А.С. Иванов

ИНФОРМАТИКА

Учебное пособие

Саратов

Издательский центр «Наука»

2009

УДК 004(075.8)
ББК 32.81я73
И18

Иванов А.С.

Информатика: Учеб. пособие. – Саратов: ООО Издательский центр
И18 «Наука», 2009. – 84 с.: ил.

ISBN 978-5-9999-0089-0

В пособии содержится учебный материал для курса лекций «Информатика и компьютерные науки».

Для студентов факультета компьютерных наук и информационных технологий Саратовского государственного университета.

Рецензенты:

Доктор физико-математических наук, доцент *Д.К. Андрейченко*

Кандидат физико-математических наук *Д.В. Кондратов*

УДК 004(075.8)
ББК 32.81я73

Работа издана в авторской редакции

ISBN 978-5-9999-0089-0

Оглавление

Глава 1. История информатики

1.1. Понятие информатики

Информатика — это наука, предметом изучения которой являются процессы сбора, преобразования, хранения, защиты, поиска и передачи всех видов информации, а также средства их автоматизированной обработки.

1.2. Роль и значение информационных революций

В истории развития цивилизации произошло несколько информационных революций — преобразований общественных отношений из-за кардинальных изменений в сфере обработки информации. Следствием подобных преобразований являлось приобретение человеческим обществом нового качества.

Первая революция связана с изобретением письменности, что привело к гигантскому качественному и количественному скачку. Появилась возможность передачи знаний от поколения к поколению.

Вторая (середина XVI в.) вызвана изобретением книгопечатания, которое радикально изменило индустриальное общество, культуру, организацию деятельности.

Третья (конец XIX в.) обусловлена изобретением электричества, благодаря которому появились телеграф, телефон, радио, позволяющие оперативно передавать и накапливать информацию в любом объеме.

Четвертая (70-е гг. XX в.) связана с изобретением микропроцессорной технологии и появлением персонального компьютера. На микропроцессорах и интегральных схемах создаются компьютеры, компьютерные сети, системы передачи данных (информационные коммуникации). Этот период характеризуют три фундаментальные инновации: переход от механических и электрических средств преобразования информации к электронным; миниатюризация всех узлов, устройств, приборов, машин; создание программно-управляемых устройств и процессов.

1.3. История развития ЭВМ

История компьютера тесным образом связана с попытками человека облегчить, автоматизировать большие объемы вычислений. Даже простые арифметические операции с большими числами затруднительны для человеческого мозга. Поэтому уже в древности появилось простейшее счетное устрой-

ство — абак, прообраз современных счётов. В XVII в. была изобретена логарифмическая линейка, упрощающая проведение сложных математических расчётов. Первая машина, способная автоматически выполнять четыре арифметических действия, также появилась в XVII в.

В 1623 г. В. Шикард изобрел машину, способную не только суммировать и вычитать числа, но и частично перемножать и делить их.

В 1642 г. французский философ и ученый Б. Паскаль изобрел арифмометр для механизации канцелярских расчетов.

В 1671 г. немецкий философ и математик Г. Лейбниц создал свою счетную машину, известную как «зубчатое колесо Лейбница».

В XIX в. английский математик Ч. Бэббидж разработал несколько проектов вычислительных механических устройств, самым известным из которых является «аналитическая машина» Бэббиджа, которая представляла собой программируемое автоматическое вычислительное устройство. Программы кодировались и переносились на перфокарты. Эту идею Бэббидж позаимствовал у французского изобретателя Ж. Жаккара, впервые применившего ее для контроля ткацких операций. По замыслу Бэббиджа такая машина должна была автоматически выполнять различные вычисления при последовательном вводе набора перфокарт, содержащих пары команд и данных. Изменяя расположение отверстий на карте и очередность следования карт, можно было менять порядок вычислений.

Меценат проекта — графиня А. Лавлейс — была программистом этой «аналитической машины». Именно она убедила Бэббиджа в необходимости использования двоичной системы счисления вместо десятичной. Ею были разработаны новые принципы программирования, предусматривающие повторение одной и той же последовательности команд и выполнение команд при определенных условиях (команды условного перехода). Ее именем назван разработанный в 1979 г. алгоритмический язык ADA.

Во второй половине XIX в. Г. Холлерит разработал машину с перфокарточным вводом, способную автоматически классифицировать и составлять таблицы данных. Наличие-отсутствие отверстия в перфокарте обнаруживалось электрическими контактными щетками, а в счетчиках применялись реле. Впервые такая машина использовалась в 1890 г. в Америке при обработке результатов переписи населения. Именно тогда стало ясно, что без создания новых процессов обработки данных невозможно выполнять обработку больших массивов информации. С тех пор машины с перфорированными картами получили широкое распространение в деловой и административной сферах. В 1896 г. Холлерит основал фирму «COMPUTING TOBULATING RECORDING COMPANY», которая стала основой для будущей IBM (это на-

звание возникло в 1924 г.) — компании, внесшей гигантский вклад в развитие мировой компьютерной техники.

Скорость вычислений в механических машинах на основе зубчатого колеса и в электрических машинах, выполненных на реле, была ограничена, поэтому в 30-х гг. начались разработки электронных вычислительных машин (ЭВМ), элементной базой которых стала трехэлектродная вакуумная лампа, изобретенная в 1906 г. Л. Форестом.

Первая треть XX в. ознаменовалась последовательным развитием и внедрением многих вычислительных устройств. Значительный вклад в этот процесс внес математик А. Тьюринг, который в 1937 г. опубликовал работу с описанием универсальной схемы вычислений. Хотя машина Тьюринга была лишь теоретическим построением и никогда серьезно не рассматривалась как экономически приемлемая машина, она привлекла внимание ряда исследователей.

Вторая мировая война дала серьезный толчок к усовершенствованию вычислительных устройств и технологий их производства. В 1944 г. Г. Айкен и группа исследователей из IBM построили электрическую вычислительную машину на релейных логических элементах.

С 1943 г. по 1946 г. в университете г. Пенсильвания (США) была построена первая полностью электронная цифровая ЭВМ, получившая название ENIAK. Главной целью при разработке этой машины было составление числовых таблиц для вычисления траектории полета снарядов и ракет. Машина весила 30 т., занимала площадь 200 кв.м., содержала 18 тыс. ламп. В ее работе использовалась десятичная система счисления. Команды по программе вводились вручную; после введения программы порядок выполнения мог быть изменен только после выполнения всей программы. Каждая новая программа требовала новой комбинации сигналов, путем установки переключателей и коммуникации разъемов. В результате на создание и выполнение даже самой простой программы требовалось очень много времени.

Сложности в программировании на ENIAK натолкнули Д. фон Неймана (1903 — 1957 гг.), бывшего консультантом проекта, на разработку новых принципов построения архитектуры ЭВМ.

Принцип I — произвольный доступ к основной памяти, состоящей из дискретных элементов — ячеек, каждая из которых может содержать набор символов, называемых словом. Время доступа (чтения или записи) не зависит от адреса ячейки.

Принцип II — хранение программы. Информация, хранимая в основной памяти, не имеет признаков принадлежности к определенному типу (программа или данные). Поэтому процессор не различает, что он обрабатывает в данный момент времени.

Указанные и другие принципы были реализованы в новой ЭВМ EDVAC, где применялась двоичная арифметика, основная память была способна хранить 1024 44-разрядных слова. Эта ЭВМ была введена в эксплуатацию в 1951 г.

Первой отечественной ЭВМ была МЭСМ (малая электронная счетная машина), выпущенная в 1951 г. под руководством С.А. Лебедева. Её номинальное быстродействие — 50 операций в секунду.

В 1959 г. были изобретены интегральные схемы (чипы), в которых все электронные компоненты вместе с проводниками помещались внутри кремниевой пластинки. Применение чипов в компьютерах позволило сократить пути прохождения тока при переключениях, скорость вычислений повысилась в десятки раз. Существенно уменьшились габариты машин.

К началу 1960-х гг. компьютеры нашли широкое применение для обработки большого количества статистических данных, производства научных расчётов, решения оборонных задач, создания автоматизированных систем управления. Высокая цена, сложность и дороговизна обслуживания больших вычислительных машин ограничивали их использование во многих сферах.

В 1970 г. сотрудник компании INTEL Э. Хофф создал первый микропроцессор, разместив несколько интегральных схем на одном кремниевом кристалле. Это революционное изобретение кардинально перевернуло представление о компьютерах как о громоздких, тяжеловесных монстрах. С микропроцессором появляются микрокомпьютеры, способные разместиться на письменном столе пользователя.

В середине 1970-х гг. начинают предприниматься попытки создания персонального компьютера — вычислительной машины, предназначенной для частного пользователя. Во второй половине 1970-х гг. появляются наиболее удачные образцы микрокомпьютеров.

ЭВМ, созданные в первой половине XX в., имели две важные особенности, которыми не обладали ранее созданные машины: возможность программирования и способность хранения информации. За последние десятилетия XX в. компьютеры проделали значительный эволюционный путь, многократно увеличили свое быстродействие и объемы перерабатываемой информации. В конце XX в. человечество вступило в стадию формирования глобальной информационной сети, способной объединить возможности компьютерных систем. Появились новые понятия.

Информационная технология (ИТ) — процесс, использующий совокупность средств и методов сбора, обработки и передачи данных (первичной информации) для получения информации нового качества о состоянии объекта, процесса или явления.

Телекоммуникации — дистанционная передача данных на базе компьютерных сетей и современных технических средств связи.

Информационный процесс — процесс, в результате которого осуществляется приём, передача (обмен), преобразование и использование информации.

Термин информация (information) возник от латинского слова informatio (разъяснение, изложение) и до середины XX в. означал сведения, передаваемые между людьми. В статье второй Федерального закона «Об информации» информация — это сведения (сообщения) о лицах, предметах, фактах, явлениях и процессах, независимо от формы их представления.

Под информацией понимают любую совокупность сигналов, воздействий или сведений, которые система или объект воспринимает извне (входная информация), выдает в окружающую среду (выходная информация) или хранит в себе (внутренняя информация).

1.4. Разделы информатики

Теоретическая информатика — раздел информатики, пограничный с математикой. Он опирается на математическую логику и включает такие разделы, как теория алгоритмов, системный анализ, теория информации и теория кодирования, исследование операций и другие.

Вычислительная техника — раздел, в котором разрабатываются общие принципы построения вычислительных систем. Речь идет не о технических деталях и электронных схемах (это лежит за пределами информатики как таковой), а о принципиальных решениях на уровне так называемой архитектуры компьютерных систем, определяющей состав, назначение, функциональные возможности и принципы взаимодействия устройств. Примеры принципиальных, ставших классическими решений в области вычислительной техники — Неймановская архитектура компьютеров первых поколений, шинная архитектура ЭВМ старших поколений, архитектура параллельной (многопроцессорной) обработки информации.

Программирование — деятельность, связанная с разработкой систем программного обеспечения. Основными разделами современного программирования являются создание системного программного обеспечения и создание прикладного программного обеспечения. Среди системного — разработка новых языков программирования и компиляторов к ним, разработка интерфейсных систем. Среди прикладного программного обеспечения общего назначения самые популярные — системы обработки текстов, электронные процессоры, базы данных. В каждой области предметных приложений информатики существует множество специализированных прикладных программ более узкого назначения.

Информационные системы — раздел, занимающийся решением вопросов об анализе потоков информации в различных сложных системах, их оптимизации, структурировании, принципах хранения и поиска. Информационно-справочные системы, информационно-поисковые системы, гигантские современные глобальные системы хранения и поиска информации, включая широко известную сеть Internet, выходят в последнее время на передний план и привлекают внимание все большего круга пользователей. Без теоретического обоснования, поиска принципиальных решений в океане информации можно просто захлебнуться. Известным примером такого решения на глобальном уровне стала гипертекстовая поисковая система www.

Искусственный интеллект — область информатики, в которой решают сложнейшие проблемы, стоящие на пересечении с психологией, физиологией, лингвистикой, другими науками: как научить компьютер мыслить так же как человек? Основные направления разработок в данной сфере: моделирование рассуждений, компьютерная лингвистика, машинный перевод, создание экспертных систем, распознавание образов и другие. От успехов работ в области искусственного интеллекта зависит, в частности, решение такой важнейшей прикладной проблемы, как создание интеллектуальных интерфейсных систем взаимодействия человека с компьютером, благодаря которым указанное взаимодействие будет походить на межчеловеческое и станет более эффективным.

1.5. Поколения ЭВМ

Историю развития вычислительных машин принято рассматривать по поколениям.

Первое поколение (1946 — 1960 гг.) — это время становления архитектуры машин фон-неймановского типа, построенных на электронных лампах с быстродействием 10—20 тыс. арифметических операций в секунду. ЭВМ отличались большими габаритами, высоким потреблением энергии, малым быстродействием, низкой надежностью, программированием в кодах. В Советском Союзе к первому поколению относится первая отечественная вычислительная машина МЭСМ (Малая Электронная Счетная Машина), созданная в 1951 г. в г. Киеве под руководством академика С.А. Лебедева, серийные машины Минск-1, Стрела, БЭСМ (Большая Электронная Счетная Машина), Урал-1, Урал-4 и др.

ЭВМ первого поколения были громоздкими, ненадежными и нуждались во вспомогательных холодильных установках. Использовались они для решения вычислительных задач научного характера. Процесс программирования

на таких машинах требовал хорошего знания устройства машины и её реакций на те или иные ситуации.

Второе поколение (1960 — 1964 гг.) — машины, построенные на транзисторах, с быстродействием до сотен тысяч операций в секунду. Появилась возможность использования библиотек стандартных программ, а процесс программирования стал более легким. Первой полупроводниковой машиной была модель RCA-501, появившаяся в 1959 г. В Советском Союзе к этому поколению относятся машины Минск-2, Минск-22, Минск-32, БЭСМ-2, БЭСМ-4, БЭСМ-6, быстродействие которых составляло миллион операций в секунду.

Третье поколение (1964 — 1970 гг.) характеризуется тем, что вместо транзисторов стали использоваться интегральные схемы (ИС) и полупроводниковая память. Для повышения эффективности использования возникла необходимость в системной программе, управляющей устройствами ЭВМ. Так была создана операционная система.

Вычислительные машины третьего поколения, как правило, образуют серии (семейства) машин, совместимых программно. Такая серия состоит из ЭВМ, производительность и объем памяти которых возрастают от одной машины серии к другой. Но программа, отлаженная на одной из машин серии, может быть сразу запущена на других машинах той же серии (на машинах большей мощности).

Первым семейством машин третьего поколения была выпущенная в 1965 г. IBM/360, имеющая свыше семи моделей.

В Советском Союзе такую серию составляли машины семейства ЕС ЭВМ (Единая Система ЭВМ), совместимых с IBM/360.

Четвертое поколение (1970 — 1980-е гг.) — машины, построенные на больших интегральных схемах (БИС), содержащих до нескольких десятков тысяч элементов на кристалле. ЭВМ четвертого поколения выполняют десятки и сотни миллионов операций в секунду. Появляются микропроцессоры, способные обрабатывать числа длинной в 16 и 32 разряда, статическая память емкостью 256 Кб. и динамическая память емкостью в 1 Мб.

ЭВМ по своим характеристикам так разнообразны, что их начинают классифицировать на сверхбольшие ЭВМ (B-7700 — фирма Барроуз, Эльбрус — СССР, Иллиак-IV — Иллинойский университет), большие (универсальные), мини-ЭВМ и микро-ЭВМ.

Пятое поколение (1980 г. — по настоящее время). В 1979 г. японскими специалистами, объединившими свои усилия под эгидой научно-исследовательского центра по обработке информации — JIPDEC, была впервые поставлена задача разработки принципиально новых компьютеров. В 1981 г. JIPDEC опубликовал предварительный отчет, содержащий детальный

многостадийный план развертывания научно-исследовательских и опытно-конструкторских работ с целью создания к 1991 г. прототипа ЭВМ нового поколения. Отчет лег в основу японской национальной программы создания ЭВМ пятого поколения. Отличительными чертами ЭВМ этого поколения являются:

- новая технология производства, не на кремнии, а на базе других материалов;
- отказ от архитектуры фон Неймана, переход к новым архитектурам (например, на архитектуру потока данных). И, как следствие, превращение ЭВМ в многопроцессорную систему (матричный процессор, процессор глобальных связей, процессор локальных связей, машины базы данных и т.п.);
- новые способы ввода-вывода информации, удобные для пользователя (например, распознавание речи и образов, синтез речи, обработка сообщений на естественном языке);
- искусственный интеллект, т.е. автоматизация процессов решения задач, получения выводов, манипулирования знаниями.

Переход к ЭВМ пятого поколения означает резкий рост «интеллектуальных» способностей компьютера, в результате чего машина сможет непосредственно «понимать» задачу, поставленную перед ней человеком. Следовательно, отпадает необходимость в составлении программы как средства «общения» с ЭВМ при решении той или иной задачи.

Предполагается, что компьютеры пятого поколения будут вести диалог с непрофессиональными пользователями на естественном языке, в том числе в речевой форме или путем обмена графической информацией - с помощью чертежей, схем, графиков, рисунков. В состав ЭВМ пятого поколения также должна войти система решения задач и логического мышления, обеспечивающая способность машины к самообучению, ассоциативной обработке информации и получению логических выводов.

1.6. Персональные компьютеры

Официальная история персональных компьютеров (ПК) берет начало с августа 1981 г., когда фирма IBM (International Business Machines Corporation) известила о создании «Персонального компьютера». К весне 1982 г. он продавался в больших количествах, причем спрос намного превышал предложение.

С самого начала появления ПК стала очевидна необходимость модели, которую можно носить с собой в небольшом чемодане, что привело к появлению фирмы Compaq Computer. Первым пополнением семейства ПК стал

компьютер, известный под названием Compaq. О его создании было официально объявлено осенью 1982 г., спустя чуть больше года после выпуска оригинального ПК.

Весной 1983 г. свой вклад в пополнение семейства ПК внесла фирма IBM. Появилась модель XT, которая добавила к ПК жесткий диск памяти большого объема. Фирма Compaq Computer ответила тем же, создав аналогичную машину в переносном варианте осенью 1983 г. Модель была названа Compaq Plus.

Летом 1984 г. появились две высокопроизводительные модели ПК. Одной из них была модель Compaq Desk Pro, первый член семейства ПК, превосходивший исходную модель по производительности вычислений. Вскоре после этого IBM выпустила компьютер AT, скорость вычислений которого намного превышала соответствующие параметры моделей исходных компьютеров и XT, и даже нового Desk Pro. Тактовая частота — 4 МГц.

Для ускорения появления новых моделей фирма IBM пользуется микропроцессорами, которые производит фирма Intel. Машина PC AT была построена на базе процессора Intel 8088. Следующие модели — на базе процессоров 80286, 80386, 80486 и 80586. В их названиях используются последние три цифры номера процессора.

1.7. Классификация компьютеров

Здесь рассматривается классификация компьютеров по обобщенному параметру, где в разной степени учтено несколько характерных признаков:

- назначение и роль в системе обработки информации;
- условия взаимодействия человека и компьютера;
- габариты;
- ресурсные возможности.

В соответствии с вышесформулированными признаками и тенденциями развития компьютерной техники предлагается следующая классификация компьютеров:

- класс больших компьютеров;
- класс суперкомпьютеров;
- класс персональных компьютеров;
- класс малых компьютеров.

При характеристике каждого класса необходимо делать сравнение отдельных моделей по таким основным техническим параметрам, как быстродействие (производительность) и объемы оперативной памяти.

Под быстродействием понимается число коротких операций, выполняемых компьютером за одну секунду. Оценка быстродействия (производительности) всегда приблизительна, особенно если учесть, что теперь широко используются многопроцессорные компьютеры.

Другая важная характеристика любого компьютера — объем (емкость) оперативной памяти, иными словами, максимальное количество хранимой в ней информации.

Помимо указанных характеристик, возможности компьютера характеризуются рядом параметров:

- разрядность и формы представления чисел;
- емкость внешней памяти;
- характеристики внешних устройств хранения, обмена и ввода-вывода информации;
- пропускная способность устройств связи узлов ЭВМ между собой;
- способность ЭВМ работать одновременно с несколькими пользователями и выполнять одновременно несколько программ;
- типы операционных систем, используемых в машине;
- программная совместимость с другими типами ЭВМ, т.е. способность выполнять программы, написанные для других типов ЭВМ;
- возможность подключения к каналам связи и к вычислительной сети;
- надежность и пр.

1.7.1. Класс больших компьютеров

История развития компьютерной техники началась с создания больших ЭВМ, элементная база которых прошла длинный путь развития. Класс больших компьютеров обладает широкими возможностями по всем техническим параметрам. Они очень дорогие, могут занимать большую площадь, но являются особенно надежными. В этом классе пока затруднительно установить четкую группировку на подклассы, так как с проникновением компьютерных сетей во все сферы деятельности человека происходит смещение акцентов по приоритетам и назначению больших компьютеров. В настоящее время в классе больших компьютеров выделяют две группы компьютеров:

- серверы;
- суперкомпьютеры.

Сервер — мощный компьютер в вычислительных сетях, который обеспечивает обслуживание подключенных к нему компьютеров и выход в другие сети.

В зависимости от назначения определяют различные типы серверов.

