, по частям? А ещё лучше пригласить всех в ресторан! Тогда ничего перепланировать не придётся.

Неизменяемость данных подобна стабильным расписаниям транспорта, занятий, привычному расположению мебели, знакомой проходимости дорог — информации о том, где открыты ворота. В чём выигрыш от стабильного расписания общественных автобусов? Прежде всего, в экономике: если расписание выдерживается с точностью до минуты, то возрастает число пассажиров, проезжающих одну-две остановки. Хотя можно подумать: пусть меняется расписание, лишь бы оно соблюдалось, и была уверенность, что автобус, в конце концов, придёт. Конечно, мало толку от постоянного расписания, если его не помнишь. Впрочем, сейчас его смотрят на сайтах. Также

удобно посмотреть на сайте, где сейчас автобус, ведь при любом расписании что-то может его нарушить.

Представьте, что кто-то взял и в вашем кабинете заменил ваш стол на новый, более модный. Почему это раздражает? Нарушены привычки ходьбы и пользования ящиками. Хотя многие скажут: «Ну и что? Вот и зря! Это же имидж, одобрение гостей! Ничего, привычки новые вскоре появятся. Сам просил улучшения условий труда, но не ожидал, что это будет неприятно».

Можете вспомнить, что вы почувствовали, когда на знакомом пути вдруг нарвались на новые запертые ворота? Ощущение грубого нарушения права двигаться привычным путём. Впрочем, если нет спешки, то можно просто вернуться и пойти другим путём или поискать лазейку. Вдруг кто-то уже её сделал. А вообще-то иногда неожиданности приятны, они украшают жизнь эмоциями.

Строгий результат можно сравнить с комплектом инструментов, суммой покупок, рюкзаком в дорогу, длиной сложного маршрута или самым вкусным пирожком. Бывает важно совершенно разные вещи собрать в одно место и после этого рассматривать их вместе как одно целое, или для серии вещей выяснить их какую-то общую характеристику. Почему, собираясь в поездку авиарейсом, мы стремимся уложить всё в одну дорожную сумку? Перед регистрацией мгновенно выясняется соответствие её веса и габаритов. Хотя конечно, 2-4 небольшие сумки удобнее нести. Порой самая нужная вещь может оказаться на дне сумки, а также при плотной упаковке вещи могут помяться.

Другой пример: маникюрный набор обычно стоит меньше, чем сумма цен отдельных инструментов. Какие имеются ещё преимущества? Каждый инструмент сразу имеет своё место, и есть инструменты почти на все случаи жизни. Впрочем, когда что-то сломается, трудно найти

эквивалент, да и в комплекте могут быть лишние, ненужные вещи.

Часто на рабочем столе используют специальные органайзеры для бумаг и мелочей. Какие действия или работы становится удобнее выполнять? Легко найти любую нужную мелочь, если всё всегда возвращать на место, и сразу видно расход материалов, ясно, что надо восполнить. Но зато приходится помнить, где что должно располагаться, и требуется приучать себя к дисциплине — всё возвращать на своё место сразу, как только вещь стала не нужна.

Все эти примеры показывают, что принципы дают выигрыш при определённых, заранее известных требованиях к решению задачи и условиях применения решений, допускающих следствия, такие как мета-программирование, верификация и выделение из автономно развиваемых модулей.

3. Неформальные иллюстрации: следствия

Мета-программирование по существу напоминает разные методы работы с текстами, такие как пересказ содержания, шифрование для защиты от стороннего доступа, перевод на другой язык. От перевода обычно требуется совпадение смысла исходного текста и результата перевода по отношению к описываемым сюжетам. Достижение такого совпадения для естественных языков — сложная задача. Вопрос: почему переводные тексты часто отличаются по оттенкам смысла от оригинального текста? Скорее всего, грамматика языка перевода слишком отличается от грамматики целевого языка. Но бывает, что переводчик не в ладах с исходным языком. Нередко встречается отсутствие владения смыслом переводимого материала, знания реалий описанной жизни, или слабое владение целевым языком.

Для пересказа не требуется слишком точного совпадения смысла, а достаточно совпадения некоторых общих схем, что можно рассматривать как соответствие некоторых ключевых моментов. В функциональном программировании такими ключевыми моментами считают совпадение результатов при определённых аргументах. Например, почему при чтении детектива так и тянет заглянуть в конец? В конце детектива часто размещён краткий пересказ происходящего. Читателю может быть лень читать все подробности, хочется скорее узнать развязку. А может, хочется проверить свои догадки. При чтении заранее хочется видеть, куда клонит автор.

Во многих видах деятельности имеют место отладочные версии с пошаговыми добавками постепенного улучшения результата. Например, зачем при постановке фильма нужен сценарий, для театрального спектакля — текст пьесы, для подготовки мероприятий — программа проведения? Эти предварительные формы дают компактное представление, позволяющее всем участникам понимать общие цели. Может показаться, что этого и не надо, фильм часто отклоняется от сценария, актёры в спектакле могут иногда импровизировать, а мероприятие может отклониться от программы. Что-то оправдывают получением оплаты труда в промежутках между съёмками.

Верификация программ, её значимость, выходит на первый план при решении вопросов достоверности, правдоподобия, логики здравого смысла, интуиции и формальных следствий. Первый пример такого рода виден на общем интересе к прогнозу погоды. По прогнозу дождь, а на небе ни облачка. Видите причину взять с собой зонтик? Мы отдаём себе отчёт, что надёжность важнее всего. Вдруг набегут тучи, и прогноз оправдается? Впрочем, многие всегда носят с собой зонтик и вынимают его из сумки при первых каплях дождя, так что и думать не надо. Бывают и другие

причины иметь при себе модный, красивый зонтик — например, хочется им похвастаться. Но встречается и неприязнь к зонтикам. Никакие причины не могут побудить взять зонтик с собой: «Не беру и всё. Не сахарный, не растаю. Терпеть не могу зонтики».

Другой пример на тему достоверности информации. Представим себе, что по карте расстояние до нужного пункта всего 1 км, а идти пришлось полчаса. В чём дело? Причины могут быть самые разные. Карта не учитывает рельеф местности и качество дорог. Но, может, дело не в достоверности — просто навалилась усталость, и ходьба была медленнее обычного, или из-за тяжёлых вещей пришлось время от времени отдыхать. А может, отвлекли красоты природы.

Дорожные реалии дают и другой пример. Много раз проверенный маршрут, возможно, не самый короткий. Чем он привлекателен? Говорят: «Знакомый путь короче». Это верно, на нём нет неожиданностей, можно рассчитать время. Но может, дело в опасении, что на другом пути можно попасть в тупик? Бывает, что хочется поискать или разведать другой путь, но это в другой раз, когда будет свободное время.

Автономность независимых модулей хорошо видна на разных вариантах сборки-разборки сложных вещей. Это выделение компонентов для рецепта, раскладка бумаг на столе с предварительной сортировкой, укладка вещей в дорогу с учётом удобства их использования в разных ситуациях.

Начнём с компонентов для рецепта. В гостях попробовали вкуснейший торт! Как бы сделать такой же? Конечно, следует поискать или выспросить рецепт и технологию. Но вдруг это почему-то невозможно. Тогда попытаемся вспомнить вкус и экспериментально его воспроизвести, может, нам повезёт и всё получится. Впрочем, если рецепт слишком сложный, то попробуем его упростить, а

потом удивимся, что и близко не получилось. А может, нет отдельных ингредиентов. Попробуем заменить их чем-нибудь, хотя ясно, что это уже будет совсем не то.

Более очевидный пример — раскладка бумаг на столе. Постепенно становится всё труднее находить срочно требуемые бумаги на столе. Что делать? Обычная практика — разложить бумаги в стопки по 4-8 темам, а там по датам. Но это долгая работа, требующая внимания, и лучше сделать её без перерыва, чтобы не забыть где и что лежит. Поэтому при поиске разобранные бумаги отложим в отдельную стопку, а остальные пусть подождут до лучших времён. Задача не решена, нам будет трудно припомнить, где искомое должно быть, и если за полчаса не находим, то считаем потерянным и прекращаем искать.

Теперь рассмотрим разные варианты при укладке вещей в дорогу. Собираясь в сложную поездку, следует продумать, как упаковать вещи, учитывая, что иногда нужно будет быстро и по-разному реагировать на возможные ситуации. Конечно, лучше разложить по отдельным пакетикам то, что надо в каждом пункте поездки (регистрация, салон самолёта, сон, ванная, гостиница и т. д.), но может не найтись на это времени. Тогда быстро всё побросаем в рюкзак, а если что-то понадобится, то всё вытряхнем, а потом всё вернём обратно. По меньшей мере, разделим вещи на багаж и ручную кладь, а там сложим их по категориям. Главное — упаковать одежду для доклада, чтобы она не помялась, а остальное как-нибудь образуется.

Такие следствия (Таблица 1) способствуют удобству продумывания процесса, возможно, на основе специально подобранных выразительных средств и дополнительных инструментов. Продумывать процессы можно, представляя их бесконечными, обратимыми, сводимыми к функциям одного аргумента, т. е. обрабатывающими одно данное. Обычно системы функционального программирования поддерживают

иллюзию таких процессов, скрытую обычными конструкциями обработки данных. Рассмотрим другие примеры средств и методов, позволяющих оперировать такими процессами.

Бесконечность — это не более чем математическая или психологическая абстракция, удобная в тех случаях, когда невозможно определить конкретные характеристики процесса или сложно манипулировать слишком большими числами. Конечно, в реальности любые процессы обладают конкретными, конечными характеристиками. Первый пример такой абстракции виден в магии перебора чёток, дающей успокоение и помогающей размышлениям. Руки автоматически работают, и это успокаивает и помогает думать о вечном. А может, бусинки просто приятны на ощупь. Их можно считать до бесконечности или таким образом с их помощью измерять интервалы времени.

Другой пример даёт обычная практика продажи товаров в магазинах. Продавец не знает, сколько покупателей выберет данный товар. Почему он так уверенно подтверждает менеджеру, что всех товаров хватит до конца дня? Возможно, он видит, что в зале покупателей заведомо меньше, чем число оставшихся упаковок товаров, и нет необходимости выяснять точное число товаров, можно считать его бесконечным. Но может быть, он вовсе не уверен в этом, а просто не хочет суетиться и получать дополнительный товар. Он знает, что уже поздно, и скорее всего больше никто не придёт, или ему не нравится общаться с этим менеджером.

Ещё один пример — шкаф для ключей в гостинице. В большой гостинице постояльцы часто берут и возвращают ключи. Может ли администратор в любой момент точно знать, какое число ключей на месте? Конечно, может! Надо лишь завести счётчик, увеличивающийся на 1 при возвращении ключа и уменьшающийся на 1, когда ключ забирают. Знать, сколько раз произошло прибавление или

вычитание, нет необходимости, и можно такой процесс считать бесконечным. Впрочем, если ключей в маленькой гостинице меньше десятка, то они видны почти мгновенно при подходящем их расположении. Иначе, пожалуй, потребуется некоторое время для подсчёта. Впрочем, какая разница сколько их, лишь бы на месте был нужный ключ.

Обратимость процессов и возможность их повторного прогона весьма характерна для программирования, в отличие от реальности, в которой все процессы уникальны. В текстовых и графических редакторах обычно имеются кнопки UNDO-REDO, позволяющие вернуть прежнее состояние редактируемого объекта. Есть и другие примеры обратимости процессов: сборка-разборка приборов, методы прогулки по незнакомому городу, а также в мире головоломок, например, сборка кубика Рубика.

Задача восстановления кубика Рубика разрешима за конечное время. Тем не менее, трудно начать осваивать этот процесс без чрезмерной затраты времени. Можно методично записывать ходы от исходной позиции, а потом идти обратно, но это скучно. Хочется поискать описание стратегий и методов, полезных для восстановления из любой, заранее неизвестной позиции. Можно и без особых изобретений пытаться, пока не получится, хотя может и непонятно будет, почему получилось. В конце концов, можно собрать хоть что-то понятное, например, углы или линии и удовлетвориться достигнутым частичным решением.

Нередко приходится выполнять сборку-разборку незнакомых приборов. Пусть предстоит чинить сложный прибор. Как исключить вероятность того, что после сборки останутся лишние детали? При современной технике проще всего делать фото каждого шага разборки. Но если рядом нет такой возможности, то можно укладывать детали по порядку или найти схему устройства прибора. А можно поискать

координаты знакомых знатоков, пусть советуют, заодно и пообщаемся.

Прогуливаясь по незнакомому городу, опасно потерять ориентировку. Как надёжно вернуться обратно? Можно методично фиксировать все приметы — статуи, красивые дома, фонтаны и пр. Такой механизм может не сработать при прогулке в сопровождении. Впрочем, о чём беспокоиться? Всегда можно поспрашивать у прохожих, хотя бывает, что прохожие покажут совсем не туда. Тогда придётся звонить знакомым. Стандартный совет — двигаться в одном направлении, пока не покажется знакомое место.

