Содержание

О новой версии классического учебника Никлауса Вирта

Новая версия учебника Н. Вирта «Алгоритмы и структуры данных» отличается от английского прототипа [1] сильнее, чем просто исправлением многочисленных опечаток и огрехов, накопившихся в процессе тридцатилетней эволюции книги. Объясняется это целями автора и переводчика при работе над книгой в контексте проекта «Информатика-21» [2], который, опираясь на обширный совокупный опыт ряда высококвалифицированных специалистов (см. списки консультантов и участников на сайте проекта [2]), ставит задачу создания единой системы вводных курсов информатики и программирования, охватывающей учащихся примерно от 5-го класса общей средней школы по 3-й курс университета. Такая система должна иметь образцом и дополнять уникальную российскую систему математического образования. Это предполагает наличие стержня общих курсов, составляющих единство без внутренних технологических барьеров (которые приводят, среди прочего, к недопустимым потерям дефицитного учебного времени) и лишь варьирующихся в зависимости от специализации, вместе с надстройкой из профессионально ориентированных курсов, опирающихся на этот стержень в отношении базовых знаний учащихся. Такая система подразумевает наличие качественных учебников (первым из которых имеет шанс стать данная книга), «говорящих» на общем образцовом языке программирования. Естественный кандидат на роль такого общего языка – Оберон/Компонентный Паскаль. Подробней об Обероне речь пойдет ниже, здесь только скажем, что Паскаль (использованный в первом издании данной книги 1975 г.), Модулу-2 (использованную во втором издании, переведенном на русский язык в 1989 г. [3]) и Оберон (использованный в данной версии) логично рассматривать соответственно как альфа-, бета- и окончательную версию одного и того же языка. Использование Оберона – самое очевидное отличие данной версии книги от предыдущего издания.

В контекст идеи о единой системе вводных курсов вписывается и узкая задача, решавшаяся новой версией учебника, – дать небольшое продуманное пособие, в котором аккуратно, но не топя читателя в болоте второстепенных деталей, прорабатывались бы традиционные темы классической алгоритмики, для полного обсуждения которых нет времени в спецкурсе, читаемом переводчиком с 2001 г. на физфаке МГУ в попытке обеспечить хотя бы минимум культуры программирования у будущих аспирантов. Здесь требуется «отлаженный» текст, пригодный для самостоятельной работы студентов. С точки зрения содержания, лучшим кандидатом на эту роль оказался прототип [1].

Что двойное переделывание программ и рассуждений в тексте (с Паскаля на Модулу-2 и затем на Оберон) не прошло безнаказанно, само по себе неудивительно. Однако затруднения, возникшие при верификации программ и текста, хотя и были преодолены, все же показались чрезмерными. Поэтому, и ввиду учебного назначения книги, встал ребром вопрос о необходимости доработки примеров. Предложения переводчика были одобрены автором на совместной рабочей сессии в апреле сего года и реализованы непосредственно в данном переводе (при первой возможности соответствующие изменения будут внесены и в прототип [1]).

Во-первых, алгоритмы поиска образца в тексте переписаны в терминах цикла Дейкстры (многоветочный while [4]). Эта фундаментальная и мощная управляющая структура поразительным образом до сих пор не представлена в распространенных языках программирования, поэтому ей посвящено новое приложение С. Раздел 1.9, в который теперь выделены эти алгоритмы, будет неплохой иллюстрацией реального применения цикла Дейкстры. Вторая группа заметно измененных программ – алгоритмы с возвратом в главе 3, в которых теперь эксплицировано применение линейного поиска и, благодаря этому, тривиализована верификация. Такое прояснение рекурсивных комбинаторных алгоритмов является довольно общим. Обсуждались – но были признаны в данный момент нецелесообразными – модификации и некоторых других программ.

Надо заметить, что программистский стиль автора вырабатывался с конца 1950-х гг., когда проблема эффективности программ висела над головами программистов дамокловым мечом, и за несколько лет до того, как Дейкстра опубликовал систематический метод построения программ [4]. В старых версиях книги заметна рефлекторная склонность к оптимизации до полного прояснения логики программ, что затрудняло эффективное применение формальной техники. Это легко объяснить: Н. Вирт осваивал только еще формирующиеся систематические методы, непосредственно участвуя в процессе создания программирования как академической дисциплины, версия за версией улучшая свои учебники.

Но и через четверть века после последней существенной переделки учебника автором аналогичная склонность к преждевременной оптимизации при не просто не вполне уверенной, а напрочь отсутствующей формальной технике – и, как следствие, запутанные циклы, – характерные черты стиля «широких программистских масс»! В профессиональных интернет-форумах до сих пор можно найти позорные дискуссии о том, нужно ли учиться писать циклы по Дейкстре, – и это в лучшем случае. Если же вообразить себе весь окружающий нас непрерывно растущий массив софта, от которого наша жизнь зависит все больше, то впору впасть в депрессию: *Quo usque tandem, Catilina*? – Сколько еще нужно *десятилетий*, чтобы система образования вышла, наконец, на уровень, давным-давно достигнутый наукой? Во всяком случае, ясно, что едва ли не главная причина проблемы – хаос, царящий в системе ИТ-образования, тормозящий создание и распространение качественных методик и поддерживаемый, среди прочего, корыстными интересами «монстров» индустрии.