Сервер приложений обрабатывает запросы от всех станций вычислительной сети и предоставляет им доступ к общим системным ресурсам (базам данных, библиотекам программ, принтерам, факсам и др.).

Файл-сервер применяется для работы с базами данных и использования файлов с информацией.

Архивационный сервер используется для резервного копирования информации в крупных многосервисных сетях. Он может использовать накопители на магнитной ленте (стриммеры) со сменными картриджами емкостью до 5 Гб. Обычно выполняет ежедневное автоматическое архивирование информации от подключенных серверов и рабочих станций.

Факс-сервер служит для организации эффективной многоадресной факсимильной связи. Он имеет несколько факсмодемных плат, специальную защиту информации от несанкционированного доступа в процессе передачи, систему хранения электронных факсов.

Почтовый сервер выполняет те же функции, что и факс-сервер, но для организации электронной почты.

Сервер печати позволяет более эффективно использовать системные принтеры.

Сервер телеконференций имеет программу обслуживания пользователей телеконференциями и новостями. Он может иметь также систему автоматической обработки видеоизображений и др.

Любой компьютер, если установить на нем соответствующее сетевое программное обеспечение, способен стать сервером. Кроме того, один компьютер одновременно может выполнять несколько функций: быть, к примеру, почтовым сервером, сервером новостей или сервером приложений и т.д.

1.7.2. Суперкомпьютеры

Первые суперкомпьютеры (модели Cray) стала выпускать компания Cray Research в середине 70-х гг. Их быстродействие исчислялось десятками и сотнями миллионов операций в секунду. Идея построения суперкомпьютера базировалась на стремлении уменьшить расстояние между всеми электронными компонентами, а также организовать параллельную работу не на одном процессоре, а сразу на нескольких.

В суперкомпьютерах используется мультипроцессорный (многопроцессорный) принцип обработки информации.

Основная идея создания мультипроцессорной обработки — расделение решаемой задачи на несколько параллельных подзадач или частей. Каждая часть решается на своем процессоре. За счет такого разделения существенно увеличивается их производительность. Параллельное вычисление особенно эффективно в задачах, где применяется большое количество операций с матрицами.

1.7.3. Персональные компьютеры

Персональными называют компьютеры, предназначенные для индивидуального пользования. Для них характерны:

- малые габариты и масса, позволяющие устанавливать их на рабочем столе пользователя;
- дружественный человеко-машинный интерфейс;
- ориентация программного обеспечения на массового пользователя и наличие большого числа готовых программных средств для различных видов профессиональной деятельности;
- возможность использования языков высокого уровня (Бейсик, Паскаль, Пролог, C++ и др.);
- наличие периферийных устройств, обеспечивающих ввод-вывод информации и ее хранение.

Совокупность указанных характеристик делает компьютер доступным непрофессиональному пользователю, определяя его персональный характер.

Персональные компьютеры используются сейчас повсеместно. За время, прошедшее с момента создания персонального компьютера, уже сменилось несколько их поколений: 8-битные, 16-битные, 32-битные, 64-битные. Многократно усовершенствовались внешние устройства, все операциональное окружение, включая системы связи, сети, системы программирования, прикладное программное обеспечение и т.п. Персональный компьютер занял нишу «персонального усилителя интеллекта» множества людей, в ряде случаев стал ядром автоматизированного рабочего места (в цехе, в банке, в билетной кассе, в школьном классе и т.д.). Их основное назначение — выполнение рутинной работы: поиск информации, составление типовых форм документации, фиксация результатов исследования, подготовка текстов разного рода от простейших документов до издательской верстки и т.д.

1.7.4. Малые компьютеры

Малые компьютеры появились в 70-х гг. Их создание было связано, с одной стороны, с тем, что для решения многих задач не требовались мощности больших ЭВМ, а с другой, необходимо было использовать возможности компьютеров для управления технологическими процессами. Такая потребность определялась повсеместным внедрением автоматизированных систем управления, где требовалось устройство, оперативно и надежно перерабатывающее информацию.

Среди малых компьютеров выделяют подкласс портативных компьютеров, которые в настоящее являются время самыми престижными в мире.

Название этого класса компьютеров происходит от латинского слова «porto» (ношу) и означает, что они легко переносимы. Они часто оформлены в виде чемоданчиков или папок и, в свою очередь, делятся на несколько типов. Самым распространенным и привычным является ноутбук — блокнотный персональный компьютер.

Ноутбук заменяет настольный персональный компьютер, поэтому он имеет высокое быстродействие, большой, четкий и яркий экран дисплея, малое энергопотребление, т.к. его источником является батарея. Если же портативный компьютер предназначен только для работы в дороге и является дополнением к основному настольному, то он должен иметь хороший модем, более длительное время работы от батареи, значительно меньший вес. Ни в какой другой компьютерной технике не достигается больше компромиссов, чем в ноутбуках!

Современные блокнотные компьютеры производят очень хорошее впечатление. Они снабжены жесткими дисками большой емкости, имеют отсеки для подсоединения CD-DVD приводов, приемлемый вес.

Существует множество типов портативных компьютеров. Каждый из них имеет свои особенности. Например, в качестве манипулятора используют не мышь, а другое устройство указания. Иногда эти функции выполняет сенсорный экран, где команда вводится в соответствии с местом прикосновения.

Другой тип — очень маленькие портативные компьютеры (всего до 200 г), органайзеры — электронные записные книжки. Они фактически являются лишь электронными устройствами, т.к. не имеют собственного процессора, но благодаря своим функциональным возможностям относятся к данному классу.

Появление и совершенствование портативных компьютеров обеспечивают людям более эффективное и своевременное использование информации.

Глава 2. Информационные системы

Информатика — наука и техника, связанные с машинной обработкой, хранением и передачей информации. Поэтому центральное понятие в информатике — информация. Точное выяснение термина «информация» существенно необходимо для понимания систем обработки информации. Вообще, понятие «информация» используется в разных смыслах. Обычно под информацией понимаются высказывания относительно событий, взаимосвязей или состояний реального мира. Например, «Волга впадает в Каспийское море».

В информатике информацией называется абстрактное содержание какого-либо высказывания, описания, указания, оператора, сообщения. Необходимо отличать информацию, т.е. абстрактное содержание, от изображения информации. Например, математическое понятие «целое число» является абстрактным понятием. Мы изображаем целые числа в виде последовательности цифр из множества 0, 1,..., 9. Можно изобразить целое число римскими цифрами или палочками, насечками. Все это изображения, т.е. внешние формы информации - представления.

Определение 2.1. Информацией называют абстрактное содержание («содержательное значение», «семантика») какого-либо высказывания, описания, указания, сообщения или известия.

Определение 2.2. Внешнюю форму изображения информации называют представлением.

Для машинной обработки информации существует много форм представления информации от условных знаков (сигналов) и произносимых слов до рисунков или последовательностей символов. Все представления информации не будут иметь смысла, если не будет известно о значениях представлений, как, например, при рассмотрении древних надписей и рисунков. Они являются внешней формой. Однако нам неизвестны значения этих изображений, поэтому нам недоступен их смысл, т.е. абстрактное значение информации. Важно установить способ для выявления значения представления.

Определение 2.3. Переход от представления к абстрактной информации, т.е. к значению представления, называют интерпретацией.

Многие формы представления информации допускают различное толкование, например «красный». Если речь идет о дорожных знаках, это признак запрещения, если о тексте — последовательность символов «к», «р» и т.д. С другой стороны, одно и то же абстрактное содержание может быть представлено различными способами, как в приведенном выше примере с представлением чисел.

Только в том случае, когда существуют единые, согласованные интерпретации, возможен обмен информацией. Примером может служить использование дорожных знаков. Поскольку значение каждого знака точно известно, водители и пешеходы понимают их смысл, что позволяет организовывать движение. И наоборот, несогласованность интерпретаций приводит к непониманию передаваемой информации. Типичным примером является использование сленга, когда одни и те же слова приобретают различный смысл и общающиеся люди не в состоянии понять друг друга, хотя используют одни и те же представления информации.

Выявление подходящих систем представления (языков) для определенных классов информации является одной из задач информатики.

Итак, мы различаем в связи с информацией:

- ее представление или изображение (внешняя форма);
- ее значение (собственно «абстрактная» информация);
- ее отношение к реальному миру (связь абстрактной информации с действительностью).

Последний вопрос, определяющий, является ли информация истинной, не рассматривается и не решается в информатике, т.к. ответ на него зависит от субъективного восприятия.

Информатика включает в себя науку о машинной обработке информации и поэтому в ней рассматриваются вопросы:

- схематизированного представления (изображения) информации: структуры объектов и данных, а также их взаимосвязи;
- правил и предписаний для обработки информации (алгоритмы, вычислительные предписания) и их представления, включая описание протекания работы (процессы).

Оба вопроса тесно связаны между собой. Программа, например, в качестве своей внешней формы имеет текстовую или графическую структуру, которая, в свою очередь, является объектом для информационной обработки. Но программа также представляет собой предписание для обработки информации. При ее выполнении в компьютере протекает процесс действий, который преобразует некоторые исходные данные в определенный результат.

В информатике рассматриваются информационные системы вида (A, R, I), где R — множество представлений с интерпретацией I во множестве A элементов (информаций). Таким образом интерпретации соответствует отображение $I : R \to A$ (интерпретация I ставит в соответствие данному представлению r некоторое абстрактное информационное содержание $I[r]$). R также называют системой представления, а A — семантической моделью.

Пример (система представления для натуральных чисел). Пусть N — множество натуральных чисел (включая число «нуль»), представляемых числом штрихов, т.е.: $\varepsilon, |, ||, |||, \ldots$, где через ε обозначена пустая последовательность. Интерпретацией I будет отображение десятичного представления в последовательность штрихов.

$I : \{0, 1, \ldots, 9\}^+ \to N$

$I[0] = \varepsilon, I[1] = |, I[2] = ||, \ldots$

Приведенный пример демонстрирует фундаментальную проблему информационной обработки: информация в ее абстрактном виде не может быть записана непосредственно, она всегда может быть только изображена.

Часто в какой-нибудь системе представления имеется много различных изображений одной и той же информации. Эти изображения называются эквивалентными. Точнее говоря, в информационной системе (A, R, I) два изображения $r_1, r_2 \in R$ называются семантически эквивалентными, если они несут одинаковую информацию: $I[r_1] = I[r_2]$.

Как правило, мы больше заинтересованы в информации, чем в ее представлении. Поэтому часто бывает удобным обходиться с представлением так, как если бы оно было непосредственно информацией. Наглядным примером этому служит классическая математика.

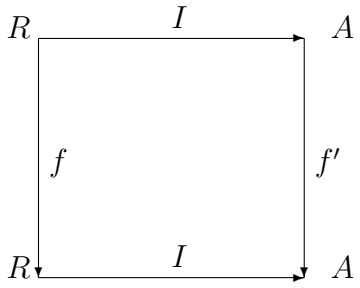
Для корректной обработки информации необходимо, чтобы в применяемой информационной системе была представима любая информация.

Пусть (A, R, I) — информационная система. Если I сюръективно, т.е. для каждой информации $a \in A$ существует представление $r \in R$ с $I[r] = a$, то каждая информация имеет представление. Обычно вызывают интерес информационные системы, обладающие этим свойством.

Отображение на множестве представлений при определенных предположениях индуцирует и отображение на множестве информаций. Пусть $f : R \to R$ отображение на множестве представлений R. Если $\forall x, y \in R$ справедливо $(I[x] = I[y]) \to (I[f(x)] = I[f(y)])$ и I сюръективно, то вследствие интерпретации $I : R \to A$ однозначным образом устанавливается отображение на множестве информаций $f' : A \to A$ по следующему правилу: $f'(a) = b$, если для некоторого $r \in R$ справедливо $I[r] = a$ и $I[f(r)] = b$. Связь между f, f' и интерпретацией можно пояснить коммутирующей диаграммой, изображенной на рисунке.

Также справедливо $I[f(r)] = f'(I[r])$. f' называют абстракцией f.

Итак информация представляется не непосредственно, а лишь изображается каким-либо образом. Однако не все эквивалентные изображения определенной информации одинаково легко интерпретируются или обрабатываются.



Пример: $\sum_{i=1}^{\infty} 1/2^i$, 0.(9), 0!, 1.

Все эти термы имеют значение «один».

Однако они различаются с точки зрения простоты, чтения, интерпретации и понимания. Простота конкретного изображения информации имеет важное значение. Часто подмножество S (изображений простой внешней формы) множества изображений R выделяется как множество нормальных форм. Тогда S называют системой нормальных форм (СНФ). Если в такой системе для каждого изображения существует по меньшей мере одна семантически эквивалентная НФ, то СНФ называется полной.

Пусть $S \subseteq R$ — СНФ. Если любое множество изображений с одинаковой интерпретацией имеет единственную НФ, т.е. отображение I_S — инъективно, то СНФ называется однозначной.

Так как на множестве однозначных НФ интерпретация есть инъективное отображение, то соответствующую информацию по мере надобности можно отождествить с ее НФ.

Глава 3. Алгоритмы

3.1. Интуитивное понятие алгоритма

Алгоритмами являются способы решения, описанные с помощью предписаний по обработке, которые удовлетворяют определенным требованиям.

Определение 3.1. Алгоритм есть способ с точным (т.е. выраженным в точно определенном языке) конечным описанием применения практически выполнимых элементарных шагов переработки информации.

Это интуитивное понятие алгоритма.

Алгоритмы типичным образом решают не только частные задачи, но и классы задач. Подлежащие решению частные задачи, выделяемые по мере необходимости из рассматриваемого класса, определяются с помощью параметров. Параметры играют роль исходных данных для алгоритма. Алгоритмы, как правило, по этим исходным данным доставляют результаты, которые в случае задач информационной обработки могут быть информацией (вернее, представлением информации) или последовательностью указаний (управляющих сигналов), по которым осуществляются определенные преобразования.

Независимо от формы описания для алгоритмов важно различать следующие аспекты:

- постановку задачи, которая должна быть решена с помощью алгоритма;
- специфичный способ, каким решается задача, при этом для алгоритма различают элементарные шаги обработки, которые имеются в распоряжении, и описание выбора отдельных подлежащих выполнению шагов.

3.2. Свойства алгоритма

Для алгоритмов справедливы некоторые общие закономерности, называемые свойствами.

Массовость. Алгоритм разрабатывается для целого класса задач.

Дискретность. Алгоритм представляется в виде конечной последовательности шагов.

Конечность. Алгоритм должен закончить работу за конечное число ш́агов.

Определенность. Каждый шаг должен быть точно и недвусмысленно определен. Перед началом каждого шага должны быть определены все необходимые ему данные.

Эффективность. Если задачу можно решить несколькими способами следует выбрать тот, который потребует меньше ресурсов.

Пример (алгоритм Евклида нахождения наибольшего общего делителя). На входе алгоритма два натуральных числа.

1. Если числа равны, выдать любое из них в качестве ответа и закончить алгоритм.
2. Заменить большее число на разность большего и меньшего.
3. Перейти к шагу 1.

Формальное понятие алгоритмов тесно связано с понятиями рекурсивных функций, машин Тьюринга, нормальных алгоритмов Маркова или систем текстовых замен (СТЗ).

Определение 3.2. Алгоритм называется *терминистическим*, если он завершается за конечное число шагов, *детерминистическим*, если нет свободы в выборе очередного шага алгоритма, *детерминированным*, если независимо от последовательности выполняемых шагов результат определяется однозначно.

3.3. Алгоритм как рекурсивная функция

Существование или не существование алгоритма может быть установлено, если найти такой математический объект, который будет существовать в точности тогда, когда и алгоритм. Таким математическим объектом может быть рекурсивная функция (РФ). Функция определяется однозначно, если известен закон, по которому каждому набору $x_1, \ldots, x_n$ ставится в соответствие одно значение функции y. Закон может быть произвольным, в том числе это может быть алгоритм вычисления значения функции. Тогда функцию называют вычислимой, а алгоритм, по которому вычисляется функция, называется алгоритмом, сопутствующим вычислимой функции. Рекурсивные функции являются частным классом вычислимых функций.

РФ строится здесь на множестве целых неотрицательных чисел следующим образом: задаются 3 базовые РФ, для которых сопутствующие алгоритмы одношаговые. Затем используются 3 приема, называемые операторами подстановки, рекурсии и минимизации, с помощью которых на основе базовых функций строятся более сложные РФ. По существу приведенные операторы — алгоритмы, комбинируя которые получают более сложные алгоритмы.

Перечислим простейшие базовые функции.

1. Функция произвольного количества аргументов тождественно равная нулю.

   Знак функции φ_n, где n — количество аргументов.

   Сопутствующий алгоритм: если знаком функции является φ_n, то значение функции равно нулю.

   Например: $\varphi_1(3) = 0$, $\varphi_3(4, 56, 78) = 0$.
2. Тождественная функция.

   Знак функции $\psi_{n,i}$, $0 < i \leq n$, где n — количество аргументов, i — номер аргумента.

   Сопутствующий алгоритм: если знак функции $\psi_{n,i}$, то значением функции является значение i-го аргумента, считая слева направо.

   Например: $\psi_{3,2}(3, 22, 54) = 22$.
3. Функция получения последователя. Функция одного независимого аргумента.

   Знак функции λ.

   Сопутствующий алгоритм: если знак функции λ, то значение функции — число, следующее за значением аргумента.

   Например: $\lambda(5) = 6$ или 5'=6.

Перечислим три приема построения сложных РФ.

1. Оператор подстановки $F(f_1, \ldots, f_n)$. Сопутствующий алгоритм: вычисляются значения функций $f_1, \ldots, f_n$ и используются как аргументы при вычислении F.

Например $\tau(y) = \lambda(\lambda(y)) = \lambda(y') = y''$.

2. Оператор рекурсии R. $f ::= R[f_1, f_2, x(y)]$.

Здесь f — функция n аргументов, f_1 — $(n-1)$ аргумента, f_2 — $(n+1)$ аргумента, причем $n-1$ аргументов функций совпадают, а 2 следующих аргумента называются дополнительными. Один из них, x — называется главным дополнительным аргументом (ГДА). Он войдет в определяющую функцию. Другой, y — вспомогательный дополнительный аргумент (ВДА), использующийся при построении новой функции.

Сопутствующий алгоритм: оператор рекурсии строит новую функцию по двум условиям:

$f(0) = f_1$

$f(i') = f_2(i, f(i))$.

Значением искомой функции при нулевом значении ГДА считать значение функции f_1, а значением новой функции для каждого последующего значения ГДА считать значение функции f_2 для предыдущего значения ГДА и для значения ВДА, совпадающего со значением искомой функции на предыдущем шаге.

Например:

$PR(x) ::= R[\varphi_0, \psi_{2,1}(x, y), x(y)]$;

$PR(0) = \varphi_0 = 0$;

$PR(1) = \psi_{2,1}(0, PR(0)) = 0$;

$PR(2) = \psi_{2,1}(1, PR(1)) = 1$;

$\ldots$

$PR(x) = x - 1$.

3. Оператор минимизации или построение по первому нулю.

$f ::= \mu[f_1, (x)]$ или $f(x_1, \ldots, x_n) = \mu[f_1(x_1, \ldots, x_n, y), (y)]$ строит новую функцию f с помощью функции f_1, имеющей $n + 1$ аргументов, и дополнительного исчезающего аргумента. Сопутствующий алгоритм: придавать последнему аргументу значения, начиная с нуля, вычисляя при этом значение функции f_1. Как только значение функции f_1 станет равным нулю, значение дополнительного аргумента принимаем за значение искомой функции для тех главных аргументов, для которых вычислялось значение функции f_1.

Например:

$r(x) ::= \mu[\psi_{2,1}(x, y), (y)]$;

$r(0) = 0$;

$r(i) = \psi_{2,1}(i, y)$ не определено для $i <> 0$.

Все базовые функции и функции, построенные без оператора минимизации, определены для всех значений дополнительных аргументов. Функции, построенные с помощью оператора минимизации, могут быть определены не для всех значений исходных данных. Большинство известных математических функций — рекурсивные.

Например:

Функция $y = x + 1$ совпадает с базовой.

Построим функцию $w = x + y$.

Получим функцию $f^*(x, y, z)$ подстановкой $(z + 1)$ вместо z в функцию $\psi_{3,3}(x, y, z)$, т.е. $f^*(x, y, z) = \psi_{3,3}(x, y, \lambda(z)) = z + 1$.

Затем воспользуемся следующим:

$S(x, y) ::= R[\psi_{1,1}(x), f^*(x, y, z), y(z)]$;

$S(x, 0) = \psi_{1,1}(x) = x$;

$S(x, 1) = f^*(x, 0, S(x, 0)) = S(x, 0) + 1 = x + 1$;

$\ldots$

$S(x, y) = x + y$.

Класс вычислимых функций исчерпывается классом РФ. Каков бы ни был алгоритм, перерабатывающий последовательность целых неотрицательных чисел в целые неотрицательные числа, найдется сопутствующий некоторой РФ алгоритм, эквивалентный данному и наоборот. Если нельзя построить РФ, то нельзя разработать алгоритм решения задачи.