Унарность или сведение к функциям одного аргумента хорошо иллюстрируется на таких примерах, как комплект для обеда в самолёте, рюкзак в походе, кошелёк или веб-приложение, сводимое к одной кнопке. Первый пример — кошелёк. Если в одежде хватает карманов, то нужен ли кошелёк? Большинство скажут «Да» — в кошельке лежат все деньги сразу, нет нужды собирать их по разным карманам, да и карманы не портятся. Но есть и другие мнения. Может, без кошелька ещё и удобнее, можно деньги рассортировать по достоинству и разложить в разные карманы. Кроме того, из карманов легче деньги доставать, не считая того, что кошелёк легко потерять со всеми деньгами, а карманы никуда не денутся.

Второй пример — рюкзак в походе. Что заставляет предпочитать рюкзак, особенно в походе, да и не только в походе? Прежде всего, руки свободны! Не для всех это главное. Многие считают, что при тяжёлой поклаже лучше сумка на колёсиках. А мне для формирования мускулатуры лучше две сумки на плечи, они уравновешивают позвоночник. И вообще, у меня красивый, модный рюкзак, всегда беру его с собой.

Третий пример — поднос в руках официанта в ресторане. Насколько поднос повышает продуктивность

работы официанта? Очевидно, что поднос позволяет компоновать заказ клиента или сокращать маршрут при обслуживании нескольких клиентов. При этом поднос провоцирует носить избыточную тяжесть, что может наоборот снизить продуктивность. Да и работа с подносом требует выучки и хорошей координации движений, что не у каждого получится, не считая того, что наличие на подносе заказов нескольких клиентов создает необходимость помнить, что кому принести.

4. Неформальные иллюстрации. Практичные компромиссы

Производственные системы функционального программирования обычно дополнены специальными средствами **практичных компромиссов,** позволяющих при минимальном отклонении от принципов повышать производительность программ. На уровне семантики такими дополнениями являются типизация данных, программируемые планы на будущее и удобные схемы циклов.

Типы данных аналогичны стандартам на гайки-болты, виды одежды, приборы для еды, методам разметки брёвен для дома. Известно, что при строительстве домов из брёвен практикуют их разметку. На каждом бревне пишут обозначение угла дома и номер венца снизу. По каким причинам нужна столь тщательная разметка? Дело в том, что все брёвна разные, на них нет стандарта, каждое бревно – отдельный тип данных. При строительстве каждое бревно обязательно располагается на своём месте, его только там можно разместить. Обычно первичная сборка дома выполняется на специальной строительной площадке, а потом дом разбирают и всё перевозят на другое место. Разметка помогает в этом процессе не перепутать расположение

брёвен. Кроме того, но это не так уж важно, по номеру можно узнать, кто отёсывал конкретное бревно, вдруг в нём что-то не так. И конечно, это традиционная технология, так делали предки.

При проведении банкета обычно все столы сервированы одинаково. Здесь уже чётко работают типы данных. Прежде всего, сразу видно, всё ли на месте. Контроль порядка не занимает особого времени. Удобно, что официант заранее знает, куда что следует положить согласно общей схеме. И можно сказать, что это красиво. Кроме того, в любой момент легко подсчитать, сколько чего надо подготовить или дополнить, если вдруг изменилось число гостей.

Ещё один пример — виды одежды для разных случаев жизни. Чем диктуется разница в выборе одежды на работу, в сад, на спортивную площадку, в поездку, в больницу и др.? На каждую обстановку существует свой тип одежды. В каждом из таких случаев мы понимаем, что придётся делать и какая одежда будет для этого удобной. Но имеет место и учёт традиций, приличий и гигиены, хотя многие скажут, что главное для них — красота. Конечно, приходится учитывать и цену одежды.

Планирование будущего обычно связано с прогнозами, разными обстоятельствами и пожеланиями: погода, выходной, отпуск, реабилитация после болезни. Прогнозы строятся по чужим данным с учётом статистики; планы на отдых или домашние работы зависят не только от пожеланий, но и от многих других обстоятельств; выздоровление после болезни зависит и от желания выздоравливать, и от особенностей организма, и от выполнения медицинских рекомендаций.

Почему, несмотря на частые ошибки, люди смотрят прогноз погоды? Дело не только в том, что «желают знать, что будет» — интересно сравнение с интуитивным прогнозом, соревнование «кто прав — они или мы?». Да и на всякий

случай, вдруг прогноз сбудется, даже если он выглядит неправдоподобным. Впрочем, вероятность ошибок современных прогнозов строго меньше половины. Хотя некоторые скажут, что никакой прогноз не нужен, лучше посмотреть в окно!

Любые планы на командировку, отдых или домашние работы обычно начинаются с составления списка необходимых вещей и действий. Насколько помогает составление списка вещей, которые следует взять с собой в командировку? На ВЦ СО АН был весьма популярен подобный список, составленный Андреем Петровичем Ершовым, часто ездившим в командировки и потому на личном опыте отладившим его детали до мелочей, которые не сразу придут в голову новичку. Это очень помогает! Резко сокращается число ситуаций, когда вас застаёт врасплох отсутствие необходимого. При стихийном составлении подобного списка ничего хорошего не получается. В список попадает слишком много вещей, которые потом оказываются лишними. Складывается впечатление, что это без толку! Всё предусмотреть невозможно. Признаём, что это полезно, если ездишь часто, и постепенно отлаживается реально практичный список.

Ковидная эпопея показала, что и при условии выполнения медицинских рекомендаций реабилитация после болезни может быть слишком долгой, превышающей терпение выздоравливающих. В чём смысл рекомендаций врача по срокам реабилитации после тяжёлой болезни, такой как ковид? Всё равно каждый выздоравливает сам, по-своему. Тем не менее, важно предупредить, что заболеть можно быстро, а выздороветь — это трудная и, может быть, долгая работа. Смысл в профилактике риска сорваться и снова заболеть. Хотя нередко кажется, что никакого смысла — говорят всем одно и то же, а мы все разные, сам я лучше знаю, у меня интуиция и внутренний голос.

Схемы циклов на практике понятны потому, что обычно они имеют аналогию в обычной жизни в форме процессов, обладающих ясной ритмикой, таких как вязание, танцы, сбивание крема, нарезка лука. Для таких процессов характерны разные условия завершения повторяющейся серии одинаковых шагов.

Например, вязание изделия выполняется до готовности по его схеме. Что отличает вязание от других видов рукоделия? Очень практично, что процесс вязания можно прервать на любой петле, а потом продолжить по раппорту до выполнения всей схемы изделия. Впрочем, обычно я вяжу, пока не надоест, а прерываясь, не знаю, соберусь ли завершить. Случается, что вяжу, пока не кончится пряжа, а продолжаю, если найду подходящую. Да и другие дела могут надолго отвлечь.

Совсем иначе выглядит схема цикла для танцев на банкете. «Танцы до упаду» — что означает эта броская фраза? Обычно — до выключения музыки, но бывает, что до усталости, когда один из партнёров может прервать танец. Впрочем, лишь бы партнёр не наступил на ногу.

Если предыдущие два примера зависят от внешних условий, то встречается и зависимость от внутренних состояний — например, при сбивании белкового крема для вкуснейшего торта. Сбивание завершается, когда крем перестаёт стекать с венчика, хотя иногда пишут, что сбивать, пока крем не станет белым или сбивать 20 минут, можно по таймеру. Совершенно типично — готовность по загустению до нужной кондиции. А что такое «нужная кондиция»?

Практичные компромиссы нацелены на повышение производительности программ. Такие средства включают в себя мемоизацию функций, восстановление и обработку состояний памяти, псевдофункции для взаимодействия программы с внешним миром.

Мемоизация функций сводится к их представлению с помощью таблиц, что аналогично записным книжкам, дневникам, справочникам, этикеткам, приборам типа компаса или термометра, приспособленным к долговременному хранению данных. Первый пример — термометр. Чем ртутный термометр удобнее спиртового? Конечно, ртутный термометр сохраняет показания, пока их не стряхнёшь. При этом, если он и удобен в чём-то, он и опасен. Кроме того, неудобно, что приходится долго ждать, пока он нагреется. Впрочем, мне он просто не нравится, значит, для меня он неудобен.

Другой пример — кулинарный справочник при мультиварке. Спрашивается, зачем в комплект поставки включён кулинарный справочник? Скорее всего, полезно каждому клиенту дать информацию о времени готовки и тем самым освободить от необходимости экспериментально его выяснять. Хотя не исключено, что это простая накрутка цены или учёт интересов рекламы производителей продуктов. Но всё это напрасно, я никогда не смотрю инструкции и всё проверяю в эксперименте.

Другие соображения связаны с этикетками на товарах в магазине. Зачем они сопровождаются этикетками и ценниками? Такая практика позволяет продавцу не помнить наименования и численные характеристики товаров, а покупателю планировать стоимость покупок. Хотя это не всегда работает, на этикетках встречаются и ошибочные наименования товара, и неправильные размеры или цены. Ценники часто не удаётся разглядеть и приходится спрашивать у продавца. Тем не менее, удобно, чтобы неискушённый покупатель узнал, как называется удививший его товар, не отвлекая продавца.

Восстановление и обработка состояний памяти аналогичны процессам стирки, химчистки, ремонта и других видов приведения в порядок или продления жизни вещей. Как

понять, что стирка успешно завершена? По инструкции достаточно полоскания в двух водах и отжима, после чего можно посмотреть, что нет нигде грязных пятен. Существует понятие «перестир», возможно, повышающее уровень чистоты.

Труднее понять, достаточно ли поработал пылесос. Иногда думают, что достаточно, если через несколько минут не появилась пыль на горизонтальных поверхностях. Можно и формальнее — если всю территорию прошли, то хватит пылесосить. Мягкая мебель не пылит, значит всё в порядке. Бывает, что нет времени — 10 минут попылесосили, а дальше в другой раз, работаем по частям**.**

Другой пример — штопка носков. Имеет ли смысл заниматься штопкой носков? Они ведь вскоре снова порвутся. Приходится иногда, если нет запасных и негде купить, например, в поездке. Впрочем, пока дырка маленькая, почему бы и не заштопать? Это не потребует много времени. И конечно, если нравится штопать, приятно смотреть, как вещь становится целой. Бывает, что это любимые носки или других таких не найти.

Псевдофункции подобны усложнению действий по пути. Заранее известен список запланированных действий, и к некоторым из них добавляется ещё какое-то действие, не нарушающее этот порядок. Например, кто-то собирается на вокзал забрать свои вещи из камеры хранения, а ему говорят: «Возьми этот свёрток с собой и заодно отнеси в камеру хранения». Или собираюсь в мастерскую отнести ключ, чтобы сделать копию. Друг просит попутно прихватить и его ключ. Почему он считает такую просьбу разумной? По его мнению, это не потребует дополнительных усилий и времени. Он оптимизирует свои процессы, чтобы самому не пришлось вскоре заботиться о копии ключа. А может и просто любит, чтобы его обслуживали или боится, что сам не найдёт времени или не сумеет найти мастерскую.

Другой пример. Иду в магазин для покупки зимней тёплой обуви. Родственник просит посмотреть цену на домашние тапки. Обязательно ли выполнять такую просьбу? Кажется, что дополнительная нагрузка небольшая, почему бы и не посмотреть. Тем не менее, это в другом отделе, пусть смотрит сам, мало ли какие тапки его интересуют. Да и мог бы позвонить по телефону в магазин и узнать, но ему лень. Не исключено, что это он просто хочет пообщаться, тапки ему не так уж нужны.

Ещё пример. Пишу письмо брату. Его приятель просит попутно отправить ему фото. Стоит ли выполнять такую просьбу? Дело секундное, почему бы и нет? Письмо от этого не пострадает, а может стать интереснее. Но лучше пусть сам напишет и посылает всё, что хочет. Брат давно ничего не писал, может, лучше фото друга отправить потом. Конечно, от меня не убудет, но мне не хочется идти на поводу.

5. Неформальные иллюстрации: параллелизм

Функциональное программирование показало себя как удобный метод подготовки многопоточных программ, смягчающий резкое расширение многих понятий программирования. Рассмотрим конкретизацию принципов ФП на параллельные вычисления. Это ленивые или частичные вычисления, пространства итераций и повторные прогоны.

Ленивые или частичные вычисления имеют разные формы и цели — оттеснение, ленивость, обработка данных по шагам, раскрутка программ. Всегда ли разумно или возможно всё сделать раз и навсегда? Такие задачи подобны уборке квартиры по шагам методом создания «дырок в сыре», накоплению мусора в ведре на выброс потом, вариантам заготовок или запасов, составлению расписаний. Что даёт ежедневная уборка по шагам — метод «дырки в сыре».

Каждый день по половине кубометра за 15-20 минут, и при дефиците времени на уборку появляется место, радующее глаз накоплением порядка. Хотя есть мнение, что если убрано не всё, то и не видно реального результата, велик риск возврата к беспорядку. Недоделанное снова расползётся — энтропия поглотит, потому что трудно выдержать границу между порядком и беспорядком.