Здесь уместно сказать о языке Оберон/Компонентный Паскаль, пропагандируемом в качестве общей платформы для предполагаемой единой системы курсов

программирования. Оберон – последний большой проект Никлауса Вирта, выдающегося инженера, ученого и педагога, вместе с Бэкусом, А. Ершовым, Дейкстрой, Хоором и другими пионерами компьютерной информатики превратившего программирование в систематическую дисциплину и лучше всего известного созданием серии все более совершенных языков программирования – Паскаля (1970), Модулы-2 (1980) и наконец Оберона (1988, 2007). В этих языках отражалось все более полное понимание проблематики эффективного программирования. Языки эти сохраняют идейную и стилевую преемственность, и коммерсант, озабоченный сохранением доли рынка, не назвал бы их по-разному (ср. зоопарк бейсиков). Чтобы подчеркнуть эту преемственность, самому популярному диалекту Оберона было возвращено законное фамильное имя – Компонентный Паскаль.

Оберон/Компонентный Паскаль унаследовал лучшие черты старого доброго Паскаля и добавил к ним промышленный опыт Модулы-2 (на которой программируются, например, российские спутники связи [5]), а также выверенный минимум средств объектно-ориентированного программирования. Принципальное достижение – удалось наконец добиться герметичности системы типов (теперь ее нельзя обойти средствами языка даже при работе с указателями). Это обеспечило возможность автоматического управления памятью (сбора мусора; до Оберона сбор мусора оставался прерогативой динамических языков – функциональных, скриптовых и т. п.) В результате диапазон эффективного применения Оберона, похоже, шире, чем у любого другого языка: это и вычислительные приложения, и системы управления любого масштаба (от беспилотников весом в 1 кг до грандиозных каскадов ГЭС), и, например, задачи символической алгебры с предельно динамичными структурами данных.

Особо следует остановиться на минимализме Оберона. Традиционно разработчики сосредоточиваются на том, чтобы снабдить свои языки, программы, библиотеки «богатым набором средств» – ведь так легче привлечь клиента, надеющегося побыстрее найти готовое решение для своих прикладных нужд. Погоня за «богатым набором средств» оборачивается ущербом качеству и надежности системы. Вместе с коммерческими соображениями это приводит к тому, что получается большая закрытая сложная система с вроде бы богатым набором средств, но хромающей надежностью и ограниченной расширяемостью, так что если пользователь сталкивается с нестандартной ситуацией в своих приложениях (что случается сплошь и рядом – ведь разнообразие реального мира превосходит любое воображение писателей библиотек), то он оказывается в тупике.

Н. Вирт еще со времен Паскаля, созданного в пику фантазийному Алголу-68 [6], пошел другим путем. Его гамбит заключался в том, чтобы, отказавшись от включения в язык *максимума* средств на все случаи жизни, тщательнейшим образом выделить *минимум* реально ключевых средств, – обязательно включив в этот минимум все, что нужно для безболезненной, неограниченной расширяемости программных систем, – и добиться высоконадежной реализации такого ядра. Этот замысел был с блеском реализован Н. Виртом и его соратником Ю. Гуткнехтом в проекте Оберон [7]. Минимализм и уникальная надежность Оберона

заставляют вспомнить автомат Калашникова. При этом вся мощь Оберона оказывается открытой даже программистам-непрофессионалам – физикам, инженерам, лингвистам.., занятым программированием изрядную долю своего рабочего времени.

Для преподавателя важно, что в Обероне достигнуты ортогональность и свободная комбинируемость языковых средств, смысловая прозрачность, а также беспрецедентно малый для столь мощного языка размер (см. полное описание синтаксиса в приложении B, а также обсуждение в [8]). В этом отношении Оберон побеждает за явным преимуществом традиционные промышленные языки, пресловутая избыточная сложность которых оказывается источником своего рода ренты, взимаемой с остального мира. Оберон скромно уходит в тень при рассмотрении любой языково-неспецифичной темы – от введения в алгоритмику до принципов компиляции и программной архитектуры. А после постановки базовой техники программирования на Обероне изучение промышленных языков зачастую сводится к изучению способов обходить дефекты их дизайна. Если уже старый Паскаль оказался настолько удачной платформой для обучения программированию, что принес своему автору высшую почесть в компьютерной информатике – премию им. Тьюринга, то понятно, что буквально вылизанный Оберон/Компонентный Паскаль называют уже «практически идеальной» платформой для обучения программированию (см. также предисловие к [12]).

Имея в виду исключительные педагогические достоинства Оберона, для всех примеров программ, приведенных в книге, обеспечена воспроизводимость в системе программирования для Компонентного Паскаля, известной как Блэкбокс (BlackBox Component Builder [9]). Это пулярный вариант Оберона, созданный для работы в распространенных операционных системах. Конфигурации Блэкбокса для использования в школе и университете доступны на сайте проекта «Информатика-21» [2]. Открытый, бесплатный и безупречно современный Блэкбокс оказывается естественной заменой устаревшему Турбо Паскалю – заменой тем более привлекательной, что, несмотря на минимализм и благодаря автоматическому управлению памятью, это более мощный инструмент, чем промышленные системы программирования на диалектах старого Паскаля. Краткое описание возможностей Блэкбокса с точки зрения использования в школьных курсах можно найти в статье [10].