3.4. Формальное описание алгоритма через замену текстов

3.4.1. Текстовые замены

Для точного описания алгоритма (которое допускает машинную обработку и выполнение) требуется формальный язык (подмножество из V^* с заданным набором знаков V) для записи алгоритмов и точное определение понятия эффективности (выполнимости) элементарных шагов переработки. В простейшем случае алгоритмы в качестве входа и выхода используют слова над некоторым набором знаков. Поскольку в виде слов может быть представлена самая различная информация, можно считать, что алгоритмы всегда оперируют со словами.

Одной из простейших концепций элементарных шагов переработки последовательностей знаков является замена определенных подслов (образцов) в обрабатываемом слове другими словами. Эта концепция ведет к алгоритмам в форме систем текстовых замен на последовательностях знаков.

Пусть V — запас знаков. V^* — множество всех конечных цепочек символов из V. Пустая цепочка, т.е. цепочка, не содержащая ни одного символа, обозначается ε. Пара $(v, w) \in V^* \times V^*$ называется заменой над V. Замена часто записывается в виде $v \to w$.

Конечное множество R замен будем в дальнейшем называть системой текстовых замен (СТЗ) над V. Элементы системы будем называть правилами текстовых замен (ПТЗ). СТЗ служат для представления алгоритмов, отдельные шаги которых состоят в применении правил замен.

Замена $s \to t$ называется *применением правила* $v \to w$, если имеются слова $a, v, w, z \in V^*$ такие, что справедливо $s = \mathrm{a} \cdot v \cdot z$, $t = \mathrm{a} \cdot w \cdot z$.

Слово $s \in V^*$ называется *терминальным* (или *терминалом*) в R, если не существует слова $t \in V^*$ такого, что справедливо следующее: замена $s \to t$ является применением какого-либо правила из R. Таким образом, к терминальному слову s нельзя больше применить никакого правила замены.

3.4.2. Вычисления

Через повторное применение ПТЗ, исходя из начально заданного слова t_0, возникают вычисления. Если $t_0, t_1, \ldots, t_n$ принадлежат V^* и $t_i \to t_{i+1}$ есть применение некоторого правила из R для всех i, $0 \leq i < n$, то последовательность $(t_i) 0 \leq i \leq n$ называют (конечным) вычислением над R для t_0. Часто вычисление записывается следующим образом: $t_0 \to t_1 \to \ldots \to t_n$.

Слово t_0 называется также *входом* для вычисления. Если t_n есть терминал, то вычисление называется *завершающимся (конечным)* с результа-

том t_n. Слово t_n называется также *выходом* для R при входе t_0. Бесконечная последовательность $(t_i)i \in N$ из слов $t_i \in V^*$, для которых $t_i \to t_{i+1}$ есть применение некоторого ПТЗ из R для всех $i \in N$, называется незавершающимся (бесконечным) вычислением.

Примеры (вычисления по СТЗ).

1. Для системы текстовых замен Q над множеством символов $\{L, O\}$, которая состоит из следующих правил:

$LL \to \varepsilon$,

$O \to \varepsilon$,

через последовательность $LOLL \to LO \to L$ задается завершающееся вычисление для входного слова $LOLL$ с результатом L.

2. Для СТЗ над $\{L, O\}$, состоящей из правил:

$O \to OO$,

$O \to L$,

для входа O последовательность вычислений $O \to OO \to OL \to LL$ является завершающимся вычислением с выходом LL, а последовательность $O \to OO \to OOO \to \ldots$ является незавершающимся вычислением.

3.4.3. Алгоритмы текстовых замен

Система текстовых замен R в силу следующего предписания определяет алгоритм текстовых замен (АТЗ), который использует слова над V в качестве входа и выхода. Для входного слова $t \in V^*$ алгоритм работает следующим образом: «Если одно из правил множества R применимо к слову t (т. е. существует слово $s \in V^*$, для которого имеет место: $t \to s$ есть применение правила из R), то примени правило к t и затем примени снова этот же алгоритм к слову s; в противном случае прекрати выполнение алгоритма».

Слово t служит входом для алгоритма. Если (после конечного числа шагов) возникает терминальное слово, т. е. слово, к которому неприменимо никакое правило, то это слово является выходом (результатом вычислений). Если такая ситуация никогда не возникает, то алгоритм не завершается. Алгоритмы, определенные таким образом с помощью СТЗ, всегда являются последовательными. При этом выбор применяемого правила является недетерминистическим.

Часто для АТЗ в качестве входов используют только слова совершенно определенной формы (нормальная форма). Определенные знаки не входят во входные и выходные слова, а используются исключительно как вспомогательные знаки в словах, возникающих в процессе вычислений.

Примеры (алгоритмы текстовых замен).

1. Сложение двух натуральных чисел, представленных в виде количества штрихов.

Натуральное число представляется в виде количества штрихов с ограничительными скобками, т. е. число $n \in N$ представляется словом $< || \ldots | >$, причем вовнутрь скобок входит n штрихов. В этом случае алгоритм состоит из одного единственного правила замены (ε обозначает пустое слово):

$$> + <\rightarrow \varepsilon$$

Для входа $< | \ldots | > + < | \ldots | >$ алгоритм дает сумму штрихов.

2. Умножение двух натуральных чисел (в таком же представлении).

Применяются вспомогательные знаки d, e, m. Алгоритм состоит из следующих правил замен:

1) $| > * <\rightarrow > * < d$
2) $d| \rightarrow |md$
3) $dm \rightarrow md$
4) $d >\rightarrow >$
5) $<> * <\rightarrow < e$
6) $e| \rightarrow e$
7) $em \rightarrow |e$
8) $e >\rightarrow >$

Для входного слова $< | \ldots | > * < | \ldots | >$ с n_1 штрихами в первом операнде и n_2 штрихами во втором операнде алгоритм дает выходное слово $< | \ldots | >$ с $n_1 * n_2$ штрихами.

Последовательности, образованные из вспомогательных знаков, представляют вполне определенные ситуации в вычислениях. Возникающие слова могут трактоваться как усложненные представления чисел. Для входного слова $< || > * < ||| >$ возникает показанный на рис. 3.1 граф возможных вычислении. Все вычисления заканчиваются получением слова $< |||||| >$. Этот алгоритм подстановки недетерминистический, но детерминированный, т. е. дает, несмотря на различные вычисления, всегда один и тот же результат.

Алгоритмы в виде текстовых замен в общем случае являются недетерминистическими и недетерминированными. Для входного слова t существуют, как правило, разные вычисления с различными результатами. При этом для одной и той же задачи могут существовать как завершающиеся, так и незавершающиеся вычисления.

3.4.4. Детерминистические алгоритмы текстовых замен

Часто предпочитают рассматривать детерминистические алгоритмы текстовых замен, т. е. алгоритмы, которые для каждого входного слова однозначно задают вычисления и, тем самым, в случае их завершения порождают вполне

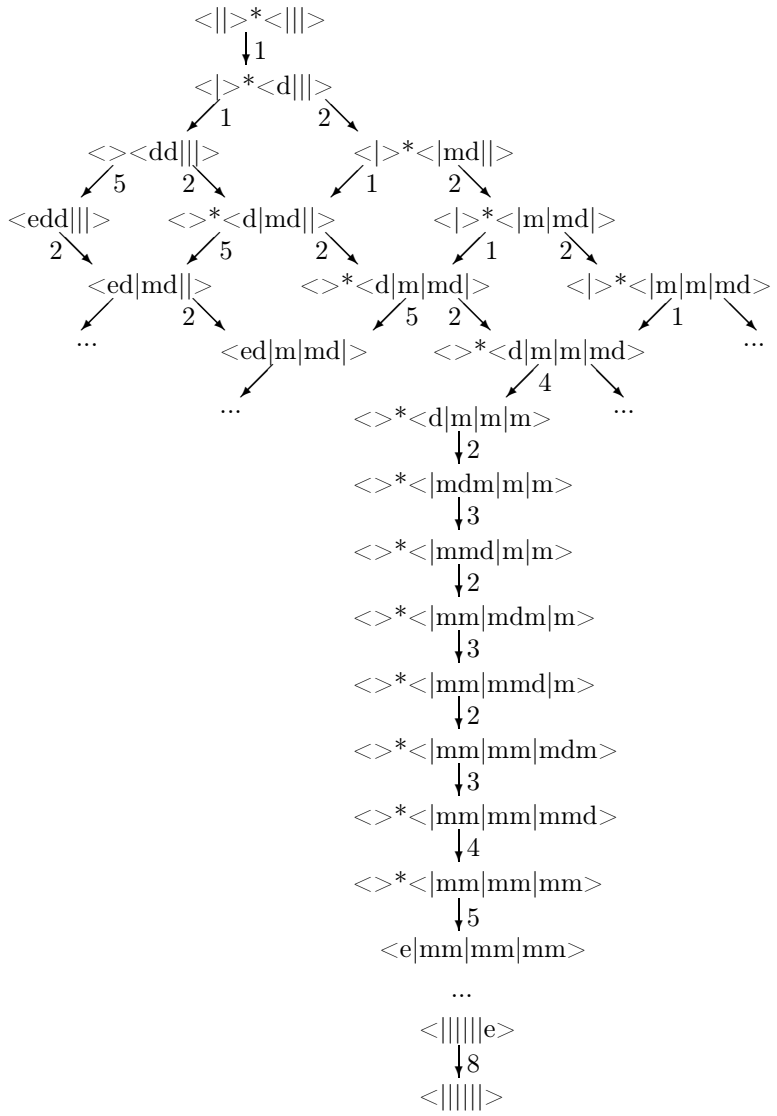


Рис. 3.1

определенный результат. Это может быть обеспечено, например, установлением приоритетов применения правил. Такие приоритеты могут быть заданы просто порядком описаний правил.

Примером детерминистических алгоритмов являются так называемые алгоритмы Маркова, названные по имени русского математика А.А. Маркова. В алгоритмах Маркова правила замен линейно упорядочены (порядок определяется последовательностью описания правил). Тогда применение правил устанавливается следующим образом.

Определение 3.3. (марковская стратегия применения ПТЗ). Если применимо несколько правил, то фактически применяется то из этих правил, которое в описании алгоритма встречается первым. Если правило применимо в нескольких местах обрабатываемого слова, то выбирается самое левое из этих мест.

Таким образом, марковские алгоритмы всегда являются детерминированными и детерминистическими. В частности, справедливо следующее:

- каждое вычисление по марковской стратегии является также общим вычислением в СТЗ;
- для каждого входного слова существует точно одно конечное или же бесконечное марковское вычисление; алгоритмы Маркова являются детерминистическими (а отсюда результат, если он существует, однозначно определен).

Пример (алгоритм Маркова). Пусть задана система текстовых замен R на множестве символов $\{\vee, \neg, true, false, (,)\}$ для редукции булевских термов, которые построены только из символов данного множества, к нормальной форме. Система состоит из следующих правил:

1) $\neg\ \neg \to \varepsilon$
2) $\neg true \to false$
3) $\neg false \to true$
4) $(true) \to true$
5) $(false) \to false$
6) $false \vee \to \varepsilon$
7) $\vee false \to \varepsilon$
8) $true \vee true \to true$

Алгоритм, определенный данной системой, работает по марковской стратегии корректно для замкнутых булевских термов (т. е. замкнутые булевские термы переводятся в семантически эквивалентные однозначные нормальные формы, состоящие из $true$ и $false$). Для терма $\neg true \vee true$ по марковской стратегии получается вычисление:

$$\neg true \vee true \to (\text{правило 2})\ false \vee true \to (\text{правило 6})\ true.$$

В общей недетерминистической стратегии дополнительно получают (с точки зрения постановки задачи корректное) вычисление:

$$\neg true \vee true \to (\text{правило 8})\ \neg true \to (\text{правило 2})\ false.$$

Благодаря марковской стратегии однозначно определяется выбор применяемого правила, что для многих задач упрощает формулирование алгоритма. В определенных случаях может также оказаться полезным введение частичного упорядочения правил (частичный порядок над правилами замен).

3.4.5. Отображения, индуцируемые алгоритмами текстовых замен

Путем сопоставления выходного слова каждому входному слову при конечном вычислении детерминистические алгоритмы вычисляют частичные функции. Функции являются частичными, так как иногда при некоторых исходных данных алгоритмы не завершаются и потому результат вычислений

не определен. Подобное имеет место также и для алгоритмов текстовых замен. Явного использования частичных функций можно избежать путем введения особого символа $\perp$ («дно»), который символизирует отсутствующий «результат» незавершающегося вычисления.

Каждый детерминированный алгоритм R в форме СТЗ на последовательностях символов V^* определяет отображение: $F_R : V^* \to V^* \bigcup \perp$ вследствие следующих правил. Пусть справедливо:

1. $F_R(t) = r$, если слово r есть результат вычислений по R для входного слова t;
2. $F_R(t) = \perp$, если вычисление по R для входного слова t не заканчивается.

Тогда мы говорим: алгоритм R вычисляет функцию F_R .

Обратим внимание, что для слов t, для которых выполнение СТЗ не завершается, отображение F_R дает результат $\perp$. Символ $\perp$, таким образом, обозначает псевдорезультат незавершающегося вычисления. С его помощью обходят явную работу с частичными отображениями.

Если слова $t \in V^*$ понимать как представления определенных информаций из множества A, т. е. существует функция интерпретации такая, что (A, V^*, I) образует информационную систему, и если функция F_R , индуцируемая алгоритмом R, согласована с интерпретацией, то R индуцирует также отображение информаций.

3.5. Вычислительные структуры

Алгоритмы работают над элементами данных, которые могут быть объединены в так называемый носитель. Для формулирования алгоритмов наряду с используемыми элементами данных весьма существенны имеющиеся в распоряжении эффективные функции над этими элементами. Фигурирующие в алгоритмах носители и операции могут трактоваться вместе как вычислительные структуры. Вычислительная структура охватывает тем самым семейство носителей (данные) и семейство отображений между ними. Вычислительные структуры обнаруживаются в самых различных проявлениях. К примеру, карманный калькулятор так же, как и мощная ЭВМ, могут математически восприниматься и описываться как вычислительные структуры.

3.5.1. Семейства функций и множеств как вычислительные структуры

Понятие вычислительной структуры близко к понятию математической структуры. Вычислительная структура состоит из семейства множеств, называемых *носителями*, и семейства отображений между носителями.

Определение 3.4. Пусть S и F — множества обозначений; вычислительная структура A состоит из семейства $\{s^A : s \in S\}$ носителей s^A и семейства $\{f^A : f \in F\}$ отображений f^A между носителями. Мы пишем $A = (\{s^A : s \in S\}, \{f^A : f \in F\})$.

Элементы $s \in S$ есть обозначения для носителей и называются *типами*. Элементы $f \in F$ есть обозначения для отображений и называются *символами функций* или *знаками операций*. Для каждого $f \in F$ существует одно $n \in N$ такое, что имеет место: f^A есть n-местная функция и существуют типы $s_1, \ldots, s_{n+1} \in S$ такие, что

$$f^A : s_1^A \times s_2^A \times \ldots \times s_n^A \to s_{n+1}^A.$$

Может быть также и $n = 0$, т. е. допускаются и «нульместные» отображения, имеющие пустой список аргументов и получающие в точности один элемент из области значений. Такие отображения называются *константами*.

Для устранения частичных отображений снова используется специальный элемент $\perp$ («дно») для представления неопределенного значения функции. Пусть M — множество, не содержащее $\perp$. Множество M' определяется как $M' = M \bigcup \{\perp\}$.

Элемент $\perp$ представляет «неопределенный» результат функции, например, в случае незавершающегося алгоритма.

Отображение $f : M_1 \times \ldots \times M_n \to M_{n+1}$ называется *строгим*, когда справедливо: если одним из аргументов функции является $\perp$, то результат функции тоже есть $\perp$. Это соответствует простому предположению, что результат применения функции к списку аргументов определен только в том случае, когда определены все аргументы. Распространение частичных отображений на все отображения путем добавления $\perp$ к носителям приводит к строгим отображениям.

Пример (вычислительная структура $BOOL$ булевских значений.) Множество S типов вычислительной структуры $BOOL$ задано так:

$S = \{bool\}$.

Множество F символов функций структуры $BOOL$ задано так:

$F = \{true, false, \neg, \vee, \wedge\}$.

Множество носителей B' сопоставлено типу $bool$, т. е. имеет место $bool^{BOOL} = B' = \{L, O, \perp\}$.

Символы из F обозначают следующие функции:

$true^{BOOL} :\to B'$,

$false^{BOOL} :\to B'$,

$\neg^{BOOL} : B' \to B'$ (бесскобочный префикс),

$\wedge^{BOOL} : B' \times B' \to B'$(инфикс),

$\vee^{BOOL} : B' \times B' \to B'$ (инфикс).

Причем для $a, b \in B$ имеет место:
$true^{BOOL} = L$,
$false^{BOOL} = O$,
$\neg^{BOOL} b = neg(b)$,
$a \vee^{BOOL} b = or(a, b)$,
a $\wedge^{BOOL} b = and(a, b)$.

Функции являются строгими и потому их значения для случая, когда один из аргументов есть $\perp$, также установлены.

3.5.2. Сигнатуры

Чтобы установить множество символов функций и типов, которые встречаются в вычислительной структуре, а также установить, каким образом символы функций содержательно могут быть связаны между собой, используются сигнатуры.

Определение 3.5. Сигнатура Σ — есть пара (S, F) множеств S и F обозначений, причем S обозначает множество типов, т. е. имен для носителей, F — множество символов (имен) функций; для каждого символа функции $f \in F$ пусть задана ее функциональность $fctf \in S^+$.

В дальнейшем с целью улучшения читаемости при задании функциональности для f будем писать $fctf = (s_1, \ldots, s_n)s_{n+1}$, чтобы выразить, что f^A в вычислительной структуре A с соответствующей сигнатурой Σ используется для обозначения отображения

$$f^A : s_1^A \times s_2^A \times \ldots \times s_n^A \to s_{n+1}^A.$$

Пример (сигнатура). Сигнатура вычислительной структуры $BOOL$ булевских значении из вышеприведенного примера дает пример сигнатуры.
$S^{BOOL} = (bool)$,
$F^{BOOL} = true, false, \neg, \vee, \wedge$,
$fct\ true = bool$,
$fct\ false = bool$,
$fct\neg = (bool)bool$,
$fct\vee = (bool, bool)bool$,
$fct\wedge = (bool, bool)bool$.

Сигнатуры допускают наглядное графическое представление в виде диаграммы, которая для каждого типа содержит узел и для каждого n-местного символа операции — дугу с n входными узлами и одним выходным узлом. Для вычислительной структуры $bool$, дополненной работой с натуральными числами, мы получаем диаграмму сигнатуры, изображенную на рис. 3.2.

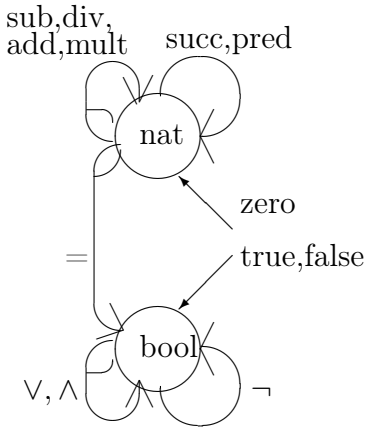


Рис. 3.2

Задания только одной сигнатуры, конечно, недостаточно для того, чтобы однозначно охарактеризовать вычислительную структуру. Имеется много различных вычислительных структур с одной и той же сигнатурой.

3.5.3. Основные термы

При заданной сигнатуре существует множество основных термов, которые могут быть образованы с помощью символов функций сигнатуры.

Пусть $\Sigma = (S, F)$ есть сигнатура. Множество основных термов типа s с $s \in S$ определяется следующим образом:

- каждый нульместный символ функции $f \in F$ с $fctf = s$ образует основной терм типа s;
- каждая последовательность символов $f(t_1, \ldots, t_n)$ с $f \in F$ и функциональностью $fctf = (s_1, \ldots, s_n)s$ есть основной терм типа s, если для всех $i, 1 \leq i \leq n$, t_i есть основной терм типа s_i.

Множество всех основных термов сигнатуры Σ обозначим через W_Σ, а множество основных термов типа s — через W_{Σ^s}. Если не существует нульместных символов функций, то множество W_Σ пусто.

Если имеется вычислительная структура A с сигнатурой Σ, то основные термы в A допускают интерпретацию. Переход от основного терма (представления) t типа s к соответствующему элементу a из множества A называют интерпретацией t в A.

Интерпретация I^A означает отображение $I^A : W_\Sigma \rightarrow \{a \in s^A; s \in S\}$.

Для каждого основного терма t запись $I^A[t]$ обозначает интерпретацию I в A. Пишут также t^A вместо $I^A[t]$. Интерпретация получается заменой в основном терме символов функций на соответствующие функции: $I^A[f(t_1, \ldots, t_n)] = f^A(I^A[t_1], \ldots, I^A[t_n])$.

В классической математике часто заданная интерпретация опускается и вместо t^A просто записывается t, разницей между основным термом и его интерпретацией там сознательно пренебрегают.

Для каждой вычислительной структуры A с сигнатурой Σ основные термы типа $s \in S$ могут использоваться как представления элементов из множества s^A, которые связаны с типом s в A. Если для каждого элемента a ($\neq \perp$) носителей из A имеется представление терма, т. е. для каждого s и каждого $a \in s^A (\neq \perp)$ существует основной терм типа s с $t^A = a$, то A называется *термопроизводимой*.

