Предисловие

Ученик плотника довольствуется лишь молотком и пилой, однако мастер использует более замысловатые инструменты. Также и компьютерное программирование, чтобы удовлетворить все возрастающим требованиям современных прикладных задач, прибегает к разнообразным утонченным средствам, и только практическая работа с ними позволяет программисту приобрести необходимую сноровку в их эффективном использовании. В этой книге в качестве фундаментальных средств разработки программ рассматриваются такие вопросы, как структурное решение задач, абстракция данных, принципы программной инженерии и сравнительный анализ алгоритмов. Для того чтобы показать, как все эти средства совместно используются для построения законченных программ, в книге детально разбирается несколько примеров значительной сложности.

Многие изучаемые нами алгоритмы и структуры данных обладают внутренней элегантностью, проявляющейся в том, что внешняя простота скрывает богатый диапазон и мощь их применений. Очень быстро обучающийся обнаруживает, что наивные методы, обычно предлагаемые во вводных курсах, нуждаются в очень серьезных улучшениях. Однако элегантность метода сочетается с неопределенностью. Вы вскоре сталкиваетесь с тем, что выбор наилучшего из нескольких возможных подходов к решению конкретной задачи далеко не очевиден. Вот здесь-то и становится оправданным рассмотрение ряда непростых проблем, интересных самих по себе и имеющих большое практическое значение, а также показ применимости математических методов к верификации и анализу алгоритмов.

Студенты, приступая к изучению программирования, часто затрудняются в практическом приложении абстрактных идей. Поэтому в настоящей книге особое внимание уделяется вопросам преобразования идей в алгоритмы, а также дальнейшего уточнения этих алгоритмов с целью получения конкретных программ, которые уже можно использовать для решения практических задач. Также на первое место перед выбором структур данных и их реализации выступает процесс спецификации и абстракции данных.

Лично я являюсь сторонником движения от конкретного к абстрактному, последовательной разработки мотивирующих примеров, дающих возможность представить идеи в более общей форме. Большинству студентов на ранних этапах их обучения требуется помощь в освоении непосредственного приложения идей, с которыми они знакомятся, и лучше

всего, если они будут писать и выполнять программы, иллюстрирующие изучаемые ими концепции. С этой целью книга содержит большое количество программных примеров, как коротких процедур, так и законченных программ значительной длины. Более того, упражнения и программные проекты составляют неотъемлемую часть книги. Многие из этих упражнений иллюстрируют изучаемую в настоящий момент тему, и их выполнение требует написания и отладки реальных программ с целью анализа и сравнения используемых алгоритмов. Другие представляют собой более объемные программные проекты, а некоторые предназначены для использования небольшой группой работающих совместно студентов.

Краткий обзор

Принципы программирования

Посредством рассмотрения первого большого проекта (игры «Жизнь» Дж. Конвея) глава 1 вводит принципы нисходящей детализации, проектирования программы, критического обзора и тестирования; демонстрация этих принципов позволит студентам использовать их при изучении последующих разделов. В то же время первый проект позволит студентам освежить свои знания языка Pascal, который выбран в качестве базового языка программирования для этой книги.

Введение в программную инженерию

В главе 2 вводятся несколько базовых принципов программной инженерии, включая проблемы спецификации и анализа, прототипирования, абстракции данных, а также разработки, детализации, верификации и анализа алгоритма. Эти принципы используются при разработке второго варианта игры «Жизнь», основанного на алгоритме, достаточно тонком для наглядной демонстрации необходимости в точной спецификации и верификации, и, кроме того, позволяющем показать важность правильного выбора структур данных.

Стеки и рекурсии

Глава 3 продолжает разъяснение концепций абстракции данных и разработки алгоритмов посредством изучения стеков как абстрактного типа данных и рекурсии как метода решения задач, а также внутренних связей между стеками, рекурсией и определенными типами деревьев. В главе 4 эти концепции иллюстрируются рассмотрением нескольких важных приложений рекурсии, включая алгоритмы с отходом, программы с древовидной структурой и рекурсивно-нисходящий синтаксический анализ.

Примеры рекурсий

Очереди

Списки

Центральной темой двух следующих глав являются очереди и списки. Здесь рассматриваются различные реализации каждого абстрактного типа данных, строятся большие прикладные программы, демонстрирующие относительные преимущества различных реализаций, и весьма неформальным образом вводятся идеи анализа алгоритмов. Основная цель этих глав — подвести студента к пониманию важности абстракции данных и к приложению методов нисходящего проектирования как к данным, так и к алгоритмам.

Поиск и сортировка

В главах 7, 8 и 9 рассматриваются алгоритмы поиска, сортировки и работы с таблицами (включая хеширование). Эти главы демонстрируют

взаимосвязь между алгоритмами и соответствующим им абстрактными типами данных, структурами данных и реализациями. Здесь вводится понятие «*O* большого» для элементарного анализа алгоритмов и подчеркивается критическая важность выбора, позволяющего наиболее эффективным образом использовать пространство памяти, время и программистские усилия.

Таблицы и извлечение информации

Этот выбор требует применения аналитических методов оценки алгоритмов, причем арсенал для проведения такого анализа должна нам предоставить комбинаторная математика. На начальных стадиях обучения мы не можем требовать от студентов ни достаточно глубоких знаний, ни математической зрелости, необходимых для оттачивания до совершенства их навыков. Моя цель, таким образом, — помочь студентам осознать важность приобретения этих навыков при изучении ими в дальнейшем соответствующих разделов математики.

Безусловно, к наиболее элегантным и полезным структурам данных следует отнести двоичные деревья. Их изучение, которому посвящена глава 10, связывает вместе концепции списков, поиска и сортировки. Относясь к рекурсивно определяемым структурам данных, двоичные деревья предоставляют студентам превосходную возможность изучить рекурсии применительно как к структурам данных, так и к алгоритмам. Глава начинается с элементарных сведений и постепенно подводит читателя к скошенным деревьям и амортизационному анализу алгоритмов.

Двоичные деревья

В главе 11 продолжается изучение более сложных структур данных, включая трай-структуры, B-деревья и красно-черные деревья. В следующей главе вводятся графы, как наиболее общие структуры, используемые при решении задач.

Многовариантные деревья

Графы

Конкретное исследование в главе 13 позволяет весьма детально рассмотреть польскую нотацию и исследовать взаимосвязь рекурсии, деревьев и стеков в качестве средств решения задач и разработки алгоритмов. Некоторые аспекты этого исследования могут служить неформальным введением в проектирование компиляторов. Как и в других местах книги, представленные алгоритмы разработаны до деталей и включены в работоспособную Pascal-программу. Эта программа принимает входные данные в виде обычного (инфиксного) выражения, преобразует это выражение в постфиксную форму и оценивает его применительно к заданным значениям переменных.

Конкретное исследование: польская нотация

Приложения посвящены темам, строго говоря, не относящимся к предмету книги, но часто отсутствующим в начальных курсах.

В приложении A дан обзор ряда вопросов дискретной математики. Последние два раздела, посвященные числам Фибоначчи и Каталана, являются более продвинутыми и необязательны для понимания основного текста книги, однако они включены с целью развития интереса к комбинаторике у студентов с математическим складом ума.

Математические методы

В приложении B обсуждаются псевдослучайные числа, программы генерации таких чисел и их приложения. Этот предмет может заинтересовать многих студентов, хотя часто он выпадает из типичных программ обучения.