Обычно ненужное не сразу попадает в мусор, а предварительно накапливается в промежуточном буфере. Потом перемещается в мешок или ведро на выброс, что позволяет всё ненужное собрать в одно место — легче потом выкинуть, не сомневаясь, что нужного там нет. Хотя всё-таки не исключено, что вдруг что-то неожиданно снова понадобится, да и нужное может случайно попасть в ненужное, не считая того, что мешка может не хватить.

Теперь подумаем, какие преимущества связаны с составлением расписаний. Расписание даёт шанс оптимизации решений, когда и что делать, хотя это дополнительная работа и лишнее напряжение по контролю времени. Приятнее делать всё по наитию, когда душа попросит, хотя в таких случаях бывают накладки с реальностью и неожиданные события.

Пространство итераций можно рассмотреть на раздаче сухого пайка в дорогу, подарков детям на праздник, футболок членам команды — всем одинаковые. Почему подарки для детских утренников делают одинаковыми для всех детей? Просто для удобства быстрой раздачи без индивидуального подхода. Также важно, чтобы дети не завидовали друг другу. Слишком трудно учитывать индивидуальность при раздаче, да и такова традиция.

Говорят, что одинаковость футболок для всех членов спортивной команды — символ единства. Что ещё хорошего в том, что у всех членов команды одинаковая форма? При игре на поле легко видеть, кто чей игрок. В какой-то мере так

красивее, как на параде, но некоторые считают, что игроки выглядят «как инкубаторские». В команде можно особо не различать, где чья одежда, ведь при необходимости возможен обмен.

Представим, что в столовой погас свет и вместо шведского стола посетителям выдали сухой паёк, одинаковый для всех. Что в этом хорошего? Такова штатная процедура, определённая в регламенте реагирования по технике безопасности: раздача в столовой не приспособлена к выдаче консервированной еды, пригодной для длительного хранения, а также легче учесть, хватит ли на всех еды (имеет место страх, что некоторые слишком много наберут с собой).

Повторы — сложные виды деятельности часто требуют разного рода повторов, иногда идентичных, иногда уточняющих или расширяющих ранее отлаженную часть, подобно заучиванию, репетициям или тренировкам. «Повторенье – мать ученья». Одни повторы дают демонстрацию выполнимости, другие нацелены на перетекание деятельности на подсознательный уровень (этологи выяснили, что разучивание циркового номера с конца даёт более устойчивые цепочки в памяти, чем заучивание с начала).

Известно, что подготовка театрального спектакля включает в себя репетиции, и особое значение придаётся генеральной репетиции. Чем отличаются репетиции друг от друга, а генеральная репетиция принципиально отличается от остальных? Репетиции позволяют уточнять решения по взаимодействию между актёрами, а генеральная репетиция позволяет убедиться, что принятых решений достаточно для успешного спектакля. Конечно, так принято — последнюю репетицию называть генеральной. Обычные репетиции можно проводить в любой обстановке и любой одежде, а генеральная репетиция проводится на настоящей сцене и в правильных костюмах. Рядовые репетиции актёры могут

проводить по мере разучивания пьесы самостоятельно, а к генеральной они уже выучили всю пьесу, и на ней обязательно присутствует режиссёр.

Главная задача такого повтора как тренировки — перетекание деятельности на подсознательный уровень, что обычно даёт повышение надёжности и скорости выполнения. Исполнительское мастерство подразумевает тренировки, на которых многократно повторяются, оттачиваются цепочки действий до автоматизма, что несколько похоже на технику превращения оперирования последовательностью действий в процедуру или функцию (Игорь Васильевич Поттосин[1] любил это называть «запроцедуривание»). Какую полезную работу в таком процессе, возможно, выполняет мозг? Мозг, по-видимому, оптимизирует цепочку действий, убирая из неё сознательный контроль порядка выполнения действий. Чрезмерно интенсивные тренировки могут надоесть или травмировать, в таких случаях мозг способен заблокировать их продолжение. Скорее всего, за время тренировок происходит запоминание цепочки действий в долговременной памяти. Тренировки тренировками, а бывают ещё и импровизации, в которых мозг, возможно, работает иначе.

Этолог Карен Прайер в работе с дельфинами и собаками заметила, что разучивание циркового номера надёжнее и быстрее происходит с конца — формируются устойчивые цепочки, автоматически срабатывающие после первого звена. Надёжно пристраивать предшественников по одному. Это напоминает известное стихотворение «Дом, который построил Джек». Она советует стихи разучивать с последнего куплета. Чем можно объяснить такую

---

[1] Один из создателей Отдела программирования в ВЦ СО АН СССР, (1958 год.) директор ИСИ СО РАН (1992-1998 годы) первый заведующий кафедрой Программирование ММФ НГУ, Заслуженный деятель науки Российской федерации (1999 год) http://alley.iis.nsk.su/person/pottosin

эффективность? По мере тренировки происходит слияние предшественника с преемником, что делает переход как бы автоматическим, не требующим внимания. Это кажется неестественным, и никто так не делает, все разучивают всё от начала к концу. Может, Карен Прайер и права, но в это трудно поверить, а на практике проверять лень. Можно подумать, что животные — это одно, а человек — другое, хотя эксперименты подтверждают её выводы.

Многопоточное программирование проявляется и на уровне конкретизации прагматики поддержки методов параллельного программирования. В их числе автоматическое распараллеливание программ, балансировка нагрузки процессоров и применение многоконтактных узлов.

Тонкости проблемы **распараллеливания программ** можно проиллюстрировать на отдельном эпизоде из жизни ВЦ СО АН. На методологическом семинаре института однажды разыгралась такая сцена: доска в конференц-зале была обустроена шторками с двух сторон, чтобы докладчик мог написать всё, что считает нужным, заранее и зашторить, чтобы слушатели не видели написанное раньше времени. Однажды докладчик не смог разместить на открытой части доски всё, что хотел и решил мешающие ему шторки задвинуть за доску. Он успешно сделал это справа и перешёл к левой стороне, чтобы и там освободить доску от шторки. Неожиданно оказалось, что только он тронул левую сторону доски, как правая шторка выскользнула вперёд и снова часть доски закрыла. Докладчик вернулся вправо, и возможно этот казус мог повториться снова, но в аудитории нашлась сообразительная особа. Она быстро вскочила со своего места, подошла к левой стороне, они вместе с докладчиком синхронно задвинули шторки за доску, и докладчик обрёл необходимое пространство. Аудитория единодушно ахнула, и прошелестел громкий шёпот: «Это истинный параллелизм».

Можно ли это рассматривать как доказательство несводимости параллельных вычислений к однопроцессорным конфигурациям? Конечно! Здесь нужны два процессора — на каждую шторку свой. С другой точки зрения, докладчику просто не хватило длины рук. Более «длиннорукий процессор» справился бы с этой задачей. Это значит, что проблема распараллеливания не может быть полностью решена на языке высокого уровня. Конечно, может просто не стоило так много писать на доске. Зачем создавать трудно решаемые проблемы? Опытный докладчик мог подумать заранее и припасти специальные прищепки.

Более массовый пример можно видеть при посадке и уборке картофеля. Обычно работу ведут два-три человека, распределив между собой функции. При посадке один копает лунки, а другой кидает в них картофелины. Потом при уборке один выкапывает картошку, а другой собирает её в ведро. Что диктует такое стандартное разделение труда? Каждая из этих функций связана с разными инструментами, применять которые одновременно неудобно. Вообще-то можно всё это делать и в одиночку, что не так уж сложно. Просто каждый ряд тогда придётся пройти два раза. И конечно, это дань традиции, все так делают, хотя партнёры могут и мешать друг другу, не выдерживая синхронизацию действий.

Другой пример — выращивание овощей по системе разных грядок. Обычно в огороде на разных грядках растут различные овощи. В чём выигрыш от такого способа выращивания овощей? Можно ведь просто рассыпать семена, а потом смотреть, где что взошло. Очевидно, что важно определять полезные растения при прополке, иначе не всегда ясно, что сорняк. И конечно, так советуют опытные люди. Однородные посадки красиво выглядят, как колонны на параде. Да и семена продают в отдельных пакетиках, вот и сеем каждый пакетик на отдельную грядку, не считая того,

что не все растения совместимы: одни могут мешать другим, угнетать их или затенять.[2]

Балансировка нагрузки — проблема, которую можно видеть при выравнивании груза в двух руках, при разливе вина по рюмкам поровну и раскладке еды по столам для банкета. В спорте принято предпочитать близких по силам партнёров в играх и соревнованиях.

Достаточно ясный пример даёт уравниловка в оплате труда. Во многих видах работ проявляется проблема оценки трудового вклада. Чаще всего её преодолевают провозглашением одинаковой оплаты без учёта сложности и интенсивности выполненной работы или пропорционально потраченному времени. Насколько такой подход может влиять на производительность? Обычно происходит равнение на среднего или слабейшего, и общая производительность постепенно снижается, за исключением исполнителей, работающих ради удовольствия. Но может именно так и надо, такой подход большинство людей воспринимает как справедливость. Такой подход благоприятен для общества тем, что всем даёт право считать себя полезными. Хотелось бы уметь оценивать реальный вклад работника, но даже если кто-то умеет это делать, ему трудно доказать, что он объективен.

Не столь очевиден пример с выравниванием груза в руках для равновесия при заболеваниях позвоночника. Почему при переноске тяжестей в руках важно, чтобы справа и слева был примерно одинаковый вес? Есть разные мнения. Позвоночник не любит сильного перекоса в одну сторону это может его повредить. Перекошенная от неравномерной нагрузки фигура выглядит смешно. Так рекомендуют инструкторы физкультуры, они знают. Лучше тяжести вообще в руках не носить, для этого существуют рюкзаки и тележки.

---

[2] Дополнение Б. Л. Файфеля

Достаточно убедительный пример — предпочтение близких по силам партнёров в спортивных играх и соревнованиях. Почему в силовых видах спорта соревнуются спортсмены одной весовой категории? При явном различии в весе результат поединка заранее предопределён, соревнования теряют зрелищность. Но может и зря — интересно, когда искусство спортсмена позволяет ему побеждать более тяжёлых противников. Скорее всего, это просто профилактика бессмысленных травм, ну или дань традиции.

Существует такое явление, как **многоконтактные узлы** в конструировании разных приборов, включая компьютеры. На уровне бытовых примеров многоконтактность можно видеть на электрических вилках-розетках, кабелях-гнёздах, многоштыревых замках, застёжках-молниях и т. п. Начнём с вилки-розетки в электрических цепях. Почему каждая вилка подходит только к своей розетке? Это же не универсально и мешает комбинаторике, зато исключён риск перепутать и соединить неподходящее. Если соединилось, значит всё в порядке. А вот если подходящей пары нет, то придётся подгонять и для этого хранить много вариантов запчастей.

Более простой пример — закручивающиеся крышки банок и бутылок. Почему у них может быть разная резьба, не любую крышку наденешь на нужную банку или бутылку? Главное — удобно при заготовках или разливе получать плотное закрывание при остывании, хотя это преимущество теряется при дефектах в резьбе. Но бывает потом трудно открыть, если крепко присосалось. Такое хозяйство требует бережного отношения, крышки могут проржаветь, или лак повредится.

Ещё проще — контейнер для яиц. Зачем для яиц нужен специально устроенный контейнер? Яйца — хрупкая вещь, без него могут повредиться. В контейнере легко видеть,

сколько яиц размещено, и проверить, все ли целые. Не всегда размер яйца подходит ячейкам контейнера, бывают яйца и поменьше, и побольше. Впрочем, контейнер — это лишний объём, можно обойтись и без него, если действовать аккуратно.

Заключение

Не исключено, что обучение функциональному программированию можно более успешно наладить, если концентрировать заметное множество задач, решаемых с помощью парадигмы функционального программирования удачнее, чем в парадигме императивного программирования, что перекликается с достаточно общим пониманием информатики как науки [11, 12]. Основная идея — трудности с овладением такой парадигмой вызваны отсутствием у большинства людей подходящей скрытой грамматики деятельности, роль которой исследована в этологии [13]. В реальной жизни наблюдаются непрерывные изменения и линейные маршруты типа обхода графа в глубину, что созвучно с императивным программированием. Для функционального программирования требуется обход графа в ширину и неизменяемость данных, что противоречит интуитивной скрытой грамматике деятельности. Идеи функционального программирования наиболее чётко были реализованы в первых LISP-системах, многие из них унаследованы современными языками программирования [1, 14].

В препринте собраны аналогии и образные примеры в качестве иллюстраций. Представленное в препринте неформальное введение адресовано новичкам, незнакомым с математическими основами программирования и методами реализации систем программирования с учётом гуманитарных основ программирования [15]. На игровых,

почти шуточных моделях можно рассмотреть основные понятия, такие как функции, рекурсия, данное, символы, их свойства и значения, стек, список. Можно показать, что для отдельных функций может существовать более одного способа перехода от определённых аргументов к конкретным результатам. Важно, что разные способы вычисления результата функции дают один и тот же результат. Это позволяет при реализации функций выбирать способы, наилучшим образом соответствующие условиям её применения. «Рекурсия» означает самоприменение некоторых конструкций, что обычно позволяет строить лаконичные определения функций. «Данные» возникают при переходе от идеальных, интуитивных конструкций для функций и значений к реальным структурам или объектам, доступным при обработке на компьютере. «Символы», их свойства и смысл — это разновидности категорий данных, различаемые по их роли в процессе обработки информации, рассматриваемой как определение или применение функций. «Стек» — удобный механизм хранения данных и доступа к ним с помощью функций. «Список» — структура данных, поддерживающая последовательную их обработку. Этого достаточно для интуитивного понимания основных принципов функционального программирования, что может помочь перейти к практическому функциональному программированию [16].