Важное приложение к книге – полный комплект программ, представленных в тексте учебника, в виде, готовом к выполнению. Программы оформлены в отдельных модулях вместе с необходимыми вспомогательными процедурами, и все такие модули собраны в папке ADru/Mod/, которая должна лежать внутри основной папки Блэкбокса (следует иметь в виду, что файлы с расширением *.odc должны читаться из Блэкбокса). Читатель без труда разберется с компиляцией и запуском программ по комментариям в модулях, читая модули в том порядке, в каком они встречаются в тексте книги (или в лексикографическом порядке имен файлов). В тексте книги в начальных строках каждого законченного программного примера справа указано имя соответствующего модуля. Например, комментарий (*ADruS18_Поиск*) означает, что данная программа содержится в модуле

ADruS18_Поиск, который в соответствии с правилами Блэкбокса хранится в файле ADru/Mod/S18_Поиск.odc. При этом речь идет о программе из раздела 1.8, а необязательный суффикс "_Поиск" служит удобству ориентации. Вся папка ADru в составе Блэкбокса имеется на диске, если диск приложен к книге, либо может быть скачана с адреса [11].

Наконец, несколько слов о собственно переводе. Старый перевод [3] был выполнен, что называется, из общих соображений. Но совсем другое дело – иметь в виду конкретных студентов, не обязательно будущих профессиональных программистов, пытающихся за минимальное время овладеть основами программирования. Поэтому в новом переводе были предприняты особые усилия, чтобы избежать размывания смысла из-за неточностей, неизбежно вкрадывающихся при неполном понимании переводчиком оригинала (ср. примечание на с. 110 в главе о сортировках в [3], где выражена надежда, что «сам читатель разберется, что хотел сказать автор»). Например, при более-менее прямолинейной пофразовой интерпретации малейшая неточность способна развалить смысл лаконичного текста Вирта из-за того, например, что после перевода могут перестать быть однокоренными слова, благодаря которым только и обеспечивалась смысловая связь между предложениями в оригинале. Поэтому добиться полного сохранения смысла при переводе оказалось проще, выполнив его с нуля.

В отношении терминологии переводам специалистов было отдано должное. Вслед за Д. Б. Подшиваловым [3] мы используем прилагательные «массивовый», «последовательностный» и «записевый». Решающий довод в пользу таких прилагательных – они естественно вписываются в грамматическую систему русского языка, чем обеспечивается необходимая гибкость выражения.

Однако даже в отношении терминологии переводы по компьютерной тематике часто демонстрируют неполное понимание существенных деталей английской грамматики. Например, при использовании существительного в качестве определения в препозиции (что, кстати, не эквивалентно русской конструкции, выражаемой родительным падежом) множественное число может нейтрализоваться, и при переводе на русский его иногда нужно восстанавливать. Так, path length должно переводиться не как «длина пути», а как «длина путей», что, между прочим, прямо соответствует математическому определению и ощутимо помогает понимать рассуждения. Optimal search tree – «оптимальное дерево поиска», а не «дерево оптимального поиска». Advanced sort algorithms – «эффективные алгоритмы сортировки», потому что буквальное значение advanced в данном случае давно нейтрализовано. Переводить на русский язык двумя словами специфичные для стилистики английского языка синонимичные пары вроде «methods and techniques» обычно неразумно. И так далее. Масса подобных неточностей снижает удобочитаемость текста и затемняет и без того непростой смысл оригинала.

Хотя по конкретным стилистическим вопросам копья можно ломать до бесконечности, все же хочется надеяться, что предпринятые усилия в основном достигли цели – не потерять точный смысл английского «исходника» этого выдержавшего проверку временем прекрасного учебника.

Троицк, Московская обл., июль 2009 Ф. В. Ткачев

[1] Wirth N. Algorithms and Data Structures. Oberon version: 2004 //http://www.inr.ac.ru/~info21/pdf/AD.pdf
[2] Информатика-21: Международный общественный научно-образовательный проект // http://www.inr.ac.ru/~info21/
[3] Н. Вирт. Алгоритмы и структуры данных / пер. с англ. Д. Б. Подшивалова. – М.: Мир, 1989.
[4] Дейкстра Э. Дисциплина программирования. – М.: Мир, 1978.
[5] Koltashev A. A., in: Lecture Notes in Computer Science 2789. – Springer-Verlag, 2003.
[6] Кто такой Никлаус Вирт? // http://www.inr.ac.ru/~info21/wirth/wirth.htm
[7] Wirth N. and Gutknecht J. Project Oberon. – Addison-Wesley, 1992.
[8] Свердлов С. В. Языки программирования и методы трансляции. – СПб.: Питер, 2007.
[9] http://www.oberon.ch/blackbox.html
[10] Ильин А. С. и Попков А. И. Компонентный Паскаль в школьном курсе информатики // http://inf.1september.ru/article.php?ID=200800100
[11] http://www.inr.ac.ru/~info21/ADru/
[12] А. Шень. Программирование: теоремы и задачи. – МЦРМО, 1995.