Случайные числа

Модули, включаемые файлы и процедуры-утилиты

Приложение C детально описывает использование модулей системы программирования Turbo Pascal и включаемых (include-) файлов при реализации абстрактных типов данных. В основном тексте книги в ряде мест используются специально разработанные процедуры-утилиты и модули. В приложении C даны развернутые описания этих программных средств.

Средства языка Pascal

Наконец, приложение D посвящено некоторым средствам программирования на языке Pascal, обычно не входящим в начальные курсы обучения этому языку, именно, записям, процедурным параметрам и упреждающим объявлениям. Далее, в приложение D включены сведения о типах указателей языка Pascal и элементарных операциях над указателями и связными списками, поскольку эти средства являются неотъемлемой частью языка Pascal, а также с целью придания разделу книги, посвященному связным спискам, большей общности. В завершающей части приложения D приведены стандартные диаграммы и таблицы, описывающие синтаксис языка Pascal, а также дополнительная информация, которая может помочь читателю в решении программистских проблем.

Изменения в третьем издании

При подготовке книги к третьему изданию весь текст был тщательно просмотрен в плане улучшения представления материала, а также с целью отражения предложений многочисленных читателей, изучавших эту книгу. Принципиальные изменения заключаются в следующем.

- Все программы были переработаны и улучшены с целью подчеркивания абстракции данных, получения повторно используемого кода и обеспечения однородности и элегантности стиля.
- Для облегчения этого процесса в книге широко используются модули Turbo Pascal, хотя и в ненавязчивой форме, чтобы обеспечить беспрепятственное преобразование в стандартный Pascal или, при необходимости, в другой диалект языка.
- Расширена документация к программам путем включения во все подпрограммы неформальных спецификаций (пред- и постусловий).
- Рекурсия стала использоваться в книге значительно раньше, а ее повторное постоянное использование подчеркивает важность этого средства.
- В книге расширено представление современных продвинутых тем включением разделов, посвященных скошенным и красно-черным деревьям и амортизационному анализу алгоритмов.
- Включены новые примеры конкретных исследований, в частности, синтаксический анализатор предложений языка Pascal и миниатюрный текстовый редактор.
- Добавлено много новых упражнений и программных проектов, включая последовательные варианты проектов, посвященных извлечению информации, которые требуют от студента сравнительного анализа производительности различных структур данных и алгоритмов.
- Материал по теории графов и алгоритмам реализации графов выделен в отдельную главу.

- Рассмотрение списков стало более логичным, во-первых, за счет упрощения спецификаций списков, и, во-вторых, перенесением элементарного обзора типов указателей языка Pascal в приложение, где этот материал изложен более подробно, чем раньше. Такое расположение материала позволило в основном тексте книги сконцентрировать внимание на основополагающих идеях, не отвлекаясь на детали реализации.
- Изъяты некоторые устаревшие темы, например, удаление рекурсии.
- Поставлявшийся ранее с книгой программный диск теперь можно приобрести у издателя; весь программный пакет также доступен в Интернете. Пакет включает в себя исходные тексты всех программ и программных фрагментов, вошедших в книгу, вместе с выполнимыми файлами (для компьютеров IBM PC и совместимых с ними), всех демонстрационных программ и почти всех программных проектов этой книги.
- Для преподавателей имеется также отдельное (бесплатное) пособие, включающее в себя полные решения всех упражнений и программных проектов, диск с упомянутым выше программным пакетом, а также второй диск с полными исходными текстами всех программных проектов.

Структура курса

требуемый исходный уровень подготовки

Для работы над этой книгой студент должен овладеть начальным курсом программирования и иметь опыт использования элементарных средств языка Pascal. В приложении D собраны некоторые продвинутые средства языка Pascal, обычно не включаемые в вводные курсы. Для анализа почти всех алгоритмов достаточно хорошего знания школьной математики, хотя знакомство с дискретной математикой (например, параллельно с чтением книги) будет весьма полезным. В приложении A дан обзор всех необходимых математических понятий.

содержание

Книга предназначена в качестве учебного пособия по таким курсам, как ACM Course CS2 (*Program Design and Implementation*, Проектирование и реализация программ), ACM Course CS7 (*Data Structures and Algorithm Analysis*, Структуры данных и анализ алгоритмов), а также курсам, являющимся комбинацией упомянутых выше. В книге дано полное освещение большинства модулей знаний ACM/IEEE[1], касающихся структур данных и алгоритмов. Сюда входят модули:

AL1 Базовые структуры данных: массивы, таблицы, стеки, очереди, деревья и графы;
AL2 Абстрактные типы данных;
AL3 Рекурсии и рекурсивные алгоритмы;
AL4 Анализ сложности с использованием нотации «*O* большого»;
AL6 Сортировка и поиск;
AL8 Практические стратегии решения задач с примерами серьезных конкретных исследований.

[1] См. *Computing Curricula 1991: Report of the ACM/IEEE-CS Curriculum Task Force*, ACM Press, New York, 1990.

Три наиболее продвинутых модуля знаний, AL5 (complexity classes and NP-complete problems, классы сложности и НП-полные задачи), AL7 (computability and undecidability, вычислимость и неразрешимость) и AL9 (parallel and distributed algorithms, параллельные и распределенные алгоритмы) не освещаются в этой книге.

Большая часть глав этой книги построена таким образом, что сначала представляются основные темы главы, сопровождаемые примерами, приложениями и практическими исследованиями. Таким образом, при нехватке времени на серьезное изучение можно без потери связности быстро переходить от главы к главе, знакомясь лишь с наиболее существенными понятиями. В дальнейшем, когда время позволит, и студент, и преподаватель смогут обращаться вразбивку к дополнительным темам и комплексным примерам.

двухсеместровый курс

Двухсеместровый курс практически покрывает всю книгу, удачно интегрируя в себе многие темы таких разделов, как решение задач, структуры данных, разработка программ и анализ алгоритмов. Для понимания общих методов студентам требуется время и практика. Комбинируя изучение абстракции данных, структур данных и алгоритмов с их реализациями в проектах реального размера, такой интегрированный курс будет служить твердой базой, на которой в дальнейшем могут основываться курсы, имеющие более теоретический характер.

Даже при не вполне исчерпывающем изучении, эта книга даст достаточно основательные знания, которые позволят интересующимся студентам по ходу своей дальнейшей работы использовать ее в качестве справочного пособия. В любом случае необходимо поручить студентам разработку основных программных проектов и предоставить им достаточное время для доведения их до рабочего состояния.

Разработка книги

Эта книга вместе с дополнительным материалом была написана с помощью собственного программного обеспечения автора под названием PreTEX, объединяющего в себе препроцессор и макропакет для полиграфической системы TEX[2]. Пакет PreTEX, используя контекстную зависимость, автоматически обеспечивает значительную часть разметки текста, требуемую программой TEX. PreTEX также предоставляет некоторые полезные для автора средства, в частности, мощную систему перекрестных ссылок, существенное упрощение набора математических выражений и листингов компьютерных программ, и автоматическую генерацию предметного указателя и содержания. В то же время PreTEX позволяет поэтапно обрабатывать текст книги, содержащийся в относительно небольших по размеру файлах. Решения, размещенные вместе с упражнениями и проектами, автоматически убираются из текста и помещаются в отдельные документы. В сочетании с языком PostScript описания страниц, PreTEX предоставляет удобные средства для цветоделения, обработки растровых изображений и других специальных приемов.