Список литературы

1. McCarthy J. Abrahams P. W.,Edwards D. J., Hart T.P. , Levin M.LISP 1.5 Programming Manual / J. McCarthy. The MIT Press, Cambridge, 1963. 106 p. https://doi.org/10.1007/978-3-662-09507-2_12
2. Backus J. Can programming be liberated from the von Neumann style? A functional stile and its algebra of programs. – Commun. ACM 21, 8, 1978, – p.613-641.
3. Лавров С.С. Функциональное программирование. // Компьютерные инструменты в образовании. – 2002, N 2-4.
4. Лавров С.С., Городняя Л.В. Функциональное программирование. Принципы реализации языка Лисп.// Компьютерные инструменты в образовании. – 2002, N5, с. 49-58.
5. Хендерсон П. Функциональное программирование. – М.: Мир, 1983. – 349 с.
6. Филд А., Харрисон П. Функциональное программирование. Перевод под редакцией В.А. Горбатова. – М.: Мир, 1993. – 638 с.
7. Городняя Л.В. Основы функционального программирования. – М.: Интернет—Университет Информационных технологий. – http://www.intuit.ru, 2004. – 272 с. http://www.intuit.ru/studies/courses/29/29/info.
8. Файфель Б.Л. http://catstail.narod.ru/homelisp/index.html
9. Городняя Л.В. О функциональном программировании / Компьютерные инструменты в образовании, 2021, No 3: 57–75. doi:10.32603/2071-2340-2021-3-57-75
10. Пойа Дж. Математическое открытие. Решение задач: основные понятия, изучение и преподавание. – М.: Наука, 1976. – 448 с.
11. Лавров С.С. Так что же такое информатика? // Компьютерные инструменты в образовании. 2000, № 1, с. 22-25.

12. Цейтин Г.С. «Ассоциативное исчисление с неразрешимой проблемой эквивалентности», Проблемы конструктивного направления в математике. 1, Сборник работ, Тр. МИАН СССР, 52, Изд-во АН СССР, М.–Л., 1958, 172-189.

13. Прайор К. «Не рычите на собаку», «Несущие ветер». Бомбора, 2019 г. , 320 с., «Селена +» , 1995, 304 с.

14. Городняя Л.В. Первые реализации языка Lisp в СССР // Материалы второй Международной конференции «Развитие вычислительной техники и её программного обеспечения в России и странах бывшего СССР» (SoRuCom-2011) с. 95-100 *https://www.computer—museum.ru/histsoft/lisp_sorucom_2011.htm*

15. Городняя Л.В. Гуманитарные факторы программирования. – Новосибирск: СО РАН, 2020, 163 с., тираж 200, ISBN 978-5-6044349-4-9.

16. Городняя Л.В. Функциональное программирование. Парадигма, модели и методы / / Сиб. Отделение Рос. Акад. наук, Ин-т систем информатики им. А.П. Ершова. г. Новосибирск: Изд-во СО РАН,2022. — 482 с. https://doi.org/10.53954/9785604782309

Приложение 1.

Реальный наглядный материал для непрофессионалов, желающих понимать идеи функционального программирования

Этот материал адресован методистам, преимущественно руководителям детских кружков, и специалистам, в планы которых не входит интересоваться практикой программирования, его математическими основами и методами реализации систем программирования.

Приведён пример тренинга по формированию интуитивной грамматики на основе работы с аналогиями и подобиями данных, присущих функциональному программированию. В центре внимания — возможность манипулирования реальными предметами, подобного цепочкам действий, возникающим при организации информационных процессов, что, возможно, надёжнее позволит формировать скрытые модели работы с информацией. Приведено несколько примеров постановок учебных задач с решениями[3], показывающими основные принципы функционального программирования, такие как универсальность, самоприменимость, равноправие параметров, а также примеры организации независимых параллельных процессов.

На четырёх небольших задачах приведены варианты их решения в функциональном стиле с применением

---

[3] Они выглядят несколько занудно, надеюсь, со временем удастся их сделать интереснее. В таком виде могут работать школьники младшего и среднего звена, старшеклассниками достаточно уяснить основные этапы работы с постановкой задачи.

рекурсивных функций и перехода к параллельным процессам. Исследовательская и проектная работа обычно проходит фазу поиска оптимального решения, что отчасти иллюстрируется на примерах в данном приложении.

Общее представление.[4] На игровых, почти шуточных, моделях рассмотрим следующие понятия:

— функции,
— процесс,
— рекурсия,
— данные,
— стек,
— символы, их свойства и значения.
— параметр,
— список,
— параллельные процессы/потоки.

Функция как понятие связана с сопоставлением аргументов и результатов, переходом от аргументов к результатам или отображением аргументов в результаты. Для отдельных функций может существовать более одного способа перехода от определённых аргументов к конкретным результатам. Важно, что разные способы вычисления результата функции дают один и тот же результат. Это позволяет при реализации функции выбирать способы, соответствующие условиям её применения.

Процесс сводится к последовательности выполнения функций, каждая из которых получает ряд аргументов и вырабатывает результат, возможно становящийся аргументом очередной функции.

---

[4] Необязательная часть адресована любителям и непрофессионалам или детям. Она нужна для интуитивного формирования скрытой грамматики функционального программирования.

Рекурсия означает самоприменение некоторых конструкций, что обычно позволяет строить лаконичные представления определений функций.

Данные возникают при переходе от идеальных, интуитивных конструкций и значений к реальным структурам или объектам, доступным для обработки на компьютере, символам или кодам. Функции, их аргументы и результаты представляются с помощью данных.

Стек — специальное данное для удобного хранения.

Символы, их свойства и смысл — разновидности данных, отличаемые по их роли в процессе обработки информации.

Параметр – разновидность символов для представления аргументов функции, смысл которых устанавливается по ходу процесса.

Список — структура данных, поддерживающая последовательный доступ к своим элементам.

Параллельные процессы/потоки — ряд из не менее чем двух процессов, возможно, происходящих одновременно. Это значит, что существует момент времени, когда ни один процесс не выполняется, и другой момент – когда все процессы ряда уже завершены. Упорядочение параллельных процессов не определено, они не зависят друг от друга.

Базовые впечатления. Можно использовать сувенир «матрёшки», бумажные карточки с надписями, карабины с цепочками для сцепления в узоры или ещё что-либо другое, что можно изготовлять самим или припасти заранее. Предлагается начальный цикл разбить на разделы:

— простая рекурсия,
— параллельные процессы,
— таблицы для связи символов с их смыслом,
— конструирование динамических структур данных.

Здесь на модели «матрёшки» отчасти проиллюстрированы первые два раздела, остальные лишь намечены.

Модель 1. Простая рекурсия. "Матрёшки".

Представим, что перед нами сувенир “матрёшки”, вкладывающиеся друг в друга. Эта модель достаточна для демонстрации типовых явлений информационных процессов, вызывающих к жизни рекурсивные функции. Такая модель позволяет устанавливать вычислимость результата как важное качество решения новых задач, когда вопросы эффективности, производительности, удобства преждевременны.

Примеры задач:

Задача 1. Требуется узнать, сколько матрёшек внутри.

Задача 2. Необходимо собрать ранее разобранные матрёшки в одну.

Задача 3. Надо выяснить, сколько матрёшек внутри, а потом снова собрать их в одну.

Задача 4. Нужно сосчитать, сколько всего матрёшек в двух или более матрёшках за минимальное время.

Решение любой задачи содержит не менее трёх фаз:

1) Формулировка вопросов относительно цели решения задачи и сопутствующих решению вспомогательных подзадач.[5]

2) Ответы на поставленные вопросы, сопровождаемые выбором простейших шагов, необходимых для решения задачи.

[5] А. Эйнштейн: «Главное — не прекращать задавать вопросы»

3) Описание рецептов решения задачи на базе выбранных простейших элементов.

Задача 1. Как узнать, сколько матрёшек внутри?

Аргумент: Матрёшка, возможно содержащая внутри какое-то число меньших матрёшек.

Надо: Выяснить, сколько матрёшек внутри.

Результат: Число спрятанных матрёшек.

Вопрос 1.1) Каким образом можно выяснить количество спрятанных матрёшек?
Ответ 1.1) Можно все их поразобрать и потом сосчитать.
Рецепт 1.1) Разбираем матрёшки по одной до тех пор, пока это удаётся.

Вопрос 1.2) До каких пор матрёшки разбираются?
Ответ 1.2) Пока не доберёмся до неразборной, самой маленькой.
Рецепт 1.2) Смотрим, разборная ли матрёшка, если да, то разбираем.

Вопрос 1.3) Всегда ли существует внутри неразборная матрёшка?
Ответ 1.3) Нет, не всегда. Может, самая внутренняя матрёшка просто пустая, а целостная просто потеряна.
Рецепт 1.3) Значит, следует всегда выяснять, имеются ли внутри меньшие матрёшки.

Вопрос 1.4) Можно ли выяснить это, не разбирая матрёшку?
Ответ 1.4) Пожалуй, можно матрёшку потрясти и по звуку понять.

Рецепт 1.4) Хорошо, проверка по звуку годится и для неразборной матрёшки.

Таким образом, получено сравнительно универсальное решение, **работоспособное** не только на безукоризненно исправных данных, но и на данных с некоторыми возможными искажениями.

> *Определение* 1. Универсальной называется функция, если она выполнима на любых возможных данных за конечное время.

Предварительный анализ и описание процесса решения:
1) Глядя на большую матрёшку, трудно сказать, сколько в ней спрятано её меньших подруг. Но можно её потрясти и по звуку понять, есть что-то внутри или она пуста.
2) Если матрёшка не пуста, то её можно разнять и вынуть внутреннюю матрёшку, а части внешней матрёшки отставить в сторонку, чтобы не мешали дальнейшему исследованию.
3) Возвращаемся к исходной задаче, но с меньшей матрёшкой, что гарантирует завершение процесса (Шаг 1) — реальные вещи не могут уменьшаться до бесконечности.
4) Если матрёшка пуста, то разобранные матрёшки можно пересчитать – они все на виду, и процесс вычисления завершён, задача решена.

Первые три шага этого процесса будут повторяться многократно, что обычно называют участком повторяемости рекурсии или цикла, четвёртый шаг выполнится один раз и даст результат решения задачи — выход из цикла или завершение функции. Такой процесс содержит возврат к самому себе. Он может быть порожден как вычисление рекурсивной функции, основанной на самоприменимости.

Определение 2. Рекурсивными называются функции, при вычислении прямо или косвенно использующие сами себя, что делает их самоприменимыми.

В описании решения задачи рекурсия сопровождается уменьшением данных до тех пор, пока уменьшение становится невозможным. В этом случае вычисление рекурсивной функции завершается.

Рецепт 1. При определении алгоритма описанного решения могут быть полезны следующие простые вспомогательные шаги-функции (действия, операции):

1. Трясём матрёшку, чтобы по звуку узнать, есть ли что-то внутри.
2. Если слышим звук, то разбираем матрёшку, отставляем её части в сторону и возвращаемся к началу (пункт 1).
3. Иначе процесс завершён, внутренних матрёшек больше не будет.
4. Подсчитываем число разобранных матрёшек. Задача решена.

Краткая запись функций для решения задачи 1:
5) Вводим данные — символы для перехода от описания решения к более удобному, лаконичному представлению программы «Число_матрёшек» с помощью четырёх функций:
Звук[6] – «Трясём матрёшку, чтобы узнать, есть ли что-то внутри».
Разбор — «Разбираем внешнюю матрёшку и отставляем её части в сторону».

[6] Полужирным шрифтом выделены имена функций

Подсчет — «Подсчитываем число разобранных матрёшек».
Число_матрёшек – «Как узнать, сколько матрёшек внутри?**»**

Для всех этих функций по умолчанию неявными **аргументами** является или матрёшка, или её части, получающиеся как **результат** разбора. Данные никуда не исчезают, они только перестраиваются в разные структуры, из которых можно собрать исходное данное – матрёшку.

6) Решение задачи 1 можно лаконично представить примерно следующим образом:

Число_матрёшек =
(ветвление[7]
(*если* **Звук** *то*[8] **Разбор** *повторить* **Число_матрёшек)**
(иначе **Подсчет)**
)

Запись решения потребовала последовательности строк, что является первым примером понятия «**список**». Кроме того, строки 2 и 3 заключены в скобки — размещённые между ними символы также образуют списки.

Кроме функций здесь ради удобочитаемости используются и другие символы. Символ «ветвление» показывает, что ход процесса вычислений представляет собой перебор ветвей, расположенных в списке вслед за ним, одна из которых может начинаться символом «иначе».