Предисловие

В последние годы признано, что умение создавать программы для вычислительных машин является залогом успеха во многих инженерных проектах и что дисциплина программирования может быть объектом научного анализа и допускает систематическое изложение. Программирование из ремесла превратилось в академическую дисциплину. Первые выдающиеся результаты на этом пути получены Дейкстрой (E. W. Dijkstra) и Хоором (C. A. R. Hoare). «Заметки по структурному программированию» Дейкстры [1] позволили взглянуть на программирование как на объект научного анализа, бросающий вызов человеческому интеллекту, а слова *структурное программирование* дали название «революции» в программировании. Работа Хоора «Аксиоматические основы программирования» [2] продемонстрировала, что программы допускают точный анализ, основанный на математических рассуждениях. И обе статьи убедительно доказывают, что многих ошибок в программах можно избежать, если программисты будут систематически применять методы и приемы, которые ранее применялись лишь интуитивно и часто неосознанно. Эти статьи сосредоточили внимание на построении и анализе программ, или, точнее говоря, на структуре алгоритмов, представленных текстом программы. При этом вполне очевидно, что систематический научный подход к построению программ уместен прежде всего в случае больших, непростых программ, работающих со сложными наборами данных. Отсюда следует, что методология программирования должна включать в себя все аспекты структурирования данных. В конце концов, программы суть конкретные формулировки абстрактных алгоритмов, основанные на конкретных представлениях и структурах данных. Выдающийся вклад в наведение порядка в огромном разнообразии терминологии и понятий, относящихся к структурам данных, сделал Хоор в статье «О структурной организации данных» [3]. В этой работе продемонстрировано, что нельзя принимать решения о структуре данных без учета того, какие алгоритмы применяются к данным, и что, обратно, структура и выбор алгоритмов часто сильно зависят от стуктуры обрабатываемых данных. Короче говоря, задачу построения программ нельзя отделять от задачи структурирования данных.

Но данная книга начинается главой о структурах данных, и для этого есть две причины. Во-первых, интуитивно ощущается, что данные предшествуют алгоритмам: нужно иметь некоторые объекты до того, как можно будет что-то с ними делать. Во-вторых, эта книга предполагает, что читатель знаком с основными понятиями программирования. Однако в соответствии с разумной традицией вводные курсы программирования концентрируют внимание на алгоритмах, работающих с относительно простыми структурами данных. Поэтому уместно посвятить вводную главу структурам данных.

На протяжении всей книги, включая главу 1, мы следуем теории и терминологии, развитой Хоором и реализованной в языке программирования *Паскаль* [4]. Сущность теории – в том, что данные являются прежде всего абстракциями реальных явлений и их предпочтительно формулировать как абстрактные струк-

туры безотносительно к их реализации в распространенных языках программирования. В процессе построения программы представление данных постепенно уточняется – в соответствии с уточнением алгоритма, – чтобы все более и более удовлетворить ограничениям, налагаемым имеющейся системой программирования [5]. Поэтому мы постулируем несколько основных структур данных, называемых фундаментальными. Очень важно, что это конструкции, которые достаточно легко реализовать на реальных компьютерах, ибо только в этом случае их можно рассматривать как истинные элементарные составляющие реального представления данных, появляющиеся как своего рода молекулы на последнем шаге уточнения описания данных. Это запись, массив (с фиксированным размером) и множество. Неудивительно, что эти базовые строительные элементы соответствуют математическим понятиям, которые также являются фундаментальными.

Центральный пункт этой теории структур данных – разграничение *фундаментальных* и *сложных* структур. Первые суть молекулы, – сами построенные из атомов, – из которых строятся вторые. Переменные, принадлежащие одному из таких фундаментальных видов структур, меняют только свое значение, но никогда не меняют ни свое строение, ни множество своих допустимых значений. Как следствие – размер занимаемой ими области памяти фиксирован. «Сложные» структуры, напротив, характеризуются изменением во время выполнения программы как своих значений, так и строения. Поэтому для их реализации нужны более изощренные методы. В этой классификации последовательность оказывается гибридом. Конечно, у нее может меняться длина; но такое изменение структуры тривиально. Поскольку последовательности играют поистине фундаментальную роль практически во всех вычислительных системах, их обсуждение включено в главу 1.

Во второй главе речь идет об алгоритмах сортировки. Там представлено несколько разных методов, решающих одну и ту же задачу. Математическое изучение некоторых из них показывает их преимущества и недостатки, а также подчеркивает важность теоретического анализа при выборе хорошего решения для конкретной задачи. Разделение на методы сортировки массивов и методы сортировки файлов (их часто называют внутренней и внешней сортировками) демонстрирует решающее влияние представления данных на выбор алгоритмов и на их сложность. Теме сортировки уделяется такое внимание потому, что она представляет собой идеальную площадку для иллюстрации очень многих принципов программирования и ситуаций, возникающих в большинстве других приложений. Похоже, что курс программирования можно было бы построить, используя только примеры из темы сортировки.

