Высшее профессиональное образование
БАКАЛАВРИАТ

Н. И. ПАРФИЛОВА, А. Н. ПЫЛЬКИН, Б. Г. ТРУСОВ

ПРОГРАММИРОВАНИЕ

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

УЧЕБНИК

Под редакцией доктора технических наук, профессора Б. Г. Трусова

Рекомендовано Федеральным государственным бюджетным образовательным учреждением высшего профессионального образования «Московский государственный технический университет имени Н. Э. Баумана» в качестве учебника для студентов высших учебных заведений, обучающихся по направлению подготовки «Информатика и вычислительнвая техника»

ACADEMIA

Москва
Издательский центр «Академия»
2012

УДК 621.391(075.8)
ББК 32.973-018я73
П189

Рецензент —
профессор Московского государственного технического университета
имени Н. Э. Баумана, доктор техн. наук *К. А. Майков*

Парфилова Н.И.

П189 Программирование. Основы алгоритмизации и программирования : учебник для студ. учреждений высш. проф. образования / Н. И. Парфилова, А. Н. Пылькин, Б. Г. Трусов ; под ред. Б. Г. Трусова. — М. : Издательский центр «Академия», 2012. — 240 с. — (Сер. Бакалавриат).

ISBN 978-5-7695-9149-5

Учебник создан в соответствии с Федеральным государственным образовательным стандартом по направлению подготовки 230100 «Информатика и вычислительная техника» (квалификация «бакалавр»).

Рассмотрены вопросы технологии разработки алгоритмов и программ, основы современных подходов к программированию, в том числе структурное и объектно-ориентированное программирование, методы отладки программ, методы решения различных предметных задач и особенности реализации основных алгоритмов. Изложены основы алгоритмизации и структурного программирования, базовые управляющие конструкции языка, технология нисходящего проектирования алгоритмов и программ различной структуры.

Для студентов учреждений высшего профессионального образования.

УДК 621.391(075.8)
ББК 32.973-018я73

ISBN 978-5-7695-9149-5

ПРЕДИСЛОВИЕ

Образовательный стандарт высшего профессионального образования по направлению подготовки 230100 «Информатика и вычислительная техника» предусматривает, что в результате изучения дисциплины «Программирование» обучаемый должен:

• **знать:** технологию разработки алгоритмов и программ, основы современных подходов к программированию, в том числе структурного и объектно-ориентированного, методы отладки программ и методы решения различных предметных задач с применением средств вычислительной техники, основные стандарты в области программной документации;

• **уметь:** выполнять формализованную постановку задачи и разрабатывать алгоритм ее решения, используя современные системы программирования, разрабатывать сопровождающие программные документы;

• **владеть:** практическими навыками разработки и отладки программ на одном из языков программирования высокого уровня.

Во *введении* рассмотрен процесс решения на ЭВМ предметных задач, даны характеристики различных его этапов.

В *главе 1* изложены основы алгоритмизации и структурного подхода к разработке алгоритмов и программ, позволяющего сформировать у обучаемого «правильный» стиль программирования и определенные практические навыки. Рассмотрены основные понятия, средства записи алгоритмов, технология нисходящего проектирования на примерах алгоритмов различной структуры.

В *главе 2* дан краткий обзор существующих языков программирования в рамках технологий структурного и объектно-ориентированного программирования. Введено понятие среды (системы) программирования, рассмотрена среда программирования Delphi языка Object Pascal и основы работы в этой среде, процесс разработки, компиляции и выполнения программ.

Глава 3 — введение в программирование, содержит описание основ императивного стиля программирования, простых типов данных и основных средств языка, позволяющих реализовать простейшие линейные алгоритмы.

Глава 4 содержит описание управляющих конструкций и реализующих их операторов для программирования алгоритмов разветвляющейся структуры; рассмотрены логические основы формирования условий разной степени сложности.

Глава 5 посвящена изучению циклических структур, принципов программирования циклов с известным и неизвестным числом повторений, итерационных циклов и сложных циклов вложенной структуры. Изложены основы и приведены примеры проектирования программ по технологии нисходящего проектирования.

Глава 6 посвящена вопросам организации ввода и вывода данных различного типа при работе со стандартными устройствами. Рассмотрены стандартная форма вывода и управление выводом при оформлении результатов работы на примерах построения таблиц, рисунков, графиков функций.

Глава 7 позволяет получить знания и практические навыки отладки программ, рассмотрены причины и типы ошибок, возникающих при их проектировании и разработке, способы и средства отладки в интегрированной среде.

Учебник носит практическую направленность. Большое внимание уделяется практике разработки алгоритмов и программ с привлечением большого числа примеров. В конце каждой главы приведены вопросы для самоконтроля и задания для самостоятельной работы.

Авторы выражают глубокую благодарность рецензентам, сделавшим ряд ценных замечаний при подготовке данного учебника, а также преподавателям и студентам, помогавшим в его написании, и всем авторам, материалы которых были использованы, принося извинения за невозможность сослаться в тексте на все случаи такого использования.

ВВЕДЕНИЕ

С общей точки зрения (разработки, отладки и выполнения программы) процесс решения любой предметной задачи с использованием средств вычислительной техники включает в себя несколько взаимосвязанных этапов:

1) разработку технического задания (постановку задачи на содержательном уровне);
2) формализацию задачи (построение математической модели);
3) выбор (или разработку) метода решения задачи;
4) разработку алгоритма (алгоритмизацию);
5) выбор языка программирования;
6) разработку программы (программирование);
7) отладку и тестирование программы;
8) оптимизацию программы;
9) документирование программы;
10) вычисления и обработки результатов.