Каждая ветвь в свою очередь представляется списком. Символы «*если*», «*то*» и «*повторить*» в представлении ветви

[7] Подчёркиванием выделены ключевые слова языка программирования
[8] Курсивом выделены необязательные слова, синтаксический «сахар»

являются вспомогательными, они используются ради удобочитаемости при показе схемы управления выполнением ветви и различения синтаксических позиций. Поэтому можно дать и более лаконичную запись, считая, что управление может не требовать вспомогательных символов, а учитывать позиции функции в строке. В первой позиции списка ветви представлено **условие** выбора ветви, а далее за ним — последовательность функций, выполняемых, если условие истинно. Условие представляется как функция, вырабатывающая значение **Истина** или **Ложь**

Число_матрёшек =

(ветвление
(Звук Разбор Число_матрёшек)
(иначе **Подсчет)**
)

Первая ветвь **зависит от успеха,** т. е. истинности функции «Звук», выполняющей роль **условия,** заданного предикатом при выборе ветви. Вторая ветвь сработает лишь при отказе первой ветви. Здесь это выражено символом «иначе», являющимся тождественно истинным предикатом.

Определение 3. **Предикат** – это функция, принимающая значения **Истина** или **Ложь.**

Символы «ветвление» и «иначе» можно рассматривать как специальные функции, определяющие общий ход процесса вычислений, структуру управления выполнением функций. Каждая ветвь начинается с условия, а вслед за ним — последовательность функций. Тождественный предикат «иначе» имеет значение Истина, он нужен для выбора ветви,

если предикаты всех предшествующих ветвей имеют значение Ложь.

Обращаем внимание, что здесь первая ветвь содержит многократно повторяемую функцию. Она завершается рекурсивным вызовом функции «Число_матрёшек». Другая ветвь вырабатывает окончательный результат этой функции. Для этого используется функция «Подсчёт». Условие «иначе» показывает, что эта ветвь будет работать, лишь если **неуспех** действия «Звук» не позволил выбрать вышестоящую ветвь. Возможен вариант с подсчётом результата вне ветвления, тогда можно сделать запись без условия «иначе».

Число_матрёшек = (ветвление
(Звук Разбор Число_матрёшек)
) Подсчёт

Рецепт 2. При определении алгоритма описанного решения могут быть полезны следующие простые вспомогательные шаги - функции:

1. Трясём матрёшку, чтобы по звуку узнать, есть ли что-то внутри.
2. Если звуков нет, внутренних матрёшек больше не будет, процесс завершён.
3. Подсчитываем число разобранных матрёшек. Задача решена.
4. Иначе разбираем внешнюю матрёшку, отставляем её части в сторону и возвращаемся к началу (шаг 1).

Для такого рецепта запись можно сделать немного по-другому. Вместо функции «Звук» взять функцию «Нет_звука».

Нет_звука – «Трясём матрёшку, чтобы узнать по отсутствию звука, что внутри ничего нет».

Это даёт несколько иной вариант решения задачи.

Число_матрёшек = (ветвление
(Нет_звука Подсчёт)
(иначе **Разбор Число_матрёшек)**
)

Число_матрёшек = (ветвление
(Нет_звука)
(иначе **Разбор Число_матрёшек)**
) Подсчёт

Одна и та же задача имеет разные решения в зависимости от смысла выбранных шагов - используемых функций. Возможен вариант рассмотрения функции «Разбор» в качестве условия выбора ветви в случае успешного разбора матрёшки, что превращает её в предикат.

Разбор - «Успешно разбираем внешнюю матрёшку и считаем, что в таком случае результатом является значение Истина, а при неразборной матрёшке – значение Ложь». Возможен вариант записи:

Число_матрёшек = (ветвление
(Нет_звука)
(Разбор Число_матрёшек)
) Подсчёт

Такая идея – вместо проверки непустоты по звуку сразу проверять возможность разбора до тех пор, пока

разбирать удаётся – может не уловить вариант завершения с разборной пустой матрёшкой, если это не оговорено в определении функции «Разбор». Но если известно, что самая маленькая матрёшка непременно целостная, то можно использовать и более лаконичную запись, без проверки на звук:

Число_матрёшек = (ветвление
(Разбор Число_матрёшек)
) Подсчёт

Сохранить универсальность можно уточнением определения функции «Разбор».

Разбор - «Успешно разбираем внешнюю матрёшку и считаем, что в таком случае результатом является значение Истина, а при неразборной **или пустой** матрёшке – значение Ложь».

Задача 2. Пусть все матрёшки разобраны. Надо их собрать в одну.

Аргумент: Множество частей разобранной матрёшки.
Надо: Надо восстановить исходную матрёшку, т. е. собрать части в одну, самую большую.
Результат: Собранная заново исходная матрёшка.

Вопрос 2.1) Если все матрёшки разобраны, то с какой начинаем сборку?
Ответ 2.1) Первой нужна самая маленькая.
Рецепт 2.1) Смотрим и выбираем самую маленькую.

Вопрос 2.2) Самая маленькая может быть размещена внутри любой матрёшки. В какую следует поместить самую маленькую?
Ответ 2.2) В качестве очередной надо выбрать наименьшую из оставшихся, т. е. вторую по величине.
Рецепт 2.2) Выбираем вторую по величине и размещаем её в очередную.

Вопрос 2.3) Можно ли быть уверенным, что будут собраны все матрёшки?
Ответ 2.3) Да, если всякий раз выбирали самую маленькую и ближайшую очередную.
Рецепт 2.3) Поочерёдно размещаем матрёшки, наименьшую в очередную, пока не соберём все, разобранных больше не осталось.

Вопрос 2.4) Всегда ли найдётся очередная?
Ответ 2.4) Нет, со временем останется всего одна.
Рецепт 2.4) Регулярно проверяем, существует ли очередная матрёшка, или уже всё собрали, и разобранных больше не осталось.

Предварительный анализ и описание процесса решения:
0) Самая маленькая матрёшка — целая или пустая — является наименьшей.
1) Сравниваем остальные матрёшки, ищем части матрёшки, ближайшей к наименьшей, т. е. второй по размеру — она очередная.
2) Прячем наименьшую матрёшку в найденную очередную, которая теперь становится самой маленькой.
3) Если ещё остались части разобранных матрёшек, то возвращаемся к исходной задаче с уменьшенным числом частей (шаг 1).

4) Если частей нет и видим всего одну матрёшку без очередной, то всё уже сделано, процесс сборки завершён, задача решена.

Здесь рекурсивная функция «Собрать_матрёшек» решения задачи включает в себя поиск подходящего элемента из набора частей. Набор уменьшается на каждом рекурсивном витке процесса. Задача решена, когда не из чего выбрать очередной элемент.

В этом описании процесса выделен шаг 0, задающий начальное данное для последующих шагов. Шаги 1-3 повторяются многократно. Четвёртый шаг выполняется один раз и является завершением процесса.

Для решения задачи 2 понадобятся дополнительные вспомогательные функции:
1. Выбираем наименьшую матрёшку.
2. Выбираем очередную матрёшку, вторую по размеру.
3. Прячем наименьшую матрёшку в большую, очередную.
4. Смотрим, остались ли части матрёшек, и при их наличии возвращаемся к началу.
5. При отсутствии частей сообщаем, что процесс завершён.

Результат второй функции не зависит от результата первой функции, а результат третьей зависит от выбора пары неравных матрёшек. При выборе функций для вычисления параметров предпочтительно, чтобы их результаты были независимы, и их можно было выполнять в любом порядке.

> *Определение* 4. Параметры являются равноправными, если при вычислении они не зависят друг от друга и их можно вычислять в произвольном порядке.

Краткая запись функций для решения задачи 2 :

5) Вводим символы для обозначения функций – действий и данных:
Наименьшая[9] — «Выделяем самую маленькую матрёшку».
Очередная — «Выбираем вторую по размеру».
(Спрятать Наименьшая **Очередн**ая**)** = «Прячем Наименьшую матрёшку в Очередную».
Части? — «Смотрим, остались ли разобранные части матрёшек».
Собрать_матрёшек — «Матрёшки разобраны. Надо их собрать в одну».
Сборка_завершена — Сообщаем о благополучном завершении.

Функции «Наименьшая» и «Очередная» вычисляют значения аргументов для функции «Спрятать». На следующем шаге функция «Спрятать» получит новый результат функции «Наименьшая», равный прежней «Очередная», а значением функции «Очередная» будет наименьшая матрёшка из оставшихся, вторая по размеру.

Список **(Спрятать Наименьш**ая **Очередн**ая**)** представляет вызов функции «Спрятать», который произойдёт после вызовов функций «Наименьшая», «Очередная». Определение функции «Очередная» специально сформулировано независимо от времени выполнения функции «Наименьшая», их можно выполнять в любом порядке. Функции «Наименьшая» и «Очередная» выполнятся раньше, чем функция «Спрятать». Для записи этого действия в виде списка потребовалось использовать скобки, внутри которых вслед за именем функции размещён перечень

[9] Для удобочитаемости синтаксис детского языка может допускать полные множества словоформ, что здесь выражено разницей шрифтов. Вместо бледных букв допустимы другие в соответствии с грамматикой русского языка.

элементов, нужных для её выполнения, т. е. **предварительно** выполняемых функций.

6) Решение задачи 2 можно представить следующим образом:

Собрать_матрёшек = (ветвление
(*если* **Части**
то (**Спрятать Наименьш**ую[10] в **Очередн**ую)
повторить **Собрать_матрёшек**
)
(иначе **Сборка_завершена**)
)

После того, как **Наименьш**ую спрятали в **Очередн**ую, **Наименьш**ей стала **Очередн**ая, а новая **Очередн**ая будет выбрана при выяснении, имеются ли оставшиеся **Части?** При отсутствии Частей, т. е. **невозможности** выбрать Очередную, ветвь «(иначе **Сборка_завершена**)» сообщает о завершении процесса. Рекурсивный вызов функции «**Собрать_матрёшек**» обусловлен **успехом в поиске** частей для упрятывания собранной части в Очередную матрёшку. Каждый рекурсивный шаг содержит свёртку двух матрёшек в одну. Можно сделать запись без ветви со словом «иначе», означающей, что процесс остановился из-за отсутствия частей для свёртки. Можно для большего лаконизма сообщение о завершении расположить вне ветвления, это более краткая запись, рассчитанная на завершение программы из-за отсутствия подходящих действий:

[10] Бледным выделены окончания, не влияющие на смысл лексемы

Собрать_матрёшек = (ветвление
(Части?
(Спрятать Наименьшую в **Очередн**ую**)**
(Собрать_матрёшек)
))
Сборка_завершена

Построенная функция «Собрать_матрёшек» делает нечто противоположное эффекту функции «Число_матрёшек». Новая функция берёт результат, полученный функцией «Число_матрёшек», и по нему строит исходный аргумент этой функции, т. е. выполняет обратное действие.

Определение 5. Функция называется обратной, если по результату некоторой другой прямой функции она восстанавливает её аргумент.

Задача 3. Надо выяснить, сколько разобрано матрёшек, а потом снова собрать их в одну.

Аргумент: Матрёшка, возможно содержащая внутри какое-то число меньших матрёшек.

Надо: Выяснить, сколько матрёшек внутри, а потом восстановить исходную матрёшку.

Результат: Число спрятанных матрёшек и восстановленная матрёшка.

На первый взгляд, решение задачи может быть простой комбинацией решений задач 1 и 2.

Вопрос 3.1) Можно ли просто решить эту задачу, последовательно решая задачу 1, а потом задачу 2?

Ответ 3.1) В принципе да, но на всякий случай рассмотрим задачу отдельно, учитывая существование готовых решений подзадач 1 и 2.
Рецепт 3.1) Анализируем задачу 3 в целом, сравнивая с последовательностью решений подзадач 1 и 2 .

Вопрос 3.2) Можно ли решения подзадач 1 и 2 **согласовать** так, чтобы решение задачи стало проще или эффективнее, чем последовательное решение подзадач 1 и 2?
Ответ 3.2) Да, надо при решении задачи 1 разбираемые части матрёшек располагать по порядку, тогда при решении задачи 2 упрощается выбор очередной матрёшки, её видно сразу.
Рецепт 3.2) Уточняем решения задач 1 и 2 так, чтобы исключить необходимость поиска ближайшей, очередной матрёшки, а просто брать её по очереди.

Вопрос 3.3) Чем оправдана дополнительная работа по уточнению готовых решений?
Ответ 3.3) Новые решения могут быть эффективнее и надёжнее.
Рецепт 3.3) При уточнении решений задач 1 и 2 оцениваем выигрыш от оптимизации функции выбора очередной матрёшки и от наследования правильности схем управления готовыми решениями.

Вопрос 3.4) Не может ли быть ошибок в расстановке частей матрёшек, мешающих гарантированному получению выигрыша от такого решения?
Ответ 3.4) Чтобы избегать таких ошибок, для размещения частей по порядку следует выстроить специальный список и каждую новую часть располагать в начале списка.
Рецепт 3.4) В определениях функций разбора и сборки учесть использование списка в качестве структуры данных,

обеспечивающей дисциплину защищённого доступа к элементам структуры.

