К. Ю. Поляков, Е. А. Еремин

# ИНФОРМАТИКА

## 9 класс

Москва
БИНОМ. Лаборатория знаний
2017

УДК 004.9
ББК 32.97
П54

**Поляков К. Ю.**

П54 Информатика. 9 класс / К. Ю. Поляков, Е. А. Еремин. — М. : БИНОМ. Лаборатория знаний, 2017. — 288 с. : ил.

ISBN 978-5-9963-3109-3

Учебное издание предназначено для изучения предмета «Информатика» в 9 классе (базовое и углублённое изучение). Входит в состав УМК по информатике для 7–9 классов, включающего авторскую программу, учебные издания, рабочие тетради, практикум, электронные ресурсы и методическое пособие.

Содержание учебного издания является продолжением курса 8 класса. В нем рассматриваются вопросы, связанные с кодированием информации, программированием, робототехникой; работа с табличными процессорами и подготовка электронных документов

Главная задача учебного издания — обеспечить освоение базовых понятий информатики и принципов работы цифровой техники, что позволяет использовать его независимо от конкретных типов компьютеров и версий программного обеспечения.

Значительное внимание уделяется систематической подготовке школьников к государственной итоговой аттестации по информатике в форме основного государственного экзамена (ОГЭ).

Предполагается широкое использование ресурсов федеральных образовательных порталов, в том числе Единой коллекции цифровых образовательных ресурсов (http://sc.edu.ru/).

Соответствует федеральному государственному образовательному стандарту основного общего образования и примерной основной образовательной программе основного общего образования.

**УДК 004.9**
**ББК 32.97**

**ISBN 978-5-9963-3109-3**

# ОТ АВТОРОВ

В этом году вы завершаете изучение предмета «Информатика» в курсе основной школы. После этого вам нужно будет выбирать, как продолжить обучение — получать профессию или заканчивать среднюю школу, гимназию или лицей. Но какой бы выбор вы ни сделали, информатика и информационные технологии будут с вами повсюду, потому что все современные профессии требуют владения компьютерной техникой.

В любом случае вам придётся учиться и после окончания школы. Информационные технологии развиваются очень быстро, и знания в этой области быстро устаревают. Поэтому важно осваивать не конкретные программы для компьютеров или смартфонов, а изучать фундаментальные (базовые, основные) идеи информатики и научиться учиться — получать знания самостоятельно.

В этом году вы подробно познакомитесь с тем, как работают компьютерные сети, в том числе сеть Интернет (*глава 1*), без которой уже сложно представить современный мир.

С помощью *главы 2* мы будем изучать основы математической логики — специального раздела математики. Используя законы логики, учёным и инженерам удалось создать практически все электронные элементы компьютеров, в том числе вычислительные блоки и элементы памяти. Можно сказать, что весь компьютер состоит из огромного количества логических элементов.

Большинство задач, которые мы решаем на компьютере, — это задачи моделирования, в которых реальные объекты и процессы заменяются их компьютерными моделями. *В главе 3* вы познакомитесь с понятием «модель» и будете исследовать модели различных типов.

*Глава 4* содержит новый материал по программированию на алгоритмическом языке системы КуМир и на языке Паскаль. Главные «программистские» темы этого года — обработка символьных строк, массивов данных и применение вспомогательных алгоритмов — процедур и функций.

В этом году вы углубите свои знания по электронным таблицам (*глава 5*) и познакомитесь с базами данных (*глава 6*).

*Глава 7* содержит материалы по истории и перспективам развития компьютеров, теории систем и рассказывает о том, что такое информационное общество.

В учебнике есть основной материал (обязательный для изучения), и дополнительный (для углублённого курса). Материал для углублённого курса обозначен чёрными горизонтальными линиями, шрифтом, значком  в начале материала и значком  — в конце, выделен шрифтом. Даже если вы изучаете информатику на базовом уровне, всегда можно заглянуть в дополнительные разделы учебника — вдруг там окажется что-то интересное.

Учебник — это не просто книга для чтения. Для того чтобы действительно изучить предмет, нужно действовать: решать задачи, выполнять практические работы. Вы должны научиться «добывать» знания, выполняя различные эксперименты, пробуя и ошибаясь (без этого тоже нельзя!), проверяя догадки, делая выводы. Именно так работают учёные, открывая новые законы природы.

При чтении учебника мы советуем сразу выполнять задания, выделенные в тексте шрифтом и отступом. Эти задания рекомендуют вам перед тем, как продолжить чтение, ответить на вопрос, выполнить небольшое упражнение в тетради или провести исследование с помощью компьютера. Задания специально подобраны так, чтобы легче было понять новый материал. Тип задания обозначается на полях навигационными значками  (вопрос), или  (письменное задание), или  (компьютерный эксперимент).

Для полноценной работы желательно использовать рабочую тетрадь (значок  ) — в ней вы будете выполнять письменные задания.

Значок  говорит о том, что при выполнении задания придётся использовать кроме учебника дополнительные источники, например сеть Интернет. Проектные и исследовательские работы, которые выполняются дома, отмечены значком  .

Значок  означает важное определение или утверждение.

Значок  служит для выделения дополнительного задания или разъяснения.

Значок  означает групповую работу.

Значок  выделяет межпредметные связи.

Задания повышенной сложности отмечены «звёздочкой» (*).

В конце каждой главы вам предлагается список электронных образовательных ресурсов из Единой коллекции цифровых образовательных ресурсов (ЕК ЦОР) **www.school-collection.edu.ru**.

Электронные материалы к учебнику (файлы для выполнения практических работ, презентации, тесты) можно загрузить с сайта поддержки учебника:

**http://kpolyakov.spb.ru/school/osnbook.htm**

В заключение нам хочется поблагодарить наших коллег, которые взяли на себя труд прочитать предварительные версии отдельных глав учебника и высказать множество полезных замечаний, позволивших сделать учебник более точным, ясным и понятным:

- А. П. Шестакова, кандидата педагогических наук, зав. кафедрой информатики и вычислительной техники Пермского государственного педагогического университета;
- М. А. Ройтберга, доктора физико-математических наук, зав. лабораторией прикладной математики Института математических проблем биологии РАН, г. Пущино;
- С. С. Михалковича, кандидата физико-математических наук, доцента кафедры алгебры и дискретной математики ЮФУ, г. Ростов-на-Дону;
- Н. Д. Шумилину, кандидата педагогических наук, доцента кафедры математики с методикой начального обучения Тверского государственного университета, г. Тверь;
- А. В. Паньгина, инженера Центра информационных технологий, г. Сосновый Бор;
- А. С. Башлакова, учителя информатики МОУ СОШ № 3, г. Унеча Брянской области;
- Н. П. Радченко, учителя информатики ГБОУ Школа № 1095, г. Москва;
- Ю. М. Розенфарба, учителя информатики МОУ Межозёрная СОШ, Челябинская область;
- О. А. Тузову, учителя информатики школы № 550, г. Санкт-Петербург;
- В. Н. Разумова, учителя информатики МОУ «Большеелховская средняя общеобразовательная школа», с. Большая Елховка, Республика Мордовия;
- А. В. Атанову, учителя информатики МАОУ СОШ № 12 им. маршала Советского Союза К. К. Рокоссовского, г. Великие Луки;
- Г. В. Роньжину, учителя информатики ГБОУ «Гимназия № 1519», г. Москва;