Другая тема, которую обычно не включают во вводные курсы программирования, но которая играет важную роль во многих алгоритмических решениях, – это рекурсия. Поэтому третья глава посвящена рекурсивным алгоритмам. Здесь показывается, что рекурсия есть обобщение понятия цикла (итерации) и что она является важным и мощным понятием программирования. К сожалению, во многих учебниках программирования она иллюстрируется примерами, для которых было бы достаточно простой итерации. Мы в главе 3, напротив, сосредоточим внимание на нескольких задачах, для которых рекурсия дает наиболее естественную формулировку решения, тогда как использование итерации привело бы к за-

путанным и громоздким программам. Класс алгоритмов с возвратом – отличное применение рекурсии, но самые очевидные кандидаты для применения рекурсии – это алгоритмы, работающие с данными, структура которых определена рекурсивно. Эти случаи рассматриваются в последних двух главах, для которых, таким образом, третья закладывает фундамент.

В главе 4 рассматриваются динамические структуры данных, то есть такие, строение которых меняется во время выполнения программы. Показывается, что рекурсивные структуры данных являются важным подклассом часто используемых динамических структур. Хотя рекурсивные определения возможны и даже естественны в этих случаях, на практике они обычно не используются. Вместо них используют явные ссылочные или указательные переменные. Данная книга тоже следует подобному подходу и отражает современный уровень понимания предмета: глава 4 посвящена программированию с указателями, списками, деревьями и содержит примеры с даже еще более сложно организованными данными. Здесь речь идет о том, что обычно (хотя и не совсем правильно) называют обработкой списков. Немало места уделено построению деревьев и, в частности, деревьям поиска. Глава заканчивается обсуждением так называемых хэш-таблиц, которые часто используют вместо деревьев поиска. Это дает возможность сравнить два принципиально различных подхода к решению часто возникающей задачи.

Программирование – это *конструирование*. Как вообще можно учить изобретательному конструированию? Можно было бы попытаться из анализа многих примеров выделить элементарные композиционные принципы и представить их систематическим образом. Но программирование имеет дело с задачами огромного разнообразия и часто требует серьезных интеллектуальных усилий. Ошибочно думать, что обучить ему можно, просто дав некий список рецептов. Но тогда в нашем арсенале методов обучения остаются только тщательный подбор и изложение образцовых примеров. Естественно, не следует ожидать, что изучение примеров будет равно полезным для разных людей. При таком подходе многое зависит от самого учащегося, от его прилежания и интуиции. Это особенно справедливо для относительно сложных и длинных примеров программ. Такие примеры включены в книгу не случайно. Длинные программы доминируют в практике программирования, и они гораздо больше подходят для демонстрации тех трудно определяемых, но существенных свойств, которые называют стилем и хорошей структурой. Они также должны послужить упражнениями в искусстве чтения программ, которым часто пренебрегают в пользу написания программ. Это главная причина того, почему в качестве примеров используются целиком довольно большие программы. Читатель имеет возможность проследить постепенную эволюцию программы и увидеть ее состояние на разных шагах, так что процесс разработки предстает как пошаговое уточнение деталей. Считаю, что важно показать программу в окончательном виде, уделяя достаточно внимания деталям, так как в программировании дьявол прячется в деталях. Хотя изложение общей идеи алгоритма и его анализ с математической точки зрения могут быть увлекательными для ученого, по отношению к инженеру-практику ограничиться только этим было бы нечестно. Поэтому я строго придерживался правила давать окончательные программы на таком языке, на котором они могут быть реально выполнены на компьютере.

Разумеется, здесь возникает проблема поиска нотации, которая одновременно позволяла бы выполнить программу на вычислительной машине и в то же время была бы достаточно машинно независимой, чтобы ее можно было включать в подобный текст. В этом отношении не удовлетворительны ни широко используемые языки, ни абстрактная нотация. Язык Паскаль представляет собой подходящий компромисс; он был разработан именно для этой цели и поэтому используется на протяжении всей книги. Программы будут понятны программистам, знакомым с другими языками высокого уровня, такими как Алгол 60 или PL/1: смысл нотации Паскаля объясняется в книге по ходу дела. Однако некоторая подготовка все же могла бы быть полезной. Книга «*Систематическое программирование*» [6] идеальна в этом отношении, так как она тоже основана на нотации Паскаля. Однако следует помнить, что настоящая книга не предназначена быть учебником языка Паскаль; для этой цели есть более подходящие руководства [7].

Данная книга суммирует – и при этом развивает – опыт нескольких курсов программирования, прочитанных в Федеральном политехническом институте (ETH) в Цюрихе. Многими идеями и мнениями, представленными в этой книге, я обязан дискуссиям со своими коллегами в ETH. В частности, я хотел бы поблагодарить г-на Г. Сандмайра за внимательное чтение рукописи, а г-жу Хайди Тайлер и мою жену за тщательную и терпеливую перепечатку текста. Я должен также упомянуть о стимулирующем влиянии заседаний рабочих групп 2.1 и 2.3 ИФИПа, и в особенности многих дискуссий, которые мне посчастливилось иметь с Э. Дейкстрой и Ч. Хоором. Наконец, нужно отметить щедрость ETH, обеспечившего условия и предоставившего вычислительные ресурсы, без которых подготовка этого текста была бы невозможной.

Цюрих, август 1975 Н. Вирт

[1] Dijkstra E. W., in: Dahl O-.J., Dijkstra E. W., Hoare C. A. R. Structured Programming. F. Genuys, Ed., New York, Academic Press, 1972. P. 1–82 (имеется перевод: Дейкстра Э. Заметки по структурному программированию, в кн.: Дал У., Дейкстра Э., Хоор К. Структурное программирование. – М.: Мир, 1975. С. 7–97).