[2] Система TEX была разработана Доналдом Кнутом (Donald E. Knuth), который внес весьма значительный вклад в наше сегодняшнее понимание структур данных и алгоритмов. (См. ссылки на его имя в предметном указателе).

Для книг, подобных этой, наиболее важной чертой PreTEX является возможность обработки им компьютерных программ. Программы не включаются в основной текст книги; они размещаются в отдельных вторичных файлах вместе с любыми поясняющими текстами и разметочной информацией. Помещая теги в соответствующие места вторичных файлов, PreTEX может извлекать в любом порядке требуемые фрагменты вторичного файла для вывода их на печать вместе с основным текстом. С помощью еще одной утилиты (с именем StripTEX) можно обработать вторичный файл, удалив из него все теги, текст и разметку, получив в результате программу, годную для компиляции. Таким образом, один и тот же исходный файл автоматически предоставляет и листинг программы для печати, и компилируемый программный код. В этом случае автор приобретает уверенность в безусловной правильности листинга компьютерной программы, включаемой в текст книги.

Для настоящего издания книги все диаграммы и рисунки были подготовлены в виде кода PostScript с помощью программы Adobe Illustrator. Такая процедура позволяет автоматически включать все иллюстрации в предварительные варианты рукописи, что сокращает конечные этапы подготовки книги к печати, устраняя необходимость ручной обработки оригинал-макета.

Благодарности

В течение ряда лет эта книга существенно улучшалась стараниями многих людей: моей семьи, друзей, коллег по работе и студентов. В первом и втором издании были перечислены некоторые из них, чей вклад в создание книги был особенно важен. После выхода в свет второго издания были опубликованы две производные книги, *Programming with Data Structures* и *Data Structures and Program Design in C*, а также переводы этих книг на ряд языков. Читатели, которых слишком много, чтобы их можно было перечислить, присылали мне свои замечания и предложения. Я выражаю всем им свою искреннюю благодарность и рад сообщить, что многие из полученных предложений нашли свое отражение в настоящем издании книги.

Пол Мейлхот (Paul Mailhot), сначала мой студент, а теперь — постоянный ассистент, внес значительный вклад в настоящее издание, преданно работая вместе со мной на всех этапах создания книги. Он написал все программы, включенные в книгу, а также в сопровождающее ее пособие *Instructor's Resource Manual*, улучшив их стиль и элегантность и обеспечив возможность повторного использования. Он также внимательно вычитал весь текст, внеся много ценных предложений. На нем лежала вся ответственность за выпуск как самой книги, так и приложений к ней; готовя книгу к печати, он в полной мере использовал возможности системы PreTEX, а с помощью Adobe Illustrator он довел все иллюстрации до совершенного состояния. Работа с таким надежным помощником доставляет истинное удовольствие.

Роберт Л. Круз

Глава 1
Принципы программирования

Эта глава представляет собой обзор важных принципов правильного программирования, в особенности, применительно к проектам значительного объема, и иллюстрирует методы разработки эффективных алгоритмов. По ходу изложения материала мы поставим ряд вопросов, касающихся программного проектирования, ответы на которые будут получены в последующих главах. Мы также коснемся многих важных средств языка Pascal посредством использования их при написании программ.

1.1. Введение

Основная трудность при разработке больших компьютерных программ заключается не в определении цели разрабатываемой программы и даже не в поиске методов, позволяющих достичь поставленной цели. Президент какой-нибудь фирмы мог бы сказать: «Давайте возьмем компьютер, и пусть он учитывает все наши товары, счета и списки сотрудников, и пусть он будет оповещать нас о необходимости произвести переучет товаров, и о перерасходе средств, и пусть он еще выписывает ведомости на зарплату, и все будет хорошо!» При достаточных затратах времени и усилий группа системных аналитиков и программистов могла бы определить, как различные сотрудники фирмы выполняют всю эту работу, и разработать программы, делающие то же самое.

проблемы больших программ

Такой подход, однако, почти неминуемо приведет к разрушительным последствиям. Беседуя со служащими, системные аналитики обнаружат, что некоторые задачи можно запрограммировать без особого труда; они напишут соответствующие программы и поставят их на компьютер. Продолжая действовать таким же образом и устанавливая на компьютер следующую порцию программ, они увидят, что новые программы опираются на старые. Однако выходные данные первой порции программ не вполне соответствуют требованиям второй порции. Эту проблему можно преодолеть, написав дополнительные программы преобразования данных, полученных от первых программ, в форму, требуемую вторыми. Программный проект начинает напоминать лоскутное одеяло. Некоторые его участки достаточно прочны, другие слабее. Одни кусочки аккуратно пришиты к соседним, другие едва приметаны. Если программистам повезет, их детище сможет в течение некоторого времени достаточно хорошо выполнять текущую работу. Если, однако, в проект потребуется внести хоть какое-то изменение, это приведет к непредсказуемым последствиям для работоспо-

собности всей системы. Потом потребуется новое изменение или возникнет неожиданная проблема, возможно даже критическая ситуация, требующая срочного решения, и весь этот программный проект окажется столь же эффективен, как лоскутное одеяло, используемое для спасения людей, прыгающих из горящего здания.

цель книги

Главной целью этой книги является описание программных методов и средств, позволяющих создавать проекты реалистичного размера, программы, имеющие значительно больший объем, чем те, которые используются для демонстрации элементарных средств программирования. Поскольку «кусочный» метод поиска решения больших задач оказывается неприемлемым, мы должны прежде всего разработать последовательный, единообразный и логичный подход, в котором необходимо соблюсти важные принципы разработки программ, принципы, часто нарушаемые при написании небольших программ, но несоблюдение которых для больших проектов окажется фатальным.

спецификация задачи

Первое серьезное препятствие при решении сложной задачи заключается в необходимости точной формулировки, в чем именно эта задача заключается. Необходимо преобразовать неясные цели, противоречивые требования и, возможно, туманные пожелания в четко сформулированный проект, который можно реализовать с помощью программ. При этом методы разделения работы, используемые людьми, не обязательно будут эффективны для компьютера. Таким образом, прежде всего следует точно описать конечные цели, а затем постепенно дробить работу на меньшие по размеру задачи до тех пор, пока они не окажутся обозримого размера.

разработка программы

Принцип, исповедуемый многими программистами, «сначала заставь свою программу работать, а после этого можешь ее совершенствовать», часто оказывается эффективным для простых программ, однако он неприменим к программам большого размера. Каждый фрагмент большой программы должен быть хорошо организован, ясно написан и понят до мельчайших подробностей, в противном случае вы рискуете через какое-то время забыть структурные особенности этого фрагмента и в результате не сможете объединить его с другими частями проекта, что часто выполняется значительно позже и, возможно, другими программистами. Поэтому стиль программирования надо рассматривать как один из важных составных элементов проектирования программы, и с самого начала работы воспитывать в себе аккуратность и тщательность.

Даже при реализации очень крупных проектов трудности могут возникнуть не из-за неспособности найти решение, а, наоборот, из-за того, что решить поставленную задачу можно самыми разными методами с применением различных алгоритмов, и трудно бывает оценить, какой из них является наилучшим, а какой, возможно, приведет к программистским проблемам или окажется безнадежно неэффективным. При этом наибольший вклад в разнообразие возможных алгоритмов вносит способ организации данных, используемых программой:

структура данных

- как данные организованы по отношению друг к другу;
- какие данные хранятся в памяти;

[. . .]