- А. В. Паволоцкого, учителя информатики ГБОУ «Гимназия № 1514», г. Москва;
- Н. Г. Неуймину, учителя информатики МАОУ «Лицей № 110» им. Л. К. Гришиной, г. Екатеринбург;
- Н. Е. Леко, учителя информатики МОУ СОШ № 9, г. Тихвин;
- И. А. Волкову, учителя информатики МОУ СОШ № 170, г. Екатеринбург;
- Н. С. Семашко, учителя информатики МБОУ «Лицей № 6», г. Дубна;
- С. В. Гриневича, учителя информатики МАОУ СОШ № 146, г. Пермь;
- Г. М. Шульгину, учителя информатики МОУ СОШ № 9, г. Пермь;
- Т. В. Дедюлькину, учителя информатики МАОУ «Гимназия № 5», г. Ростов-на-Дону;
- С. В. Гайсину, методиста ЛОИРО, г. Санкт-Петербург.

*С уважением, авторы:*
*Константин Юрьевич Поляков,*
*Евгений Александрович Еремин*

# Глава 1
# КОМПЬЮТЕРНЫЕ СЕТИ

## § 1
## Как работает компьютерная сеть?

***Ключевые слова:***

- компьютерная сеть
- локальная сеть
- глобальная сеть
- протокол
- пакет
- контрольная сумма
- сервер
- клиент

### Что такое компьютерная сеть?

**Компьютерная сеть** — это группа компьютеров, объединённых линиями связи.

Все устройства, которые соединены в сеть, называются **узлами** сети (по аналогии с узлами рыболовной сети). Кроме компьютеров к ним относятся вспомогательные устройства, участвующие в передаче данных.

Для связи узлов между собой используются различные **каналы связи**:

- *электрические кабели* (данные передаются с помощью электрических сигналов);
- *оптические кабели* (данные передаются с помощью световых лучей);
- *радиоканалы* (данные передаются с помощью радиоволн).

Объединяя компьютеры в сеть, мы получаем следующие *преимущества*:

- *быстрый обмен данными* между компьютерами (не нужно использовать для переноса данных съёмные диски, флэш-диски);
- компьютеры в сети могут использовать *общие ресурсы*:
  - — общие данные могут быть размещены на одном компьютере;
  - — можно запускать программы с другого компьютера;
  - — все компьютеры могут использовать общие внешние устройства (например, принтер);
- *электронную почту и другие способы сетевого общения* (чаты, форумы и т. п.).

В то же время при организации сети:

- необходимы *денежные затраты* на сетевое оборудование (кабели, вспомогательные устройства) и программное обеспечение (например, операционную систему специального типа);
- *снижается безопасность* данных, поэтому компьютеры, на которых ведутся секретные разработки, не должны быть подключены к сети;
- необходим высококвалифицированный специалист — *системный администратор*, который занимается настройкой сети и обеспечивает её работу.

**Системный администратор** (на практике часто используют сокращения «сисадмин» или «админ») обычно решает следующие задачи:

- устанавливает и настраивает программное обеспечение (в том числе и несетевое);
- устанавливает права доступа пользователей к ресурсам сети;
- обеспечивает защиту информации;
- предотвращает потерю данных в случае сбоя электропитания;
- периодически делает резервные копии данных на DVD-дисках или съёмных жёстких дисках;
- устраняет неисправности в сети.

В некоторых крупных организациях кроме системных администраторов есть также **сетевой администратор**, который занимается только работой сети.

## Типы компьютерных сетей

По «радиусу охвата» обычно выделяют следующие типы компьютерных сетей:

- *персональные сети* объединяют устройства одного человека (сотовые телефоны, карманные компьютеры, смартфоны, ноутбук и т. п.) в радиусе не более 30 м; самый известный стандарт таких сетей — *Bluetooth*;
- *локальные сети* (от англ. *local* — местный) связывают, как правило, компьютеры в пределах одного или нескольких соседних зданий; для создания беспроводных локальных сетей используется технология *Wi-Fi*;
- *корпоративные сети* — сети компьютеров одной организации (возможно, находящиеся в разных районах города или даже в разных городах);
- *городские сети*, объединяющие компьютеры в пределах города;
- *глобальные сети*, объединяющие компьютеры в разных странах (например, сеть *Интернет*).

Используя дополнительные источники, найдите ответы на вопросы.
– Что означает сокращение *PAN*?
– Откуда произошли обозначения *Bluetooth* и *Wi-Fi*?

## Обмен данными

Для того чтобы люди могли полноценно общаться, нужно, чтобы они говорили на одном языке. Это правило действует и для компьютерных систем, где вместо слова «язык» используется термин «протокол».

**Протокол** — это набор правил, определяющих порядок обмена данными в сети.

В современных сетях пересылаемые данные делятся на части — **пакеты**. Дело в том, что чаще всего одна линия связи используется для обмена данными между несколькими узлами. Если передавать большие файлы целиком, то получится, что сеть будет заблокирована, пока не закончится передача очередного файла. Кроме того, в этом случае при сбое весь файл нужно передавать заново, это увеличивает нагрузку на сеть.

Если передавать отдельные пакеты, время ожидания сокращается до времени передачи одного пакета (это доли секунды), по сети одновременно передаются пакеты, принадлежащие нескольким файлам. На рисунке 1.1 по одной линии связи (между узлами 3 и 4) одновременно выполняется передача данных от узла 2 к узлу 5 (эти пакеты обозначены чёрными прямоугольниками) и от узла 1 к узлу 6 (белые прямоугольники).

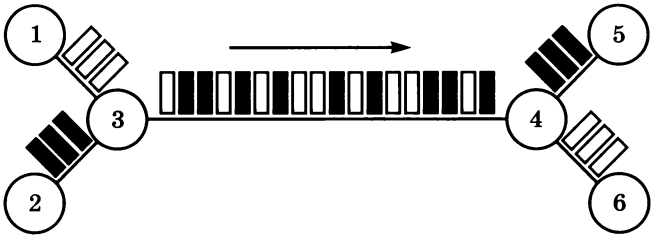



**Рис. 1.1**

Вместе с каждым пакетом передаётся его **контрольная сумма** — число, найденное по специальному алгоритму и зависящее от всех данных пакета. Узел-приёмник рассчитывает контрольную сумму полученного блока данных, и если она не сходится с контрольной суммой, указанной в пакете, фиксируется ошибка, и этот пакет (а не весь файл!) передаётся ещё раз.