Последовательное выполнение перечисленных этапов составляет полный цикл разработки, отладки и выполнения программы. Рассмотрим более подробно наиболее важные из них.

Постановка задачи. Постановка задачи представляет собой *содержательный анализ* существа задачи, изучение общих свойств рассматриваемого физического явления или объекта, определение конечной цели и результатов работы, анализ известной информации и определение исходных данных, *выработку общего подхода* к исследуемой проблеме: выяснению, существует ли решение поставленной задачи и единственно ли оно, и, наконец, анализ возможностей используемой вычислительной среды. На этом этапе требуется знание исследуемой предметной области и глубокое понимание существа поставленной задачи. Правильно сформулировать задачу не менее сложно, чем ее решить.

Формализация задачи. Как правило, формализация задачи сводится к построению *математической модели* рассматриваемого явления или процесса, когда в результате предыдущего анализа существа решаемой задачи устанавливается ее принадлежность к одному из известных классов задач и выбирается соответствующий математический аппарат, определяется формат исходных данных и результатов работы, вводится определенная система условных обозначений. При этом нужно уметь сформулировать на языке математики

конкретные задачи физики, механики, экономики, технологии и т. д. Для успешного преодоления этого этапа требуется не только знание предметной области, но и определенное знание вычислительной математики, тех ее разделов и методов, которые могут быть использованы при решении данной задачи.

Математическая модель — это система математических соотношений, учитывающих наиболее существенные взаимосвязи в изучаемом классе объектов/явлений и их свойства, в совокупности с определенной областью допустимых значений исходных данных и областью допустимых значений искомых результатов.

Построение любой модели представляет собой грамотное упрощение поставленной задачи и требует знания того, какие факторы и параметры наиболее существенны для изучаемой задачи и должны быть учтены при построении модели. При этом следует учитывать два противоречивых требования: требуемую простоту (математических зависимостей, вычислительных методов и расчетных схем, способов получения или измерения требуемых исходных данных, возможных способов представления результатов работы) и адекватность модели (полноту и точность) исследуемому явлению или процессу.

Если известных средств недостаточно, тогда необходимо попытаться разработать новый подход, новые методы исследования.

Выбор метода решения задачи. После математической постановки задачи мы отвлекаемся от ее предметной сущности и далее оперируем с абстрактными математическими понятиями, величинами, формулами.

Следующий шаг — анализ и оценка известных методов решения поставленной *математической задачи* и выбор наиболее приемлемого. При выборе метода надо учитывать: во-первых, характеристики самого метода, сложность формул и соотношений, связанных с этим численным методом; во-вторых, необходимую точность вычислений.

На выбор метода большое влияние оказывают вкусы и знания самого пользователя. А между тем, этот этап — важнейший в процессе решения задачи, так как оказывает существенное влияние на трудоемкость всей последующей процедуры разработки и реализации алгоритма и программы, их качество, на точность получаемых результатов.

При решении задачи на ЭВМ необходимо помнить, что любой получаемый результат является приближенным! Если известен алгоритм точного решения, то возможны ошибки, связанные с ограниченной точностью представления вещественных чисел в памяти ЭВМ. При вычислениях с заданной степенью точности (при использовании численных методов) возникает дополнительная погрешность, определяемая выбранным методом и которую, по возможности, оценивают на данном этапе.

Разработка алгоритма. *Процесс алгоритмизации* — это отдельный этап, и мы придаем ему особую значимость. Именно на

этом этапе осуществляется процесс разработки структуры алгоритма, реализующего выбранный метод решения, и осуществляется запись «придуманной» структуры на языке, «понятном» самому разработчику. Можно использовать различные способы описания алгоритмов, отличающиеся по простоте и наглядности. В практике программирования наибольшее распространение получили словесная запись алгоритмов, схемы алгоритмов (блок-схемы) и структурограммы (диаграммы Насси—Шнейдермана).

Построение алгоритма заключается в разложении вычислительного процесса решения задачи на возможные составные части, установлении порядка их следования, описании содержания каждой такой части в той или иной форме и последующей проверке, которая должна показать, обеспечивается ли реализация выбранного метода. Разработать структуру алгоритма порой непросто и получить «правильный», а тем более «хороший» алгоритм в большинстве случаев удается только после *многих шагов анализа и коррекции* его структуры.

Как правило, в процессе разработки алгоритм проходит несколько этапов детализации. Первоначально составляется укрупненная схема алгоритма, в которой отражаются наиболее важные и существенные связи между исследуемыми процессами (или частями процесса). На последующих этапах раскрываются (детализируются) выделенные на предыдущих этапах части вычислительного процесса, имеющие некоторое самостоятельное значение. Кроме того, на каждом этапе детализации выполняется многократная проверка и исправление (отработка) схемы алгоритма. Подобный подход позволяет во многом избежать возможных ошибочных решений.

Ориентируясь на крупноблочную структуру алгоритма, можно быстрее и проще разработать несколько различных его вариантов, провести их анализ, оценку и выбрать наилучший (оптимальный).

Эффект *поэтапной детализации алгоритма* во многом зависит от того, как осуществляется его структуризация: разделение общего алгоритмического процесса на составные части, что должно определяться не произволом пользователя (программиста), а внутренней логикой самого процесса. Каждый элемент крупноблочной схемы алгоритма должен быть максимально самостоятельным и логически завершенным в такой степени, чтобы дальнейшую его детализацию можно было выполнять независимо от детализации остальных элементов. Это упрощает процесс проектирования алгоритма и позволяет осуществлять его разработку по частям одновременно нескольким людям.