программных средств, поэтому его легко освоить, однако в то же время он обладает мощными возможностями обработки данных, которые облегчают переход от общих алгоритмов к конкретным программам их реализации.

Несколько разделов этой и последующих глав неформально вводят некоторые средства языка Pascal по мере того, как они будут встречаться в составляемых нами программах. Строгое описание синтаксиса (грамматики) языка Pascal можно найти в приложении D или в учебниках по программированию на этом языке.

1.2. Игра «Жизнь»

Возьму на себя смелость перефразировать старую пословицу:

Одна конкретная задача стоит тысячи нереализованных абстракций.

конкретное исследование

В этой и последующей главах мы сконцентрируемся на конкретном исследовании, которое, будучи не очень крупным по критериям реальной жизни, позволит тем не менее проиллюстрировать и методы проектирования программ, и подводные камни, которых нам следует научиться избегать. Иногда пример подвигнет нас на формулировку общих принципов; в других случаях мы начнем, наоборот, с дискуссии на общие темы; однако всегда конечной целью будет обнаружение общих методов, ценность которых будет проявляться в определенном диапазоне практических применений. В дальнейших главах мы применим тот же подход при рассмотрении более крупных проектов. Здесь же мы воспользуемся в качестве примера игрой, получившей название «Жизнь», которая была предложена английским математиком Дж. Х. Конвеем (J. H. Conway) в 1970 г.

1.2.1. Правила игры «Жизнь»

описания

Программа представляет собой модель жизни некоторого сообщества, а не игру нескольких игроков. Действия разворачиваются в неограниченном пространстве решетки, или сетки, в которой каждая ячейка может быть либо занята организмом, либо нет. Занятые ячейки будут называться ***живыми***; незанятые — ***мертвыми***. Расположение и количество живых ячеек изменяется от поколения к поколению в соответствии с количеством соседних живых ячеек следующим образом:

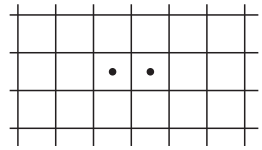
правила переходов

1. Соседями данной ячейки являются восемь ячеек, соприкасающихся с ней по горизонтали, вертикали или диагонали.
2. Если ячейка является живой, но либо имеет лишь одну живую ячейку-соседа, либо не имеет живых соседей вообще, в следующем поколении она умирает от одиночества.
3. Если ячейка является живой и имеет четырех или более живых соседей, в следующем поколении она умирает от перенаселенности.
4. Живая ячейка, имеющая двух или трех живых соседей, остается живой в следующем поколении.

5. Если ячейка мертва, то в следующем поколении она станет живой, если она имеет точно трех (не больше и не меньше) живых соседей. Все остальные мертвые ячейки остаются мертвыми в следующем поколении.
6. Все рождения и смерти имеют место точно в одни и те же моменты, поэтому умирающая ячейка может помочь родиться другой, но не может предотвратить смерть других ячеек из-за уменьшения перенаселенности; рождающаяся ячейка не может предотвратить смерть или, наоборот, убить ячейки, живые в предыдущем поколении.

1.2.2. Примеры

В качестве первого примера рассмотрим сообщество



Число живых соседей для конкретных ячеек выглядит следующим образом:

0	0	0	0	0	0
0	1	2	2	1	0
0	1	•1	•1	1	0
0	1	2	2	1	0
0	0	0	0	0	0

пример гибели

В соответствии с правилом 2 обе живые ячейки умрут в следующем поколении, а по правилу 5 ни одна ячейка не оживет; таким образом, в данном примере сообщество вымирает.

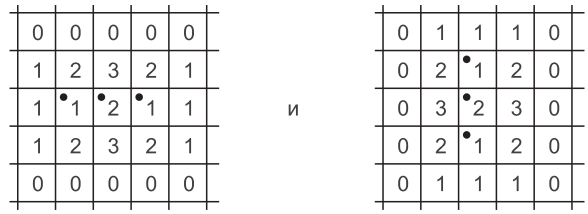
С другой стороны, сообщество

0	0	0	0	0	0
0	1	2	2	1	0
0	2	•3	•3	2	0
0	2	•3	•3	2	0
0	1	2	2	1	0
0	0	0	0	0	0

стабильность

имеет, как это показано на рисунке, другое распределение числа живых соседей. Каждая живая ячейка имеет трех живых соседей и в силу этого остается живой, но все мертвые ячейки имеют в качестве соседей две или меньше живых ячеек, в силу чего остаются мертвыми.

Два сообщества



изменения

будут изменяться от поколения к поколению, в чем можно убедиться, изучив распределение живых соседей для отдельных ячеек.

Любопытно, что из очень простых начальных конфигураций могут развиваться весьма сложные живые сообщества, существующие на протяжении многих поколений, причем обычно очень не просто предугадать, какие изменения возникнут в следующем поколении. Некоторые очень маленькие начальные конфигурации развиваются в большие сообщества; другие медленно вымирают; многие достигают стадий, когда они перестают изменяться или проходят через повторяющиеся конфигурации каждые несколько поколений.

многообразие

популярность

Вскоре после изобретения этой модели Мартин Гарднер (Martin Gardner) начал обсуждать игру «Жизнь» на своей колонке в журнале *Scientific American*, и с тех пор она очаровывала многих людей, отчего в течение нескольких лет даже существовал ежеквартальный информационный бюллетень, посвященный этой игре. Она идеально подходит для демонстрации на домашних компьютерах.

Разумеется, нашей первой задачей будет написание программы, которая покажет, как исходное сообщество будет изменяться от поколения к поколению.

1.2.3. Решение

Несколько минут размышлений покажут, что решение задачи «Жизнь» настолько простое, что оно вполне может служить упражнением для членов кружка начинающих программистов, которые только что освоили понятие массива. Все, что нам нужно — это объявить большой прямоугольный массив[1], члены которого соответствуют ячейкам «Жизни», и которые должны обладать состоянием, т. е. помечаться либо как живые, либо как мертвые. Тогда для определения того, что произойдет в следующем поколении, мы должны просто посчитать число живых ячеек в окрестностях каждой ячейки и применить описанные выше правила. Поскольку, однако, для продвижения по прямоугольному массиву мы будем использовать циклы, нам важно не нарушить правило 6, т. е. не позволить произведенным уже изменениям повлиять на распределение числа живых соседей для ячеек, обрабатываемых позже. Простейший способ избежать этой ловушки заключается в объявлении второго прямоуголь-

метод

[1] Массив с двумя индексами называется **прямоугольным**. Первый индекс определяет **строку** массива, а второй — его **столбец**.

ного массива, который будет отображать сообщество в следующем поколении и после полного завершения вычисления этого поколения будет копироваться в исходный массив.

Теперь давайте изобразим на бумаге этот метод в виде неформального алгоритма.

алгоритм

Инициализируем прямоугольный массив, который мы назовем map, и который будет содержать начальную конфигурацию живых ячеек.
Будем повторять следующие шаги произвольное число раз:
Для каждой ячейки в прямоугольном массиве выполним следующее:
Подсчитаем число живых соседей данной ячейки.
Если число живых соседей равно 0, 1, 4, 5, 6, 7 или 8, то установим соответствующую ячейку в другом прямоугольном массиве, названном newmap, в живое состояние; если число живых соседей равно 2, установим соответствующую ячейку в newmap в то же состояние, что и в прямоугольном массиве map (поскольку состояние ячейки с числом живых соседей 2 не изменяется).
Скопируем прямоугольный массив newmap в прямоугольный массив map.
Выведем прямоугольный массив map на экран пользователя.