Казалось бы, чем меньше размер пакета, тем лучше. Однако это не так, потому что любой пакет кроме «полезных» данных содержит служебную информацию: адреса отправителя и получателя, контрольную сумму. Поэтому в каждом случае есть некоторый оптимальный (наилучший) размер пакета, который зависит от многих условий (например, от уровня помех, количества компьютеров в сети, передаваемых данных и т. д.). Чаще всего для обмена данными в локальных сетях и в Интернете используются пакеты размером не более 1,5 Кбайт.

www Используя дополнительные источники, выясните, какое семейство протоколов используется для обмена данными в Интернете.

## Серверы и клиенты

В любой сети одни компьютеры используют ресурсы других. Для описания роли компьютеров в обмене данными вводят два термина: *сервер* и *клиент*.

**Сервер** — это компьютер, предоставляющий свои ресурсы (файлы, программы, внешние устройства и т. д.) в общее использование.
**Клиент** — это компьютер, использующий ресурсы сервера.

Обычно *серверы* — это специально выделенные мощные компьютеры, которые используются только для обработки запросов большого числа клиентских компьютеров **(рабочих станций)** и, как правило, включены постоянно. Чаще всего они находятся в отдельных помещениях, куда пользователи не имеют доступа; это повышает защищённость данных.

В крупных локальных сетях используют несколько серверов, каждый из которых решает свою задачу:

- *файловый сервер* хранит данные и обеспечивает доступ к ним;
- *сервер печати* обеспечивает доступ к общему принтеру;
- *почтовый сервер* управляет электронной почтой;
- *серверы приложений* (например, серверы баз данных) выполняют обработку информации по запросам клиентов.

Часто понятия «сервер» и «клиент» относятся не к компьютерам, а к программам. **Программа-сервер** получает запросы от клиентов, ставит их в очередь, и после выполнения посылает каждому клиенту ответ с результатами выполнения запроса. Задача **программы-клиента** — послать серверу запрос в определённом формате и после получения ответа вывести результаты на монитор пользователя. Такая технология называется **клиент-сервер**. Её используют, например, все веб-сайты в Интернете: программа-брау-

зер (клиент) посылает запрос веб-серверу и выводит его ответ (веб-страницу) на экран. Как правило, при желании программу-сервер и программу-клиент можно запустить на одном компьютере.

В некоторых организациях применяют *терминальные серверы* — мощные компьютеры, которые предоставляют пользователям свои ресурсы (процессорное время, оперативную и дисковую память). Рабочие станции (*терминалы* или *«тонкие» клиенты*) в таких системах выполняют только две задачи:

- передают серверу данные, введённые пользователем с помощью клавиатуры и мыши;
- выводят на экран изображение рабочего стола, полученное от сервера.

Поэтому в качестве терминалов можно использовать маломощные и устаревшие компьютеры.

## Выводы

- Компьютерная сеть — это группа компьютеров, объединённых линиями связи.
- Протокол — это набор правил, определяющих порядок обмена данными в сети.
- Сервер — это компьютер, предоставляющий свои ресурсы (файлы, программы, внешние устройства и т. д.) в общее использование.
- Клиент — это компьютер, использующий ресурсы сервера.

## Интеллект-карта

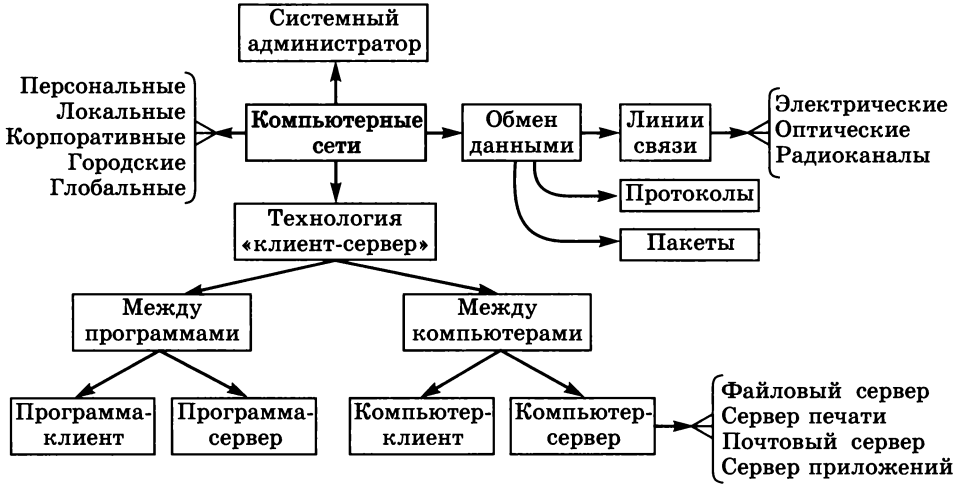



**Рис. 1.2**

### Вопросы и задания

1. Какие компьютерные сети окружают вас? Какие каналы связи они используют?
2. Зачем нужны протоколы?
3. Может ли один компьютер выполнять роли сервера и клиента?
4. Зачем данные, передаваемые по сети, делятся на пакеты?
5. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «Пакетная передача данных»
б) «Технология "клиент-сервер"»
в) «Протоколы Интернета»

# § 2
# Структуры сетей

***Ключевые слова:***

- общая шина
- звезда
- кольцо
- коммутатор

Для обмена данными в сети очень важно, как именно связаны компьютеры линиями связи. Существуют три основные схемы соединения компьютеров в сети: общая шина, звезда и кольцо. Каждая из них обладает своими достоинствами и недостатками.

## Общая шина

В схеме **«общая шина»** (рис. 1.3) все компьютеры обмениваются данными с помощью одного канала связи. Например, они могут быть присоединены к одному кабелю.

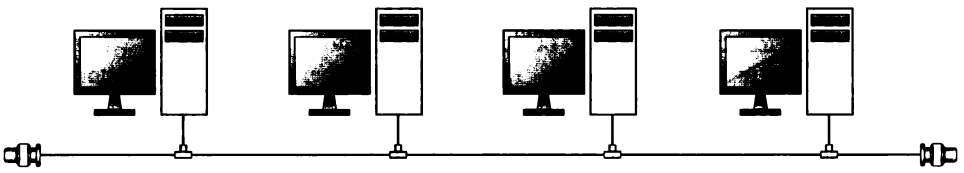


**Рис. 1.3**

Так как существует всего одна линия связи, компьютеры передают данные по очереди. Сигнал, который идёт по шине, получают все компьютеры, но каждый из них обрабатывает только те данные, которые ему предназначены.