Разработка алгоритмов является в значительной степени творческим, эвристическим процессом, как правило, требует большой эрудиции, изобретательности, нестандартных и нетрадиционных подходов к решению задачи. Но даже этот творческий процесс включает с себя чисто технологические вопросы и, придерживаясь

определенной дисциплины (технологии) в разработке алгоритма, можно облегчить себе и сам процесс его разработки, и дальнейшую работу с этим алгоритмом.

Разработка программы. *Процесс программирования* — это запись разработанного алгоритма на специальном языке (*языке программирования*) — представление алгоритма на языке, «понятном» исполнителю (вычислительной машине), т.е. в форме, допускающей ввод в машину и последующий перевод на машинный язык (в коды машины).

Язык программирования — это строго формализованный язык для описания процесса решения задачи на ЭВМ, представляет собой совокупность ограниченного набора символов и строгих правил их использования. Составленная программа вводится в ЭВМ и затем автоматически переводится на язык машины с помощью специальных программных средств, позволяющих автоматизировать этот процесс. Перевод — *трансляция* исходного текста программы выполняется служебной программой — *транслятором*, который осуществляет синтаксический контроль текста программы и последующий его перевод.

Трансляторы могут быть компилирующего типа — компиляторы и интерпретирующего типа — интерпретаторы.

Компилятор анализирует и преобразует исходный текст в так называемый объектный код (промежуточное состояние программы в относительных адресах и с неразрешенными внешними ссылками) с использованием всей логической структуры программы. Затем программа, представленная в объектном коде, обрабатывается служебной программой — *компоновщиком*, который осуществляет подключение внешних подпрограмм/разрешение внешних ссылок и выполняет дальнейший перевод программы пользователя в коды машины (в абсолютный/загрузочный код — с абсолютной адресацией машинных команд). Программа в абсолютном коде может быть сохранена (в *.exe*-файле) и выполнена на компьютере. Загрузка программы из *.exe*-файла в память машины для ее выполнения осуществляется служебной программой *загрузчиком*.

Интерпретатор сразу производит анализ, перевод (в машинный код) и выполнение программы строка за строкой. Поэтому интерпретатор должен находиться в оперативной памяти в течение всего времени выполнения программы пользователя. При интерпретации скорость выполнения программы существенно снижается, однако весь процесс прохождения программы на ЭВМ упрощается и имеется возможность организации диалогового (*интерактивного*) режима отладки и выполнения программы.

Выбор языка программирования определяется многими факторами: типом решаемой задачи, располагаемыми вычислительными средствами, вкусами и знаниями заказчика и разработчика.

Язык программирования *Паскаль* был создан профессором, директором института информатики Швейцарской высшей политех-

нической школы г. Цюриха *Никлаусом Виртом*, в 1968 г. впервые опубликовано предварительное описание языка, в 1970 г. представлен компилятор. Язык назван в честь известного математика *Блеза Паскаля* (1623—1662), автора первой механической вычислительной сумматорной машины. Язык Паскаль был создан автором специально для обучения дисциплине программирования в высшей школе как язык поддержки структурного программирования и средство формирования у обучаемого определенного стиля и практических навыков программирования. Основной тезис его разработки: «язык должен быть очевидным и естественным отражением фундаментальных и наиболее важных концепций алгоритмов». Широкое распространение языка, его современных диалектов свидетельствует о его практической ценности в различных сферах применения и, прежде всего, в сфере начального обучения программированию и формирования профессиональных навыков будущего специалиста в области IT-технологий.

Отладка и тестирование программы. *Отладка программы* — это процесс поиска и устранения ошибок. Часть ошибок формального характера, связанных с нарушением правил записи конструкций языка или отсутствием необходимых описаний, обнаруживает транслятор, производя синтаксический анализ текста программы. Транслятор выявляет ошибки и сообщает о них, указывая их тип и место в программе. Такие ошибки называются *ошибками времени трансляции* или *синтаксическими ошибками*.

Ошибочные ситуации могут возникнуть и при выполнении программы, например деление на нуль или извлечение корня квадратного из отрицательного числа. Такие ошибки называются *ошибками времени выполнения*.

Программа, не имеющая ошибок трансляции и выполнения, может и не дать верных результатов из-за логических ошибок в алгоритме, т. е. *алгоритмических*, *или семантических*, *ошибок*. Ошибки подобного рода могут возникнуть на любом этапе разработки программы: постановки задачи, разработки математической модели или алгоритма.

Необходим действенный контроль над процессом вычислений, позволяющий предотвращать или своевременно обнаруживать ошибки подобного рода. Для этого используются как качественный анализ задачи, основанный на различного рода интуитивных соображениях и правдоподобных рассуждениях, так и контрольный просчет, тестирование программы.

Тестирование программы — это выполнение программы на наборах исходных данных (*тестах*), для которых известны результаты, полученные другим методом. Система тестов подбирается таким образом, чтобы

а) проверить все возможные режимы работы программы;

б) по-возможности, локализовать ошибку.

При тестировании программы простой и действенный метод дополнительного контроля над ходом ее выполнения — получение *контрольных точек*, т. е. контрольный вывод промежуточных результатов.

Для проверки правильности работы программы иногда полезно также выполнить проверку выполнения условий задачи (например, для алгебраического уравнения найденные корни подставляются в исходное уравнение и проверяются расхождения левой и правой частей).

Для сложных по структуре программ плохо спланированные процессы алгоритмизации и программирования приводят к ошибкам, которые могут быть обнаружены лишь после многократных проверок, и процесс отладки и тестирования может потребовать значительно больше машинного времени, чем собственно само решение задачи на ЭВМ.