Предварительный анализ и описание процесса решения:
При описании решения этой задачи сразу учитываем, что понадобится не только разбирать матрёшек, но и собрать их обратно, что можно сделать с помощью уточнённой функции «Число_матрёшек 3».

1) Заново решаем задачу 1 с небольшой поправкой в пункте 2:
1.2.1. Если матрёшка не пуста, то её можно разнять и вынуть внутреннюю матрёшку, а части внешней матрёшки ставить рядом по порядку, чтобы потом их было удобно брать для сборки — сразу видно Наименьшую. Получится ряд по убыванию роста матрёшек.

2) Заново решаем задачу 2, но с учетом того, что части разобранных матрёшек стоят по порядку, от самой большой до самой маленькой. Значит, пункт 2 можно уточнить:
1.2.2. Брать матрёшки просто в порядке, обратном разбору, начиная с последней, как из **стека.**

3) Вводим уточнённые обозначения:
Разбор3 - «Разбираем внешнюю матрёшку и ставим её части по порядку в ряд, организованный как список».
Очередная3 - «Выбираем очередную по порядку, т. е. вторую по величине матрёшку от начала списка».
Разбор_и_сборка - «Выяснить, сколько матрёшек, и снова собрать их в одну».

Уточнение действий «**Разбор**» и «**Очередн**ая» вызвано тем, что задачи 1 и 2 решаются совместно, они согласованы и подчинены целям задачи 3, допускающим **оптимизацию.**

Порядок размещения и выбора в функциях **Разбор3** и **Очередная3** взаимно противоположный. При выполнении функции Разбор3 формируется промежуточное данное — список, специально устроенный так, чтобы упростить выполнение функции **Очередная3**.

> *Определение* 6. Преобразование представления функции называется оптимизацией, если полученная в результате преобразования функция обладает улучшенными характеристиками, например, выполняется быстрее.

Краткая запись решения задачи 3.
4) Решение общей задачи можно представить следующим образом:

Число_матрёшек3 = (ветвление
(Звук Разбор3 Число_матрёшек3)
) Подсчёт

Собрать_матрёшек3 = (ветвление
(Части? (Спрятать Наименьшую в **Очередную3)**
Собрать_матрёшек3)
) Сборка_завершена

Разбор_и_сборка = Число_матрёшек3
Собрать_матрёшек3

Важно, что в результате уменьшилась техническая сложность решения второй задачи – выбор матрёшек по порядку из списка проще, чем поиск ближайшей по размеру из произвольного множества. Согласование решений двух задач для более общей третьей задачи заключается в предварительной договорённости о расположении

промежуточных результатов, достигнутой при формулировке вопросов и анализе возможных решений. Примерно так устроена работа стека для организации вычислений в языках высокого уровня. Нечаянная перестановка частей матрёшки могла бы помешать процессу решения задачи.

Если нужно результатом функции **Разбор_и_сборка** получить число подсчитанных матрёшек, то может помочь вспомогательная функция **Результат1**, выдающая результат первой функции в качестве общего результата последовательности функций.

Результат1 – функция выбора результата первой функции в качестве результата последовательности функций.

Разбор_и_сборка_результат – «Выяснив, сколько матрёшек, снова собрать матрёшек в одну, а результатом сделать ранее полученное число матрёшек».

Представление функции несколько изменится в предположении, что результатом функции является результат, декларированный объемлющей функцией Результат1:

(Разбор_и_сборка_результат) = (Результат1
Число_матрёшек3
Собрать_матрёшек3
)

В этих задачах проявляются такие понятия и особенности, как выход из рекурсии, шаг рекурсии и использование структур данных для оптимизации рекурсивных программ. Повысить надёжность процесса удаётся с помощью специально сконструированных структур, само устройство которых гарантирует четкость

информационной обработки. При необходимости можно управлять выбором общего результата.

Задача 4. Нужно сосчитать, сколько всего матрёшек внутри двух или более матрёшек за минимальное время. Рассмотрим случай двух матрёшек.

Аргументы: Две матрёшки, возможно, содержащие внутри разное число меньших матрёшек. Имеется возможность использовать более одного процессора.

Надо: Выяснить, сколько в них суммарно внутренних матрёшек.

Результат: Общее число спрятанных матрёшек.

Вопрос 4.1) Почему бы просто по очереди не подсчитать число для той и другой, а потом числа сложить?

Ответ 4.1) Сложность процесса будет сравнима с суммарным числом спрятанных матрёшек. Желательно это число сократить, используя доступные процессоры.

Рецепт 4.1) Выстроить комплекс из двух процессоров и на каждом выполнить подсчёт для одной и другой независимо, а потом полученные числа сложить.

Вопрос 4.2) Как устроить комплекс так, чтобы на каждый разместить по одной матрёшке?

Ответ 4.2) Надо список матрёшек отобразить в список процессоров.

Рецепт 4.2) Дать разные имена матрёшкам, записать список имён и по этому списку создать комплекс для независимого подсчёта числа матрёшек.

Вопрос 4.3) Каким образом от комплекса перейти к результатам подсчёта?

Ответ 4.3) На каждый элемент комплекса надо загрузить функцию подсчёта, а результаты отобразить в список, подобный списку матрёшек.
Рецепт 4.3) Подсчёт выполняем параллельно, на каждом процессоре независимо, и по завершении всех процессов получим список чисел, соответствующих результатам подсчёта.

Вопрос 4.4) Как по завершении подсчёта получить общее число матрёшек?
Ответ 4.4) Для получения итогового результата придётся просуммировать элементы полученного списка.
Рецепт 4.4) По завершении всех параллельных процессов подсчитать сумму результатов, собранных в списке.

Предварительный анализ и описание процесса решения:
1) При описании решения этой задачи сразу учитываем, что такая задача, скорее всего, будет использоваться и для произвольного числа матрёшек.
2) Запись решения должна допускать отображение списка матрёшек в список процессоров и в список результатов подсчёта.
3) Схема решения включает в себя неявное копирование функции подсчёта на процессоры.
4) Имена матрёшек можно рассматривать как значения одноимённых функций размещения матрёшек на процессоры для независимого подсчёта их характеристик.
5) Итоговый результат может быть получен после завершения всех независимых параллельных процессов. Это означает, что время работы может быть несколько больше, чем число, ожидаемое по наибольшему числу матрёшек.

Определение 7. Функция называется отображением, если её результат строится как подобие аргумента и известна зависимость элемента результата от соответствующего элемента аргумента.

Определение 8. Функция называется отображающей, если она определяет зависимость элемента результата от соответствующего элемента аргумента.

6) Вводим дополнительные обозначения:

Общее_число — функция для подсчёта общего числа в ряде матрёшек.

Сумма — функция суммирования чисел в ранее созданном списке, полученном как отображение результатов параллельных процессов.

(Матрёшки Галя Валя) — объявление функций для загрузки конкретных матрёшек с известными именами. Каждое имя становится функцией для **доступа** к именованной матрёшке.

Определение 9. Данное считается именованным, если существует метод доступа к нему по объявленному имени.

(Галя Валя) — список матрёшек для подсчёта. При необходимости расширить или изменить состав матрёшек достаточно отредактировать их список, выполняющий роль пространства итераций, по которому строится многопроцессорный комплекс (подобно представлению распараллеливаемых циклов в функциональном языке программирования SISAL).

Определение 10. Список выполняет роль пространства итераций, если он специально сформирован для отображения в многопроцессорный комплекс.

7) Краткая запись решения задачи 4.

Общее_число = (для (Галя Валя) *выполнить*
Число_матрёшек)
Сумма

Здесь символ «для» представляет специальную функцию, которая **отображает** список матрёшек в список процессоров, затем размещает на эти процессоры копии функции «Число_матрёшек» и инициирует их выполнение и сборку результатов в списке. Символ «*выполнить*» — вспомогательный, для удобочитаемости. Функция «Сумма» суммирует элементы списка, собранные в него предшествующим отображением «для».

Все эти отображения здесь упоминаются по умолчанию неявно. Для явного их представления нужны некоторые уточнения.

Дать_процесс – выделение процессора для независимого параллельного процесса, связанного со значением ранее объявленной переменной.

Разместить – загрузка на процессор данных, являющихся значением переменной, для предстоящей их обработки в пределах отдельной итерации.

X **из** (Галя Валя) – объявление **переменной** X, которая принимает значения **из списка**, для каждого из которых будет выстроена своя независимая итерация их обработки — **последовательность** выполнения функций.

Все эти конструкции неявно связаны использованием общей **переменной**. Она объявлена, получает значения из списка, а затем считается доступной в своей итерации, отдельной для каждого значения переменной. В пределах итерации переменная значения не меняет.

> *Определение* 11. Имя выполняет роль **переменной**, если в разных позициях или ситуациях она имеет разные значения.

8) Подробная запись решения задачи 4.

```
Общее_число = (для X из (Галя Валя)
                    (Дать_процесс
                      Разместить X
                      Число_матрёшек)
                    )
               Сумма
```

Полученную таким образом многопроцессорную программу более наглядно можно представить следующей таблицей:

Таблица 2.Табличная форма представления многопоточной программы

Общее_число		*Имя функции*
для X **из** (Галя Валя)		*Пространство итераций*
X = Галя	X = Валя	*Итерации для разных значений X*
(**Дать_процесс Разместить** X **Число_матрёшек**)	(**Дать_процесс Разместить** X **Число_матрёшек**)	*Список выполняемых функций, результаты которых образуют список, синхронизованный с именами обрабатываемых значений X.*
Сумма		*Общий результат обеих итераций*

В целом, описанные выше четыре задачи и их решения дали пищу ещё для дюжины определений, не считая специальных функций (ветвление, иначе, для) и ряда чисто синтаксических форм.

Приложение 2

Вопросы для контроля уровня знаний

Предлагаются вопросы, на которые даны правдоподобные ответы[11]. Из них надо выбрать ответ, наиболее соответствующий идеям функционального программирования.

Комплект 1

Задача 1. Выберите примеры аналогов принципу «универсальность функций».

Вариант 1. В каких случаях универсальные приборы удобнее, чем специализированные?

+ Кухонный комбайн удобнее, чем отдельно мясорубка, миксер, блендер, соковыжималка и т. п. Он меньше занимает места.

- Если места достаточно, то удобнее иметь отдельные приборы.

- Применение комбайнов обычно сложнее, чем отдельных приборов.

- Трудно разместить все насадки так, чтобы в любой момент можно было взять любую, иными словами — обеспечить к ним произвольный доступ.

Вариант 2. Универсальный складной нож — незаменимый инструмент в походных условиях

+ В дороге удобно, что в одном корпусе ножа имеются не только одно-два лезвия, но ещё и ножницы, шило, штопор, а

[11] Это отличается от традиции предлагать смесь из правильных и неправильных ответов, из которых следует выбирать правильные.

может и ещё что-нибудь, что может пригодиться совершенно неожиданно.
- При большом числе элементов нож становится хрупким, может неожиданно сломаться.
- В универсальных ножах труднее воспользоваться отдельным нужным лезвием, чем в простом ноже с одним лезвием.
- Многие прилады в таких ножах никогда не находят применения и просто остаются лишним грузом.

Вариант 3. Стиральный автомат.
+ Нет ничего удобнее при дефиците свободного времени и отсутствии возможности контролировать процесс по шагам.
- Отдельно простая машина и центрифуга дешевле.
- Если сижу обычно дома, никуда не спешу, то вполне могу более гибко управлять ходом стирки.
- Общий процесс стирки довольно длинный, и он выполняется полностью от начала до конца, приходится ждать, пока он завершится.

Задача 2. Рассмотрите, в каких случаях востребованы аналоги принципа «строгий результат».

Вариант 1. Вес дорожной сумки.
Почему в поездку укладываем всё в одну дорожную сумку?
+ Перед регистрацией мгновенно выясняется соответствие веса сумки.
- 2-4 небольшие сумки удобнее нести.
- Самая нужная вещь может оказаться на дне сумки.
- При плотной упаковке вещи могут помяться.

Вариант 2. Маникюрный набор.
Маникюрный набор обычно стоит меньше, чем сумма цен отдельных инструментов. Какие имеются ещё особенности?

+ Каждый инструмент сразу имеет своё место.
+ Имеются инструменты почти на все случаи жизни.
- Когда что-то сломается, трудно найти эквивалент.
- В комплекте могут быть лишние, ненужные вещи.

Вариант3. Органайзер.
Используя специальный органайзер для бумаг и мелочей, какие действия или работы становится удобнее выполнять?
+ Легко найти любую нужную мелочь, если всё всегда возвращать на место.
+ Сразу видно расход материалов, ясно, что надо восполнить.
- Приходится помнить, где что должно располагаться.
- Требуется приучать себя к дисциплине — всё возвращать на своё место сразу, как только вещь стала не нужна.

Задача 3. Насколько часто в жизни приходится проверять достоверность, правдоподобие, логику здравого смысла?

Вариант 1.
По прогнозу дождь, а на небе ни облачка, но беру с собой зонтик.
+ Вдруг набегут тучи, и прогноз оправдается?
- Всегда ношу с собой зонтик, так что и думать не надо.
- У меня модный, красивый зонтик, хочу им похвастаться.
- Никаких причин, не беру и всё. Терпеть не могу зонтики.