Это самая простая и дешёвая схема, она позволяет легко подключать новые компьютеры, расход кабеля небольшой. В то же время при разрыве кабеля вся сеть не работает. Каждый компьютер «видит» все данные, которые идут по сети, поэтому данные легко перехватить. Так как используется один канал связи, при увеличении числа компьютеров работа сети замедляется. Поэтому количество компьютеров в такой сети ограничено (обычно не более 10–15).

Схема «общая шина» в современных кабельных сетях не встречается, но фактически применяется в беспроводных сетях, которые используют один канал связи.

Используя дополнительные источники, выясните значение слова «терминатор». Зачем используют терминаторы в сетях типа «общая шина»?

## Звезда

В схеме **«звезда»** (рис. 1.4) есть центральное устройство, через которое идёт весь обмен данными. Чаще всего в центре находится **коммутатор** (его часто называют *«свитч»*). Коммутатор передаёт принятый пакет только адресату, а не всем компьютерам в сети.

Используя дополнительные источники, выясните, от какого иностранного слова произошло слово «свитч» и что оно обозначает.

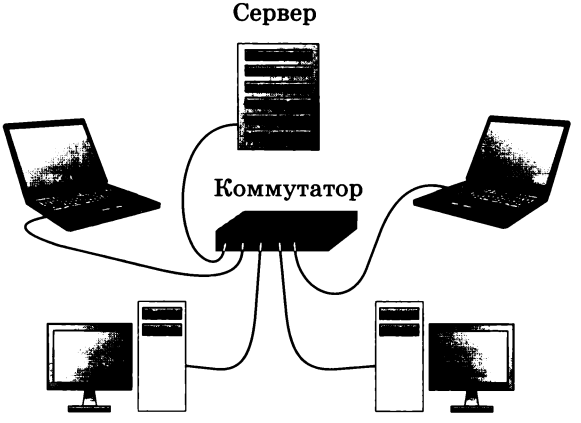



**Рис. 1.4**

Схема «звезда» имеет много достоинств. При выходе из строя любой рабочей станции сеть остаётся работоспособной. Каждая рабочая станция получает только «свои» данные, поэтому перехватить «чужую» информацию очень сложно.

Вместе с тем для каждого компьютера нужно проложить отдельную линию связи с коммутатором, поэтому расход кабеля в схеме «звезда» большой. Многое зависит от коммутатора: если он неисправен, то сеть не работает.

Для компьютерной сети в вашем классе сравните расход кабеля при использовании схем соединения «звезда» и «общая шина».

Узнайте (у учителя или с помощью программного обеспечения) скорость передачи данных в вашей компьютерной сети. Вычислите примерное время передачи:
- фотографии размером 5 Мбайт;
- фильма размером 450 Мбайт.

## Кольцо

В схеме **«кольцо»** (рис. 1.5) каждый компьютер соединяется с двумя соседними, причём от одного он только получает данные, а другому только передаёт. Таким образом, пакеты движутся по кольцу в одном направлении. Для повышения надёжности обычно используют «двойное кольцо», в котором каждая линия связи дублируется. По второму кольцу данные могут передаваться в обратном направлении.

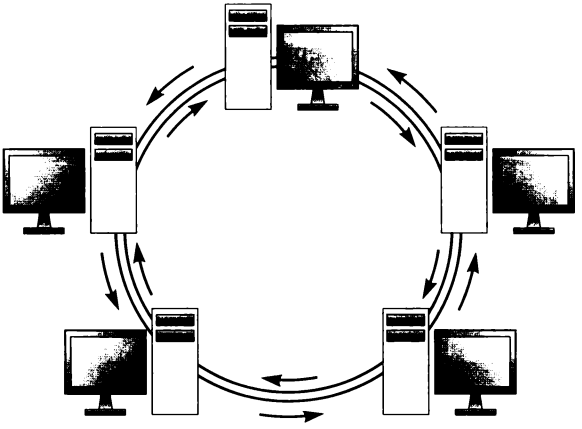


**Рис. 1.5**

Каждый компьютер участвует в передаче сигнала и усиливает его, поэтому размер сети может быть очень велик (до 20 км). Сеть хорошо работает при большом потоке данных, поэтому схема «кольцо» применяется для соединения коммутаторов между собой. В то же время такую сеть сложнее настраивать и обслуживать. Для подключения нового компьютера нужно останавливать работу сети. Так как данные проходят по кольцу через несколько компьютеров, обеспечить надёжную защиту информации в этой схеме довольно сложно.

### Выводы

- Существуют три основные схемы соединения компьютеров в сети: общая шина, звезда и кольцо.
- В схеме «общая шина» все компьютеры используют для обмена данными общую линию связи.
- В схеме «звезда» есть центральное устройство (коммутатор), к которому присоединяются все компьютеры.
- В схеме «кольцо» данные передаются по кругу от одного компьютера к другому.

Нарисуйте в тетради интеллект-карту этого параграфа.

### Вопросы и задания

1. Найдите в тексте достоинства и недостатки каждой структуры сети.
2. Почему схема «общая шина» сейчас не используется в кабельных сетях?
3. Какие достоинства и недостатки имеет схема «двойное кольцо» в сравнении с «одиночным»?
4. Какая схема обеспечивает лучшую защиту данных? Почему?
5. Выполните по указанию учителя задания в рабочей тетради.

# § 3
# Локальные сети

***Ключевые слова:***

- локальная сеть
- одноранговая сеть
- сеть с выделенным сервером
- беспроводная сеть
- Bluetooth
- Wi-Fi
- точка доступа
- сетевая карта
- коммутатор
- патч-корд
- маршрутизатор

## Типы локальных сетей

Локальной сетью обычно называют компьютерную сеть в одном или нескольких соседних зданиях. В небольших организациях часто используют **одноранговые сети** (на 10–15 компьютеров), в которых все компьютеры равноправны: каждый может выступать как в роли клиента, так и в роли сервера.

Пользователь может открыть *общий доступ* к некоторым ресурсам своего компьютера (папкам на диске, принтерам), так что ими могут пользоваться другие. Каждому пользователю можно дать свои права для работы с ресурсом, например разрешить только читать данные, но не изменять их.

Одноранговые сети недороги, просты в настройке и обслуживании, не требуют сложного программного обеспечения. Все компьютеры независимы друг от друга, отключение одного из них не нарушает работу остальных в сети.