Документирование программы. Разработка программной документации является важным аспектом процесса решения задач на ЭВМ. Одним из показателей квалификации программиста принято считать умение и навыки документирования программных продуктов. Главная цель документации состоит в том, чтобы помочь стороннему пользователю понять программу. Наличие документации помогает сэкономить значительное время и позволяет избежать многих недоразумений. В общем случае текст готовой программы снабжается внешней и программной документацией.

Внешняя документация представляет собой сведения о программе, не содержащиеся в самой программе. В зависимости от размеров и сложности программы внешняя документация может принимать различные формы:

- схемы алгоритмов;
- инструкции для пользователей;
- образцы входных и выходных данных;
- полное описание процесса построения программы;
- словесное описание алгоритма;
- ссылки на источники информации и др.

Программная документация реализуется в основном с помощью соответствующих комментариев (в начале программы в виде заголовка и вводных комментариев, и поясняющих — внутри текста программы), а также рационального выбора имен, применения стандартных методов структурирования программ.

Документирование программы регламентируется стандартами Единой системы программной документации (ЕСПД) вне зависимости от назначения и области их применения (ГОСТ 19.001—77, ГОСТ 19.105—78). Виды программ и программных документов для вычислительных машин, комплексов и систем установлены ГОСТ 19.101—77. *Виды программных документов*, разрабатываемые на различных стадиях проектирования программ (ГОСТ 19.102—77), их

содержание и коды приведены в табл. В.1. *Виды эксплуатационных документов*, их коды и соответствующие им стандарты, определяющие содержание работ и документов, представлены в табл. В.2.

Таблица В.1. **Виды программных документов для этапа проектирования программ**

Вид программного документа	Код вида документа	Содержание программного документа
Ведомость держателей подлинников	05	Перечень предприятий, на которых хранят подлинники программных документов
Текст программы	12	Запись программы с необходимыми комментариями
Описание программы	13	Сведения о логической структуре и функционировании программы
Программа и методика испытаний	51	Требования, подлежащие проверке при испытании программ, а также порядок и методы их контроля
Техническое задание	—	Назначение и область применения программы, технические, технико-экономические и специальные требования, предъявляемые к программе, необходимые стадии и сроки разработки, виды испытаний
Пояснительная записка	81	Схема алгоритма, общее описание алгоритма и функционирования программы, а также обоснование принятых технических и технико-экономических решений
Эксплуатационные документы	—	Сведения для обеспечения функционирования и эксплуатации программ

Таблица В.2. **Виды эксплуатационных документов**

Вид документа	Код вида документа	Стандарт ЕСПД
Ведомость эксплуатационных документов	20	ГОСТ 19.507—79
Формуляр	30	ГОСТ 19.501—78

Окончание табл. В.2

Вид документа	Код вида документа	Стандарт ЕСПД
Описание применения	31	ГОСТ 19.502—78
Руководство системного программиста	32	ГОСТ 19.503—79
Руководство программиста	33	ГОСТ 19.504—79
Руководство оператора	34	ГОСТ 19.505—79
Описание языка (программы)	35	ГОСТ 19.506—79
Руководство по техническому обслуживанию	46	ГОСТ 19.508—79

Вычисления и обработка результатов. Только после того как появится полная уверенность, что программа обеспечивает получение правильных результатов, можно приступать непосредственно к расчетам.

После завершения расчетов наступает этап использования полученных результатов вычислений в практической деятельности. Интерпретация результатов вычислений и их использование снова относятся к той предметной области знаний, откуда возникла задача.

Глава 1
ОСНОВЫ АЛГОРИТМИЗАЦИИ

1.1. Понятие алгоритма

Термин «алгоритм» произошел от имени арабского математика аль-Хорезми (IX в.), которым были описаны общие правила (названные позднее алгоритмами) выполнения основных арифметических действий в десятичной системе счисления. Эти алгоритмы изучаются в начальных разделах школьной математики. К числу алгоритмов школьного курса математики относятся также правила решения определенных видов уравнений или неравенств, правила построения различных геометрических фигур и т. п. Понятие алгоритма используется не только в математике, но и во многих областях человеческой деятельности: например, говорят об алгоритме управления производственным процессом, алгоритме управления полетом ракеты, алгоритме пользования бытовым прибором. Причем интуитивно под алгоритмом понимают некоторую систему правил, обладающих определенными свойствами.

Далее, изучая понятие алгоритма, мы будем предполагать, что его исполнителем является автоматическое устройство — ЭВМ. Это накладывает на запись алгоритма целый ряд обязательных требований. Сформулируем эти требования в виде *перечня свойств*, которыми должен обладать алгоритм, адресуемый к исполнению на ЭВМ.

1. Первым свойством алгоритма является *дискретный* (пошаговый) характер определяемого им процесса. Возникающая в результате такого разбиения запись алгоритма представляет собой упорядоченную последовательность отдельных предписаний (директив, команд), образующих прерывную/дискретную структуру алгоритма: только выполнив требования одного предписания можно приступить к исполнению следующего.

2. Исполнитель может выполнить алгоритм, если он ему *понятен*, т. е. записан на понятном ему языке и содержит предписания, которые исполнитель может выполнить. Набор действий, которые могут быть выполнены исполнителем, называется *системой команд исполнителя*. Алгоритм не должен содержать описания действий, не входящих в систему команд исполнителя, т. е. своей структурой

команд и формой записи алгоритм должен быть ориентирован на конкретного исполнителя.

3. Алгоритмы, предназначенные для исполнения техническим устройством, не должны содержать предписаний, приводящих к неоднозначным действиям.