Интерпретация («значение») основного терма допускает соответственно вычисление терма. Один из простых способов организации такого вычисления представляют собой схемы.

3.5.4. Вычисления основных термов: схемы

Основной терм имеет характерную внутреннюю структуру. Он образуется из символов функций и последовательности (иногда пустой) основных термов («подтермов»), являющихся термами-аргументами.

Схема (или формуляр) для основного терма - графическое представление вычислений при его интерпретации. Схема состоит из прямоугольников, в которые заносится интерпретация основных термов, и подсхем для вычислений подтермов. Вычисление интерпретации основного терма допускает его удобное проведение по схеме. Поскольку интерпретация основного терма производится через значения интерпретаций его подтермов, то интерпретация подтермов упорядочивается с помощью схемы, структура которой аналогична структуре самого основного терма.

Примеры (схемы).

1. Основному терму $((1 + 2) * 3) - 4$ с интерпретацией в N соответствует схема, показанная на рис. 3.3.

2. Основному терму $(true \wedge false) \vee (false \vee ((\neg false \wedge true))$ с интерпретацией в $BOOL$ соответствует схема, приведенная на рис. 3.4.

Основные термы применяются для представления элементов из множества носителей вычислительной структуры. Для определения отображения между этими элементами используются термы с идентификаторами.

3.5.5. Термы с (свободными) идентификаторами

Идентификатор («обозначатель», «переменная», «неизвестное») – держатель места («имя») для терма (или элемента), который может быть подставлен на это место. Идентификаторы могут пониматься как имена термов или элементов, которые позднее будут конкретизированы.

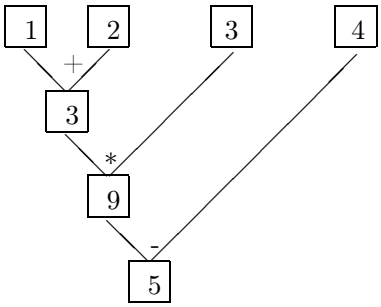


Рис. 3.3

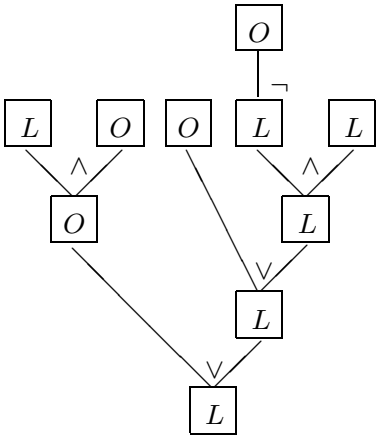


Рис. 3.4

Пусть $\Sigma = (S, F)$ — сигнатура и $X = \{X_s : s \in S\}$ — семейство множеств идентификаторов. Пусть множества X_s идентификаторов попарно не пересекаются и отличны от символов функций в F. $W_\Sigma(X)$ обозначает алгебру термов, распространенную на X, т. е. $W_{\Sigma I}$ с $\Sigma I = (S, F \bigcup x \in X_s : s \in S)$ и $fct\ x = s$ для $x \in X_s$, где X_s обозначает множество идентификаторов (держателей мест, обозначений) типа s.

Примеры (термы с идентификаторами).

1. Уравнения с «неизвестными» в математике — это уравнения между термами с идентификаторами, например $ax^2 + bx + \mathrm{c} = 0$.

2. Часто термы снабжаются свободными идентификаторами, чтобы определять функции. Функция f может быть определена через отображение $f : N \to N$ с $f(x) = 2x + 1$.

Термы со свободными идентификаторами называются также *полиномами*. Вместо идентификаторов в термы могут подставляться другие термы. Соответствующее отображение называется *подстановкой в термы с (свободными) идентификаторами.*

Пусть t — терм с идентификаторами, x — идентификатор типа s и r — терм типа s; через $t[r/x]$ обозначается терм, который получается, когда идентификатор x заменяется на r. Этот процесс называется *подстановкой.*

Подстановки описываются формально аналогично построению термов с помощью следующих уравнений:

$x[t/x] = t$,

$y[t/x] = y$, если x и y - различные идентификаторы,

$f(t_l, \ldots, t_n)[t/x] = f(t_1[t/x], \ldots, t_n[t/x])$, где $f \in F$ с функциональностью $fctf = (s_1, \ldots, s_n)s_{n+1}$ и термы t_i имеют типы s_i.

Через $t[t_1/x_1, \ldots, t_n/x_n]$ обозначается терм, который возникает из терма t при одновременной подстановке t_i вместо (попарно различных) идентификаторов x_i.

Пусть t — терм с (свободными) идентификаторами. Терм r назовем *экземпляром* t, если r получается из t путем замены (подстановки) в нем определенных (свободных) идентификаторов.

3.5.6. Интерпретация термов с идентификаторами

Пусть A — вычислительная структура с сигнатурой $\Sigma = (S, F)$, а X — семейство множеств идентификаторов. Отображение

$b : \{x \in X_s : s \in S\} \to \{a \in s^A : s \in S\}$,

которое каждому идентификатору x в X типа s ставит в соответствие элемент $a \in s^A$ структуры данных s^A типа s, называется конкретизацией X (в A).

Для каждой конкретизации b определяется интерпретация I_b^A терма t со свободными идентификаторами из X с помощью следующих равенств:

$I_b^A[x] = b(x)$,

$I_b^A[f(t_1, \ldots, t_n)] = f^A(I_b^A[t_1], \ldots, I_b^A[t_n])$.

Для $n = 0$ получается $I_b^A[f] = f^A$.

3.5.7. Термы с (свободными) идентификаторами как схемы

Термы со свободными идентификаторами могут играть роль схем, в которых не все значения определены.

Пример (из геометрии). Площадь S кольца s с внутренним радиусом r и внешним радиусом R получается по формуле $S = \pi(R^2 - r^2)$.

Терму в правой части формулы соответствует схема, приведенная на рис. 3.5.

Терм со свободными идентификаторами определяет вычислительную схему. Схемы можно найти (в несколько иной форме) во многих сферах человеческой деятельности, например в управлении. Есть схемы для начисления зарплаты.

Пример (схема с многократно применяемым промежуточным результатом). Терм $(x - y) * (x + y)$ обладает схемой, показанной на рис. 3.6.

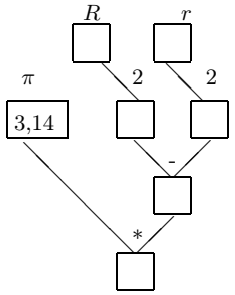


Рис. 3.5

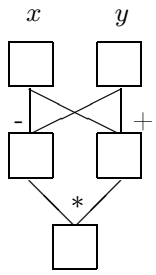


Рис. 3.6

Термы могут использоваться в системах замены термов для представления алгоритмов.

3.5.8. Алгоритмы как системы подстановки термов

Наиболее наглядный метод описания алгоритмов как системы текстовых замен предлагают системы подстановки термов. Ранее было показано, что термы могут строиться над заданной сигнатурой по определенным, четким правилам. К сигнатуре и термам над нею могут быть заданы интерпретации над вычислительной структурой, т. е. над алгеброй. Как и в случае булевских термов, возникает информационная система, при которой термы выступают как представления. Через интерпретацию задается семантическая эквивалентность на термах. Тогда правила преобразования термов могут быть определены так, что термы всегда будут переводиться в семантически эквивалентные термы.

Множества правил для алгоритмов могут быть предъявлены в форме системы подстановки термов. Специальная структура термов, которая получается из их построения, может использоваться для задания правил подстановок и их применения.

3.5.9. Правила подстановки термов

Для заданной сигнатуры Σ и заданного семейства X множеств идентификаторов пара (t, r) термов t, r одинакового типа с (свободными) идентификаторами из X называется *подстановкой термов*, *правилом подстановки термов (ППТ)*, а также *схемой подстановки термов*. Правило записывается в виде $t \to r$.

В большинстве случаев для ППТ требуется, чтобы все идентификаторы, встречающиеся в r, также входили в t. Если в t и r заменяют определенные идентификаторы $x_1, \ldots, x_n$ на термы $t_1, \ldots, t_n$ подходящих типов, то получают экземпляр (частный, конкретный случай) правила. Соответственно $t[t_1/x_1, \ldots, t_n/x_n] \to r[t_1/x_1, \ldots, t_n/x_n]$ называется *экземпляром правила* $t \to r$. Если t и r — основные термы, то экземпляр называется *полным*.

Пример (экземпляры правил подстановок термов).

Для правила $pred(succ(x)) \to x$ примерами экземпляров являются:

$pred(succ(zero)) \to zero$,

$pred(succ(succ(y))) \to succ(y)$.

Благодаря тому, что правило применяется к любому подтерму имеющегося терма, из правила получаются шаги подстановки термов .

Пусть $t \to r$ есть экземпляр правила. Пусть задан терм C, в котором встречается свободный идентификатор x, тогда $C[t/x] \to C[r/x]$ называется *(безусловным) применением правила* (к терму $C[t/x]$). Терм t называется редексом, а вхождение x в C — *местом применения*.

Пример (применение правила замены термов).

К терму $succ(succ(pred(succ(zero))))$ может быть применено правило предыдущего примера, Тогда получим

$$succ(succ(pred(succ(zero)))) \rightarrow succ(succ(zero)).$$

Аналогом алгоритмов в виде подстановки текстов являются алгоритмы в виде систем подстановки термов.

3.5.10. Системы подстановки термов

Множество (в общем случае конечное) R правил подстановки термов на сигнатуре Σ называется *системой подстановки термов* (СПТ) над Σ. Если для последовательности термов (t_i) $0 \leq i \leq n$ справедливо для $i = 0, \ldots, n-1$ $t_i \rightarrow t_{i+1}$ есть применение правила из системы подстановки термов R, то последовательность термов является вычислением в R для t_0.

Терм t называется *терминальным* для системы R, если не существует терма r такого, что $t \rightarrow r$ есть применение правила из R.

Если в вычислении, заданном с помощью термов (t_i) $0 \leq i \leq n$, терм t_n является терминальным, то вычисление называется *терминированным* (*завершающимся*), а t_n — *результатом* или *выходом* вычисления для входа t_0.

Бесконечная последовательность (t_i) $i \in N$ термов, удовлетворяющих условию $t_i \rightarrow t_{i+1}$, $i \in N$ есть применение правила из системы подстановки термов R, называется *нетерминированным (незавершающимся) вычислением* в R для t_0. Система R называется *терминированной*, если не существует незавершающихся вычислений.

Терминальные основные термы определяют нормальную форму. Они часто также называются термами в нормальной форме относительно R. Над системой R мы можем терму t поставить в соответствие терминальный терм r в качестве нормальной формы, если r есть результат вычислений с входом t. Как правило, система нормальных форм, индуцированных через СПТ, не является ни однозначной, ни полной. Термы, для которых существуют только бесконечные вычисления, не имеют нормальных форм. Для определенных термов могут существовать вычисления с различными результатами.

Пример (СПТ для вычислительной структуры BOOL). Ниже символы функций $\neg, \vee$ будут использоваться в скобочной префиксной и инфиксной формах записи. Правила подстановок термов гласят:

$(\neg true) \rightarrow false$,
$(\neg false) \rightarrow true$,
$(false \vee x) \rightarrow x$,
$(x \vee false) \rightarrow x$,
$(true \vee true) \rightarrow true$.

Эта СПТ редуцирует каждый основной терм типа *bool* к терму $true$ или $false$. Повторное применение ППТ из заданного множества правил можно трактовать как алгоритм, работающий с термами в качестве входа и выхода.

3.5.11. Алгоритмы подстановки термов

Пусть задана система подстановок R. R определяет алгоритм в силу следующего предписания (алгоритм получает в качестве входного слова основной терм t).

1. Если R содержит правило подстановки с применением $t \to r$, то далее алгоритм продолжается с использованием r вместо t.
2. Если R не содержит правила подстановки с применением $t \to r$, то алгоритм заканчивается с t в качестве результата.

Алгоритм подстановки термов (АПТ) тем самым выполняет вычисление в R для каждого основного терма t. Часто вычисление состоит в решении задачи преобразования заданного основного терма в определенную заранее заданную нормальную форму.

Если АПТ начинает работать с заданным основным термом t, то t называют также *входом* для алгоритма; если алгоритм завершается основным термом r, то r называется также *выходом* или *результатом*.

Алгоритмы подстановки термов работают над основными термами сигнатуры. Они осуществляют преобразование основных термов посредством вычислений. При этом не рассматривается, насколько проведенные преобразования соответствуют определенным численным свойствам символов функций. Все же благодаря концепции вычислительных структур возможна особенно ясная, простая концепция корректности СПТ.

Приведем пример работы алгоритма термовых замен. Определим сначала сигнатуру NAT, описывающую работу с целыми неотрицательными ́числами.

$S = \{nat\}$,
$F = \{zero, succ, pred, add, mult\}$,
$fct\ zero = nat$,
$fct\ succ = (nat)\ nat$,
$fct\ pred = (nat)\ nat$,
$fct\ add = (nat,\ nat)\ nat$,
$fct\ mult = (nat,\ nat)\ nat$.

Знак операции $zero$ определяет константу ноль, $succ$ и $pred$ — увеличение и уменьшение на 1, а add и $mult$ — сложение и умножение. Далее следует выбрать нормальные формы, к которым следует приводить исходные термы. В данном примере нормальные формы имеют вид $succ(succ(\ldots succ(zero)))$. Причем, для простоты можно использовать запись $succ^n(zero)$ для обозначения терма, содержащего n знаков операции $succ$.

Система термовых замен состоит из следующих правил:

1) $succ(pred(x)) \to x$,

2) $add(zero,\ x) \to x$,

3) $add(succ(x),\ y) \to succ(add(x,\ y))$,

4) $mult(zero,\ x) \to zero$,

5) $mult(succ(x),\ y) \to add(y,\ mult(x,\ y))$.

Для исходного терма $mult(succ(succ(zero)),\ succ(succ(succ(zero))))$ вычисление будет выглядеть следующим образом:

$mult(succ(succ(zero)),\ succ(succ(succ(zero))))$ $\xrightarrow{\text{сократим выражение}}$

$\to\ mult(succ(succ(zero)),\ succ^3(zero))$ $\xrightarrow{\text{правило 5}}$

$\to\ add(succ^3(zero),\ mult(succ(zero),\ succ^3(zero)))$ $\xrightarrow{\text{3 раза правило 3}}$

$\to\ succ^3(add(zero,\ mult(succ(zero),\ succ^3(zero))))$ $\xrightarrow{\text{правило 2}}$

$\to\ succ^3(mult(succ(zero),\ succ^3(zero)))$ $\xrightarrow{\text{правило 5}}$

$\to\ succ^3(add(succ^3(zero),\ mult(zero,\ succ^3(zero))))$ $\xrightarrow{\text{правило 4}}$

$\to\ succ^3(add(succ^3(zero),\ zero))$ $\xrightarrow{\text{3 раза правило 3}}$

$\to\ succ^6(add(zero,\ zero))$ $\xrightarrow{\text{правило 2}}$ $succ^6(zero)$.

Таким образом, выходной терм будет иметь следующий вид:

$succ(succ(succ(succ(succ(succ(zero))))))$.

Глава 4. Основы машинной арифметики

4.1. Системы счисления и способы перевода чисел

Системой счисления называют систему приемов и правил, позволяющих устанавливать взаимнооднозначное соответствие между любым числом и его представлением в виде совокупности конечного числа символов. Множество символов, используемых для такого представления, называют цифрами.

В зависимости от способа изображения чисел с помощью цифр системы счисления делятся на *позиционные* и *непозиционные*.

В непозиционных системах любое число определяется как некоторая функция от численных значений совокупности цифр, представляющих это число. Цифры в непозиционных системах счисления соответствуют некоторым фиксированным числам. Примером непозиционной системы служит римская система счисления. В вычислительной технике непозиционные системы не применяются.

Систему счисления называют *позиционной*, если одна и та же цифра может принимать различные численные значения в зависимости от номера разряда этой цифры в совокупности цифр, представляющих заданное число. Пример такой системы — арабская десятичная система счисления.

В позиционной системе счисления любое число записывается в виде последовательности цифр:

$A = a_{m-1}a_{m-2}\ldots a_k\ldots a_0, a_{-1}\ldots a_{-l}$.

Позиции, пронумерованные индексами $k(-l \le k \le m-1)$ называются *разрядами* числа. Сумма $m+l$ соответствует общему количеству разрядов числа (m — число разрядов целой части числа, l — дробной части).

Каждая цифра в записываемой последовательности может принимать одно из N возможных значений. Количество различных цифр N, используемых для изображения чисел в позиционной системе счисления, называется *основанием системы счисления*. Основание N указывает, во сколько раз единица $k+1$-го разряда больше единицы k-го разряда, а цифра a_k соответствует количеству единиц k-го разряда, содержащихся в числе.

Таким образом, число может быть представлено в виде суммы:

$$(A)_N = a_{m-1}N^{m-1} + \ldots + a_kN^k + \ldots + a_0N^0 + a_{-1}N^{-1} + \ldots + a_{-l}N^{-l}. \quad (4.1)$$

Основание позиционной системы счисления определяет ее название. В вычислительной технике применяются двоичная, восьмеричная, десятичная и

шестнадцатеричная системы. В дальнейшем, чтобы явно указать используемую систему счисления, будем число заключать в скобки и в индексе указывать основание системы счисления.

В двоичной системе счисления используются только две цифры: 0 и 1. Любое двоичное число может быть представлено в следующей форме:

$$(A)_2 = a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \ldots + a_k 2^k + \ldots + a_0 2^0 + a_{-1}2^{-1} + \ldots + a_{-l}2^{-l}.$$

Например, двоичное число

$(10101,101)_2 = 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 0\times 2 + 1 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3}$.

В восьмеричной системе счисления для записи чисел используется восемь цифр (0, 1, 2, 3, 4, 5, 6, 7), а в шестнадцатеричной - шестнадцать (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

Для хранения и обработки данных в ЭВМ используется двоичная система, так как она требует наименьшего количества аппаратуры по сравнению с другими системами. Все остальные системы счисления применяются только для удобства пользователей.

Многоразрядные числа складываются, вычитаются, умножаются и делятся по тем же правилам, что и в десятичной системе счисления.

Перевод числа из одной системы в другую выполняется по универсальному алгоритму, заключающемуся в последовательном делении целой части числа и образующихся целых частных на основание новой системы счисления, записанное в исходной системе счисления, и в последующем умножении дробной части и дробных частей получающихся произведений на то же основание, записанное в исходной системе счисления.

При переводе целой части получающиеся в процессе последовательного деления остатки представляют цифры целой части числа в новой системе счисления, записанные цифрами исходной системы счисления. Последний остаток является старшей цифрой переведенного числа.

При переводе дробной части числа целые части чисел, получающихся при умножении, не участвуют в последующих умножениях. Они представляют собой цифры дробной части исходного числа в новой системе счисления, изображенные числами старой системы. Значение первой целой части является первой цифрой после запятой переведенного числа.

Пример перевода числа 30,6 из десятичной системы в двоичную представлен на рис. 4.1.

Если при переводе дробной части получается периодическая дробь, то производят округление, руководствуясь заданной точностью вычислений.

Пример перевода числа 111110,01 из двоичной системы в десятичную приведен на рис. 4.2.

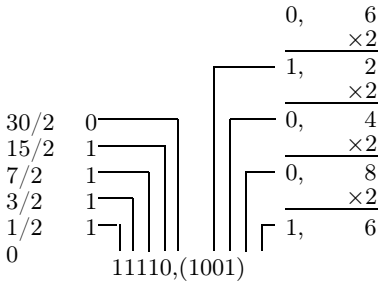


Рис. 4.1

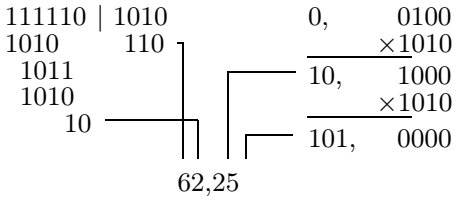


Рис. 4.2

Примечание 1: 1010 - основание десятичной системы счисления в двоичной записи.

Примечание 2: десятичные эквиваленты разрядов искомого числа находим по таблице.

Двоичные числа	Восьмеричные числа	Десятичные числа	Шестнадцатеричные числа
0,0001	0,04	0,0625	0,1
0,001	0,1	0,125	0,2
0,01	0,2	0,25	0,4
0,1	0,4	0,5	0,8
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10

При переводе чисел из любой системы счисления в десятичную удобнее пользоваться непосредственно формулой (4.1):

$(775)_8 = 7 \times 8^2 + 7 \times 8 + 5 = (509)_{10}$.

Для осуществления автоматического перевода десятичных чисел в двоичную систему счисления необходимо вначале каким-то образом ввести их

в машину. Для этой цели обычно используется двоично-десятичная запись чисел или их представление в кодах ASCII.

При двоично-десятичной записи каждая цифра десятичного числа заменяется четырехзначным двоичным числом (тетрадой):

$(983,65)_{10} = (1001\ 1000\ 0011, 0110\ 0101)_{2-10}$.

При записи чисел в кодах ASCII цифрам от 0 до 9 поставлены в соответствие восьмиразрядные двоичные коды от 00110000 до 00111001.