В одноранговой сети нет единого центра управления, поэтому на каждом компьютере приходится создавать учётные записи всех пользователей сети и настраивать права доступа. Это усложняет обслуживание сети и защиту данных. Чем больше компьютеров в сети, тем сложнее ею управлять.

В крупных организациях используют **сети с выделенными серверами**, в которых один или несколько мощных компьютеров играют роль серверов (пользователи на них не работают), а остальные (клиенты, рабочие станции) используют их ресурсы.

В таких сетях основная обработка данных выполняется на серверах, а на рабочие станции передаются готовые результаты. Это уменьшает поток данных в сети. На компьютерах-клиентах можно использовать самое разное оборудование и операционные системы; главное, чтобы они смогли обращаться к серверу по нужному протоколу. Повышается уровень безопасности данных, потому что права на доступ устанавливаются только на серверах, где хранятся данные. Рабочие станции не должны быть очень мощными, для ускорения работы всей сети нужно усилить только серверы.

В то же время серверное оборудование стоит достаточно дорого, для настройки и обслуживания серверов нужны грамотные специалисты.

На серверах обычно используют специальные **серверные операционные системы** (*Windows Server, Linux, FreeBSD*). В таких ОС основное внимание уделяется стабильной и надёжной работе с большим количеством клиентов, а не пользовательскому интерфейсу. Они обеспечивают работу пользователей, веб-узла, электронной почты, систем управления базами данных и т. п.

Важная возможность серверных ОС — *терминальный доступ*, при котором пользователь со своей рабочей станции запускает программу на сервере и получает на своем экране результаты её работы.

Используя текст параграфа, запишите в таблицу достоинства и недостатки одноранговых сетей и сетей с выделенными серверами.

## Беспроводные сети

Беспроводные сети используются там, где создание кабельной сети невозможно или невыгодно, например за пределами зданий, в исторических помещениях и т. п. Через них мобильные компьютеры (ноутбуки, планшетные компьютеры, смартфоны) могут легко подключаться к сети и получать доступ к Интернету. Обмен данными происходит с помощью радиоволн сверхвысокой частоты.

Стандарт **Bluetooth** для **беспроводных персональных сетей** позволяет обмениваться данными восьми устройствам. Это могут быть настольный и планшетный компьютеры, мобильный телефон, ноутбук, принтер, цифровой фотоаппарат, мышь, клавиатура, наушники.

Радиус действия сети *Bluetooth* обычно не более 20 м, он зависит от мощности передатчиков, а также от преград и помех. Максимальная скорость[1)] обмена данными может достигать 24 Мбит/с. В будущем с помощью *Bluetooth* можно будет связывать любые электронные устройства, включая холодильники и стиральные машины. Для обеспечения защиты данных от перехвата приёмник и передатчик 1600 раз в секунду одновременно меняют частоту сигнала.

Скорость передачи данных по сети Bluetooth равна 3 Мбит/с. Оцените время передачи видеофайла объёмом 25 Мбайт.

В **локальных беспроводных сетях** применяют стандарт **Wi-Fi** (от англ. *Wireless Fidelity* — беспроводная точность). Компьютеры подключаются к беспроводной сети через специальное устройство — **точку доступа** (рис. 1.6). Одна точка доступа обычно обслуживает не более 15 компьютеров (при увеличении этого количества падает скорость передачи данных). Часто главная задача точки доступа — обеспечить мобильным компьютерам доступ к кабельной сети и выход в Интернет.

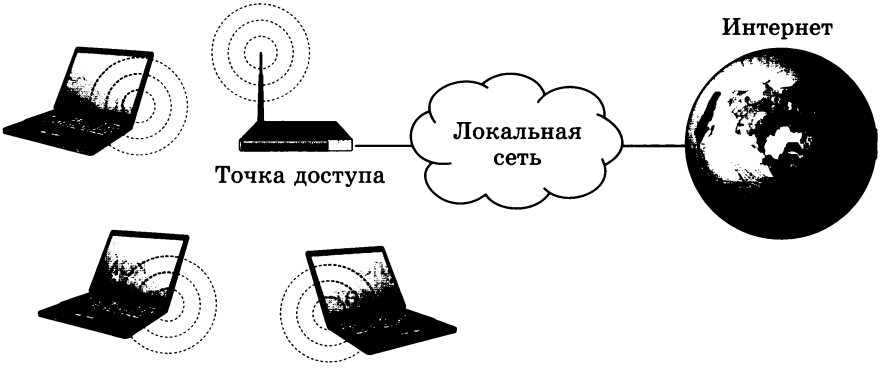



**Рис. 1.6**

1) При указании скорости передачи данных используются десятичные приставки (а не двоичные, как при измерении количества информации), например 1 Мбит/с = $10^6$ бит/с.

Скорость передачи данных в сетях *Wi-Fi* может достигать 600 Мбит/с. Радиус действия сети в помещениях не превышает 45 м, а вне зданий — 450 м.

Скорость передачи данных по сети *Wi-Fi* равна 50 Мбит/с. Оцените время передачи видеофайла объёмом 25 Мбайт.

Технология *Wi-Fi* широко используется как в офисах, так и в домашних сетях. Бесплатный выход в Интернет через *Wi-Fi* предоставляют многие библиотеки, университеты, кафе (для привлечения посетителей), гостиницы. Такие зоны доступа называют *«хот-спот»*. В некоторых гостиницах и аэропортах эта услуга платная.

Используя дополнительные источники, выясните, от каких иностранных слов произошло выражение «хот-спот» и как переводятся эти слова.

Сети *Wi-Fi* работают в радиоэфире, так что любое приёмное устройство, настроенное на нужную частоту, может перехватить сигнал. Поэтому в беспроводных сетях важно обеспечить защиту данных. Для этого используют специальные алгоритмы кодирования сигналов и шифрование.

Используя текст параграфа, заполните в тетради таблицу характеристик беспроводных сетей.

## Оборудование для локальных сетей

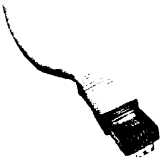
В современных локальных сетях используется технология пакетной передачи данных, которая называется **Ethernet**. Для связи компьютеров могут применяться электрические кабели или оптоволокно. Существующий стандарт определяет скорости передачи данных до 100 Гбит/с.

Используя дополнительные источники, выясните, от какого слова произошло название *Ethernet*.

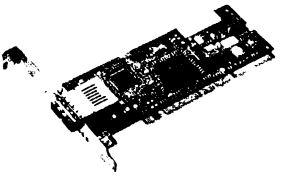
Для того чтобы подключить компьютер к кабельной сети, он должен иметь **сетевую карту** (**сетевой адаптер**, англ. *network interface card*) — рис. 1.7. В современных материнских платах настольных компьютеров и в ноутбуках обычно уже есть встроенная сетевая карта, поддерживающая стандарт *Ethernet* со скоростью до 1 Гбит/с.