[2] Hoare C. A. R. *Comm. ACM*, 12, No. 10 (1969), 576–83.

[3] Hoare C. A. R., in Structured Programming [1]. P. 83-174 (имеется перевод: Хоор К. О структурной организации данных, в кн. [1]. С. 98–197).

[4] Wirth N. The Programming Language Pascal. *Acta Informatica*, 1, No. 1 (1971), 35–63.

[5] Wirth N. Program Development by Stepwise Refinement. *Comm. ACM*, 14, No. 4 (1971), 221–27.

[6] Wirth N. Systematic Programming. Englewood Cliffs, N. J. Prentice-Hall, Inc., 1973 (имеется перевод: Вирт Н. Систематическое программирование. Введение. – М.: Мир, 1977).

[7] Jensen K. and Wirth N. PASCAL-User Manual and Report. Berlin, Heidelberg, New York; Springer-Verlag, 1974 (имеется перевод: Йенсен К., Вирт Н. Паскаль. Руководство для пользователя и описание языка. – М.: Финансы и статистика, 1988).

Предисловие к изданию 1985 года

В этом новом издании сделано много улучшений в деталях, а также несколько более серьезных модификаций. Все они мотивированы опытом, приобретенным за десять лет после первого издания. Однако основное содержание и стиль текста не изменились. Кратко перечислим важнейшие изменения.

Главное изменение, повлиявшее на весь текст, касается языка программирования, использованного для записи алгоритмов. Паскаль был заменен на Модулу-2. Хотя это изменение не оказывает серьезного влияния на представление алгоритмов, выбор оправдан большей простотой и элегантностью синтаксиса Модулы-2, что часто приводит к большей ясности представления структуры алгоритма. Кроме того, было сочтено полезным использовать нотацию, которая приобретает популярность в довольно широком сообществе по той причине, что она хорошо подходит для разработки больших программных систем. Тем не менее тот очевидный факт, что Паскаль является предшественником Модулы, облегчает переход. Для удобства читателя синтаксис Модулы суммирован в приложении.

Как прямое следствие замены языка программирования был переписан раздел 1.11 о последовательной файловой структуре. В Модуле-2 нет встроенного файлового типа. В пересмотренном разделе 1.11 понятие последовательности как структуры данных представлено в более общем виде, и там также вводится набор программных модулей, которые явно реализуют идею последовательности конкретно в Модуле-2.

Последняя часть главы 1 является новой. Она посвящена теме поиска и, начиная с линейного и двоичного поиска, подводит к некоторым недавно изобретенным быстрым алгоритмам поиска строк. В этом разделе подчеркивается важность проверок промежуточных состояний (assertions) и инвариантов цикла для доказательства корректности представляемых алгоритмов.

Новый раздел о приоритетных деревьях поиска завершает главу, посвященную динамическим структурам данных. Эта разновидность деревьев была неизвестна во время выхода первого издания. Такие деревья допускают экономное представление и позволяют выполнять быстрый поиск по множествам точек на плоскости.

Целиком исключена вся пятая глава первого издания. Это сделано потому, что тема построения компиляторов стоит несколько в стороне от остальных глав и заслуживает более подробного обсуждения в отдельной книге.

Наконец, появление нового издания отражает прогресс, глубоко повлиявший на издательское дело в последние десять лет: применение компьютеров и изощренных алгоритмов для подготовки и автоматического форматирования документов. Эта книга была набрана и сформатирована автором с помощью компьютера Lilith и редактора документов Lara. Без этих инструментов книга не только стала бы дороже, но, несомненно, даже еще не была бы закончена.

Пало Альто, март 1985 г. Н. Вирт

Нотация

В книге используются следующие обозначения, взятые из работ Дейкстры.

В логических выражениях литера & обозначает конъюнкцию и читается как «и». Литера ~ обозначает отрицание и читается как «не». Комбинация литер or обозначает дизъюнкцию и читается как «или». Литеры **A** и **E**, набранные жирным шрифтом, обозначают кванторы общности и существования. Нижеследующие формулы определяют смысл нотации в левой части через выражение в правой. Интерпретация символа «...» в правых частях оставлена интуиции читателя.

Ai: $m \le i < n : P_i$ $\quad P_m \ \&\ P_{m+1} \ \&\ \ldots \ \&\ P_{n-1}$

Здесь P_i – некоторые предикаты, а формула утверждает, что выполняются все P_i для значений индекса i из диапазона от m до n, но не включая само n.

Ei: $m \le i < n : P_i$ $\quad P_m \text{ or } P_{m+1} \text{ or } \ldots \text{ or } P_{n-1}$

Здесь P_i – некоторые предикаты, а формула утверждает, что выполняются некоторые из P_i для каких-то значений индекса i из диапазона от m до n, но не включая само n.

Si: $m \le i < n : x_i = x_m + x_{m+1} + \ldots + x_{n-1}$

MIN i: $m \le i < n : x_i =$ минимальное среди значений $(x_m, \ldots, x_{n-1})$

MAX i: $m \le i < n : x_i =$ максимальное среди значений $(x_m, \ldots, x_{n-1})$

Глава 1

Фундаментальные структуры данных

1.1. Введение

Современные цифровые компьютеры были изобретены для выполнения сложных и длинных вычислений. Однако в большинстве приложений предоставляемая таким устройством возможность хранить и обеспечивать доступ к большим массивам информации играет основную роль и рассматривается как его главная характеристика, а возможность призводить вычисления, то есть выполнять арифметические действия, во многих случаях стала почти несущественной.