1.2.4. Life: главная программа

Приведенная выше схема алгоритма для игры «Жизнь» приводит к такой Pascal-программе, которую мы будем называть Life (жизнь), а в этом, первом, варианте — Life1.

главная программа

```
program Life1 (input,output);                    { Игра "Жизнь", вариант 1 }
{ Pre:  Пользователь должен предоставить начальную конфигурацию
        живых ячеек.
  Post: Программа выводит последовательность решеток,
        отображающих изменения в конфигурации живых ячеек
        согласно правилам игры "Жизнь".
  Uses: Использует функции UserSaysYes, NeighborCount и процедуры
        WriteMap, Initialize }
uses Utility;                    { модуль, содержащий процедуры-утилиты,
                                                специфичные для Turbo Pascal }
const                  { maxrow и maxcol определяются размерами экрана }
  maxrow = 20;                    { максимально допустимое число строк }
  maxcol = 60;                  { максимально допустимое число столбцов }
type
  rowrange = 0 .. (maxrow + 1);
  colrange  = 0 .. (maxcol + 1);
       { Эти диапазоны увеличены на 1, чтобы представить место
         для границы решетки. Необходимость этого действия будет
         пояснена в разделе 1.4.2. }
  state = (dead, alive);                                  { состояние ячейки }
  grid = array [rowrange, colrange] of state;        { прямоугольный массив }
var
  map,                                  { конфигурация текущего поколения }
  newmap: grid;                      { конфигурация следующего поколения }
```

[. . .]

- функция UserSaysYes будет запрашивать пользователя, следует ли выполнить переход к следующему поколению;
- процедура NeighborCount(map, row, col) будет определять число населенных ячеек, соседних с ячейкой, находящейся в клетке row,col прямоугольного массива map (под населенными ячейками понимаются ячейки с живыми обитателями).

функционирование программы

Функционирование программы Life осуществляется очевидным образом. Прежде всего мы считываем исходную ситуацию, чтобы установить первую конфигурацию занятых ячеек. Затем мы начинаем цикл, выполняющий один проход для каждого поколения. Внутри этого цикла мы сначала выполняем вложенную пару циклов вдоль строк row и столбцов col, которые просмотрят все ячейки в прямоугольном массиве map. Тело этих вложенных циклов состоит из предложения множественного выбора **case ... end**. В настоящем варианте функция NeighborCount(map, row, col) вернет одно из значений 0, 1, ..., 8, и для каждого из этих случаев (**case**) мы можем предусмотреть отдельное действие или, как в нашей программе, некоторые случаи могут приводить к одному и тому же действию. Вы должны удостовериться в том, что действие, предписанное для каждого случая, точно соответствует правилам 2, 3, 4 и 5 раздела 1.2.1. Наконец, после выполнения вложенных циклов и предложения **case**, которые заполняют значениями прямоугольный массив newmap, предложение присваивания map := newmap копирует его в прямоугольный массив map, и процедура WriteMap(map) выводит результат.

Упражнения 1.2

Определите путем вычислений вручную, что будет происходить с сообществами, изображенными на рис. 1.1, на протяжении пяти поколений. [Подсказка: расставьте конфигурацию программы Life на шашечной доске. Используйте для живых ячеек шашки одного цвета, а шашками второго цвета обозначайте ячейки, которые в следующем поколении родятся или умрут.]

1.3. Стиль программирования

Перед тем, как начать писать подпрограммы для игры Life, рассмотрим несколько принципов, которые надо будет обязательно использовать в программировании.

1.3.1. Имена

В легенде о создании мира (Книга Бытия 2:19) Всевышний привел всех животных к Адаму, чтобы тот дал им имена. Согласно древнему иудейскому верованию, животные ожили только после того, как Адам назвал их. Эта легенда позволяет сформулировать важный принцип компьютерного программирования: даже если данные или алгоритмы уже существуют, только после того, им будут даны значащие имена, можно будет определить их расположение в программе, и только после этого они начнут жить самостоятельной жизнью.

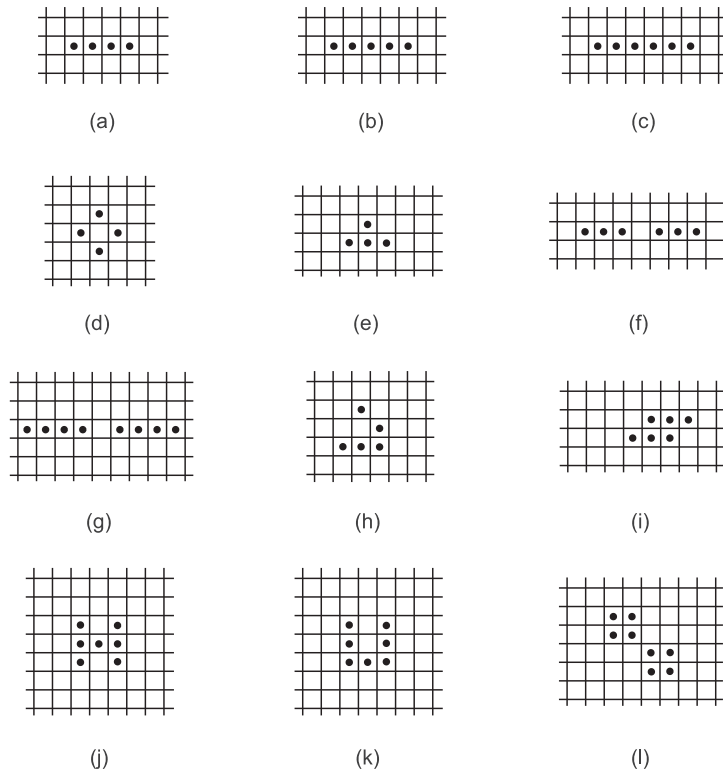


Рис. 1.1. Простые конфигурации для программы Life

цель наглядных наименований

Чтобы программа функционировала правильно, чрезвычайно важно точно знать, что представляет собой каждая переменная, и отчетливо представлять, что делает каждая подпрограмма. Поэтому в программу всегда необходимо включать документацию с описанием назначения переменных и подпрограмм. Имена переменных и подпрограмм следует выбирать со всей тщательностью, чтобы они ясно и точно отражали свое назначение. Найти разумные имена не всегда просто, однако это достаточно важная задача, чтобы выделить ее в качестве второго программистского принципа:

> ***Программистский принцип***
> *Всегда именуйте свои переменные и подпрограммы с максимальным тщанием и детально объясняйте их назначение*

Язык Pascal даже навязывает нам выполнение этого принципа, требуя, чтобы в программе содержалась секция с объявлениями переменных. При этом Pascal, в отличие от большинства других языков программирования, допускает более интенсивное использование имен. Константам, используемым в различных местах программы, следует давать имена, и то же относится к типам данных, чтобы компьютер мог обнаружить ошибки, которые в противном случае найти было бы затруднительно.

[. . .]