В большинстве кабельных локальных сетей данные передаются со скоростью 100 Мбит/с. Для соединения компьютеров используется восьмижильный кабель **«витая пара»**, который представляет собой четыре пары скрученных проводов. Восьмиконтактный разъём с защёлкой часто называют *RJ-45* (см. рис. 1.7).

Сетевой кабель «витая пара»



Разъём RJ-45

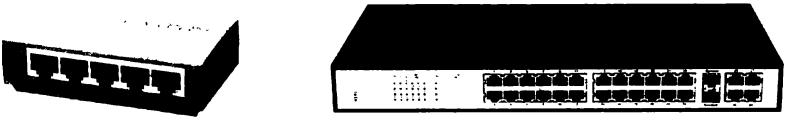


Сетевая карта

**Рис. 1.7**

На большие расстояния данные передают по **оптоволоконным кабелям**, в которых информацию переносит луч света. Свет идёт внутри кабеля, отражаясь от стенок стеклянного или пластикового цилиндра-световода.

Компьютеры объединяются в единую сеть по схеме «звезда» с помощью коммутаторов («свитчей») — рис. 1.8.



Коммутаторы

**Рис. 1.8**

Компьютер соединяют с коммутатором отрезком кабеля с двумя разъёмами RJ-45, который называется **«патч-корд»**.

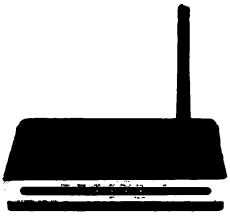
Используя дополнительные источники, выясните, от каких иностранных слов произошло название «патч-корд». Что оно означает? www

Обычно все компьютеры, входящие в локальную сеть, получают доступ к Интернету через один канал связи. Для связи локальной сети с другими сетями необходим **маршрутизатор**. Задача маршрутизатора — определить дальнейший маршрут движения пакета и направить его на нужный выход (порт). Роль маршрутизатора в локальной сети может выполнять обычный компьютер с несколькими сетевыми картами.

Используя дополнительные источники, выясните, как иначе называют маршрутизатор. От какого слова образован этот термин? www

Для подключения к беспроводным сетям компьютер должен иметь **адаптер Wi-Fi**, который есть во всех современных переносных устройствах — ноутбуках, планшетных компьютерах, смартфонах. Если встроенного адаптера нет, можно использовать дополнительный адаптер, который подключается к USB-порту. Устройство подключается к беспроводной сети и получает доступ в Интернет через точки доступа или беспроводные маршрутизаторы (рис. 1.9).

USB-адаптер Wi-Fi



Точка доступа

Маршрутизатор

**Рис. 1.9**

## Выводы

- Одноранговые сети — это сети, в которых все компьютеры равноправны. Они используются при небольшом количестве компьютеров.
- В сетях с выделенными серверами один или несколько компьютеров выполняют роль серверов, пользователи на них не работают.
- Для персональных беспроводных сетей используется технология *Bluetooth*, для локальных — *Wi-Fi*. Главная проблема беспроводных сетей — обеспечение защиты данных.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Как вы думаете, можно ли использовать технологию «клиент-сервер» в одноранговых сетях?
2. Почему существует две беспроводные технологии: *Bluetooth* и *Wi-Fi*?
3. Можно ли сказать, что в беспроводных сетях используется структура «общая шина»? Обоснуйте свой ответ.
4. Как защищается информация в беспроводных сетях?
5. Выясните, какая сеть используется в вашей школе — одноранговая или с выделенным сервером. Почему было принято такое решение?
6. Выясните, какое оборудование используется для локальной сети вашей школы. С какой скоростью передаются данные в сети?
7. Выполните по указанию учителя задания в рабочей тетради.

## Подготовьте сообщение

а) «Серверные операционные системы»
б) «Что такое терминальный сервер?»

# § 4
# Глобальная сеть Интернет

***Ключевые слова:***

- маршрутизатор
- провайдер
- протоколы TCP/IP
- IP-адрес
- домен
- система доменных имён

## Что такое Интернет?

Вы уже знаете, что это **Интернет** — это **глобальная компьютерная сеть.** Слово «*Интернет*» (англ. *Internet*) возникло как сокращение от *Interconnected Networks* — «объединённые сети» или «сеть сетей».

Первая версия сети была построена в 60-х годах XX века американскими военными. Перед разработчиками поставили задачу: создать сеть, которая осталась бы работоспособной при разрушении 70% узлов (в случае ядерной войны). У такой сети не должно быть центра, от которого зависит её работа. В то же время невозможно соединить каждый компьютер с каждым — на это нужно слишком много линий связи. В результате было найдено решение: сделать сеть, состоящую из ячеек, как рыболовная сетка (рис. 1.10). Такие сети называются **распределёнными.**

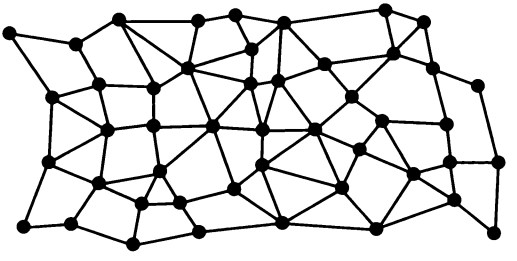


**Рис. 1.10**

В узлах сети стоят специальные компьютеры — **маршрутизаторы**, которые направляют каждый пакет данных кратчайшим маршрутом по адресу, указанному в заголовке пакета. Если же этот канал связи вышел из строя, то пакет будет отправлен по более длинному, но работающему пути.

Используя дополнительные источники, найдите ответы на вопросы.
- Как называлась сеть, в результате развития которой появился Интернет?
- Когда произошёл первый сеанс связи в этой сети?
- Какое сообщение было тогда передано?

В Интернете нет единого центра управления. Если бы он существовал, то для разрушения всей сети было бы достаточно уничтожить этот центр.

Информация в Интернете хранится на **серверах**, связанных скоростными линиями связи (оптоволоконными, спутниковыми). Практически все услуги Интернета основаны на использовании **технологии «клиент-сервер»**: программа-клиент на компьютере пользователя запрашивает данные, сервер возвращает ответ.