Алгоритм рассчитан на чисто механическое исполнение, и если применять его повторно к одним и тем же исходным данным, то всегда должен получаться один и тот же результат; при этом и промежуточные результаты, полученные после соответствующих шагов алгоритмического процесса, тоже должны быть одинаковыми. Это свойство определенности и однозначности — *детерминированности* — алгоритма позволяет использовать в качестве исполнителя специальные машины-автоматы.

4. Основополагающим свойством алгоритма является его *массовость*, применимость к некоторому классу объектов, возможность получения результата при различных исходных данных на некоторой области допустимых значений. Например, исходными данными в алгоритмах аль-Хорезми могут быть любые пары десятичных чисел. Конечно, его способ не всегда самый рациональный по сравнению с известными приемами быстрого счета. Но смысл массовости алгоритма состоит как раз в том, что он одинаково пригоден для всех случаев, требует лишь механического выполнения цепочки простых действий и при этом исполнителю нет нужды в затратах творческой энергии.

5. Цель выполнения алгоритма — получение конечного *результата* посредством выполнения указанных преобразований над исходными данными. В алгоритмах аль-Хорезми исходными данными являются два десятичных числа, результатом — также десятичное число. Причем при точном исполнении всех предписаний алгоритмический процесс должен заканчиваться за *конечное число шагов*. Это обязательное требование к алгоритмам — требование их *результативности или конечности*.

В математике известны вычислительные процедуры алгоритмического характера, не обладающие свойством конечности. Например, процедура вычисления числа π. Однако если мы введем условие завершения вида «закончить после получения n десятичных знаков числа π», то получим алгоритм вычисления n десятичных знаков числа π. На этом принципе построены многие вычислительные алгоритмы.

6. Если алгоритм должен быть выполнен не просто за конечное время, а за разумное конечное время, то речь идет об *эффективности* алгоритма. Время выполнения алгоритма — очень важный параметр, однако понятие эффективности алгоритма трактуется шире, включая такие аспекты, как сложность, необходимые ресурсы, информационно-программное обеспечение. Эффективность алгоритма нередко определяет возможность его практической реализации.

Перечисленные свойства алгоритма, по существу, являются неформальным его определением. Объединяя их в одно целое, мы можем сформулировать это определение следующим образом: **алгоритм** — это полное и точное описание на некотором языке конечной последовательности правил, указывающих исполнителю действия, которые он должен выполнить, чтобы за конечное время перейти от (варьируемых) исходных данных к искомому результату.

1.2. Алгоритмическая система

Понятие «алгоритмическая система» дает формальный ответ на вопрос, что должно быть известно и доступно разработчикам алгоритмов.

Алгоритмическая система — это набор средств и понятий, позволяющих строить некоторое множество алгоритмов для решения определенного класса задач. Алгоритмическая система определяется наличием четырех составляющих ее частей:

1) множеством входных объектов или исходных данных, подлежащих обработке алгоритмами данной системы;

2) множеством выходных объектов или результатов выполнения алгоритмов данной системы;

3) системой команд исполнителя, т. е. набором тех действий, которые может выполнять исполнитель и которые мы можем описывать в алгоритмах, что собственно является ориентацией алгоритмической системы на конкретного исполнителя;

4) языком описания алгоритмов — языком исполнителя; язык, на котором описан алгоритм, должен быть понятен исполнителю и не должен включать в свой состав указания на невозможные для исполнителя действия, а также обращения к входным или выходным объектам, не принадлежащим к множеству входных или выходных объектов данной алгоритмической системы.

В качестве примера рассмотрим алгоритмическую систему, предназначенную для построения алгоритмов обработки данных — алгоритмов обработки символьных последовательностей (строк) из ограниченного алфавита символов. Входными объектами такой системы являются строки символов конечной длины. С помощью специальных приемов можно преобразовать в строки символов практически любую информацию, в том числе формулы, таблицы, рисунки. Результат обработки данных также представляет собой строки символов. Алгоритмические системы для обработки данных строятся на одном и том же множестве входных и выходных объектов.

Исполнителем в современных системах обработки данных является вычислительная машина. Набор операций, выполняемых ЭВМ, весьма ограничен, однако, комбинируя их в нужной последователь-

ности, можно строить весьма сложные алгоритмы решения множества самых различных задач. Язык, на котором записываются алгоритмы, адресованные вычислительной машине, опирается на систему команд данной ЭВМ. Алгоритм, написанный на машинном языке, представляет собой закодированную специальным образом последовательность команд, адресованных различным устройствам ЭВМ.

Отметим принципиальную особенность алгоритмических систем обработки данных. В таких системах текст алгоритма также является последовательностью символов, которую можно преобразовать в той же алгоритмической системе. Следовательно, открывается возможность составлять алгоритмы преобразования алгоритмов, обрабатывая при этом тексты, реализующие преобразуемые алгоритмы. Это и создает ту удивительную логическую гибкость, которая превратила ЭВМ в принципиально новый инструмент обработки данных, обладающий колоссальными возможностями. Одним из примеров преобразования алгоритма с помощью ЭВМ из одной формы в другую является его *трансляция* — перевод с некоторого алгоритмического языка на язык машины.

1.3. Алгоритмизация

Алгоритмизация — это процесс разработки и описания алгоритма решения какой-либо задачи. Пусть мы имеем некоторую математическую задачу, которая может быть решена одним из известных математических методов. Как приступить к процессу построения алгоритма решения такой задачи?