1.3.2. Документация и форматы

назначение документации

Многие студенты полагают, что документацией следует заниматься лишь после завершения работы над программой, и то лишь для того, чтобы преподаватель мог ознакомиться с программой, и чтобы удачные находки студента не остались бы незамеченными. Разумеется, автор небольшой программы может держать все детали в уме, и документация ему нужна лишь для того, чтобы объяснить роль программы кому-либо другому. Однако в случае больших программ (да и малых по истечение нескольких месяцев) становится невозможным запомнить, как каждая деталь программы соотносится с другой деталью, и поэтому при написании больших программ существенно включать в каждый фрагмент программы соответствующую документацию. Полезной привычкой является составление документации по мере разработки программы, и еще лучшей, как это будет показано позже — подготовка части документации еще до того, как программист приступит к написанию текста программы.

Не любые пояснения годятся в качестве документации. Почти так же часто, как программы с малым объемом документации или с пометками, понятными только для их автора, встречаются программы с многословными пояснениями, которые в действительности мало дают для понимания программы. Отсюда следует третий программистский принцип:

> ***Программистский принцип***
> *Старайтесь, чтобы ваша документация была краткой, но содержательной*

Стиль документации, как и вообще стиль любого письменного документа, зависит от личности автора, и самые разные стили могут оказаться вполне удовлетворительными. Тем не менее, имеются общепринятые рекомендации, которых желательно придерживаться:

рекомендации

1. В начало каждой подпрограммы помещайте пролог, который включает в себя
   (a) Идентификационные данные (имя программиста, дату, номер версии)[2].
   (b) Назначение программы и используемый в ней метод.
   (c) Результаты работы подпрограммы и используемые ею данные.
   (d) Ссылки на другие виды документации, внешние по отношению к программе.
2. При объявлении каждой переменной, константы или типа объясняйте, что собой представляют эти объекты и как они будут использоваться. Еще лучше, если эти сведения можно извлечь из имени объекта.
3. Перед каждой существенной секцией (блоком или подпрограммой) программы включайте краткий комментарий со сведениями о назначении или действии данной секции.
4. Отмечайте конец каждой существенной секции, если это не очевидно.

[2] Ради экономии места программы, приведенные в этой книге, не содержат идентификационных строк или других частей пролога, поскольку необходимую информацию можно найти в окружающем тексте.

[. . .]

Вполне возможно, что существуют другие ситуации, которые так и не были протестированы даже после многих пробных прогонов. Для любой программы значительного размера невозможно выполнить полностью исчерпывающие тесты, хотя тщательный выбор тестовых данных существенно повышает вашу уверенность в ее правильной работе. Например, все, конечно, уверены, что типичный компьютер может правильно сложить два числа с плавающей точкой, однако эта уверенность, разумеется, не основана на исчерпывающем тестировании компьютера, когда его заставляют складывать все возможные пары чисел с плавающей точкой с проверкой каждого результата. Если число с плавающей точкой двойной точности занимает 64 бит, то всего могут существовать 2^{128} различных пар складываемых чисел. Это число астрономически велико: все компьютеры, выпущенные до настоящего времени, выполнили лишь ничтожную долю такого количества операций сложения. Наша уверенность в том, что компьютер выполняет сложение правильно, основана на тестировании отдельных компонентов схемы суммирования, т. е. проверки правильности сложения каждого из 64 разрядов и правильности учета переноса в следующий разряд.

методы тестирования

Существуют по меньшей мере три общих принципа, используемых при выборе тестовых данных.

1. Метод черного ящика

Большинству пользователей большой программы не интересны детали ее функционирования; им нужно только получить результат. Другими словами, они желают рассматривать программу как черный ящик, откуда и произошло название метода отладки. Точно так же при проверке правильности работы программы тестовые данные должны выбираться исходя из спецификаций решаемой задачи, безотносительно к внутренним деталям программы. Тестовые данных должны выбираться, по меньшей мере, следующим образом:

выбор данных

1. ***Простые значения***. Программу следует отлаживать с помощью легко проверяемых данных. Не один студент, пытавшийся проверить свою программу только со сложными, запутанными данными, испытывал смущение, когда преподаватель предъявлял ему результат тривиального примера.
2. ***Типичные, реалистичные значения***. Всегда проверяйте работу программы с типичными для ее применения данными. При этом данные должны быть достаточно просты, чтобы результаты можно было проверить вручную.
3. ***Крайние значения***. Многие программы имеют тенденцию ошибаться на границах своей применимости. Например, превышение счетчиком предельного значения или выход за пределы массива может иметь драматические последствия.
4. ***Недопустимые значения***. «Мусор на входе, мусор на выходе» — это известное в компьютерных кругах изречение не следует воспринимать как руководство к действию. Если на вход правильно написанной программы поступает мусор, на выходе программы по меньшей мере

должно появиться разумное сообщение об ошибке. Разумеется, программа должна иметь средства обнаружения типичных ошибок ввода и выполнять только те вычисления, которые имеют смысл после отбрасывания ошибочно введенных входных данных.

2. Метод стеклянного ящика

проверка путей выполнения

Второй подход к выбору тестовых данных основывается на очевидном утверждении, что программу вряд ли можно считать оттестированной, если в ней остаются участки кода, которые никогда не выполнялись. В методе ***стеклянного ящика*** анализируется логическая структура программы, и для каждой возможной альтернативы ее поведения выбираются тестовые данные, которые порождают эту альтернативу. Так, например, тестовые данные должны обеспечить обработку всех вариантов в каждом предложении **case**, всех предложений в каждой конструкции **if** и условия завершения каждого цикла. Если в программе имеется несколько предложений выбора или итераций, для проверки всех возможных путей выполнения потребуются различные комбинации тестовых данных. На рис. 1.3 показан короткий программный сегмент с возможными путями выполнения.

Для больших программ метод стеклянного ящика, очевидно, не применим, но для отдельных коротких модулей он является превосходным

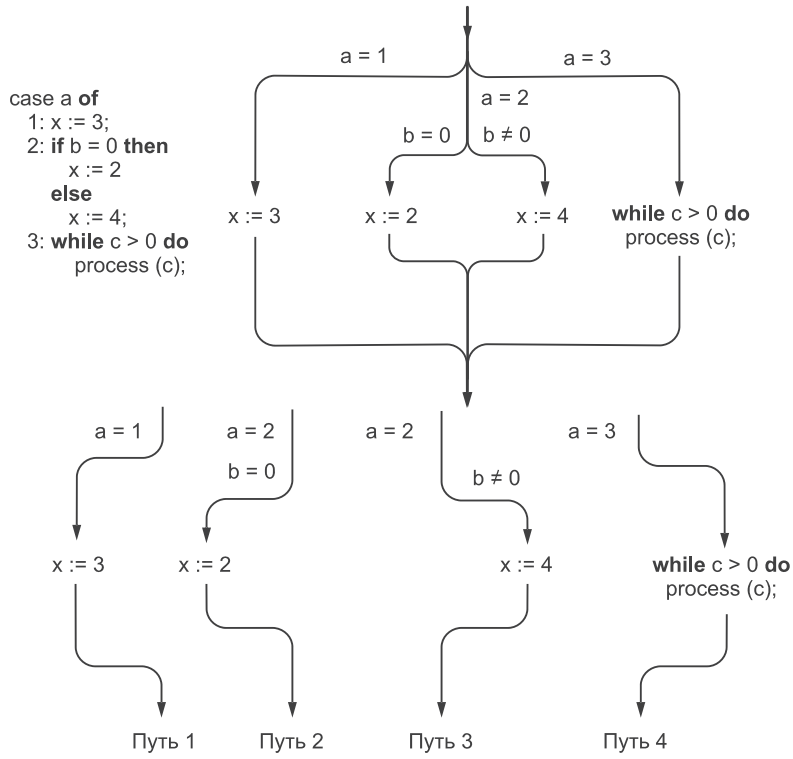


Рис. 1.3. Программный сегмент и возможные пути его выполнения

средством отладки и тестирования. В хорошо разработанной программе в каждом модуле будет лишь небольшое количество циклов и альтернативных путей выполнения. В результате нескольких тщательно выбранных комбинаций тестовых данных будет достаточно для тестирования каждого модуля по отдельности.