## Как подключиться к Интернету?

Пользователь получает доступ к глобальной сети через **провайдера** — фирму, локальная сеть которой непосредственно связана с Интернетом. Существует несколько **способов подключения к провайдеру**:

- с помощью *ADSL-модема*, который использует телефонную линию, но позволяет одновременно разговаривать по телефону и работать в Интернете; скорость передачи данных из Интернета к пользователю может достигать 25 Мбит/с;
- через *кабельную локальную сеть* провайдера (если она существует в вашем доме); в этом случае телефонная линия не используется;
- с помощью *оптических сетей* с высокой пропускной способностью (англ. **PON**: *Passive Optical Network* — пассивная оптическая сеть); в таких сетях для передачи данных со скоростью до 2,5 Гбит/с используются оптоволоконные кабели и оптические разветвители, которые не требуют питания и обслуживания;
- с помощью *беспроводных модемов* (*USB-модемов* — рис. 1.11), которые используют сети сотовых операторов и работают везде, где доступна мобильная связь; скорость передачи данных для сетей 3-го поколения (англ. **3G** : $3^{rd}$ *generation*) достигает 10 Мбит/с, а в сетях 4-го поколения (**4G**) — до 1 Гбит/с;

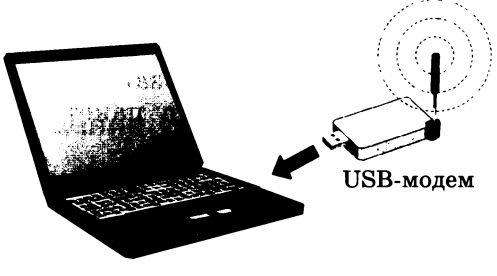



**Рис. 1.11**

- с помощью *беспроводных каналов связи* по технологии *Wi-Fi* (со скоростью до 54 Мбит/с).

Постройте в тетради или в электронных таблицах столбчатую диаграмму, на которой сравниваются наибольшие скорости передачи данных в кабельных, оптических и беспроводных сетях различного типа.

## Протоколы Интернета

Вы уже знаете, что для передачи информации источник и приёмник должны использовать один и тот же протокол — набор правил, определяющих порядок обмена данными в сети. В Интернете используется **семейство протоколов TCP/IP**. Название TCP/IP происходит от двух самых важных протоколов — TCP и IP.

Используя дополнительные источники, найдите ответы на вопросы.
- Когда было разработано семейство протоколов TCP/IP?
- От каких иностранных выражений образованы сокращения TCP и IP? Что они означают?

С помощью **протокола TCP** компьютер устанавливает соединение с другим компьютером и обеспечивает доставку данных. Блок данных, который нужно передать, разбивается на пакеты (размер пакета обычно не превышает 1,5 Кбайта).

**IP-протокол** устанавливает правила построения пакета и систему IP-адресов, с помощью которой маршрутизаторы определяют маршруты движения пакетов.

Кроме TCP и IP службы Интернета (например, Всемирная паутина, электронная почта и др.) используют свои протоколы «верхнего уровня», но об этом мы поговорим чуть позже.

## IP-адреса

В Интернете любые два компьютера могут связаться друг с другом. Для этого каждый из них должен иметь уникальный адрес. С «точки зрения» компьютеров, удобнее работать с числовыми адресами, каждый из которых занимает одинаковое место в памяти. Такие адреса (их называют **IP-адресами**, потому что они используются IP-протоколом) представляют собой 32-битные числа, например

$$3232262259 = 11000000101010000110100001110011_2$$

Для удобства обычно разбивают это число на группы из 8 двоичных разрядов (*октеты*):

11000000.10101000.01101000.01110011

и записывают каждую группу в десятичной системе счисления:

192.168.104.115

В IP-адресе закодированы номер сети и номер компьютера в сети. Такая структура чем-то напоминает обычный почтовый адрес: индекс определяет номер почтового отделения, а адрес — конкретные улицу, дом и квартиру.

Определите, в каком диапазоне должно находиться каждое из четырёх чисел, составляющих IP-адрес. Как вы рассуждали?

Какие последовательности не могут быть IP-адресами?

| | | |
|---|---|---|
| 101.123.278.211 | 156.21.0.1 | 257.212.100.1 |
| 112.345.0.43 | 23.32.12.11 | 101.1.201.2 |

В связи с бурным развитием Интернета адресов, которые можно использовать при таком кодировании, уже не хватает для всех желающих. Поэтому разработана новая система IP-адресов, в которой на каждый адрес отводится 128 бит, а не 32, как сейчас. Такой адрес записывается в виде восьми групп по четыре шестнадцатеричные цифры, разделённых двоеточиями, например:

2001:0DB8:11A3:09D7:1F34:8A2E:07A0:765D

Адреса такого типа использует новая, шестая версия IP-протокола, которая называется *IPv6*. Полный переход на *IPv6* займёт несколько лет, он потребует больших денежных затрат и замены всех устаревших устройств.

IP-адрес присваивается не компьютеру, а каналу связи (*интерфейсу*). Поэтому один компьютер может иметь несколько IP-адресов, например если у него есть сетевая карта и адаптер *Wi-Fi* или две сетевые карты.

## Доменные имена

В отличие от компьютеров человеку неудобно работать с числовыми адресами. Они плохо запоминаются, при вводе IP-адреса легко сделать ошибку. Поэтому в 1984 году была разработана **система доменных имён (DNS)**, которая позволила использовать *символьные имена сайтов*, например: **www.mail.ru**.

Используя дополнительные источники, выясните, от каких иностранных слов образовано сокращение *DNS* и что они означают.

**Домен** — это группа символьных адресов в Интернете. Домены образуют многоуровневую структуру (*иерархию*, *дерево*), вкладываются друг в друга, как матрёшки (рис. 1.12).

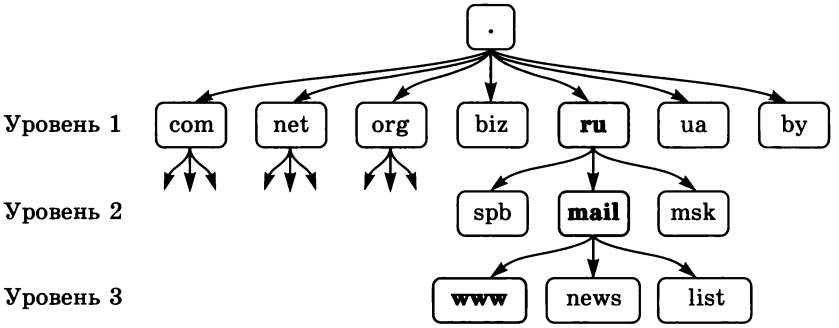



**Рис. 1.12**

Чем-то такая система напоминает почтовый адрес, в котором указывается страна, город, улица, дом, квартира.

Найдите в словарях разные значения слова «домен».

Точка в корне дерева — это **корневой домен. Домены верхнего уровня** могут обозначать страну, каждая страна имеет свой двухбуквенный домен. Например, домены **ru**, **рф** и **su** принадлежат России.

Используя дополнительные источники, выясните, почему России принадлежит домен **su**.