Поскольку речь идет о разработке алгоритма для ЭВМ, то нужно сначала проанализировать возможность его машинной реализации, оценить ресурсы и возможности конкретной ЭВМ, имеющейся в распоряжении (в том числе, допустимую точность вычислений, объем запоминающих устройств, быстродействие, информационно-программное обеспечение).

Непосредственная разработка алгоритма начинается с осознания существа поставленной задачи, с анализа того, что нам известно, что следует получить в качестве результата, в какой форме нужно представить исходные данные и результаты вычислений. Следующая ступень — разработка общей идеи алгоритмического процесса и анализа этой идеи. После этого можно приступить к более детальной разработке уже задуманного конкретного алгоритма. И вот этот *процесс разработки* конкретного *алгоритма*, в соответствии с определением самого понятия «алгоритм», заключается в последовательном выполнении пунктов:

1) разложение всего вычислительного процесса на отдельные шаги — возможные составные части алгоритма, что определяется внутренней логикой самого процесса и системой команд исполнителя;

2) установление взаимосвязей между отдельными шагами алгоритма и порядка их следования, приводящего от известных исходных данных к искомому результату;

Таблица 1.1. **Целые числа**

5	1	3	8
10	9	6	1
5	10	1	1

3) полное и точное описание содержания каждого шага алгоритма на языке выбранной алгоритмической системы;

4) проверка составленного алгоритма на предмет, действительно ли он реализует выбранный метод и приводит к искомому результату.

В результате проверки могут быть обнаружены ошибки и неточности, что вызывает необходимость доработки и коррекции алгоритма — возвращение к одному из предыдущих пунктов. Как правило разработка алгоритма включает в себя многократно повторяющуюся процедуру его анализа и коррекции.

Процедура анализа и коррекции алгоритма производится не только с целью устранения ошибок, но и с целью улучшения, т. е. оптимизации алгоритма. При определенном методе решения задачи оптимизация проводится с целью сокращения алгоритмических действий и упрощения по возможности самих этих действий. При этом алгоритм должен оставаться «эквивалентным» исходному.

Будем называть два алгоритма *эквивалентными*, если:

1) множество допустимых исходных данных одного из них является множеством допустимых исходных данных и другого; из применимости одного алгоритма к каким-либо исходным данным следует применимость и другого алгоритма к этим данным;

2) применение этих алгоритмов к одним и тем же исходным данным дает одинаковые результаты.

Приведем пример двух эквивалентных алгоритмов. Пусть нам надо подсчитать общую сумму чисел, приведенных в табл. 1.1.

Эквивалентными будут алгоритмы подсчета общей суммы по строкам: 5 + 1 + 3 + 8 + 10 + 9 + 6 + 1 + 5 + 10 + 1 + 1 = 60, и по столбцам: 5 + 10 + 5 + 1 + 9 + 10 + 3 + 6 + 1 + 8 + 1 + 1 = 60. Заметим, что для данной таблицы считать проще по столбцам.

1.4. Средства записи алгоритмов

1.4.1. Уровень языка описания алгоритма

Характер языка, используемого для записи алгоритмов, определяется тем, для какого исполнителя предназначен алгоритм. Возможности исполнителя алгоритмов определяют *уровень* используемых языковых средств, т. е. степень детализации и формализации пред-

писаний в алгоритмической записи. Если алгоритм предназначен для исполнителя-человека, то его запись может быть не полностью формализована и детализирована, но должна оставаться понятной и корректной. Для записи таких алгоритмов может использоваться *естественный* язык. Для записи алгоритмов, предназначенных для исполнителей-автоматов, необходимы строгая формализация средств записи и определенная детализация алгоритмических предписаний. В таких случаях применяют специальные формализованные языки.

Поскольку одним из пользователей языка описания алгоритмов, так или иначе, остается человек, то, говоря об уровне языка, имеют в виду также и уровень его доступности для человека.

К настоящему времени в информатике сложились вполне определенные традиции в представлении алгоритмов.

1.4.2. Словесная запись алгоритмов

Самой распространенной формой представления алгоритмов, адресуемых человеку, является обычная *словесная запись*. В этой форме могут быть выражены любые алгоритмы. Но если такой алгоритм предназначен для его дальнейшей реализации на вычислительном устройстве, то принято придерживаться более формализованного способа построения фраз с тщательно отобранным набором слов. Кроме того, необходимо указывать начало и конец алгоритма, отмечать момент ввода в вычислительное устройство значений исходных данных и вывода/печати полученного результата. В вычислительных алгоритмах широко используется общепринятая математическая символика, язык формул. Вводится необходимая в вычислительной практике *операция присваивания*:

$$y := A$$

(читается: «y присвоить значение A»), где y — переменная; A — некоторое выражение/формула.

Следует сначала выполнить все действия, предусмотренные формулой A, а затем полученный результат сохранить в качестве значения переменной y. Выражение A в частном случае может быть переменной или числом. Например:

$x := \sin a$ — присвоить переменной x значение синуса;

$z := \ln a + 3b^3$ — присвоить переменной z значение суммы;

$y := x$ — присвоить переменной y значение переменной x;

$z := 5.7$ — считать значением переменной z число 5,7;

$k := k + 1$ — значение переменной k увеличить на единицу.

Введенные соглашения позволяют представлять словесные алгоритмы в достаточно компактной и наглядной форме.

Пример 1.1. Составим алгоритм вычисления коэффициентов приведенного квадратного уравнения $x^2 + px + q = 0$, корни которого x_1, x_2 известны. Коэффициенты такого уравнения определяются по следующим формулам:

$$p = -(x_1 + x_2);$$
$$q = x_1 x_2.$$

Предположим, что правила выполнения арифметических операций сложения, вычитания и умножения известны исполнителю. Тогда искомым алгоритмом будет следующая система предписаний:

Начало.
1. Ввести x_1, x_2.
2. $p := -(x_1 + x_2)$.
3. $q := x_1 x_2$.
4. Вывести p, q.
Конец.