модульное тестирование

При тестировании методом стеклянного ящика преимущества модульной структуры программы становятся очевидными. Рассмотрим типичный пример проекта, включающего 50 подпрограмм, каждая из которых реализует 5 различных случаев или альтернатив. Если бы мы тестировали такую программу как единое целое, то для проверки всех возможных альтернатив потребовалось бы 5^{50} наборов тестовых данных. Каждый же модуль в отдельности требует только 5 тестовых наборов, а всего таких наборов потребуется $5 \times 50 = 250$. В результате проблема немыслимого размера была сведена к процедуре тестирования вполне умеренной, для большой программы, сложности.

сравнение

Если вы уверились в том, что метод стеклянного ящика всегда является предпочтительным, мы должны заметить, что на практике тестирование методом черного ящика обычно оказывается более эффективным в плане обнаружения ошибок. Возможно, одна из причин этого связана с тем, что большая часть тонких программных ошибок возникает не внутри подпрограмм, а на их границах, в интерфейсах между подпрограммами, в силу неправильного или неполного понимания условий и правил обмена информацией между подпрограммами. Поэтому при отладке крупных проектов наиболее разумной стратегией тестирования будет приложение метода стеклянного ящика к каждому небольшому модулю сразу после его написания, и использование метода черного ящика с соответствующими тестовыми данными для тестирования более крупных секций программы по мере их завершения.

ошибки в интерфейсе

3. Метод тикающего ящика

В заключение этого раздела упомянем еще одну стратегию тестирования программ, стратегию, которая, к сожалению, распространена очень широко. Эту стратегию можно было бы назвать методом ***тикающего ящика***. Она состоит в том, что после достаточно тщательной отладки проекта он совсем не тестируется, а вместо этого передается потребителю для пробного использования и приемки. Результатом, разумеется, будет бомба замедленного действия.

Упражнения 1.4

E1. Если вы подозреваете, что программа Life содержит ошибки, где в тексте главной программу лучше всего разместить вставки? Какую информацию следует вывести на экран?

E2. Обратитесь к своему решению упражнения E7 из раздела 1.3 (разработка программы для вывода точечного графика) и укажите наилучшие места включения вставок, если они потребуются.

E3. Найдите наилучшие наборы тестовых данных при тестировании методом черного ящика для каждого из следующих случаев:

(a) Функция возвращает наибольший из своих трех параметров, являющихся действительными числами.

(b) Функция возвращает квадратный корень из действительного числа.

(c) Функция возвращает наименьшее общее кратное своих двух параметров, которые должны быть целыми положительными числами (***Наименьшее общее кратное*** есть наименьшее целое число, которое кратно обоим параметрам. Например, наименьшее общее кратное для 4 и 6 есть 12, для 3 и 9 — 9, а для 5 и 7 — 35.)

(d) Процедура упорядочивает три целых числа, задаваемые в качестве ее параметров, в восходящем порядке.

(e) Процедура упорядочивает массив A целых чисел, индексируемых от 1 до переменной n, в восходящем порядке; значения A и n являются параметрами процедуры.

E4. Найдите подходящий набор тестовых данных при тестировании методом стеклянного ящика следующих фрагментов:

(a) Предложения

```
if a < b then if c > d then x := 1 else if c = d then x := 2
else x := 3 else if a = b then x := 4 else if c = d then x:=5
else x := 6;
```

(b) Функции NeighborCount(map, row, col).

Программные проекты 1.4

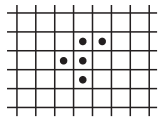
P1. Реализуйте программу Life из этой главы на своем компьютере и убедитесь, что она работает правильно.

P2. Протестируйте программу Life с примерами, показанными на рис. 1.1.

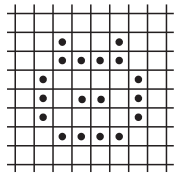
P3. Выполните прогоны программы Life с исходными конфигурациями, приведенными на рис. 1.4. Некоторые из этих конфигураций проходят через большое число поколений до достижения стабильной конфигурации или конфигурации с предсказуемым поведением.

Подсказки и ловушки

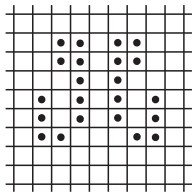
1. Приступая к решению задачи, удостоверьтесь в том, что вы ее понимаете.
2. Приступая к написанию программы, удостоверьтесь в том, что вы понимаете, какой алгоритм надо в ней использовать.
3. При возникновении затруднений разделите программу на секции и проанализируйте функционирование каждой секции в отдельности.
4. Старайтесь, чтобы ваши подпрограммы были простыми и короткими; одиночная подпрограмма редко занимает более одной страницы текста.
5. При написании подпрограмм включайте в них тщательно продуманную документацию, как это описано в разделе 1.3.2.
6. Следите за тем, чтобы для каждой подпрограммы точно описывались пред- и постусловия.