Шестнадцатеричная и восьмеричная системы счисления используются только программистами и операторами ЭВМ, так как представление в них чисел более компактное, чем в двоичной, и перевод из них в двоичную и обратно выполняется очень просто (их основания представляют собой целую степень числа 2).

Для перевода восьмеричного числа в двоичное достаточно каждый восьмеричный разряд представить тремя двоичными (триадой), а для перевода шестнадцатиричного числа - четырьмя (тетрадой):

$(376,51)_8 = (011\ 111\ 110, 101\ 001)_2$;

$(1AF8)_{16} = (0001\ 1010\ 1111\ 1000)_2$.

4.2. Формы представления чисел в ЭВМ

Разряд двоичного числа представляется в ЭВМ некоторым техническим устройством, например, триггером, двум различным состояниям которого приписываются значения 0 и 1. Группа таких устройств, предназначенная для представления в машине многоразрядного числа, называется *регистром.*

На рис. 4.3 изображена структура двоичного регистра, представляющего в машине n-разрядное слово.

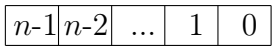


Рис. 4.3

Отдельные запоминающие элементы пронумерованы от 0 до $n-1$. Количество разрядов регистра определяет точность представления чисел. Путем соответствующего увеличения числа разрядов регистра может быть получена любая точность вычислений, однако это сопряжено с увеличением количества аппаратуры (в лучшем случае зависимость линейная, в худшем — квадратичная).

В ЭВМ применяются две основные формы представления чисел: *полулогарифмическая* с плавающей запятой и *естественная* с фиксированным положением запятой.

При представлении чисел с фиксированной запятой положение запятой закрепляется в определенном месте относительно разрядов числа и сохраняется неизменным для всех чисел, изображаемых в данной разрядной сетке. Обычно запятая фиксируется перед старшим разрядом или после младшего. В первом случае в разрядной сетке могут быть представлены только числа, которые по модулю меньше 1, во втором – только целые числа.

Для кодирования знака числа используется старший («знаковый») разряд.

При выполнении арифметических действий над правильными дробями могут получаться двоичные числа, по абсолютной величине большие или равные единице, что называется *переполнением* разрядной сетки. Для исключения возможности переполнения приходится масштабировать величины, участвующие в вычислениях.

Диапазон представления правильных двоичных дробей:
$2^{-(n-1)} \leq \mid A \mid \leq 1 - 2^{-(n-1)}$.

Числа, которые по абсолютной величине меньше единицы младшего разряда разрядной сетки, называются *машинным нулем*. Диапазон представления целых двоичных чисел со знаком в n-разрядной сетке:
$0 \leq \mid A \mid \leq 2^{n-1} - 1$.

Использование представления чисел с фиксированной запятой позволяет упростить схемы машины, повысить ее быстродействие, но представляет определенные трудности при программировании. В настоящее время представление чисел с фиксированной запятой используется как основное только в микроконтроллерах.

В универсальных ЭВМ основным является представление чисел с плавающей запятой, которое в общем случае имеет вид $A = m * N^p$, где N — основание системы счисления, p — целое число, называемое *порядком* числа A, m — *мантисса* числа A ($\mid m \mid < 1$).

Так как в ЭВМ применяется двоичная система счисления, то $A = m * 2^p$, причем порядок и мантисса представлены в двоичной форме.

Двоичное число называется *нормализованным*, если его мантисса удовлетворяет неравенству $\frac{1}{2} \leq \mid m \mid < 1$.

Неравенство показывает, что двоичное число является нормализованным, если в старшем разряде мантиссы стоит единица. Например, число $0,110100 * 10^{100}$ — нормализованное, а $0,001101 * 10^{110}$ — ненормализованное.

Нормализованное представление чисел позволяет сохранить в разрядной сетке большее количество значащих цифр и, следовательно, повышает точность вычислений. Однако современные ЭВМ позволяют при необходимости выполнять операции также и над ненормализованными числами.

Широкий диапазон представления чисел с плавающей запятой удобен для научных и инженерных расчетов. Для повышения точности вычислений во многих ЭВМ предусмотрена возможность использования формата двойной длины, но тогда происходит увеличение затрат памяти на хранение данных и замедляются вычисления.

4.3. Представление отрицательных чисел в ЭВМ

Для кодирования знака двоичного числа используется старший («знаковый») разряд (ноль соответствует плюсу, единица — минусу). Такая форма представления числа называется *прямым кодом.*

Формула для образования прямого кода правильной дроби имеет вид
$[A]_{\text{пр}} = A$, если $A \geq 0$,
$[A]_{\text{пр}} = 1 - A$, если $A < 0$.
Примеры:
$A = 0,110111 \rightarrow [A]_{\text{пр}} = 0,110111$;
$A = -0,110111 \rightarrow [A]_{\text{пр}} = 1 - (-0,110111) = 1,110111$.
Прямой код целого числа получается по формуле
$[A]_{\text{пр}} = A$, если $A \geq 0$,
$[A]_{\text{пр}} = 10^{n-1} - A$, если $A < 0$,
где 10 — число 2 в двоичной системе счисления, n – количество позиций в разрядной сетке.

Например, при $n = 8$ $A = 110111 \rightarrow [A]_{\text{пр}} = 00110111$;
$A = -110111 \rightarrow [A]_{\text{пр}} = 10000000 - (-110111) = 10110111$.

В ЭВМ прямой код применяется только для представления положительных двоичных чисел. Для представления отрицательных чисел применяется либо дополнительный, либо обратный код, так как над отрицательными числами в прямом коде неудобно выполнять арифметические операции.

Формула для образования дополнительного кода дроби
$[A]_{\text{доп}} = 10 + A$.
Формула для образования обратного кода дроби
$[A]_{\text{обр}} = 10 - 10^{-(n-1)} + A$.
Например, при n=8
$A = -0,1100001 \rightarrow [A]_{\text{доп}} = 10 + (-0,1100001) = 1,0011111$;
$[A]_{\text{обр}} = 10 - 10^{-7} + (-0,1100001) = 1,1111111 - 0,1100001 = 1,0011110$.
Формула для образования дополнительного кода целого числа
$[A]_{\text{доп}} = 10^n + A$.
Формула для образования обратного кода целого числа
$[A]_{\text{обр}} = 10^n - 1 + A$.
Например, при n = 8
$A = -1100001 \rightarrow [A]_{\text{доп}} = 100000000 + (-1100001) = 10011111$;
$[A]_{\text{обр}} = 100000000 - 1 + (-1100001) = 11111111 - 1100001 = 10011110$.

Таким образом, правила для образования дополнительного и обратного кода состоят в следующем:

- для образования дополнительного кода отрицательного числа необходимо в знаковом разряде поставить единицу, а все цифровые разряды инвертировать (заменить 1 на 0, а 0 — на 1), после чего прибавить 1 к младшему разряду;
- для образования обратного кода отрицательного числа необходимо в знаковом разряде поставить единицу, а все цифровые разряды инвертировать.

Примечание: при данных преобразованиях нужно учитывать размер разрядной сетки.

Замена вычитания двоичных чисел $A_1 - A_2$ сложением с дополнениями $[A_1]_{\text{пр}} + [-A_2]_{\text{доп}}$ или $[A_1]_{\text{пр}} + [-A_2]_{\text{обр}}$ позволяет оперировать со знаковыми разрядами так же, как и с цифровыми. При этом перенос из старшего знакового разряда, если он возникает, учитывается по разному для обратного и дополнительного кодов:

- при использовании дополнительного кода единица переноса из знакового разряда отбрасывается;
- при использовании обратного кода единица переноса из знакового разряда прибавляется к младшему разряду суммы (осуществляется так называемый *циклический перенос*).

Пример: складываем числа $A_1 = 0,10010001$ и $A_2 = -0,01100110$.
При использовании обратного кода получим
$[A_1]_{\text{пр}} = 0,10010001$
$+$
$[A_2]_{\text{обр}} = 1,10011001$
10,00101010
Результат: 0,00101011.
При использовании дополнительного кода получим
$[A_1]_{\text{пр}} = 0,10010001$
$+$
$[A_2]_{\text{доп}} = 1,10011010$
Результат: 0,00101011.

Если знаковый разряд результата равен нулю, то получено положительное число, которое представлено в прямом коде. Если в знаковом разряде единица, то результат отрицательный и представлен в обратном или дополнительном коде.

Для того чтобы избежать ошибок при выполнении бинарных операций, перед переводом чисел в обратные и дополнительные коды необходимо выравнивать количество разрядов прямого кода операндов.

4.4. Методы контроля

При передаче информации большую роль играют вопросы правильности полученной информации, обнаружения и исправления ошибок и т.д. Рассмотрим некоторые методы контроля.

Контроль четности/нечетности заключается в том, что к сообщению добавляется один контрольный бит. Значение его устанавливается так, чтобы общее число единиц в сообщении (с учетом контрольного бита) было четным/нечетным. При проверке сообщения подсчитывается количество единиц, и, если четность сохранилась, делается вывод о правильности передачи информации. В противном случае констатируется наличие ошибки. Разряд, в котором произошла ошибка, определить нельзя.

Другой разновидностью контроля является циклический остаточный код или *CRC-код* (Cyclic Redundancy Code). Над сообщением проводится ряд математических преобразований (например, сообщение делится на несколько частей, которые складываются по модулю 2), результат которого дописывается к сообщению. При проверке указанные математические преобразования повторяются и результат сравнивается с записанным. В зависимости от совпадения или несовпадения CRC-кодов делается вывод о наличии ошибок в сообщении.

Большой вклад в развитие методов контроля внес Р. Хэмминг. Он предложил несколько кодов, способных не только обнаруживать наличие ошибок в сообщениях, но и исправлять их.

Дистанцией Хэмминга между двумя сообщениями одинаковой длины называется количество попарно различных бит в соответствующих разрядах этих сообщений.

Пример. Пусть $A = 110011$, а $B = 111001$. Дистанция Хэмминга между данными сообщениями равна 2 (третий и пятый биты).

Если каждая пара сообщений имеет дистанцию Хэмминга не менее 3, то можно определить до двух ошибок, а исправить одну. Действительно, изменение одного бита в одном из двух сообщений изменит дистанцию Хэмминга между ними на 1 (либо увеличит, либо уменьшит). Тогда сообщение с одной ошибкой будет иметь с истинным сообщением дистанцию 1, а со всеми остальными не менее 2. Соответственно, если все пары имеющихся сообщений имеют дистанцию Хэмминга не менее 5, то можно определить до 4 ошибок, а исправить до 2 и т.д.

Пример. Пусть имеется следующий набор сообщений:
$A = 110011$,
$B = 000010$,
$C = 011110$.

Дистанция Хэмминга между сообщениями A и B равна трем, между B и C также трем, а между A и C — четырем.

Допустим пришло сообщение $X = 100010$. Дистанция Хэмминга между сообщениями A и X равна двум, между B и X одному, а между C и X — четырем. Если в передаваемом сообщение возможно возникновение не более одной ошибки, можно сделать вывод, что пришло сообщение B с одной ошибкой.

Ниже приводится пример самовосстанавливающегося кода Хэмминга, обладающего способностью обнаружить одну ошибку и определить номер её разряда.

Предположим, что имеется код, содержащий m информационных разрядов. В него добавляется k контрольных разрядов. Число k определяется как минимальное целое число, удовлетворяющее соотношению

$$2^k > m + k. \tag{4.2}$$

Контрольные разряды располагаются в позициях, номера которых представляют собой степень двойки, т.е. 1, 2, 4, 8, 16 и т.д.

Каждый из k контрольных разрядов обеспечивает контроль четности для одной из k групп. В группу i-го контрольного разряда входят те разряды, в двоичной записи номеров которых есть единица на том же месте, что и в двоичной записи номера i-го контрольного разряда. Таким образом в первую группу войдут разряды с номерами 1, 3, 5, 7, 9, 11, 13, 15 и т.д. Во второй группе окажутся элементы с номерами 2, 3, 6, 7, 10, 11, 14, 15 и т.д. В третьей — 4, 5, 6, 7, 12, 13, 14, 15, 20, ..., в четвертой — 8, 9, 10, 11, 12, 13, 14, 15, 24, ..., и т.д.

При проверке сообщения определяется четность каждой из k групп. Если все группы дали значение 0, т.е. в каждой из них оказалось четное количество единиц — сообщение без ошибок. В противном случае полученные значения чётностей, записанные справа налево, дадут двоичную запись номера разряда, в котором произошла ошибка.

Пример. Пусть имеется сообщение 11011100100101, содержащее 14 информационных разрядов. Тогда из соотношения (4.2) находим, что $k = 5$.

Разместим сначала информационные разряды.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
k_1	k_2	m_1	k_3	m_2	m_3	m_4	k_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	k_5	m_{12}	m_{13}	m_{14}
		1		1	0	1		1	1	0	0	1	0	0		1	0	1

Теперь определим, какие разряды войдут в каждую из 5 контрольных групп.

k_1: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19.
k_2: 2, 3, 6, 7, 10, 11, 14, 15, 18, 19.
k_3: 4, 5, 6, 7, 12, 13, 14, 15.
k_4: 8, 9, 10, 11, 12, 13, 14, 15.
k_5: 16, 17, 18, 19.

После заполнения контрольных бит таким образом, чтобы количество единиц в каждой группе было четным, сообщение примет вид

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
k_1	k_2	m_1	k_3	m_2	m_3	m_4	k_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	k_5	m_{12}	m_{13}	m_{14}
1	0	1	1	1	0	1	1	1	1	0	0	1	0	0	0	1	0	1

Так будет выглядеть исходное сообщение, закодированное самовосстанавливающимся кодом Хэмминга. Допустим, при передаче данных произошла ошибка в 11 разряде. Ниже приведено пришедшее сообщение.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
k_1	k_2	m_1	k_3	m_2	m_3	m_4	k_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	k_5	m_{12}	m_{13}	m_{14}
1	0	1	1	1	0	1	1	1	1	1	0	1	0	0	0	1	0	1

Подсчитаем количество единиц в каждой контрольной группе и выпишем их справа налево.

k_5	k_4	k_3	k_2	k_1
0	1	0	1	1

Получившаяся последовательность есть ничто иное, как двоичная запись числа 11, то есть номера, в котором произошла ошибка. Естественно, такой код может восстановить только одну ошибку, однако ясно демонстрирует возможность создания самовосстанавливающихся кодов.

Глава 5. Аппаратные средства персональных компьютеров

5.1. История создания персонального компьютера

В истории вычислительной техники можно выделить два больших этапа развития. Для первого этапа характерно применение в компьютерах не микросхем, а радиоламп и транзисторов, не оперативной памяти на динамических элементах, а памяти на ферритовых кольцах, не миниатюрных накопителей на магнитных дисках, а огромных стоек с накопителями на сменных магнитных дисках.

Второй этап развития компьютерной техники ознаменовался началом выпуска ЭВМ на модулях микропроцессоров. В 1971 г. был выпущен 4-разрядный микропроцессор Intel 4004, который обладал рядом положительных особенностей, свойственных современным микропроцессорам: универсальностью, гибкостью, миниатюрностью, малой потребляемой мощностью и низкой стоимостью. Подобные микропроцессоры свыше десятилетия использовались как в зарубежной, так и в отечественной вычислительной технике.

Первыми персональными компьютерами принято считать микроЭВМ Altair 8800 производства MITS, которые поступили в продажу в 1975 г. в виде наборов для радиолюбителей «Сделай сам». В них использовались интегральные микросхемы малой (до 100 транзисторов на кристалл) и средней (до 1000 транзисторов на кристалл) степени интеграции, а также 8-разрядный микропроцессор Intel 8080.

В 1977 г. корпорация Apple выпустила в продажу персональный компьютер AppleII, который отличался совершенным (по тем временам) программным обеспечением, рассчитанным на самых неподготовленных пользователей. Простота освоения работы на компьютере позволяла использовать такую технику не только специалистам-радиолюбителям и профессионалам, но и всем желающим приобщиться к миру электроники.

Начало 80-х гг. знаменуется разделением мира ПК на две фракции несовместимых друг с другом ПК — Macintosh (наследники AppleII) и IBM PC. Первые ПЭВМ имели существенные недостатки, ограничивающие распространение ПК в среде неподготовленных пользователей, и в основном отвечали интересам узкого круга профессионалов. Все они имели закрытую архитектуру, аппаратные средства были представлены одним неразъемным устройством. Совершенствование компьютера было уделом профессионалов-разработчиков. В августе 1981 г. фирма IBM выпустила первый ПК, в котором осуществила модульный принцип построения, использовавшийся ранее

в больших вычислительных системах. Такое построение, называющееся открытой архитектурой, обеспечивало возможность сборки ЭВМ из независимо изготовленных частей. На системной (материнской) плате размещены только те блоки, которые осуществляют обработку информации. Схемы, управляющие всеми остальными устройствами компьютера, реализованы на отдельных платах, которые вставляются в стандартные разъемы на системной плате — слоты, что позволяет легко заменять дополнительные устройства на новые при старении прежних, увеличивать объем памяти, добавлять, по мере необходимости, новые устройства.

Начиная с 1981 г. лидерство на компьютерном рынке захватила корпорация IBM с компьютером IBM PC, который на долгие годы стал стандартом для производителей персональных компьютеров.

5.2. Архитектура ПК

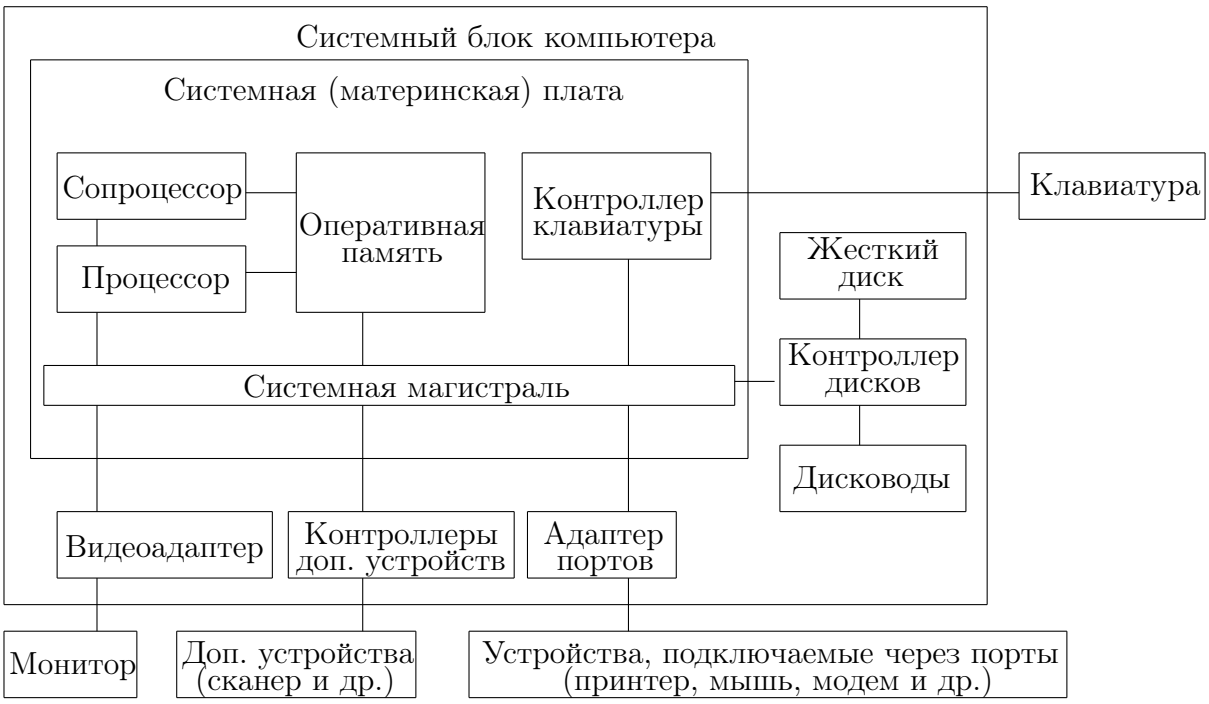
Архитектура ПК — структурная схема внутренней организации и взаимодействия основных функциональных модулей компьютера (центрального процессора, чипсета, системы памяти, контроллеров периферийных устройств и самих периферийных устройств). В понятие архитектуры ПК входят также принципы организации вычислительного процесса и переработки информации. Детальный анализ архитектуры компьютера содержит описание представления программ и данных: систему счисления, информационные форматы и организацию вычислительного и обменного процессов. Он затрагивает также рассмотрение структуры памяти, методики выполнения машинных операций, системы размещения информации в памяти, системы диагностирования и контроля, а также управления вычислительным процессом.

Как видно, архитектура ПК — ёмкое понятие, рассматривающее тонкости взаимодействия всех его компонентов, объясняющее тончайшие нюансы вычислительных и обменных процессов.

На рисунке изображена упрощенная блок-схема, дающая общее представление об архитектуре и основных компонентах компьютера. Рассмотрим подробнее их назначение.