Вариант 2. По карте расстояние 1 км, почему идти пришлось полчаса?
+ Карта не учитывает рельеф местности и качество дорог.
- Просто навалилась усталость. Вот причина медленного хода.
- Из-за тяжёлых вещей пришлось время от времени отдыхать.

- Отвлекали красоты природы.

Вариант 3. Много раз проверенный маршрут, возможно, не самый короткий. Чем он привлекателен?
+ «Знакомый путь короче».
+ Нет неожиданностей, можно рассчитать время.
- Опасение, что на другом пути можно попасть в тупик.
- Хочется поискать другой путь, но это в другой раз.

Задача 4. Насколько проще думать, используя модель непрерывности или бесконечности данных и процессов их обработки, чем выяснять их возможные или допустимые размеры?

Вариант 1. В чём магия перебора чёток, помогающая размышлениям?
+ Руки автоматически работают, и это успокаивает и помогает думать.
- Бусинки приятны на ощупь.
- Можно считать до бесконечности.
- Таким образом можно измерять интервалы времени.

Вариант 2. Продавец не знает, сколько покупателей выберет данный товар. Почему он подтверждает менеджеру, что товаров хватит?
+ Он видит, что в зале покупателей меньше, чем число товаров.
- Вовсе он не уверен, просто не хочет суетиться.
- Он знает, что уже поздно и, скорее всего, больше никто не придёт.
- Ему не нравится общаться с этим менеджером.

Вариант 3. Место для ключей. В большой гостинице постояльцы часто берут и возвращают ключи. Можно ли в любой момент знать, какое число ключей отсутствует?
+ Конечно можно! Надо лишь завести счётчик, увеличивающийся на 1 при возвращении ключа и уменьшающийся на 1, когда ключ забирают.
- Если их меньше десятка, то это видно почти мгновенно.
- Пожалуй, требуется время для подсчёта.
- Какая разница сколько их, лишь бы на месте был нужный ключ.

Задача 5. Многие процессы обладают цикличностью. Бывают разные схемы циклов и разные условия завершения повторяющейся серии одинаковых шагов. В каких случаях предпочтительна конкретная схема цикла и определённое условие его завершения?

Вариант 1. Вязание до готовности изделия по его схеме. Что отличает вязание от других видов рукоделия?
+ Процесс можно прервать на любой петле, а потом продолжить.
- Вяжу, пока не надоест, не зная, соберусь ли завершить.
- Пока не кончится пряжа, я вяжу, а продолжу, если найду подходящую.
- Другие дела могут надолго отвлечь.

Вариант 2. Танцы на банкете.
«Танцы до упаду» — что означает эта броская фраза?
+ Обычно — до выключения музыки.
- Бывает, что до усталости.
- Другой партнёр может прервать танец.
- Лишь бы партнёр не уронил или не наступил на ногу.

Вариант 3. Сбивание белкового крема для вкуснейшего торта.
+ Крем перестаёт стекать с венчика.
- Пока крем не станет белым.
- Сбивать 20 минут, можно по таймеру.
- Готовность по загустению до нужной кондиции.

Задача 6. Восстановление данных или продление жизни вещей нередко требует особого внимания и специальной заботы. Почему такие накладные расходы признают целесообразными?

Вариант 1. Как понять, что стирка успешно завершена?
+ После отжима нет нигде грязных пятен.
+ «Перестир» не всегда повышает уровень чистоты.
- Полоскание в двух водах и отжим.
- «Воду видела — значит чисто»[12].

Вариант 2. Достаточно ли поработал пылесос?
+ Через несколько минут не появилась пыль на поверхностях.
- Вроде всю территорию прошли, значит хватит.
- Мягкая мебель не пылит, значит, всё в порядке.
- 10 минут попылесосили, а дальше в другой раз.

Вариант 3. Имеет ли смысл заниматься штопкой носков?
+ Приходится, если нет запасных, например, в турпоездке.
+ Пока дырка маленькая, почему бы и не заштопать?
- Конечно, если нравится штопать.
- Пожалуй, если это любимые носки или других таких не найти.

[12]Речь о варианте, когда до чистой воды дело не дошло, но процесс стирки прерван

Задача 7. Идея ленивых вычислений и частичных процессов позволяет выполнять планирование и выбор наиболее удачных моментов для выполнения конкретных действия, обеспечивая оттеснение, опережение, обработку по шагам, раскрутку программ и многое другое. Всегда ли это разумно или лучше всё сделать раз и навсегда в обычной жизни?

Вариант 1. Что даёт ежедневная уборка по шагам?
+ Каждый день по полкубометра за 15-20 минут и постепенно всё приведём в порядок.
- Если убрано не всё, то и не видно результата, есть риск возврата.
- Недоделанное снова расползётся — энтропия поглотит.
- Трудно выдержать границу между порядком и беспорядком.

Вариант 2. Оттеснение мусора в промежуточный буфер, для перемещения в мешок или ведро на выброс потом.
+ Всё ненужное в одно место — легче потом убрать.
- Всё-таки не исключено, что вдруг что-то снова понадобится.
- Нужное может случайно попасть в ненужное.
- Мешка может не хватить.

Вариант 3. Какие преимущества даёт составление расписаний?
+ Оптимизация решений, когда и что делать.
- Лишнее напряжение по контролю времени.
- Лучше делать всё по наитию, когда душа попросит.
- Бывают накладки в реальности, так что от расписаний мало толку.

Задача 8. В чём проявляется выигрыш от таких усложнённых конструкций, как многоконтактные узлы,

например, вилка-розетка, кабели-гнёзда, многоштыревые замки и т. п.?

Вариант 1. Вилка-розетка в электрических сетях.
Почему каждая вилка подходит только к своей розетке?
+ Исключён риск перепутать и соединить неподходящее.
+ Если соединилось, значит, всё в порядке.
- Если подходящей пары нет, то придётся подгонять.
- Приходится хранить много вариантов запчастей.

Вариант 2. Закручивающиеся крышки банок или бутылок. Почему у них может быть разная резьба?
+ Удобно при заготовках получать плотное закрывание.
- Всё теряется при дефектах в резьбе.
- Бывает потом трудно открыть, если крепко присосалось.
- Требуют бережного отношения, могут проржаветь.

Вариант 3. Контейнер для яиц. Зачем нужен контейнер?
+ Хрупкая вещь, без такого устройства яйца могут повредиться.
+ Легко видеть, сколько яиц размещено, и проверить, все ли целые.
- Не всегда размер подходит, бывают яйца и поменьше, и побольше.
- Лишний объём, главное — не спотыкаться и не падать.

Комплект 2.

Задача 1. Насколько в реальности заметна гибкость границ между физическими объектами?

Вариант 1. На кухне стоит два холодильника, хотя вся провизия хранится в одном, а второй выключен и стоит пустым. Чем может быть полезным второй холодильник?

+ Быстро всё перемещаем в пустой холодильник и включаем его, а первый выключаем до полной разморозки.
+ Можно не спеша попутно выбросить устаревшее, а нужное рассортировать, сразу размещая в резервном холодильнике.
- Лучше после разморозки вернуть всё как было.
- Быстрая разморозка, потом возврат в первый холодильник.

Вариант 2. Очень бывает досадно, если срочно надо вымыть фрукты, а раковина переполнена грязной посудой. Можно ли фрукты ополоснуть в любой момент?
+ Опытные хозяйки при интенсивной готовке моют посуду при каждом удобном случае, чтобы раковина была свободна.
- Мыть понемногу, не допуская переполнения.
- Просто всё выставить на стол и помыть фрукты.
- Помыть не всё, а часть, чтобы места хватило для мытья фруктов.

Вариант 3. Предстоит многолюдный домашний праздник. Можно ли квартиру превратить в банкетный зал?
+ В Болгарии часто в квартирах внутренние стены делают как раздвижные двери-купе.
- Можно сделать несколько столов в соседних комнатах.
- Отчего бы не разделить гостей на группы и праздновать по частям?
- Лучше всего пригласить всех в ресторан!

Задача 2. Часто ли обычные вещи удобно рассматривать как состоящие из независимых составляющих?

Вариант 1. Компоненты для кулинарного рецепта. Вкуснейший торт!
+ Ищу или выспрашиваю рецепт и технологию.

- Пытаюсь вспомнить вкус и экспериментально его воспроизвести.
- Рецепт слишком сложный. Попробую его упростить.
- Нет отдельных ингредиентов. Попробую заменить чем-нибудь.

Вариант 2. Всё труднее находить нужные бумаги на столе. Что делать?
+ Разложу в стопки по 4-8 темам, а там по датам.
- При поиске разобранные бумаги отложу в отдельную стопку.
- Постараюсь припомнить, где что должно быть.
- Если за полчаса не найду, то скажу что потеряно.

Вариант 3. Разные карманы в сумке при укладке вещей в дорогу. Собираюсь в сложную поездку. Как упаковать вещи, учитывая, что иногда нужно будет быстро реагировать на разные ситуации?
+ Разложу по отдельным пакетикам то, что надо в каждом пункте поездки (регистрация, салон самолёта, сон, ванная, гостиница и т. д.).
- Быстро всё побросаю в рюкзак, а если что-то понадобится, то всё вытряхну, а потом всё верну обратно.
- Разделю вещи на багаж и ручную кладь, а там сложу их по категориям.
- Главное — упаковать парадную одежду для выступления.

Задача 3. Нередко в разных сложных работах возникает острое желание вернуться на несколько шагов назад и переделать всё заново.

Вариант 1. Задача восстановления кубика Рубика разрешима. Тем не менее, сделать это для произвольной конфигурации весьма непросто.

+ Записывать ходы от исходной позиции, а потом идти обратно.
- Поискать описание стратегий и методов.
- Пытаться, пока не получится.
- Собрать хоть что-то понятное, например, углы или линии.

Вариант 2. Сборка-разборка сложных приборов.
+ Делать фото каждого шага разборки.
- Укладывать детали по порядку.
- Найти схему устройства прибора.
- Поискать координаты знакомых знатоков, пусть советуют.

Вариант 3. Прогуливаясь по городу, опасно потерять ориентировку.
+ Методично фиксировать все приметы — статуи, фонтаны и пр.
- О чём беспокоиться? Всегда можно поспрашивать у прохожих.
- Придётся звонить знакомым.
- Двигаться в одном направлении, до знакомого места.

Задача 4. Отказ от перебора ненужных работ обычно подобен предварительному определению типов данных, позволяющих сразу отбраковать бесперспективные действия. Иногда проверки типов данных воспринимаются как накладные расходы. В чём можно видеть пользу от такого механизма?

Вариант 1. Разметка брёвен для дома. При строительстве домов из брёвен практикуют их разметку. На каждом бревне пишут обозначение угла дома и номер венца снизу. По каким причинам нужна столь тщательная разметка?

+ При строительстве нет стандарта на размеры брёвен. Каждое из них индивидуально подгоняется к своему месту.
+ Обычно первичная сборка выполняется на специальной строительной площадке, а потом дом разбирают и всё перевозят на другое место.
- По номеру можно узнать, кто отёсывал конкретное бревно.
- Традиционная технология, так делали предки.

Вариант 2. Приборы для еды. Почему предпочитают типовую схему раскладки приборов и закусок на все столы?
+ Сразу видно, всё ли на месте.
- Официант заранее знает, куда и что следует положить.
- Это красиво!
- Легко подсчитать, сколько чего надо подготовить.

Вариант 3. Чем диктуется разница в выборе одежды на работу, в сад, на спортивную площадку, в поездку, в больницу и др.?
+ Пониманием того, что там придётся делать.
- Прежде всего, учётом традиций, приличий и гигиены.
- Главное — красота, идёт или нет.
- Приходится учитывать цену одежды.

Задача 5. Всегда ли удаётся дополнительную нагрузку на основные действия выполнять без ущерба для целевого процесса?

Вариант 1. Собираюсь в мастерскую отнести ключ, чтобы сделать копию. Друг просит попутно прихватить и его ключ. Почему?
+ По его мнению, это не потребует дополнительных усилий и времени.
+ Оптимизирует, чтобы самому не пришлось это делать.

- Любит, чтобы его обслуживали.
- Боится, что сам не выберет времени.

Вариант 2. Иду в магазин для покупки тёплой обуви. Родственник просит посмотреть цену на домашние тапки.
+ Нагрузка небольшая, почему бы и не посмотреть.
- Пусть смотрит сам, мало ли, какие тапки его интересуют.
- Мог бы позвонить по телефону в магазин и узнать, но ему лень.
- Это он просто хочет пообщаться, тапки ему не так уж нужны.

Вариант 3. Пишу письмо брату. Его приятель просит попутно отправить ему фото. Стоит ли выполнять такую просьбу?
+ Дело секундное, почему бы и нет?
- Лучше пусть сам напишет и пошлёт всё, что хочет.
- Брат давно ничего не писал, может лучше потом.
- От меня не убудет, но мне почему-то не хочется.