В приведенной записи алгоритма ни в одном из предписаний нет указания на то, к какому следующему предписанию надо перейти. Предполагается, что в случае отсутствия специальных указаний предписания алгоритма выполняются в порядке их следования. Такое выполнение предписаний в алгоритме называется *естественным ходом* выполнения алгоритма, а сам алгоритм называется *линейным*.

Пример 1.2. Составим алгоритм определения максимального числа из трех: $z := \max(a, b, c)$.

Решение задачи на ЭВМ можно получить, действуя следующим образом. Сначала найдем наибольшее из двух чисел, например, сравнив между собой a и b. Предположим, что исполнитель может выполнить операцию сравнения «больше». Найденное наибольшее число «запомним» в качестве значения переменной z. Далее сравним значение переменной z с оставшимся числом c. Если c больше z, то присвоим z новое значение — значение c, в противном случае значение z останется прежним. В результате переменная z будет равна наибольшему из a, b, c и будет являться искомым результатом.

Изложенные рассуждения можно представить в виде следующей словесной записи алгоритма:

Начало.
1. Ввести a, b, c.
2. Если $a > b$, то $z := a$;
   иначе $z := b$.
3. Если $c > z$, то $z := c$.
4. Вывести z.
Конец.

Ход выполнения алгоритма зависит от результатов проверки условий $a > b$ и $c > z$. Если для введенных значений a, b действительно $a > b$, то выполняется операция $z := a$; если нет, то выполняется $z := b$. Таким об-

разом, в зависимости от результата проверки условия $a > b$ требуется выполнить различные действия. В алгоритме на этом шаге предусмотрены оба возможных направления дальнейших вычислений. При проверке условия $c > z$ операция $z := c$ может выполняться, если действительно $c > z$, или не выполняться, в противном случае.

Вычислительный процесс, который в зависимости от выполнения некоторых условий реализуется по одному из нескольких возможных, заранее предусмотренных направлений, называется *разветвляющимся*. Каждое отдельное направление называется ветвью вычислений. Выбор той или иной ветви осуществляется при выполнении алгоритма в результате проверки этих условий и определяется свойствами исходных данных и промежуточных результатов. При разработке алгоритма должны быть учтены все возможные ветви вычислений.

Пример 1.3. Составим алгоритм определения остатка от деления двух целых неотрицательных чисел A и B, где $B \neq 0$. Рассматривая деление как многократное вычитание делителя B из делимого A, получим следующий алгоритм:

Начало.
1. Ввести A, B.
2. Если $A < B$, то перейти к п. 5;
   иначе перейти к п. 3.
3. $A := A - B$.
4. Перейти к п. 2.
5. Ост $:= A$.
6. Вывести Ост.

Конец.

Пункты 2, 3, 4 записаны в алгоритме один раз, а могут выполняться многократно. Многократно повторяемые участки вычислений образуют так называемый *цикл*. Вычислительный процесс, содержащий многократные вычисления по одним и тем же математическим зависимостям, но для различных значений входящих в них переменных, называется *циклическим*. Переменные, изменяющиеся в цикле, называются *переменными цикла*, а действия, повторяемые в цикле, — *телом цикла*. Количество повторений цикла определяется значениями переменных, входящих в условие его окончания.

Чтобы лучше понять характер циклических процессов, рассмотрим подробнее структуру приведенного алгоритма. Первый шаг алгоритма представляет собой *подготовку цикла*: задание начальных значений переменным цикла перед первым его выполнением. *Тело цикла* — действия, многократно повторяемые в цикле (п. 2, 3, 4). В каждом конкретном случае число повторений этих действий будет различным. Пункт 3 обеспечивает *модификацию* (изменение) значений переменной цикла A, входящей в условие его окончания. *Управление циклом* осуществляется в п. 2. Проверяется условие окончание цикла ($A < B$), и осуществляется либо выход из цикла, либо возврат на его повторение. Очень важно правильно сформулировать условие окончания цикла.

Поняв сущность и структуру циклических алгоритмов, мы могли бы записывать их в более компактном виде, например:

Начало.
1. Ввести A, B.
2. Пока $A \geq B$ выполнять $A := A - B$.
3. Ост $:= A$.
4. Вывести Ост.
Конец.

Предписание «Пока $A \geq B$ выполнять $A := A - B$» надо понимать следующим образом: если A больше или равно B, то выполнить операцию $A := A - B$ и перейти опять на проверку условия $A \geq B$; если A меньше B, то перейти к выполнению следующего предписания (п. 3), не выполняя операцию $A := A - B$.

1.4.3. Схемы алгоритмов

Схема алгоритма — это графический способ его представления с элементами словесной записи. Каждое предписание алгоритма изображается с помощью плоской геометрической фигуры — *блока*. Отсюда название: *блок-схема*. Переходы от предписания к предписанию изображаются линиями связи — *линиями потоков информации*, а направление переходов — *стрелками*. Различным по типу выполняемых действий блокам соответствуют различные геометрические фигуры. Приняты определенные стандарты ГОСТ 19.701—90 (ИСО 5807—85) графических изображений блоков (табл. 1.2).

Рассмотрим ***общие правила построения схем алгоритмов***.