5.3. Процессор

Центральный процессор (ЦП) — электронный модуль, выполняющий в компьютерной системе основную вычислительную работу. Он управляет взаимодействием между всеми блоками и системами компьютера. Именно к ЦП стягиваются все магистрали компьютерной системы. ЦП находится в функцио-



нальном центре компьютерной системы, окруженном аппаратными функциональными блоками и подсистемами. Вообще говоря, процессор — отдельный модуль компьютерной системы, реализующий определенные вычислительные и обменные процессы. В компьютере может быть несколько процессоров. Одни из них управляют вводом-выводом данных и называются *процессорами ввода-вывода*. Другие процессоры выполняют вычисления с вещественными числами и называются *математическими сопроцессорами*. Третьи генерируют изображения на экран монитора и называются *видеопроцессорами*. Но в любом персональном компьютере есть ЦП, который управляет всей компьютерной системой. Основными характеристиками процессора являются тактовая частота, определяющая его быстродействие, и разрядность (размер внутренних ячеек памяти).

5.4. Контроллер

Контроллеры предназначены для управления доступом из системы к какому-либо из устройств, а также для выполнения соответствующих операций информационного обмена. Каждое внешнее устройство имеет свой контроллер. После получения соответствующих команд от центрального процессора контроллер выполняет операции по обслуживанию внешнего устройства.

5.5. Шины

Все электронные элементы компьютера обмениваются информацией друг с другом и взаимосвязаны с помощью шин — совокупности линий и микросхем, осуществляющих передачу электрических сигналов определенного функционального назначения между различными компонентами ПК. Совокупность всех шин информационно-вычислительной системы называется *системной магистралью*.

По шинам передаются сигналы трех групп: адресные, управляющие и данные. Соответственно различают следующие шины.

Шина данных предназначена для передачи данных между электронными модулями ПК.

Шина адреса обеспечивает пересылку кодов адресной информации к оперативному запоминающему устройству (ОЗУ) или электронным модулям ПК для доступа к ячейкам памяти или к устройствам ввода-вывода.

Шина управления включает линии, по которым передаются сигналы управления, обмена, запросы на прерывания, передачи управления, синхронизации и т.д.

Шины характеризуются *разрядностью*, т.е. количеством линий, составляющих шину. Другое определение разрядности — количество одновременно передаваемых по линиям шины битов информации.

Бывают шины *последовательные* и *параллельные*. Последовательная шина состоит из одной линии данных, которые передаются последовательно, бит за битом, и нескольких линий управления и адреса. Параллельная шина состоит из нескольких линий данных, адреса и управления. *Пропускной способностью* шины называется количество информации, которое может быть передано по каналу за единицу времени.

5.6. Видеосистема

Видеоадаптер (видеокарта, видеопамять) служит для хранения изображения, передаваемого на монитор. Он нужен для того, чтобы процессору не было необходимости постоянно формировать изображение и он мог бы заниматься другими задачами, отвлекаясь на работу с видеоизображением только по мере надобности.

Монитор компьютера предназначен для вывода на экран текстовой и графической информации. Мониторы бывают цветные и монохромные, т.е. использующие для создания изображения на черном фоне точки одного цвета различной яркости. Они могут работать в текстовом и графическом режиме. Текстовый режим отличается очень экономным расходованием системных

ресурсов. Экран монитора условно разбивается на отдельные участки, чаще всего 25 строк на 80 символов. В каждом участке может быть выведен один из заранее заданных символов.

Графический режим монитора предназначен для вывода на экран рисунков, графиков, анимации и т.д. Экран монитора состоит из точек (пикселов). Каждая точка может быть темной, светлой, либо иметь один из цветов. На цветных мониторах для создания точки нужного цвета используют три стоящих рядом точки красного, синего и зеленого цветов. Меняя их яркость можно добиться любого необходимого цвета. Количество точек на экране называют *разрешающей способностью* или *разрешением* монитора. Например: 800x600 означает, что у монитора 800 точек по горизонтали и 600 по вертикали. Еще одной важной характеристикой монитора является его размер, измеряемый в дюймах.

5.7. Система памяти

Система памяти ПК включает следующие компоненты: оперативную память, сверхоперативную память, кэш-память, память базовой системы ввода-вывода и память системного конфигуратора и часов реального времени.

Оперативная память — основная память компьютера, предназначенная для хранения текущих данных и выполняемых программ, а также копий отдельных модулей операционной системы. Большинство программ в процессе выполнения резервируют часть ОЗУ для хранения своих данных. К данным, хранящимся в ОЗУ, ЦП может обращаться непосредственно, используя хост-шину. После отключения питания компьютера все содержимое ОЗУ стирается.

Каждая ячейка ОЗУ может хранить данные объемом в один байт и имеет свой уникальный адрес. Для удобства управления оперативная память разбивается на банки (memory banks). Ёмкость и разрядность используемых в банках микросхем зависит от конструкции материнской платы.

На первых ЭВМ объем оперативной памяти составлял всего 1 Мб. Причем под программы пользователя отводилось только 640 Кб. (так называемая основная память). Остальная часть памяти резервировалась для служебных целей: для передачи изображения на экран; для обеспечения взаимодействия с дополнительными устройствами компьютера и др. Эта часть памяти носит название *верхней памяти*.

Отдельной строкой стоит **BIOS - базовая система ввода-вывода (Base Input-Output System)**. Ее основные функции:

- инициализация и начальное тестирование аппаратных средств;

- настройка и конфигурирование аппаратных средств и системных ресурсов;
- выполнение основных низкоуровневых операций ввода-вывода;
- загрузка операционной системы.

В настоящий момент разработано очень много различных типов дополнительных устройств (например, много различных видов жестких дисков). Если машине каждый раз придется определять, какие устройства с какими параметрами присутствуют в схеме, то подготовка ЭВМ к работе заняла бы слишком много времени, поэтому некоторые характерные особенности устройств (в частности дисководов и жестких дисков) хранятся в BIOS.

Вся оперативная память, кроме BIOS, является энергозависимой, т.е. при выключении компьютера ее содержимое очищается. Только BIOS питается от специальных батареек или аккумуляторов, что гарантирует сохранение настроек.

Микросхемы памяти имеют четыре основные характеристики:

- тип (обозначает статическую или динамическую память);
- объем (показывает емкость микросхемы);
- структуру (количество ячеек памяти и разрядность каждой из них);
- время доступа (характеризует скорость работы микросхемы памяти).

Сверхоперативное запоминающее устройство (СОЗУ) расположено внутри самого ЦП. Это память очень маленького объема (всего несколько десятков байт) без которой ПК работать не будет. Дело в том, что после включения компьютера операционная система загружает в СОЗУ всю информацию, необходимую для работы процессора. Можно сказать, что операционная система программирует процессор. В процессе работы ЦП сохраняет в СОЗУ результаты своей вычислительной деятельности.

Кэш-память выполняет промежуточное хранение данных при обмене между ЦП и ОЗУ и служит как бы «карманом» между ними. Она позволяет повысить скорость информационного обмена в целом.

Кэш работает следующим образом. Все свои запросы ЦП адресует одновременно ОЗУ и кэш. Если адреса обращения идентичны, то констатируется попадание в кэш. Тогда данные или команды считываются из более быстрой памяти, что повышает общую производительность ПК. Если кэш не располагает информацией по искомому адресу, процессор обращается за ней к ОЗУ. В таком случае скорость обмена данными замедляется. Однако параллельно с передачей данных в ЦП весь блок данных, содержащий требуемое, перемещается в кэш, поскольку существует большая вероятность, что либо это данное, либо находящееся в памяти рядом с ним в скором времени снова понадобится процессору. Поэтому в следующий раз затребованные данные уже

будут считываться из кэша. По мере заполнения из него удаляются давно или редко используемые данные. Таким образом, современные ЦП — высокотехнологичные электронные изделия, выполняющие сотни миллионов операций в секунду и позволяющие ПК решать очень сложные задачи за короткие промежутки времени.

Память часов реального времени и конфигурационных установок системы предназначена для организации часов реального времени и хранения сведений о системе. Содержимое этой памяти не должно уничтожаться после выключения компьютера, поэтому она имеет постоянное питание от автономного источника — аккумулятора. Данные в этой памяти пользователь может прочитать или изменить с помощью программы Setup. Часы реального времени используют для хранения информации 14 байт. Информация о конфигурации системы хранится в 114 байтах.

5.8. Накопители

Накопители в зависимости от способа записи на носитель делятся на:

- накопители на магнитной ленте;
- накопители на дисках;
- флеш-носители.

Различают ленточные накопители на:

- катушечной магнитной ленте;
- кассетах с магнитной лентой.

Дисковые накопители делятся на:

- накопители на гибких магнитных дисках (НГМД, FDD, дискеты);
- накопители на жеских магнитных дисках (НЖМД, HDD, винчестер);
- оптические диски (CD-ROM, DVD-ROM);
- записываемые диски (CD-R, DVD-R);
- многократно перезаписываемые диски (CD-RW, DVD-RW).

Ленточные накопители используются достаточно редко, в основном для хранения больших архивов данных, чтобы в случае выхода компьютера из строя можно было после ремонта восстановить исходные программы и данные.

НГМД или дискеты служат для переноса информации с одного компьютера на другой. Сейчас используются дискеты размером 3,5 дюйма (89 мм), их ёмкость обычно составляет 1,44 Мб. На некоторых компьютерах еще можно встретить и 5,25-дюймовые дисководы (133 мм), но, имея плохую защиту от физических повреждений, они гораздо чаще выходят из строя. Из-за малой емкости дискеты почти вышли из употребления.

НЖМД предназначены для постоянного хранения информации, используемой при работе с компьютером. Его часто называют *винчестером.* Одной из главных характеристик жесткого диска является его объем, измеряемый в мегабайтах. Вторая существенная для пользователя характеристика жесткого диска — время доступа к информации. От него зависит, насколько быстро будет производиться работа с диском.

Компакт-диски удобны как для переноса информации, так и для её постоянного хранения. Невысокая цена и простота работы с ними сделали их одним из наиболее популярный сейчас видов носителей. Однако появившиеся недавно флеш-носители составляют им серьёзную конкуренцию.

Глава 6. Сетевые технологии

6.1. Топология сети

Термин «топология» или «топология сети» характеризует физическое расположение компьютеров, кабелей и других компонентов сети. Топология — стандартный термин, используемый профессионалами при описании основной компоновки сети. Кроме термина «топология» для описания физической компоновки употребляют также следующие:

- физическое расположение;
- компоновка;
- диаграмма;
- карта.

Многие характеристики сети определяются её топологией. В частности, выбор той или иной топологии влияет на:

- состав необходимого сетевого оборудования;
- характеристики сетевого оборудования;
- возможности расширения сети;
- способ управления сетью.

Чтобы совместно использовать ресурсы или выполнять другие сетевые задачи, компьютеры должны быть подключены друг к другу, для чего в большинстве сетей применяется кабель.

Однако просто подключить компьютер к кабелю, соединяющему другие компьютеры, недостаточно. Различные типы кабелей в сочетании с различными сетевыми платами, сетевыми операционными системами и другими компонентами требуют и различного взаимного расположения компьютеров.

Каждая топология сети налагает ряд условий. Например, она может диктовать не только тип кабеля, но и способ его прокладки. Топология может также определять способ взаимодействия компьютеров в сети. Различным видам топологий соответствуют различные методы взаимодействия, которые оказывают большое влияние на сеть.

6.2. Базовые топологии

Все сети строятся на основе трех базовых топологий:

- шина (bus);
- звезда (star);
- кольцо (ring).

Если компьютеры подключены вдоль одного кабеля [сегмента (segment)], топология называется *шиной*. В том случае, когда компьютеры подключены к сегментам кабеля, исходящим из одной точки, или концентратора, топология называется *звездой*. Если кабель, к которому подключены компьютеры, замкнут в кольцо, такая топология носит название *кольца*. Хотя сами по себе базовые топологии просты, в реальности часто встречаются довольно сложные их комбинации, объединяющие свойства нескольких топологий.

6.2.1. Шина

Топологию «шина» (рис. 6.1) часто называют «линейной шиной» (linear bus). Данная топология относится к наиболее простым и широко распространенным топологиям. В ней используется один кабель, именуемый *магистралью* или *сегментом*, вдоль которого подключены все компьютеры сети.

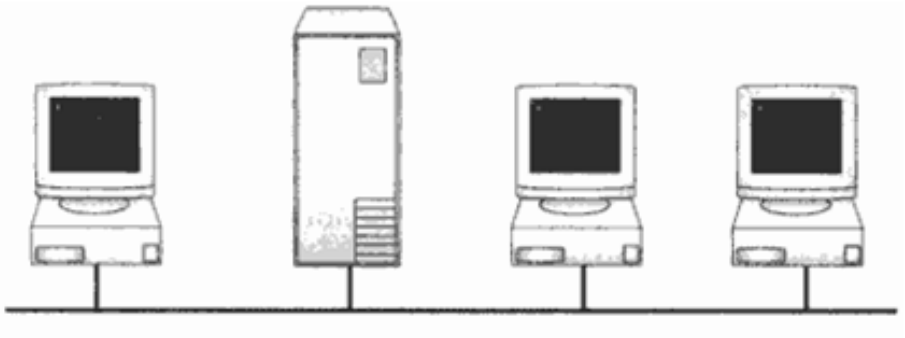


Рис. 6.1

Взаимодействие компьютеров. В сети с топологией «шина» компьютеры адресуют данные конкретному компьютеру, передавая их по кабелю в виде электрических сигналов. Чтобы понять процесс взаимодействия компьютеров по шине, нужно уяснить следующие понятия:

- передача сигнала;
- отражение сигнала;
- терминатор.

Передача сигнала. Данные в виде электрических сигналов передаются всем компьютерам сети, однако информацию принимает только тот, адрес которого соответствует адресу получателя, зашифрованному в них. Причем в каждый момент времени только один компьютер может вести передачу.

Так как данные в сеть передаются лишь одним компьютером, ее производительность зависит от количества компьютеров, подключенных к шине. Чем их больше, т.е. чем больше компьютеров, ожидающих передачи данных, тем

медленнее работает сеть. Однако вывести прямую зависимость между пропускной способностью сети и количеством компьютеров в ней нельзя. Кроме числа компьютеров на быстродействие сети влияет множество факторов, в том числе:

- характеристики аппаратного обеспечения компьютеров в сети;
- частота, с которой компьютеры передают данные;
- тип работающих сетевых приложений;
- тип сетевого кабеля;
- расстояние между компьютерами в сети.

Шина — пассивная топология, т.е. компьютеры только «слушают» передаваемые по сети данные, но не перемещают их от отправителя к получателю. Поэтому, если один из компьютеров выйдет из строя, это не скажется на работе остальных. В активных топологиях компьютеры регенерируют сигналы и передают их по сети.

Отражение сигнала. Данные или электрические сигналы распространяются по всей сети от одного конца кабеля к другому. Если не предпринимать никаких специальных действий, сигнал, достигая конца кабеля, будет отражаться и не позволит другим компьютерам осуществлять передачу. Поэтому, после того как данные достигнут адресата, электрические сигналы необходимо погасить.

Терминатор. Чтобы предотвратить отражение электрических сигналов, на каждом конце кабеля устанавливают терминаторы (terminators), поглощающие эти сигналы (рис. 6.2). Все концы сетевого кабеля должны быть к чему-нибудь подключены, например, к компьютеру или к баррел-коннектору для увеличения длины кабеля. К любому свободному, неподключенному концу кабеля должен быть подсоединен терминатор, чтобы предотвратить отражение электрических сигналов.

Нарушение целостности сети. Разрыв сетевого кабеля происходит при его физическом разрыве или отсоединении одного из его концов. Возможна также ситуация, когда на одном или нескольких концах кабеля отсутствуют терминаторы, что приводит к отражению электрических сигналов в кабеле и прекращению функционирования сети. Сеть «падает». Сами по себе компьютеры в сети остаются полностью работоспособными, но до тех пор, пока сегмент разорван, они не могут взаимодействовать друг с другом.

6.2.2. Звезда

При топологии «звезда» все компьютеры с помощью сегментов кабеля подключаются к центральному компоненту, именуемому концентратором (hub). Сигналы от передающего компьютера поступают через концентратор ко всем

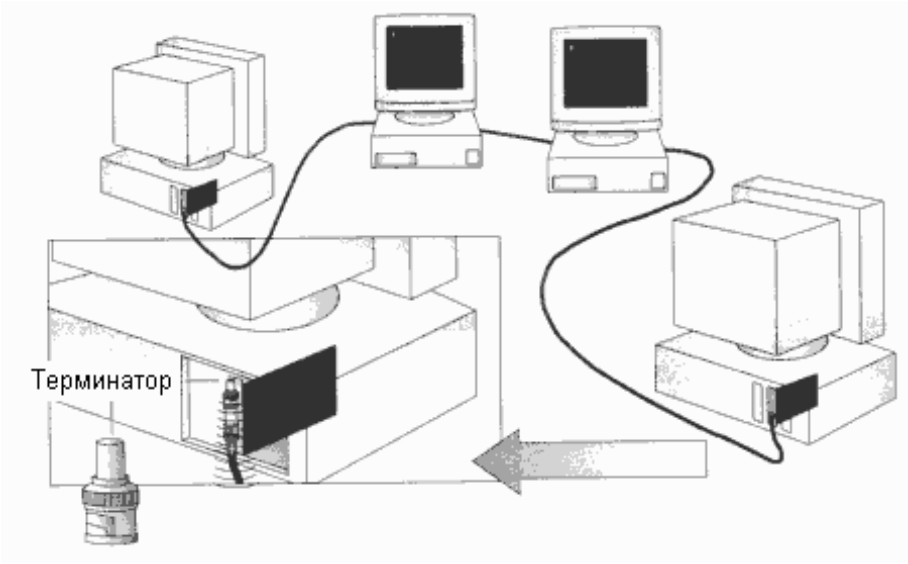


Рис. 6.2

остальным. Такая топология возникла на заре вычислительной техники, когда компьютеры были подключены к центральному, главному компьютеру (рис. 6.3).

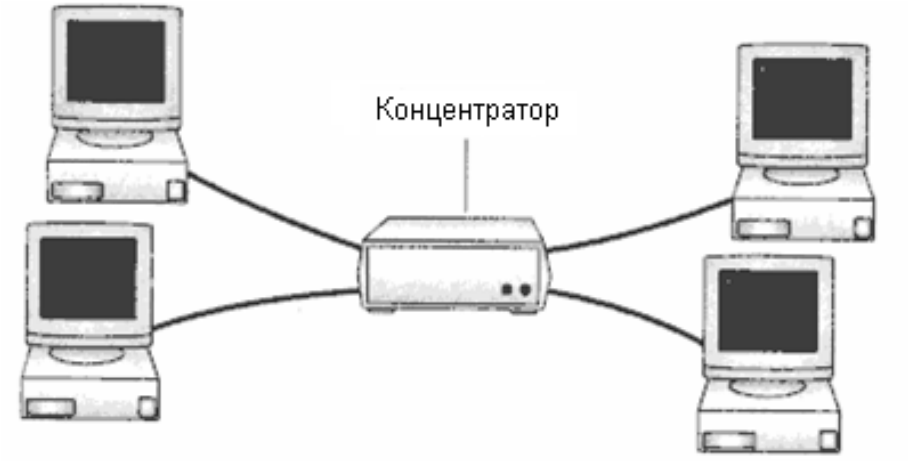


Рис. 6.3

В сетях с топологией «звезда» подключение кабеля и управление конфигурацией сети централизованны. Но есть и недостаток: так как все компьютеры подключены к центральной точке, для больших сетей значительно увеличивается расход кабеля. К тому же, если центральный компонент выйдет из строя, нарушится работа всей сети. А если выйдет из строя только один компьютер (или кабель, соединяющий его с концентратором), то лишь он не сможет передавать или принимать данные по сети. На остальные компьютеры в сети это не повлияет.

6.2.3. Кольцо

При топологии «кольцо» компьютеры подключаются к кабелю, замкнутому в кольцо (рис. 6.4). У кабеля просто не может быть свободного конца, к которому надо подключать терминатор. Сигналы передаются по кольцу в одном направлении и проходят через каждый компьютер. В отличие от пассивной топологии «шина», здесь каждый компьютер выступает в роли репитера, усиливая сигналы и передавая их следующему компьютеру. Поэтому, если выйдет из строя один компьютер, прекращает функционировать вся сеть.