В таких приложениях большой массив обрабатываемой информации является в определенном смысле абстрактным представлением некоторой части реального мира. Информация, доступная компьютеру, представляет собой специально подобранный набор данных, относящихся к решаемой задаче, причем предполагается, что этот набор достаточен для получения нужных результатов. Данные являются абстрактным представлением реальности в том смысле, что некоторые свойства реальных объектов игнорируются, так как они несущественны для этой задачи. Поэтому абстракция – это еще и упрощение реальности.

В качестве примера можно взять файл с данными о служащих некоторой компании. Каждый служащий (абстрактно) представлен в этом файле набором данных, который нужен либо для руководства компании, либо для бухгалтерских расчетов. Такой набор может содержать некоторую идентификацию служащего, например имя и зарплату. Но в нем почти наверняка не будет несущественной информации о цвете волос, весе или росте.

Решая задачу с использованием компьютера или без него, необходимо выбрать абстрактное представление реальности, то есть определить набор данных, который будет представлять реальную ситуацию. Этот выбор можно сделать, руководствуясь решаемой задачей. Затем нужно определиться с представлением информации. Здесь выбор определяется средствами вычислительной установки. В большинстве случаев эти два шага не могут быть полностью разделены.

Выбор представления данных часто довольно сложен и не полностью определяется имеющимися вычислительными средствами. Делать такой выбор всегда нужно с учетом операций, которые нужно выполнять с данными. Хороший пример – представление чисел, которые сами суть абстракции свойств некоторых объектов. Если единственное (или основное) действие, которое нужно выполнять, – сложение, то хорошим представлением числа n может быть n черточек. Правило сложения при таком представлении – очевидное и очень простое. Римская нотация основана на этом принципе простоты, и правила сложения просты для маленьких чисел. С другой стороны, представление арабскими цифрами требует неочевидных правил сложения (для маленьких чисел), и их нужно запоминать. Однако ситуация меняется на противоположную, если нужно складывать большие числа или выполнять умножение и деление. Разбиение этих операций на более простые шаги гораздо проще в случае арабской нотации благодаря ее систематической позиционной структуре.

Хорошо известно, что компьютеры используют внутреннее представление, основанное на двоичных цифрах (битах). Это представление непригодно для использования людьми, так как здесь обычно приходится иметь дело с большим

числом цифр, но весьма удобно для электронных схем, так как два значения 0 и 1 можно легко и надежно представить посредством наличия или отсутствия электрических токов, зарядов или магнитных полей.

Из этого примера также видно, что вопрос представления часто требует рассматривать несколько уровней детализации. Например, в задаче представления положения объекта первое решение может касаться выбора пары чисел в, скажем, декартовых или полярных координатах. Второе решение может привести к представлению с плавающей точкой, где каждое вещественное число x состоит из пары целых, обозначающих дробную часть f и показатель e по некоторому основанию (например, $x = f \times 2^e$). Третье решение, основанное на знании, что данные будут храниться в компьютере, может привести к двоичному позиционному представлению целых чисел. Наконец, последнее решение может состоять в том, чтобы представлять двоичные цифры электрическими зарядами в полупроводниковом устройстве памяти. Очевидно, первое решение в этой цепочке зависит главным образом от решаемой задачи, а дальнейшие все больше зависят от используемого инструмента и применяемых в нем технологий. Вряд ли можно требовать, чтобы программист решал, какое представление чисел использовать или даже какими должны быть характеристики устройства хранения данных. Такие решения низкого уровня можно оставить проектировщикам вычислительного оборудования, у которых заведомо больше информации о существующих технологиях, чтобы сделать разумный выбор, приемлемый для всех (или почти всех) приложений, где играют роль числа.

В таком контексте выявляется важность языков программирования. Язык программирования представляет абстрактный компьютер, допускающий интерпретацию в терминах данного языка, что может подразумевать определенный уровень абстракции по сравнению с объектами, используемыми в реальном вычислительном устройстве. Тогда программист, использующий такой язык высокого уровня, будет освобожден от заботы о представлении чисел (и лишен возможности что-то сделать в этом отношении), если числа являются элементарными объектами в данном языке.

Использование языка, предоставляющего удобный набор базовых абстракций, общих для большинства задач обработки данных, влияет главным образом на надежность получающихся программ. Легче спроектировать программу, опираясь в рассуждениях на знакомые понятия чисел, множеств, последовательностей и циклов, чем иметь дело с битами, единицами хранения и переходами управления. Конечно, реальный компьютер представляет любые данные – числа, множества или последовательности – как огромную массу битов. Но программист может забыть об этом, если ему не нужно беспокоиться о деталях представления выбранных абстракций и если он может считать, что выбор представления, сделанный компьютером (или компилятором), разумен для решаемых задач.