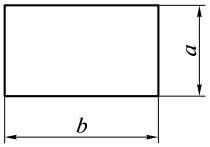
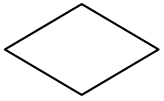
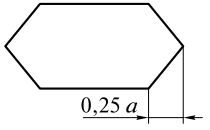
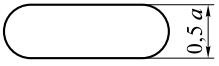
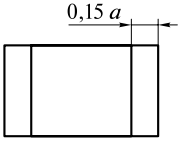
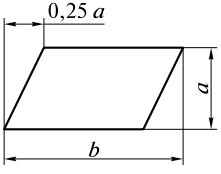
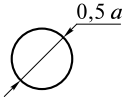
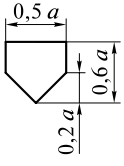
1. Для конкретизации содержания блока и уточнения выполняемого действия внутри блока помещаются краткие пояснения — словесные записи с элементами общепринятой математической символики (рис. 1.1, *а*).

2. Основное направление потока информации в схемах может не отмечаться стрелками. Основное направление — сверху вниз и слева направо. Если очередность выполнения блоков не соответствует этому направлению, то возможно применение стрелок (рис. 1.1, *б*).

3. По отношению к блоку линии могут быть входящими и выходящими. Количество входящих линий принципиально не ограничено. Количество выходящих линий регламентировано и зависит от типа блока. Например, логический блок должен иметь не менее двух выходящих линий, каждая из которых соответствует одному из возможных направлений вычислений. Блок модификации должен иметь две выходящие линии, одна соответствует повторению цикла, вторая — его окончанию.

4. Допускается разрывать линии потока информации, размещая на обоих концах разрыва специальный символ «соединитель»

Таблица 1.2. **Изображение блоков в схемах алгоритмов**

Символ	Обозначение и размеры	Функция
Процесс (вычислительный блок)		Выполнение операции или группы операций, в результате которых изменяются значение, форма представления или расположение данных
Решение (логический блок)		Выбор направления выполнения алгоритма в зависимости от некоторых условий
Модификация (заголовок цикла)		Выполнение операций по управлению циклом — повторением команды или группы команд алгоритма
Пуск-останов (начало-конец)		Начало или конец выполнения программы или подпрограммы
Предопределенный процесс (вызов подпрограммы)		Вызов и использование ранее созданных и отдельно описанных алгоритмов (подпрограмм)
Ввод-вывод		Общее обозначение ввода или вывода данных в алгоритме безотносительно к внешнему устройству
Соединитель		Указание прерванной связи между блоками в пределах одной страницы
Межстраничный соединитель		Указание прерванной связи между блоками, расположенными на разных листах

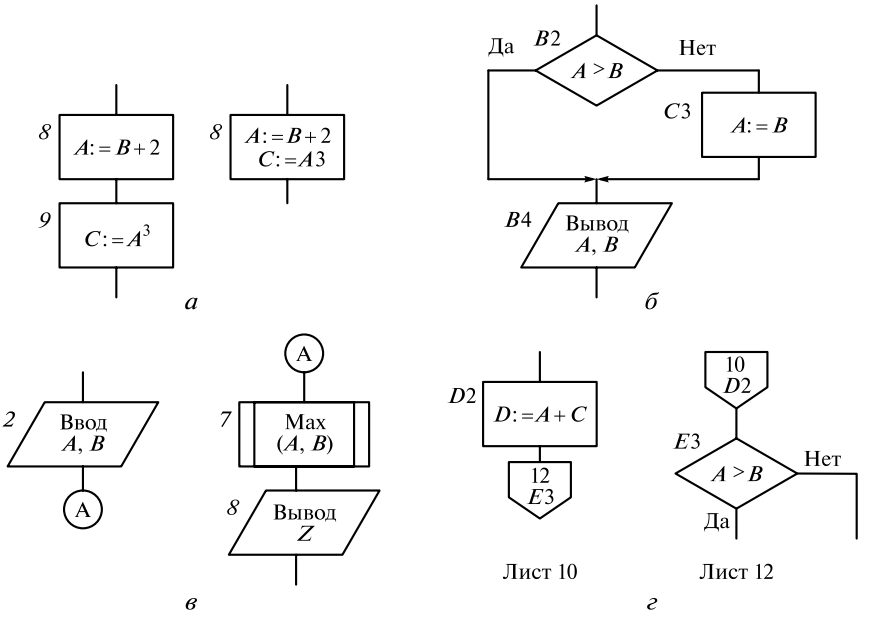


Рис. 1.1. Примеры изображения элементов схем алгоритмов:

а — линейные вычисления; *б* — ветвление; *в* — соединитель на странице; *г* — межстраничный соединитель

(рис. 1.1, *в*, *г*). В пределах одной страницы используется символ обычного соединителя, во внутреннем поле которого помещается маркировка разрыва либо отдельной буквой, либо буквенно-цифровой координатой блока, к которому подходит линия потока.

Если схема располагается на нескольких листах, переход линий потока с одного листа на другой обозначается с помощью символа «межстраничный соединитель». При этом на листе с блоком-источником соединитель содержит номер листа и координаты блока-приемника, а на листе с блоком-приемником — номер листа и координаты блока-источника.

5. Нумерация блоков осуществляется либо в левом верхнем углу блока в разрыве его контура, либо рядом слева от блока (см. рис. 1.1). Принцип нумерации может быть различным, наиболее простой — сквозная нумерация. Блоки начала и конца не нумеруются.

6. Для блоков приняты следующие размеры: a = 10, 15, 20 мм; $b = 1{,}5a$. Если необходимо увеличить размер блока, то допускается увеличение на число, кратное пяти. Необходимо выдерживать минимальное расстояние 3 мм между параллельными линиями потоков и 5 мм между остальными символами.