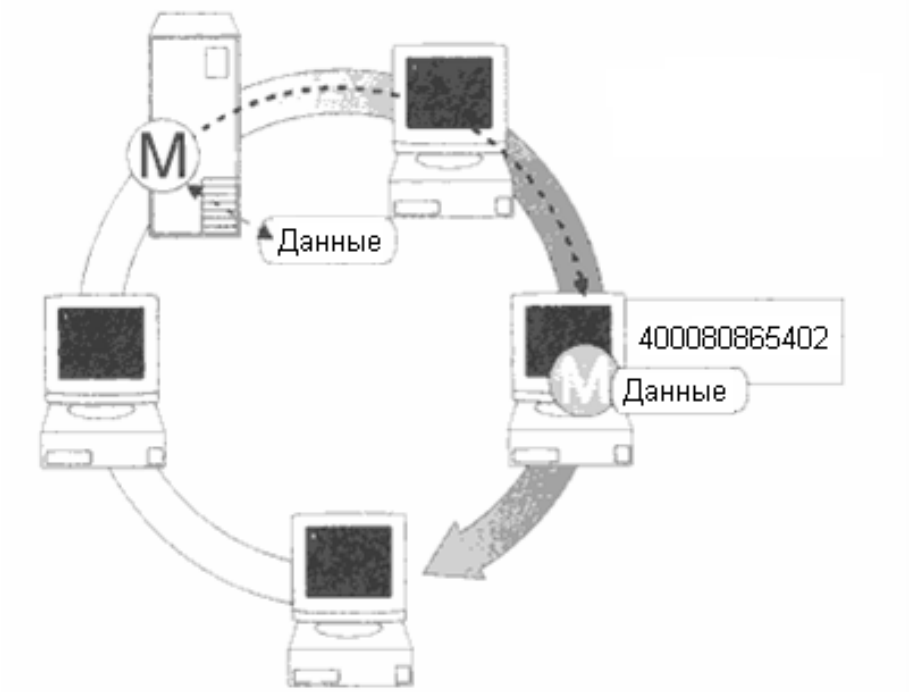


Рис. 6.4

Передача маркера. Один из принципов передачи данных в кольцевой сети носит название *передачи маркера*. Суть его такова. Маркер последовательно, от одного компьютера к другому, передается до тех пор, пока его не получит тот, который «хочет» передать данные. Передающий компьютер изменяет маркер, помещает электронный адрес в данные и посылает их по кольцу.

Данные проходят через каждый компьютер, пока не окажутся у того, чей адрес совпадает с адресом получателя, указанным в данных. Затем принимающий компьютер посылает передающему сообщение, где подтверждает факт приёма данных. Получив подтверждение, передающий компьютер создаёт новый маркер и возвращает его в сеть. На первый взгляд кажется, что передача маркера отнимает много времени, однако на самом деле маркер передвигается с очень большой скоростью. В кольце диаметром 200 м маркер может циркулировать с частотой 10 000 оборотов в секунду.

6.3. Модель ISO/OSI

В модели ISO/OSI сетевые функции распределены между семью уровнями, изображенными на рис. 6.5.

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Рис. 6.5

В разработке эталонной модели участвовали семь комитетов, и для каждого из них был создан один уровень. Схема OSI — не просто абстрактная модель. Её сопровождает реальный набор «стандартных» протоколов. Создание системы OSI началось в первой половине 80-х гг. и растянулось на многие годы. Пока комитеты ISO спорили о своих стандартах, за их спиной менялась вся концепция организации сетей, по всему миру внедрялся протокол TCP/IP.

Каждому уровню соответствуют различные сетевые операции, оборудование и протоколы, выполняются определенные сетевые функции, которые взаимодействуют с функциями соседних уровней, вышележащего и нижележащего. Например, сеансовый уровень должен взаимодействовать только с представительным и транспортным. Рассмотрим уровни модели подробней.

Уровень 7. Прикладной (Application Layer) — самый верхний уровень модели OSI. Он представляет собой окно для доступа прикладных процессов к сетевым услугам. Прикладной уровень обеспечивает доступ прикладных процессов в среде OSI. Функции прикладного уровня разделяются на две группы: общие и специальные. Первые дают средства взаимодействия, используемые различными приложениями, например, средства организации связи между прикладными процессами. Вторые обеспечивают определенные потребности конкретных приложений, например, обмен файлами, доступ к базам данных и электронную почту.

Уровень 6. Представительный (Presentation Layer) уровень предназначен для представления данных, подлежащих передаче между прикладными объектами, структур данных, на которые ссылаются прикладные объекты, методов, которые могут использоваться для манипулирования и обработки данных. Представительный уровень имеет дело с синтаксисом, т.е. с формальным представлением данных. Семантика, т.е. способ интерпретации данных, их смысл — прерогатива только прикладного уровня. Наличие представительного уровня освобождает приложения от необходимости заботиться о проблеме общего представления данных и обеспечивает независимость от синтаксиса. Это позволяет прикладным объектам использовать любой локальный синтаксис, представительный уровень обеспечивает преобразование локальных синтаксисов в согласованный обоими прикладными объектами. Преобразования синтаксисов выполняются локально и видны для других о́ткрытых систем. Поэтому представительные протоколы не стандартизируются.

Функции представительного уровня включают:

- запрос на установление сеанса;
- передачу данных;
- согласование и пересогласование выбора синтаксиса;
- преобразование синтаксиса, включая преобразование данных;
- форматирование и специальные преобразования (сжатие, шифрование/дешифрование).

Сущность второй и третьей функции заключается в следующем. Существует три варианта синтаксиса данных: синтаксис отправителя, синтаксис получателя и синтаксис, используемый объектами представительного уровня (синтаксис передачи). Любые два из них могут быть идентичными. Уровень представления содержит функции, необходимые для преобразования между синтаксисом передачи и каждым из синтаксисов прикладных объектов по мере необходимости. Единого синтаксиса передачи для всей OSI не существует, поэтому представительные объекты-корреспонденты согласуют синтаксис в процессе установления соединения. Представительный объект должен знать синтаксис своего прикладного объекта и согласованный синтаксис передачи. Согласование синтаксиса передачи осуществляется в процессе диалога между объектами представительного уровня либо в процессе установления соединения, либо в любое время в процессе передачи данных.

Представительный уровень отвечает за преобразование протоколов, трансляцию данных, их шифрование, смену или преобразование применяемого набора символов (кодовой таблицы) и расширение графических команд. Он, кроме того, управляет сжатием данных для уменьшения количества передаваемых битов.

Уровень 5. Сеансовый (Session Layer) уровень предназначен для организации и синхронизации диалога и управления обменом данными. С этой целью уровень предоставляет услуги по установлению сеансового соединения между двумя представительными объектами и поддержанию упорядоченного взаимодействия при обмене данными между ними. При этом сеанс отображается на транспортное соединение и использует его. Сеанс может быть расторгнут сеансовыми или представительными объектами. Функции сеансового уровня сводятся к установлению и расторжению сеансового соединения, обмену нормальными и срочными данными, управлению взаимодействием, синхронизации и восстановлению сеанса. Перечисленные функции тесно связаны с сеансовым сервисом, поскольку собственные, не инициированные со стороны верхнего уровня действия практически отсутствуют. Синхронизацию между пользовательскими задачами сеансовый уровень обеспечивает посредством расстановки в потоке данных контрольных точек (chekpoints). Таким образом, в случае сетевой ошибки потребуется заново передать только данные, следующие за последней контрольной точкой. Выполняется управление диалогом между взаимодействующими процессами, т.е. регулируется, какая из сторон осуществляет передачу, когда, как долго и т.д.

Уровень 4. Транспортный (Transport Layer) уровень обеспечивает прозрачную передачу данных между сеансовыми объектами и освобождает их от функций, связанных с надежностью и эффективностью передачи данных. Уровень оптимизирует использование имеющихся сетевых ресурсов, представляя транспортный сервис при минимальной стоимости. Оптимизация выполняется при ограничениях, накладываемых всеми взаимодействующими в пределах сети сеансовыми объектами, с одной стороны, и возможностях и параметрах сетевого сервиса, который используется транспортным уровнем, с другой. Протоколы транспортного уровня предназначены для межконцевого (point-to-point) взаимодействия, где концы определяются как транспортные объекты-корреспонденты. Транспортный уровень освобождается от маршрутизации и ретрансляции и занимается исключительно обеспечением взаимодействия между оконечными открытыми системами. Транспортные функции зависят от сетевого сервиса и включают:

- отображение транспортного адреса на сетевой адрес;
- мультиплексирование и расщепление транспортных соединений на сетевые;
- установление и расторжение транспортных соединений;
- управление потоком на отдельных соединениях;
- обнаружение ошибок и управление качеством сервиса;
- исправление ошибок;

- сегментирование, блокирование и сцепление;
- передачу срочных блоков данных.

Транспортный уровень гарантирует доставку пакетов без ошибок, в той же последовательности, без потерь и дублирования. На этом уровне сообщения переупаковываются: длинные разбиваются на несколько пакетов, а короткие объединяются в один, что увеличивает эффективность передачи пакетов по сети. На транспортном уровне узла-получателя сообщения распаковываются, восстанавливаются в первоначальном виде и посылается сигнал подтверждения приема. Транспортный уровень управляет потоком, проверяет ошибки и участвует в решении проблем, связанных с отправкой и получением пакетов.

Уровень 3. Сетевой (Network Layer) уровень отвечает за адресацию сообщений и перевод логических адресов и имен в физические адреса. Сетевой уровень обеспечивает установление, поддержание и разъединение сетевых соединений между системами, содержащими взаимодействующие прикладные объекты, а также предоставляет функциональные и процедурные средства для блочного обмена данными между транспортными объектами по сетевым соединениям.

Сетевой уровень определяет маршрут от транспортного объекта-отправителя к транспортному объекту-получателю и обеспечивает независимость от особенностей маршрутизации и ретрансляции, связанных с установлением и использованием данного сетевого соединения. Это тот случай, когда несколько подсетей используются последовательно или параллельно.

Здесь решаются такие задачи и проблемы, связанные с сетевым трафиком, как коммутация пакетов, маршрутизация и перегрузки. Если сетевой адаптер маршрутизатора не может передавать большие блоки данных, посланные компьютером-отправителем, на сетевом уровне они разбиваются на меньшие. Сетевой уровень компьютера-получателя собирает эти данные в исходное состояние.

Функции сетевого уровня:

- маршрутизация и ретрансляция;
- организация сетевых соединений;
- мультиплексирование сетевых соединений на канальное соединение;
- сегментирование и блокирование;
- обнаружение и исправление ошибок;
- управление потоком;
- передача срочных данных;
- возврат к исходному состоянию.

Сетевые соединения могут иметь различную конфигурацию: от простого двухточечного соединения до сложной комбинации подсетей с различными характеристиками. Обычно сетевые функции разделяются на подуровни. Более подробно такое разделение описано в оригинальных документах ISO, описывающих модель OSI.

Уровень 2. Канальный (Data Link Layer) уровень осуществляет передачу кадров (frames) данных от сетевого уровня к физическому. Кадры — это логически организованная структура, в которую можно помещать данные. Канальный уровень узла-получателя упаковывает «сырой» поток битов, поступающих от физического уровня, в кадры данных.

Канальный уровень обеспечивает функциональные и процедурные средства для установления, поддержания и расторжения канальных соединений между сетевыми объектами и передачи блоков данных. Канальное соединение (канал передачи данных) строится на одном или нескольких физических соединениях.

Канальный уровень обнаруживает и в большинстве случаев исправляет ошибки, которые могут возникнуть на физическом уровне, что позволяет сетевому уровню считать передачу данных по сетевому соединению фактически безошибочной. Помимо этого канальный уровень позволяет сетевому управлять взаимными соединениями физических каналов. Обычно, когда канальный уровень посылает кадр, он ожидает со стороны получателя подтверждения приема. Канальный уровень получателя проверяет наличие возможных ошибок передачи. Кадры, получение которых не подтверждено, а также поврежденные при передаче, посылаются вторично.

Функции канального уровня:

- установление и расторжение канального соединения;
- расщепление канального соединения на несколько физических;
- обнаружение и исправление ошибок;
- управление потоком данных;
- управление соединением физических каналов передачи данных.

Уровень 1. Физический (Physical Layer) — самый нижний уровень в модели OSI. Он осуществляет передачу неструктурированного, «сырого» потока битов по физической среде (например, по сетевому кабелю). Физический уровень обеспечивает механические, электрические, функциональные и процедурные средства активизации, поддержания и деактивизации физических соединений для передачи данных между канальными объектами. Функции уровня сводятся к активизации и деактивизации физического соединения, а также передачи данных. Здесь реализуются электрический, оптический, механический и функциональный интерфейсы с кабелем. Физический уровень формирует сигналы, которые переносят данные, поступившие от всех

вышележащих уровней. На этом уровне определяется способ соединения сетевого кабеля с платой сетевого адаптера, в частности, количество контактов в разъемах и их функции. Кроме того, здесь определяется способ передачи данных по сетевому кабелю. Физический уровень предназначен для передачи битов (нулей и единиц) от одного компьютера к другому. Уровень отвечает за кодирование данных и синхронизацию битов, гарантируя их достоверность, устанавливает длительность каждого бита и способ его перевода в соответствующие электрические или оптические импульсы, передаваемые по сетевому кабелю.

Нижние уровни модели (1-й и 2-й) определяют физическую среду передачи данных и сопутствующие задачи, такие как передача битов данных через плату сетевого адаптера и кабель, самые верхние — способ осуществления доступа приложений к услугам связи. Чем выше уровень, тем более сложную задачу он решает. Каждый уровень предоставляет несколько услуг (т.е. выполняет несколько операций), подготавливающих данные для доставки по сети на другой компьютер. Уровни отделяются друг от друга границами — интерфейсами. Все запросы от одного уровня другому передаются через интерфейс. Каждый уровень использует услуги нижележащего уровня и предоставляет услуги вышележащему уровню, маскируя детали реализации этих услуг. А все вместе они обеспечивают передачу информации.

Глава 7. Основные композиции действий и их правила вывода

7.1. Соотношения для корректности программ

Каждый алгоритм имеет как статическую, так и динамическую структуру. Статическая структура представляется текстом (или структурной схемой) программы, описывающим действия и проверки, которые должны быть выполнены при решении данной задачи. Текст не зависит от значений исходных данных. Напротив, динамическая структура в значительной степени определяется выбором исходных данных, поскольку при выполнении алгоритма будут сделаны различные переходы в зависимости от значений входных переменных. Динамическая структура отражает процесс вычислений и состоит из последовательности состояний вычислений.

Состояние вычислений в любой момент времени включает в себя значения всех программных переменных в этот момент и, таким образом, зависит от начальных значений переменных и этапов вычислений алгоритма, которые ему предшествовали. Текущая инструкция или изменяет состояние вычислений (значения некоторых переменных) и передает управление следующей инструкции в последовательности, или производит проверку состояния вычислений (сравнивает значения определенных переменных) и на основе её результата передает управление другой инструкции. Пример алгоритма нахождения $x\ div\ y$ и $x\ mod\ y$ с приведенными состояниями вычислений представлен на рис. 7.1.

Язык программирования предоставляет как простые операторы, так и методы композиции, которые позволяют формировать структурные операторы из других простых или составных операторов. Все это ставит две связанные между собой задачи.

1. Определить виды используемых в языке C++ простых операторов, а также часто применяемые методы композиции решений подзадач.
2. Обеспечить правила вывода, позволяющие определить эффект воздействия простого оператора на состояние вычисления, а также вывести определенные свойства составного оператора из свойств составляющих его компонент.

Рассмотрим подробнее рис. 7.1. Заметим, что состояние вычислений, связанное с точкой входа, определяется значениями входных переменных x и y, но после выполнения первых двух операторов состояние вычислений расширяется, включая значения четырех переменных: x, y, q и r. (Значения x и y

могут быть различными в зависимости от применения алгоритма, но остаются постоянными в течение одного применения). Ясно, что следующая за проверкой $r \geq y$ передача управления будет зависеть от текущего состояния вычислений.

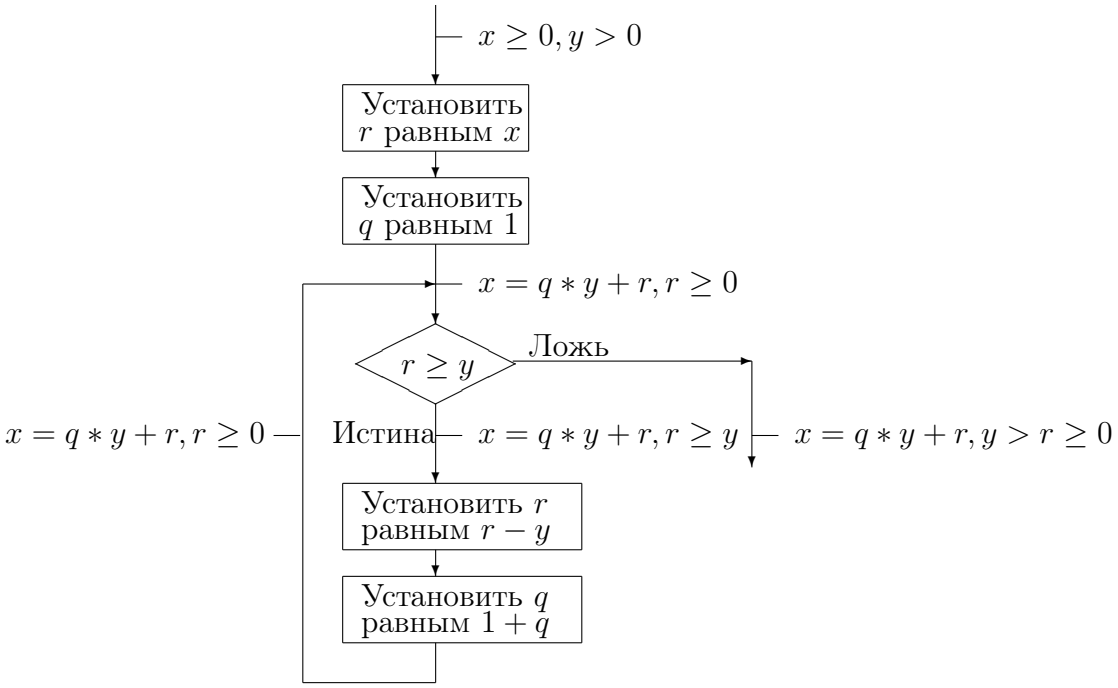


Рис. 7.1

Фундаментальное свойство всех видов композиции, которые будут рассматриваться, заключается в том, что они дают возможность объединить в одну сложную структурную схему с одним входом и одним выходом одну или более схем также с одним входом и одним выходом. Как показано на рис. 7.1, процедура проектирования объединяет с вышеупомянутыми точками некоторые соотношения, которые описывают существенные аспекты состояний вычислений в этих точках. Каждая такая структурная схема имеет вид, приведенный на рис. 7.2, где S может быть отдельным действием ЭВМ или сложной схемой. Чтобы выразить ее в более краткой форме, используется следующая нотация:

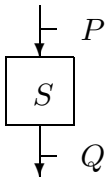


Рис. 7.2

$$\{P\}S\{Q\}. \tag{7.1}$$

Соотношение (7.1) — спецификация программы со следующим смыслом: если соотношение P истинно перед выполнением S, то Q будет истинно после завершения выполнения S. Другими словами, если P истинно на входе, то Q будет истинно на выходе.

Если S — программа, корректность которой устанавливается, то нотация (7.1) — это то, что требуется доказать, причем P — соотношение, которому должны удовлетворять начальные значения переменных, а Q — соотношение, которому должны удовлетворять конечные значения переменных. Например, если S — алгоритм, изображенный на рис. 7.1, то соотношение, которое надо доказать, таково: $\{(x \geq 0) \wedge (y > 0)\}S\{(x = q * \mathrm{y} + r) \wedge (0 \leq r < \mathrm{y})\}$, где знак $\wedge$ используется в качестве формальной нотации для «и».

Как уже отмечалось, соотношение $\{P\}S\{Q\}$ означает, что если P истинно перед выполнением S, то Q должно быть истинно при завершении S. Это означает, что $\{P\}S\{Q\}$ тождественно истинно, если S не завершается, т. е. если на рис. 7.2 точка выхода никогда не достигается. Другими словами, (7.1) определяет только частичную корректность S. Частично корректной является такая программа, в которой гарантируется получение требуемого результата при условии ее завершения. Но действительно ли завершается программа для некоторых исходных значений — другой вопрос. Если дополнительно можно показать, что программа завершается для всех исходных значений, удовлетворяющих соотношению P, то говорят, что программа полностью корректна. Чтобы доказать завершение программы, получаемой применением правил композиции, введенных ранее, необходим анализ циклов, т.е. операторов итерации.

Можно подвести итог рассматриваемому подходу к проектированию программ.

1. Проектирование должно начинаться со спецификации $\{P\}S\{Q\}$, которой должна удовлетворять проектируемая программа, т.е. с ясного и недвусмысленного определения того, когда программа должна использоваться (предусловие P), и результата ее использования при этом (постусловие Q).
2. Процесс проектирования сверху вниз определяет спецификации $\{P_i\}S_i\{Q_i\}$ для компонентов S_i, из которых строится программа.
3. Проектирование программы осуществляется одновременно с доказательством корректности указанных спецификаций.

7.2. Правила вывода для простых операторов

Правила вывода — схемы рассуждений, позволяющие доказывать свойства программ. Правила вывода имеют вид

$$\frac{H_1, \ldots, H_n}{H}. \tag{7.2}$$

Если $H_1, \ldots, H_n$ — истинные утверждения, то H — также истинное утверждение. Рассмотрим два простейших правила вывода. Первое утверждает,

что если выполнение программы S обеспечивает истинность утверждения R, то оно также обеспечивает истинность каждого утверждения, которое следует из R, т. е.

$$\frac{\{P\}S\{R\}, R \to Q}{\{P\}S\{Q\}} \tag{7.3}$$

Например, из двух выражений

$$\{(x > 0) \wedge (y > 0)\}S\{(z + u * y = x * y) \wedge (u = 0)\}, \tag{7.4}$$

$$(z + u * y = x * y) \wedge (u = 0) \to (z = x * y) \tag{7.5}$$

заключаем, используя 7.3, что

$\{(x > 0) \wedge (y > 0)\}S\{z = x * y\}$.