Задача 6. Пространство итераций — типичный пример однородных многопоточных программ. Насколько часто встречаются задачи, удобные для решения в виде серии одинаковых действий?

Вариант 1. Подарки детям на утренник делают одинаковыми для всех.
+ Просто для удобства быстрой раздачи без индивидуального подхода.
- Важно, чтобы дети не завидовали друг другу.
- Слишком трудно учитывать индивидуальность при раздаче.
- Такова традиция.

Вариант 2. Футболки членам спортивной команды.

+ При игре на поле легко видеть, кто чей игрок.
- Так красивее, как на параде.
- Вот и зря — выглядят, как «инкубаторские».
- В команде можно особо не различать, где чья одежда.

Вариант 3. В столовой погас свет и вместо шведского стола посетителям выдали сухой паёк, одинаковый для всех.
+ Это штатная процедура, она определена в регламенте реагирования.
- Раздача в столовой не приспособлена к выдаче разной еды.
- Опасения, что некоторые слишком много наберут с собой.
  – Легче учесть, хватит ли на всех еды.

Задача 7. Автоматическое распараллеливание похоже на методы распределения работ в коллективе исполнителей.

Вариант 1. На методологическом семинаре ВЦ СО АН однажды разыгралась такая сцена. Доска в конференц-зале была обустроена шторками с двух сторон, чтобы докладчик мог заранее её зашторить, и слушатели не увидели бы написанное раньше времени. Однажды докладчик решил мешающие ему шторки задвинуть за доску. Он успешно сделал это справа и перешёл к левой стороне. Неожиданно оказалось, что только он тронул левую сторону доски, как правая шторка выскользнула вперёд и снова часть доски закрыла. Докладчик вернулся вправо и, возможно, этот казус мог повториться многократно, но в аудитории нашлась сообразительная особа. Она быстро вскочила со своего места, подошла к левой стороне, они вместе с докладчиком синхронно задвинули шторки за доску, и докладчик обрёл необходимое пространство. Аудитория единодушно ахнула, и прошелестел громкий шёпот: «Это истинный параллелизм».

Можно ли это рассматривать как доказательство несводимости параллельных вычислений к однопроцессорным конфигурациям?
+ Конечно! Здесь нужны два процессора — на каждую шторку свой.
- Докладчику просто не хватило длины рук.
- Ну и не стоило так много писать на доске.
- Докладчик мог подумать заранее и припасти прищепки.

Вариант 2. При посадке и уборке картофеля обычно работу ведут два-три человека, распределив между собой функции.
+ Каждая из этих функций связана с разными инструментами.
- Можно всё это делать и в одиночку, не так уж сложно.
- Это дань традиции.
- Партнёры могут и мешать друг другу.

Вариант 3. Обычно в огороде на разных грядках растут различные овощи. В чём выигрыш от такого способа выращивания овощей?
+ Легко определить полезные растения при прополке.
- Так советуют опытные люди, овощи не все совместимы.
- Однородные посадки красиво выглядят, как колонны на параде.
- Сеем каждый пакетик на отдельную грядку.

Задача 8. Рекурсия как метод самоприменимости функций — одна из самых популярных схем в математике. Насколько техника рекурсии полезна при решении каких-нибудь задач в обычной жизни?

Вариант 1. Можно ли описать процесс выяснения числа матрёшек?
+ Да, можно: прибавить 1 к числу внутренних матрёшек.

- Да, разнимая матрёшку, можно считать число разобранных матрёшек.
- Можно сначала все разобрать, а потом полученные детали сосчитать.
- Зачем считать? На упаковке, возможно, написано число матрёшек.

Вариант 2. Как узнать в какой из двух коробок карточек больше?
+ Если обе коробочки пусты, то одинаково. Если одна пуста, а в другой карточки имеются, то в ней карточек больше. Если обе заполнены, то вынимаем по карточке и сравниваем то, что осталось.
- Вынимаем карточки из коробок в стопки и сравниваем их по высоте.
- Подсчитываем число карточек и сравниваем полученные числа.
- Вынимаем из каждой коробки по одной карточке до тех пор, пока одна из коробок не окажется пустой.

Вариант 3. При сборе урожая заполнено несколько мешков разного размера, и потому они разного веса. Требуется узнать вес урожая.
+ Берём и взвешиваем первый попавшийся мешок. Полученное число прибавим к результату взвешивания оставшихся мешков.
- Взвешиваем мешки и суммируем полученные веса.
- Упорядочиваем мешки по объёму, записываем веса и их суммируем.
- Находим большие весы и на них загружаем все мешки сразу.

Комплект 3.

Задача 1. Можно ли принцип «От перемены мест слагаемых сумма не меняется» распространить на рецептуру блюда, меню в ресторане?

Вариант 1. Можно ли правило перестановочности слагаемых распространить на репертуар театра?
+ Конечно, не так уж важно, в каком порядке перечислены спектакли.
- Всё-таки удобнее, когда репертуар упорядочен по датам.
- Лучше, когда репертуар начинается с самых интересных спектаклей.
- Можно упорядочить и по алфавиту, легче искать.

Вариант 2. От порядка элементов кулинарный рецепт не меняется.
+ Только при записи рецепта, а при изготовлении порядок важен.
- Удобнее упорядочить по весу или цене.
- Хорошо, когда выделена группа близких по смыслу ингредиентов.
- Люблю, когда учтён порядок обязательности или дефицита.

Вариант 3. Насколько важен порядок блюд в категории меню?
+ Если внутри категории блюд немного, их хорошо видно.
- Всё-таки лучше упорядочивать по ценам.
- Хорошо бы упорядочить по времени готовки.
- Можно поддерживать стандартное расположение.

Задача 2. Чем полезно стабильное расписание?

Вариант 1. В чём выигрыш от стабильного расписания автобусов?

+ Возрастает число пассажиров, проезжающих одну-две остановки.
- Пусть меняется, лишь бы соблюдалось.
- Толку от постоянства, если его не помнишь.
- Надёжнее на сайте смотреть, где сейчас автобус.

Вариант 2. В вашем кабинете заменили ваш стол на новый.
+ Нарушены привычки ходьбы и пользования ящиками.
- Вот и зря! Это же имидж, одобрение гостей!
- Ничего, привычки новые вскоре появятся.
- Сам просил, не ожидал, что это будет неприятно.

Вариант 3. Что вы почувствуете, нарвавшись на новые запертые ворота?
+ Произошло нарушение прав на движение привычным путём.
- Если не спешу, то просто вернусь и пойду другим путём.
- Поищу лазейку. Может кто-то уже её сделал.
- Люблю неожиданности, они украшают жизнь эмоциями.

Задача 3. Удаётся ли в делах видеть, что результат достигнут?

Вариант 1. Почему перевод текста часто отличается от оригинала?
+ Грамматика языка оригинала отличается от грамматики целевого языка.
- Переводчик не в ладах с исходным языком — обычная версия.
- Просто переводчик особо не вникал в смысл и речевые стереотипы.
- Отсутствует владение смыслом переводимого материала.

Вариант 2. Почему при чтении детектива тянет заглянуть в конец?
+ В конце детектива часто размещён краткий пересказ происходящего.
- Лень читать все подробности, хочется скорее узнать развязку.
- Хочется проверить свои догадки.
- При чтении хочется видеть, куда клонит автор.

Вариант 3. Зачем при постановке фильма нужен сценарий?
+ Компактное представление позволяет участникам понимать цели.
- Может и не надо, фильм часто отклоняется от сценария.
- Зато сценарист получит гонорар!
- Так принято делать, хотя, может, это не обязательно.

Задача 4. Часто ли сводят ряд действий к работе с одним объектом?

Вариант 1. Если в одежде хватает карманов, то нужен ли кошелёк?
+ Да, в кошельке все деньги сразу, да и карманы не портятся.
- Может, без кошелька удобнее деньги разложить в разные карманы.
- Из карманов легче деньги доставать.
- Кошелёк легко потерять со всеми деньгами.

Вариант 2. Что заставляет предпочитать рюкзак, особенно в походе?
+ Руки свободны!
- Ничего подобного, лучше сумка на колёсиках.
- А мне лучше две сумки на плечи для равной нагрузки на позвоночник.

- У меня красивый рюкзак, всегда беру его с собой для красоты.

Вариант 3. Насколько поднос улучшает работу официанта?
+ Поднос позволяет компоновать заказы или сокращать маршруты.
- Поднос провоцирует носить избыточную тяжесть.
- Работа с подносом требует выучки, что не у каждого получится.
- Комплектация на подносе создает необходимость помнить, что кому принести.

Задача 5. Что пользы смотреть прогнозы, они не всегда сбываются.

Вариант 1. Почему, несмотря на ошибки, все смотрят прогноз погоды?
+ «Желают знать, что будет» — сравнение с интуитивным прогнозом.
+ Вероятность ошибок строго меньше половины.
+ На всякий случай.
- Ничего подобного, лучше посмотреть в окно!

Вариант 2. Зачем пишут список вещей, нужных в командировку?
+ Резко сокращается число ситуаций, застающих врасплох.
+ Полезно, если ездишь часто — будет практичный список.
- В список попадает слишком много лишних вещей.
- Без толку! Всё предусмотреть невозможно.

Вариант 3. В чём смысл рекомендаций врача, если каждый выздоравливает сам, по-своему?

+ Заболеть можно быстро, а выздороветь — это трудно и долго.
+ Смысл в профилактике риска сорваться и снова заболеть.
- Никакого смысла! Говорят всем одно и то же, а мы все разные.
- Сам я лучше знаю, у меня интуиция и внутренний голос.

Задача 6. Методика запоминания результатов действий позволяет не повторять такие действия.

Вариант 1. Чем ртутный термометр удобнее спиртового?
+ Ртутный термометр сохраняет показания, пока их не стряхнёшь.
-Он удобен в чём-то, но и опасен.
- Ничего удобного, приходится долго ждать, пока нагреется.
- Мне он просто не нравится, значит и не удобен.

Вариант 2. Зачем в комплект мультиварки входит кулинарный справочник?
+ Полезно каждому клиенту дать информацию о времени готовки.
- Простая накрутка цены.
- Учёт интересов рекламы производителей продуктов.
- Никогда не смотрю инструкции, всё проверяю в эксперименте.

Вариант 3. Зачем товар в магазине сопровождается этикетками?
+ Такая практика помогает покупателю знать стоимость покупок.
- На этикетках встречаются и неправильные размеры.
- Ценники часто не удаётся разглядеть.

- Неискушённый покупатель так может узнать название товара.

Задача 7. Какие проблемы требуют уменьшения разницы между объёмами работы отдельных исполнителей?

Вариант 1. Проблема оценки трудового вклада.
+ Происходит равнение на среднего, и продуктивность снижается.
- Такой подход большинство людей воспринимает как справедливость.
- Такой подход даёт право считать себя полезными.
- Даже если кто-то умеет оценивать, ему трудно доказать, что он прав.

Вариант 2. При переноске тяжестей важно равновесие справа и слева.
+ Позвоночник не любит сильного перекоса в одну сторону.
- Перекошенная от неравномерной нагрузки фигура выглядит смешно.
- Так рекомендуют инструкторы физкультуры, они знают.
- Лучше тяжести вообще в руках не носить, существуют тележки.

Вариант 3. Соревнуются спортсмены одной весовой категории.
+ При явном различии в весе соревнования теряют зрелищность.
- Искусство спортсмена может побеждать любых противников.
- Скорее всего, это просто профилактика бессмысленных травм.
– Наверное, это дань традиции.

Задача 8. Насколько часто нужны повторы, репетиции, тренировки?

Вариант 1. Чем отличаются репетиции друг от друга, а генеральная репетиция принципиально отличается от остальных?
+ Репетиции позволяют уточнять решения, а генеральная репетиция позволяет убедиться, что принятых решений достаточно.
- Просто так принято, последнюю репетицию называть генеральной.
- Генеральная репетиция проводится на настоящей сцене и в правильных костюмах.
- К генеральной репетиции все актёры уже выучили пьесу, и на ней обязательно присутствует режиссёр.

Вариант 2. Исполнительское мастерство оттачивает цепочки действий до автоматизма.
+ Мозг, по-видимому, убирает сознательный контроль за порядком выполнения действий и паузами между действиями.
+ Мозг способен заблокировать продолжение тренировок.
- Происходит запоминание цепочки действий в памяти.
- Бывают ещё и импровизации, в которых мозг работает иначе.

Вариант 3. Чем можно объяснить эффективность разучивания животными сложных цирковых номеров с последнего элемента?
+ Так происходит слияние предшественника с преемником, не требуя внимания и сокращая время исполнения.
- Никто так не делает, все разучивают всё от начала к концу.
- Может и так, но в это трудно поверить.
- Животные — это одно, а человек — другое.

Утверждено к печати в электронном виде
Редакционным советом Института систем информатики СО РАН

Л.В. Городняя

НЕФОРМАЛЬНОЕ ВВЕДЕНИЕ В ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ
(ОБРАЗЫ, АНАЛОГИИ, ПОДОБИЯ)

Препринт
188

Редактор А.А. Шелухина
Рецензент Т.А. Андреева