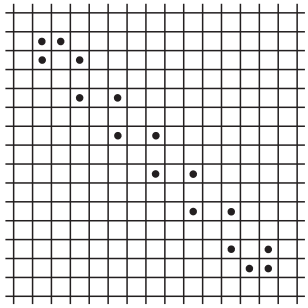
Пентамино для символа r



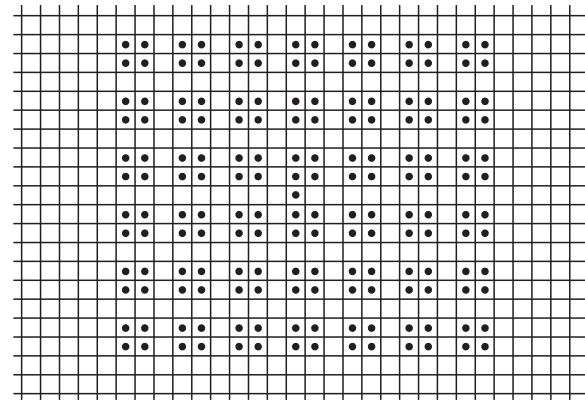
Чеширский кот



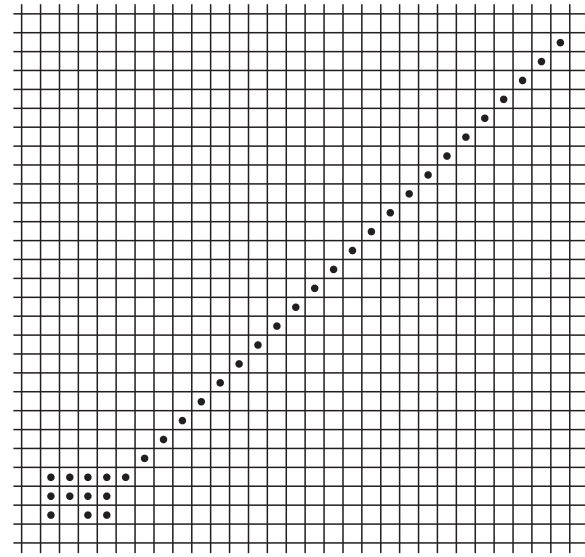
Бокал



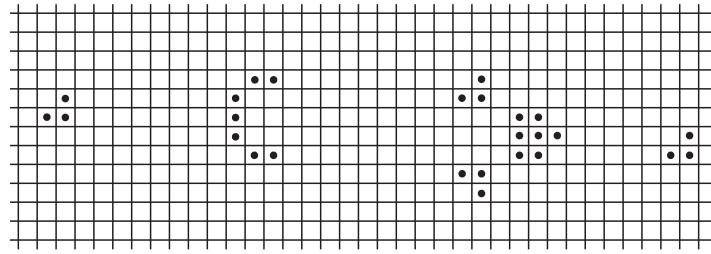
Столбик-символ парикмахерской у ее вход



Вирус



Уборка урожая



Катапульта для планера

Рис. 1.4. Конфигурации для программы Life

7. В начало каждой подпрограммы включайте проверку на фактическое выполнение указанного в ней предусловия.
8. При каждом использовании подпрограммы спрашивайте себя, почему вы уверены в выполнении ее предусловий.
9. Для упрощения отладки используйте заглушки и драйверы, а также тестирование методами черного ящика и стеклянного ящика.
10. Для локализации ошибок повсеместно используйте вставки.
11. Программируя массивы, опасайтесь выхода индексов за пределы массивов хотя бы на 1. Всегда при отладке программ, использующих массивы, прибегайте к проверке на граничные значения.
12. По мере написания программы придерживайтесь аккуратного форматирования ее исходного текста; это серьезно облегчит дальнейшую отладку.
13. Следите за тем, чтобы ваша документация соответствовала вашему коду, и, уточняя текст программы, вносите изменения в код, а не только в комментарии.
14. Постарайтесь объяснить вашу программу кому-нибудь еще. При этом вы сами поймете ее лучше.
15. Помните программистские принципы!

Обзорные вопросы

Большинство глав этой книги завершаются набором вопросов, подобранных так, чтобы помочь вам вспомнить основные идеи данной главы. Ответы на все эти вопросы фактически содержатся в тексте книги; если вы затрудняетесь в ответе, прочитайте заново соответствующий раздел.

1. В каких случаях разумно использовать однобуквенные имена переменных?
2. Назовите четыре вида информации, которую следует включать в документацию к программе.
3. В чем состоит различие между *внешней* и *внутренней* документацией?
4. Что представляют собой пред- и постусловия?
5. Назовите три вида параметров. В чем состоят особенности их обработки в языке Pascal?
6. Почему следует избегать появления в программе побочных эффектов?
7. Что представляет собой программная заглушка?
8. В чем заключается разница между заглушками и драйверами, и когда следует использовать те и другие?
9. Что представляет собой структурированный разбор программы?
10. Что такое вставка, и когда надо включать вставки в программу?
11. Опишите способ реализации безопасного программирования.
12. Назовите два метода тестирования программы и опишите, когда следует использовать каждый из них.

13. Как вы подойдете к решению задачи, если вы не можете сразу осознать все детали этой задачи?

Литература для дальнейшего изучения

Pascal

Язык программирования Pascal был разработан Никлаусом Виртом (Niklaus Wirth), который впервые опубликовал его описание в 1971 г. Для старых версий языка стандартным руководством была книга

K. Jensen and N. Wirth, *PASCAL User Manual and Report*, second edition, Springer-Verlag, Berlin, Heidelberg, New York, 1974, 167 pages.

Имеется русский перевод: Йенсен К., Вирт Н. Паскаль: Руководство для пользователя. — М.: Компьютер, 1993. — 255 с.

В дальнейшем Международная организация по стандартизации (International Standards Organization, ISO) разработала спецификацию стандартной версии языка Pascal, которой придерживается большинство современных компиляторов. Этот стандартный Pascal кратко, но точно описан в следующей книге, к которой вы можете обращаться при возникновении у вас тонких языковых проблем:

Doug Cooper, *Standard Pascal User Reference Manual*, W. W. Norton, New York, 1983, 176 pages.

Многие хорошие учебники посвящены более развернутому описанию языка Pascal, причем их слишком много, чтобы их можно было здесь перечислить. В этих учебниках можно также найти массу примеров и приложений. Однако некоторые книги, предназначенные для начинающих, не содержат описания важных «продвинутых» средств языка Pascal, которые будут использоваться в этой книге. Поэтому проследите за тем, чтобы выбранный вами учебник описывал полный синтаксис стандартного языка Pascal.

Программистские принципы

Приведем три книги, содержащие много полезных советов, касающихся стиля программирования и правильного составления программ, а также примеры хорошего и плохого программирования:

Brian Kernighan and P. J. Plauger, *The Elements of Programming Style*, second edition, McGraw-Hill, New York, 1978, 168 pages.

Henry F. Ledgard, Paul A. Nagin, and John F. Hueras, *Pascal with Style: Programming Proverbs*, Hayden Book Company, Hasbrouk Heights, N.J., 1979, 210 pages.

Dennie Van Tassel, *Program Style, Design, Efficiency, Debugging, and Testing*, second edition, Prentice-Hall, Englewood Cliffs, N.J., 1978, 323 pages.

Эдсгер У. Дейкстра (Edsger W. Dijkstra) был родоначальником движения, известного под названием структурного программирования, которое требует применения к проектированию и написанию программ тща-

тельно разработанного нисходящего подхода. Эти принципы были им сформулированы в опубликованном им в марте 1968 г. письме под названием «Go To Statement Considered Harmful» («О вреде предложения Goto») в журнале *Communications of the ACM* (том 11, стр. 147–148). С тех пор Дейкстра опубликовал несколько статей и книг, весьма полезных с точки зрения изучения правильного метода программирования. Одной из наиболее интересных является

Edsger W. Dijkstra, A Discipline of Programming, Prentice-Hall, Englewood Cliffs, N.J., 1976, 217 pages.

Имеется русский перевод: Э. Дейкстра. Дисциплина программирования. — М.: Мир, 1978.

Игра «Жизнь»

Выдающийся английский математик Дж. Х. Конвей (J. H. Conway) внес значительный вклад в столь различающиеся области математики, как теория конечных простых групп, логика и комбинаторика. Он придумал игру «Жизнь», начав с изучения клеточных автоматов и разработав правила воспроизводства, которые затрудняют безграничное разрастание популяции, но при которых многие конфигурации проходят через интересные последовательности. Конвей, однако, не опубликовал свои наблюдения, а только сообщил о них Мартину Гарднеру (Martin Gardner). Популярность игры стремительно возросла, когда появилось ее обсуждение в работе

Martin Gardner, «Mathematical Games» (regular column), *Scientific American* 223, no 4 (October 1970), 120–123; 224, no 2 (February 1971), 112–117.

Примеры в конце разделов 1.2 и 1.4 взяты из этой работы, которая была перепечатана со включением дальнейших результатов в

Martin Gardner, *Wheels, Life and Other Mathematical Amusements*, W. H. Freeman, New York, 1983, pp 214–257.

Имеется русский перевод: Гарднер М., Математические досуги. — 2-е изд., испр. и доп. — М.: Мир, 2000.

В настоящей книге приведена библиография статей, посвященных игре «Жизнь». В течение нескольких лет даже выпускался ежеквартальный бюллетень под название *Lifeline*, который публиковал для приверженцев этой игры последние разработки и статьи на смежные темы.