Второе правило утверждает, что если R — известное предусловие программы S, приводящей к результату Q после завершения своего выполнения, то это же относится к любому другому утверждению, из которого следует R:

$$\frac{P \to R, \{R\}S\{Q\}}{\{P\}S\{Q\}}. \tag{7.6}$$

Правила (7.3) и (7.6) называются *правилами консеквенции*. Теперь обратимся к специфическому виду, который принимает $\{P\}S\{Q\}$, если что-то известно о S. Рассмотрим случай, когда S — простой оператор. Простейшей формой простого оператора является пустой оператор — тот, который не оказывает никакого воздействия на значения программных переменных. Для любого P имеем правило вывода

$$\{P\}\{P\}. \tag{7.7}$$

Из простых операторов, оказывающих влияние на значения программных переменных и, следовательно, на булевы значения утверждений P и Q в $\{P\}S\{Q\}$, сначала рассмотрим присваивание. Итак, пусть $x = e$ есть оператор присваивания, который устанавливает x равным значению выражения e. Тогда можем сделать вывод, что для любого P

$$\{P_e^x\}x = e\{P\}. \tag{7.8}$$

Здесь утверждается, что если P истинно для подстановки e вместо x перед выполнением присваивания, то P должно быть истинным, когда переменной x присвоили ее новое значение. Это правило можно пояснить примерами, данными на рис. 7.3.

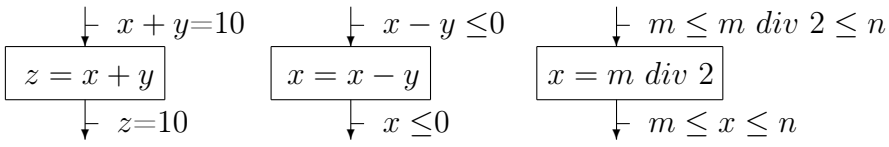


Рис. 7.3

Основу языков программирования составляют структурные операторы, позволяющие из простых операторов комбинировать достаточно сложные конструкции.

7.3. Составные и условные операторы

Предположим, мы хотим доказать, что имеет место $\{P\}S\{Q\}$, когда S является структурным оператором. Для каждого типа композиции операторов необходимо правило, позволяющее вывести свойства сложного (структурного) оператора на основе установленных свойств его компонент. Далее рассматриваются такие правила для составных и условных операторов.

7.3.1. Составные операторы

Простейшей формой структурирования является создание так называемых составных операторов путем последовательной композиции, состоящей из действия S_1, за которым следует действие S_2. Можно обобщить это определение последовательной композиции на случай произвольного конечного числа действий $S_1, S_2, \ldots, S_n$. В языке C++ последовательно соединенные операторы обычно заключают в скобки { и }, которые указывают, что полученный таким образом оператор является единым, хотя и структурным. Такой оператор имеет вид

$$\{S_1; S_2; \ldots; S_n;\} \tag{7.9}$$

и называется *составным оператором*. Он может быть представлен структурной схемой, изображенной на рис. 7.4.

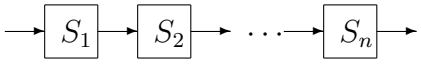


Рис. 7.4

Правило вывода для последовательной композиции гласит: если S есть $\{S_1; S_2;\}$ и если имеют место $\{P\}S_1\{R\}$ и $\{R\}S_2\{Q\}$, то истинно и $\{P\}\{S_1; S_2;\}\{Q\}$. Формально это правило может быть выражено следующим образом:

$$\frac{\{P\}S_1\{R\}, \{R\}S_2\{Q\}}{\{P\}\{S_1; S_2;\}\{Q\}}. \tag{7.10}$$

Правило вывода (7.10) обобщается следующим очевидным образом:

$$\frac{\{P_{i-1}\}S_i\{P_i\}\text{для } i = 1, \ldots, n}{\{P_0\}\{S_1; \ldots; S_n;\}\{P_n\}}. \tag{7.11}$$

7.3.2. Условные операторы

Если S_1 и S_2 — операторы, а B — логическое выражение, то

$$if(B)S_1; else\ S_2; \tag{7.12}$$

есть оператор, обозначающий следующее действие: вычисляется B; если его значение есть истина, то должно быть выполнено действие, описываемое S_1, в противном случае — действие, описываемое S_2 Выражение (7.12) может быть графически представлено структурной схемой, изображенной на рис. 7.5.

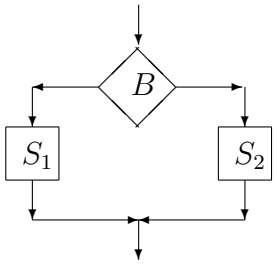


Рис. 7.5

Выработаем правило вывода для условного оператора (7.12). Если требуется установить истинность $\{P\}if(B)S_1; else\ S_2; \{Q\}$, то необходимо доказать два утверждения.

1. Если B истинно, то выполняется S_1. Так как P справедливо перед выполнением (7.12), то делаем вывод, что в этом случае $P \wedge B$ также справедливо перед выполнением S_1. Если Q справедливо после выполнения (7.12), то должно быть справедливо и $\{P \wedge B\}S_1\{Q\}$. Итак, мы должны доказать $\{P \wedge B\}S_1\{Q\}$.
2. Если B ложно, то будет выполняться S_2. Так как P было истинно перед выполнением (7.12), делаем вывод, что $P \wedge \neg B$ справедливо перед выполнением S_2. С этим утверждением в качестве предусловия S_2 требуется доказать, что после выполнения S_2 будет справедливо Q, т.е. доказать, что $\{P \wedge \neg B\}S_2\{Q\}$.

Если мы доказали как $\{P \wedge B\}S_1\{Q\}$, так и $\{P \wedge \neg B\}S_2\{Q\}$, то можно утверждать, что если P справедливо перед выполнением (7.12), то Q будет справедливо по окончании его выполнения независимо от того, какой оператор (S_1 или S_2) был выбран для выполнения.

Итак, можно сформулировать следующее правило вывода:

$$\frac{\{P \wedge B\}S_1\{Q\}, \{P \wedge \neg B\}S_2\{Q\}}{\{P\}if(B)S_1; else\ S_2; \{Q\}}. \tag{7.13}$$

Теперь рассмотрим второй вариант условного оператора и обеспечим для него правило вывода. Он получается, когда мы замечаем, что в (7.12) S_2 может быть любым и, в частности, пустым оператором. Пустой оператор определяет тождественную операцию, т. е. операцию, не воздействующую на значения переменных. Если S_2 — пустой оператор, то (7.12) запишется так:

$$if(B)S; \tag{7.14}$$

Действие, которое определяется оператором (7.14), вначале состоит в вычислении B. Если B истинно, то должно быть выполнено действие, определяемое S, иначе должна быть выполнена тождественная операция. Выражение (7.14) представлено структурной схемой на рис. 7.6.

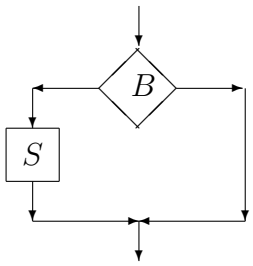


Рис. 7.6

Теперь определим соответствующее правило вывода. Если мы хотим доказать $\{P\}if(B)S;\{Q\}$, то необходимо обеспечить два условия. Первое из них заключается в том, что если S выполняется, то Q справедливо после его завершения. Так как при условии истинности B для выполнения выбирается S, то выводим, что $P \wedge B$ справедливо перед выполнением S, если P справедливо перед выполнением (7.14). Итак, требуется доказать $\{P \wedge B\}S\{Q\}$. Второй случай имеет место, когда S не выполняется, т.е. когда после вычисления B получается значение ложь. Итак, в этой точке справедливо $P \wedge \neg B$, и если Q справедливо после выполнения (7.14), то необходимо доказать, что $P \wedge \neg B \rightarrow Q$.

Правило вывода формулируется следующим образом:

$$\frac{\{P \wedge B\}S\{Q\}, P \wedge \neg B \to Q}{\{P\}if(B)S;\{Q\}}. \tag{7.15}$$

Рассмотрев составные и условные операторы переходим к исследованию циклических конструкций.

7.4. Операторы итерации

Теперь обратимся к структурам, которые приводят к циклам. Сначала изучим конструкцию *while*, а затем *do-while*.

7.4.1. Оператор while

Если B — логическое выражение и S — оператор, то

$$while(B)S; \tag{7.16}$$

обозначает итерационное выполнение оператора S, пока B истинно. Если B ложно с самого начала, то S не будет выполняться совсем. Процесс итерационного выполнения S оканчивается, когда B становится ложным. Эта ситуация изображена на рис. 7.7.

Задав любое начальное неотрицательное целое значение n, можно воспользоваться этим оператором для вычисления суммы $1^2 + 2^2 + \ldots + n^2$, получаемой как конечное значение h в следующей программе:

$$h = 0; while(n > 0)\{h+ = n * n; n - -;\} \tag{7.17}$$

Теперь необходимо изучить утверждение $\{P\}while(B)S;\{Q\}$, для чего рассмотрим рис. 7.7 более подробно, т. е. перейдем к рис. 7.8.

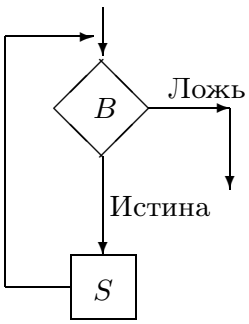


Рис. 7.7

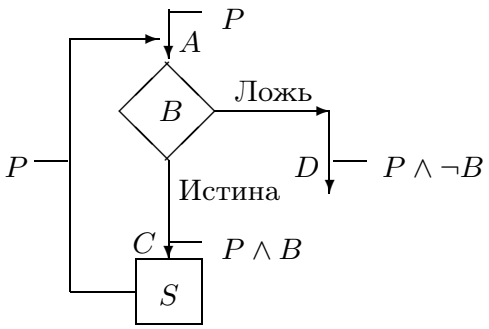


Рис. 7.8

Если P справедливо, когда мы впервые входим в цикл в точке A, то ясно, что $P \wedge B$ будет справедливо, если мы достигнем точки C. Тогда если мы

хотим быть уверенными, что P снова выполняется при возврате в точку A, нам необходимо обеспечить истинность для S утверждения $\{P \wedge B\}S\{P\}$.

В этом случае ясно, что P будет выполняться не только тогда, когда точка A на рис. 7.8 достигается первый или второй раз, но и после произвольного числа итераций. Аналогично $P \wedge B$ выполняется всякий раз, когда достигается точка C. Когда достигается точка выхода D (см. рис. 7.8), справедливо не только P, но и $\neg B$. Итак, получаем следующее правило вывода:

$$\frac{\{P \wedge B\}S\{P\}}{\{P\}while(B)S;\{P \wedge \neg B\}}. \tag{7.18}$$

Заметим, что (7.18) устанавливает свойство инвариантности P для цикла. Если P выполняется для начального состояния вычисления, то оно будет выполняться для состояния вычисления, соответствующего каждому прохождению цикла. P составляет сущность динамических процессов, которые происходят при выполнении (7.16). Далее будем называть P инвариантом цикла.

7.4.2. Оператор do-while

Другая форма оператора итерации:

$$do\ S\ while(B); \tag{7.19}$$

где S — последовательность операторов, а B — логическое выражение. Оператор (7.19) определяет, что S выполняется перед вычислением B. Затем, если B истинно, процесс итерационного выполнения S продолжается, в противном случае он завершается. Структура оператора итерации (7.19) представлена на рис. 7.9.

Сравнивая рис. 7.9 и 7.7 замечаем, что на рис. 7.9 S должен быть выполнен по меньшей мере один раз, в то время как на рис. 7.7 он может не выполняться совсем.

Пример оператора *do-while* дан в программе (7.20), которая вычисляет сумму $h = 1^2 + 2^2 + \ldots + n^2$ для заранее заданного значения n:

$$h = 0; do\{h+ = n * n; n - -; \}while(n); \tag{7.20}$$

Предположим, что допускаются отрицательные значения n. Тогда можно использовать (7.17) и (7.20) для иллюстрации фундаментальной проблемы, связанной с итерационной композицией, когда мы заинтересованы в полной, а не в частичной корректности. Если $n \leq 0$, то (7.20) является бесконечным циклом. С другой стороны, программа (7.17) завершается, даже если $n \leq 0$. Таким образом, видим, что неправильно спроектированный оператор итерации может фактически описывать бесконечное вычисление.

Чтобы установить правило вывода для *do S while*(B), рассмотрим рис. 7.10.

Предположим, доказано, что $\{P\}S\{Q\}$ и $Q \wedge B \to P$. Если P истинно в точке входа, то Q будет истинно, когда точка C достигается в первый раз.

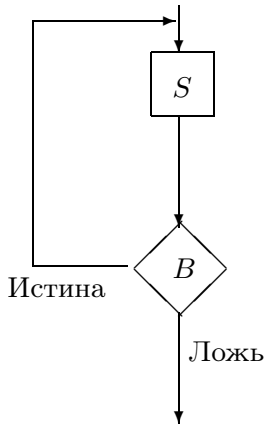


Рис. 7.9

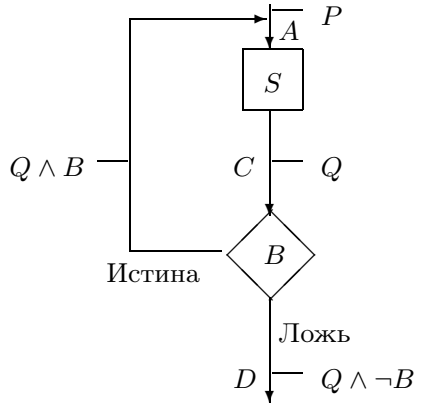


Рис. 7.10

Если B истинно, то истинно $Q \wedge B$ и при прохождении через цикл вновь достигается точка A. Так как $Q \wedge B \to P$, то, следовательно, P удовлетворится при достижении точки A во второй раз (если это вообще происходит). Но тогда вновь на основании $\{P\}S\{Q\}$ будет удовлетворяться Q, когда точка C достигается во второй раз и т.д. Итак, видно, что при сделанных предположениях P будет истинно всякий раз, когда достигается точка A и Q будет истинно всякий раз, когда достигается точка C, независимо от числа повторений цикла. Когда итерационный процесс заканчивается, т.е. когда достигается точка выхода (точка D на рис. 7.10), должно быть истинно утверждение $Q \wedge \neg B$. Это рассуждение приводит к следующему правилу вывода:

$$\frac{\{P\}S\{Q\}, Q \wedge B \to P}{\{P\}do\ S\ while(B); \{Q \wedge \neg B\}}. \tag{7.21}$$

7.5. Использование основных правил вывода

Приведем пример использования основных правил вывода для доказательства корректности программы. Рассмотрим программу нахождения *div* и *mod* двух целых неотрицательных чисел:

$$q = 0;\ r = x;$$
$$\text{while } (r \geq y)\ \{$$
$$r = r - y;$$
$$q = 1 + q;\}$$

Теперь посмотрим, каким соотношениям должны удовлетворять переменные на разных участках кода.

$$\{((x \geq 0) \wedge (y > 0))\}$$
$$q = 0;$$
$$r = x;$$
$$\{((x = q * y + r) \wedge (0 \leq r))\}$$
$$\text{while } (r \geq y) \{$$
$$\{((x = q * y + r) \wedge (0 < y \leq r))\}$$
$$r = r - y;$$
$$q = 1 + q; \}$$
$$\{((x = q * y + r) \wedge (0 \leq r < y))\}$$

Критерий корректности программы выражается соотношением

$$(x = q * y + r) \wedge (0 \leq r < y),$$

которое должно иметь место при завершении программы, если соотношение $(\mathrm{x} \geq 0) \wedge (\mathrm{y} > 0)$ справедливо перед ее выполнением. Именно это и следует доказать.

Рассмотрим сначала цикл в приведенной программе, представленный на рис. 7.11. В этом примере B имеет вид $(r \geq \mathrm{y})$. Если теперь заметить, что выходное соотношение на рис. 7.11 может быть переписано как $(x = q * \mathrm{y} + r) \wedge (r \geq 0) \wedge \neg(r \geq \mathrm{y})$, то ясно, что это как раз и есть $P \wedge \neg B$, где

$$P - \text{это } (x = q * y + r) \wedge (r \geq 0). \tag{7.22}$$

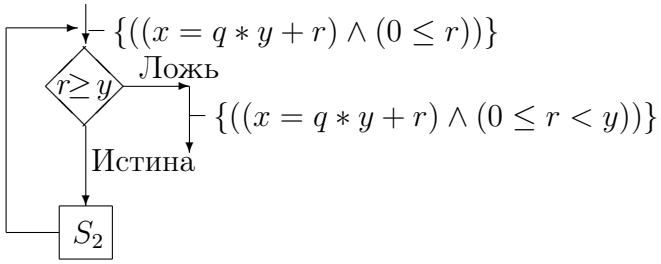


Рис. 7.11

Другими словами, наша задача при верификации рис. 7.11 показать, что $\{P\}while(r \geq y)S_2\{P \wedge \neg(r \geq y)\}$, где инвариант цикла P задается соотношением (7.22). Но правило вывода (7.18) гарантирует, что это утверждение должно быть истинным постольку, поскольку можно доказать

$$\{P \wedge (r \geq y)\}S_2\{P\} \tag{7.23}$$

для P, задаваемого (7.22) и S_2, задаваемого $\{r = r - \mathrm{y}; q = 1 + q; \}$.

Теперь заметим, что

$$(x = q * y + r) \wedge (r \geq 0) \wedge (r \geq y) \rightarrow (x = (1 + q) * y + (r - y)) \wedge (r - y \geq 0).$$

Вспоминая правило вывода для оператора присваивания, получим:

$$\{(x = (1 + q) * y + r - y) \wedge (r - y \geq 0)\} r = r - y; \{(x = (1 + q) * y + r) \wedge (r \geq 0)\}$$

$$\{(x = (1 + q) * y + r) \wedge (r \geq 0)\} q = 1 + q; \{(x = q * y + r) \wedge (r \geq 0)\}.$$

Из правила вывода для составного оператора и правила консеквенции следует:

$$\{P \wedge (r \geq y)\}\{r = r - y; q = 1 + q; \}\{P\}, \tag{7.24}$$

которое есть в точности выражение (7.23), необходимое для доказательства. Таким образом, получаем

$$\{P\} while(r \geq y) S_2; \{P \wedge \neg(r \geq y)\}. \tag{7.25}$$

Следующий шаг состоит в проверке выполнения условия

$$\{(x \geq 0) \wedge (y > 0)\}\{q = 0; r = x; \}\{(x = q * y + r) \wedge (r \geq 0)\}. \tag{7.26}$$

Действительно,

$$(x \geq 0) \wedge (y > 0) \rightarrow (x = 0 * y + x) \wedge (x \geq 0);$$

$$\{(x = 0 * y + x) \wedge (x \geq 0)\} q = 0; \{(x = q * y + x) \wedge (x \geq 0)\};$$

$$\{(x = q * y + x) \wedge (x \geq 0)\} r = x; \{(x = q * y + r) \wedge (r \geq 0)\}.$$

Из последних трех выражений, используя правило консеквенции и правило вывода для составного оператора получаем (7.26).

Теперь из (7.25), (7.26) и правила вывода для составного оператора следует:

$$\{(x \geq 0) \wedge (y > 0)\}$$

$$q = 0; r = \text{x};$$

$$while(r \geq \text{y})\{r = r - \text{y}; q = 1 + q; \}$$

$$\{(x = q * y + r) \wedge (0 \leq r < y)\}.$$

Условие $y > 0$ не использовалось при доказательстве частичной корректности, однако оно будет играть существенную роль при доказательстве того, что алгоритм действительно завершается.

Список рекомендуемой литературы

Алагич С., Арбиб М. Проектирование корректных структурированных программ. М.: Радио и связь, 1984. 284 с.
Ахо А., Ульман Д., Хопкрофт Д. Структуры данных и алгоритмы. М.: Вильямс, 2007. 400 с.
Брой М. Информатика: В 3 ч. Ч. 1. М.: Диалог-МИФИ, 1996. 300 с.
Брукшир Д. Введение в компьютерные науки. М.: Вильямс, 2001. 688 с.
Гук М. Аппаратные средства IBM PC. СПб.: Питер, 2006. 816 с.
Кнут Д. Искусство программирования. Т. 1. Основные алгоритмы. М.: Вильямс, 2006. 720 с.
Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. СПб.: Питер, 2001. 672 с.
Фомин С.В. Системы счисления. М.: Наука, 1987. 48 с.

Учебное издание

Иванов Александр Сергеевич

ИНФОРМАТИКА

Учебное пособие

Оригинал-макет подготовлен А.С. Ивановым в пакете LaTeX 2_ε

Подписано в печать 19.11.2009. Формат 60× 84 1/16. Бумага офсетная.
Гарнитура Times New Roman. Печать RISO. Объем 5,25 печ. л. Тираж 200 экз. Заказ 293.

ООО Издательский центр «Наука»
410600, г. Саратов, ул. Пугачевская, д. 117, оф. 50

Отпечатано с готового оригинал-макета
Центр полиграфических и копировальных услуг
Предприниматель Серман Ю.Б. Свидетельство № 3117
410600, г. Саратов, ул. Московская, д. 152, оф. 19, тел. 26-18-19, 51-16-28