Чем ближе абстракции к конкретному компьютеру, тем легче сделать выбор представления инженеру или автору компилятора и тем выше вероятность, что единственный выбор будет подходить для всех (или почти всех) мыслимых приложений. Это обстоятельство устанавливает определенные пределы на «высоту»

используемых абстракций по сравнению с уровнем реального «железа». Например, неразумно включать в язык общего назначения геометрические фигуры, так как из-за внутренне присущей им сложности их подходящее представление будет сильно зависеть от действий, выполняемых с ними. Однако природа и частота таких действий неизвестна проектировщику языка программирования общего назначения и соответствующего компилятора, и любой выбор проектировщика может оказаться плохим для некоторого класса приложений.

Эти соображения определили выбор нотации для описания алгоритмов и соответствующих данных в настоящей книге. Разумеется, нам хотелось бы использовать знакомые понятия математики, такие как числа, множества, последовательности и т. д., а не машинно зависимые сущности вроде строк битов. Но нам также хотелось бы использовать нотацию, для которой существуют эффективные компиляторы. Неразумно использовать язык, в сильной степени машинно зависимый, но также недостаточно и описывать программы в абстрактной нотации, в которой проблемы представления остаются нерешенными. Язык программирования Паскаль был спроектирован в попытке найти компромисс между этими двумя крайностями, а его наследники Модула-2 и Оберон учитывают опыт, накопленный за десятилетия [1.3]. Оберон сохраняет базовые понятия Паскаля с некоторыми усовершенствованиями и добавлениями; он используется на протяжении этой книги [1.5]. Оберон был успешно реализован для ряда компьютеров, при этом было продемонстрировано, что его нотация достаточно близка к реальному «железу», чтобы выбранные средства и их представления можно было объяснить с полной ясностью. Язык также близок к другим языкам, так что уроки, усвоенные здесь, могут быть с равным успехом применены и при их использовании.

1.2. Понятие типа данных

В математике переменные обычно классифицируются по некоторым важным характеристикам. Проводится четкое различие между вещественными, комплексными и логическими переменными, или между переменными, представляющими отдельные значения, множества значений, множества множеств, или между функциями, функционалами, множествами функций и т. д. Такая классификация не менее, если не более, важна в обработке данных. Мы будем придерживаться того принципа, что каждая константа, выражение или функция имеет определенный *тип*. В сущности, тип характеризует множество значений, к которому принадлежит константа, или которые может принимать переменная или выражение, или которые могут порождаться функцией.

В математических текстах тип переменной обычно можно определить просто по шрифту, без учета контекста; но это невозможно в компьютерных программах. На вычислительной установке обычно доступен только один шрифт (латинские буквы). Поэтому часто следуют правилу явно вводить соответствующий тип в *объявлении* константы, переменной или функции, причем такое объявление должно предшествовать использованию этой константы, переменной или функции. Это правило тем более разумно, что компилятор должен выбрать пред-

ставление объекта в памяти компьютера. Очевидно, что объем памяти, отведенной под переменную, должен быть выбран в соответствии с диапазоном значений, которые может принимать переменная. Если эта информация доступна компилятору, то можно избежать так называемого динамического размещения. Очень часто этот пункт оказывается ключевым для эффективной реализации алгоритма.

Сущность понятия типа, как оно используется в данном тексте и реализуется в языке программирования Оберон, выражается в следующих утверждениях [1.2]:

1. Тип данных определяет множество значений, которому принадлежит значение константы, или в котором принимает значения переменная или выражение, или которому принадлежат значения, порождаемые операцией или функцией.
2. Тип значения, обозначенного константой, переменной или выражением, может быть выведен из их объявлений и вида выражения без выполнения вычислений.
3. Каждая операция или функция требует аргументов определенных типов и дает результат некоторого, тоже определенного типа. Если операция допускает аргументы нескольких типов (например, + используется для сложения как целых, так и вещественных чисел), то тип результата может быть определен на основе особых правил языка программирования.

Компилятор может использовать такую информацию о типах для проверки законности различных конструкций. Например, ошибочное присваивание булевского (логического) значения арифметической переменной может быть обнаружено без выполнения программы. Подобная избыточность текста программы весьма полезна при ее разработке и может рассматриваться как главное преимущество хороших языков высокого уровня по сравнению с машинным кодом (или кодом символического ассемблера).

Очевидно, в конечном итоге данные будут представлены огромным количеством двоичных цифр независимо от того, была ли написана исходная программа на языке высокого уровня, использующего понятие типа, или на ассемблере, где типов нет. Для компьютера память представляется однородной массой битов без явной структуры. Но именно абстрактная структура позволяет человеку-программисту видеть смысл в монотонном пейзаже компьютерной памяти.

Теория, о которой идет речь в данной книге, и язык программирования Оберон дают некоторые способы определения типов данных. В большинстве случаев новый тип данных строится из других типов, уже определенных (назовем их *составляющими*). Значения такого типа – это обычно агрегаты значений-компонент, принадлежащих ранее определенным составляющим типам, и такие значения называются *составными*, или *структурированными*. Если используется только один составляющий тип, то есть все компоненты принадлежат одному типу, то этот тип называют *базовым*. Число различных значений типа T называют его *мощностью*. Мощность позволяет определить объем памяти для представления переменной x, имеющей тип T, что обозначается как x: T.

Поскольку составляющие типы, в свою очередь, могут быть составными, то могут выстраиваться целые иерархии структур. Впрочем, очевидно, что наимень-