Существуют и «общие» домены верхнего уровня, не принадлежащие никакой стране, например: **com**, **net**, **org**, **biz**, **info**, **name**, **museum** и др.

Используя дополнительные источники, найдите ответы на вопросы.
- Какие восемь общих доменов верхнего уровня появились в 1984 году?
- Какие домены верхнего уровня были добавлены в 2001 году?
- Каково назначение доменов верхнего уровня **edu**, **name**, **eco**, **jobs**, **mobi**, **travel**?

Таким образом, сейчас в Интернете используется две системы адресов: IP-адреса и доменные имена. Чтобы установить соответствие между ними, на специальных серверах, которые называются **DNS-серверами**, хранятся таблицы, состоящие из пар «IP-адрес — доменное имя». Их задача — по запросу компьютера-клиента вернуть IP-адрес для заданного доменного имени (или наоборот).

Когда вы вводите адрес сайта (доменное имя) в адресной строке браузера, сначала отправляется запрос на DNS-сервер, цель которого — определить IP-адрес сервера. Если это удалось, направляется запрос на получение главной (домашней) страницы сайта, причём для этого используется полученный IP-адрес, а не доменное имя.

Одному доменному имени может соответствовать несколько IP-адресов. Такой приём применяется для распределения нагрузки на сайты с большим количеством посетителей (например, **www.yandex.ru**, **www.google.com**).

В то же время несколько небольших сайтов могут размещаться на одном компьютере и иметь один и то же IP-адрес.

Запишите порядок обмена данными между компьютерами после ввода адреса **www.yandex.ru** в адресной строке браузера.

## Выводы

- Интернет не имеет единого центра управления и никому не принадлежит.
- Маршрутизатор — это компьютер, пересылающий пакеты данных между участками сети.
- Для обмена данными в Интернете используются протоколы семейства TCP/IP. Кроме того, каждая служба имеет свой протокол верхнего уровня (уровня приложения).
- IP-адрес — это числовой адрес узла сети.
- Доменное имя — это символьное имя сайта. Для преобразования доменных имён в IP-адреса используют DNS-серверы.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Вспомните, зачем данные, которые передаются по Интернету, разбиваются на пакеты.
2. Какова роль маршрутизаторов?
3. Какими способами можно подключить ноутбук к Интернету? Если все варианты доступны, какой вы выберете? Почему?

4. Приведите пример, когда компьютер может иметь несколько IP-адресов.
5. Сколько битов в памяти нужно выделить для хранения IP-адреса?
6. Зачем нужны доменные адреса?
7. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «История развития Интернета»
б) «Протоколы Интернета»
в) «Служба DNS»
г) «Домены верхнего уровня»

# § 5
# Службы Интернета

***Ключевые слова:***

- гипертекст
- адрес документа (URL)
- веб-сайт
- почтовый сервер
- файловый архив
- форум
- мессенджер
- облачный сервис
- синхронизация данных
- информационная система

**Службы (сервисы)** Интернета — это услуги, которые предоставляются пользователям. Самые популярные службы Интернета — Всемирная паутина и электронная почта.

## Всемирная паутина (WWW)

Всемирная паутина, или **«веб» (WWW)**, — это служба для доступа к гипертекстовым документам **(веб-страницам)**, хранящимся на серверах.

Используя дополнительные источники, выясните, от каких иностранных слов образовано сокращение WWW.

Как вы знаете, **гипертекст** — это текст, в котором есть активные ссылки **(гиперссылки)** на другие документы. На веб-страницах встречается не только текст, но и графика, звук, видео, причём каждый элемент может быть гиперссылкой. Такие документы называются **гипермедиа-документами.**

Всемирная паутина использует специальный протокол «верхнего» уровня **HTTP**. Полный адрес документа в Интернете содержит четыре части: протокол, адрес сервера, каталог и имя файла. Например, адрес

*http://example.com/doc/new/vasya-new.htm*

говорит о том, что для обращения к документу *vasya-new.htm*, который находится в каталоге */doc/new* на сервере *example.com*, нужно использовать протокол HTTP. Такой адрес часто называют английским сокращением — **URL**.

Используя дополнительные источники, выясните, от каких иностранных слов образованы сокращения HTTP и URL.

Иногда каталог и имя файла не указывают, например: *http://example.com*. Это означает, что мы обращаемся к главной странице веб-сайта.

**Веб-сайт** (или просто **сайт**) — это группа веб-страниц, которые расположены на одном сервере, объединены общей идеей и связаны с помощью гиперссылок.

Чтобы сайт стал доступен другим компьютерам, на сервере должна быть запущена специальная программа — **веб-сервер**.

Для просмотра веб-страниц на экране используются программы-**браузеры**. Браузер отправляет веб-серверу запрос, содержащий URL-адрес документа (веб-страницы, рисунка, файла и т. п.), а сервер в ответ передаёт запрошенные данные. Обмен обычно происходит по протоколу HTTP, однако для безопасного обмена секретной информацией, например для выполнения финансовых операций через Интернет, применяют протокол **HTTPS**, предусматривающий шифрование всех передаваемых данных.

Используя дополнительные источники, выясните, что означает последняя буква S в сокращении HTTPS.

Особенность современного Интернета — привлечение пользователей к наполнению сайтов информацией и её корректировке, к сотрудничеству, совместной деятельности в сети. Это привело к появлению термина **Веб 2.0**.

Сайты, использующие технологии Веб 2.0, например социальные сети, как правило, требуют регистрации пользователей, для этого необходим действующий адрес электронной почты. Любой

желающий может создать «личный кабинет» с собственными настройками и хранить там файлы, фотографии, видео, заметки. Другие могут комментировать эти материалы.

Пользователи объединяются в группы (сообщества) для того, чтобы вместе обсуждать интересующие их вопросы. Часто участники могут оценивать сообщения друг друга, таким образом, изменяется «репутация» (или «карма») участников, появляется некоторое соперничество.

Активно развиваются **вики-системы** — веб-сайты, структуру и содержимое которых пользователи могут изменять с помощью инструментов, которые есть на самом сайте. Самый известный вики-сайт — это свободная энциклопедия **Википедия** (русская версия размещена на сайте **ru.wikipedia.org**).

С одной стороны, Веб 2.0 расширяет возможности пользователей. С другой стороны, нужно понимать, что размещённые данные хранятся где-то на серверах, куда в принципе может получить доступ злоумышленник. Известны случаи массовых взломов учётных записей в социальных сетях и блогах. Поэтому не следует размещать в Интернете информацию, опубликование которой как-то может вам повредить, даже теоретически.

Фактически на таких сайтах пользователи сами заполняют базу данных о себе, своих друзьях, карьере и даже личной жизни. Изучая и анализируя эти данные, владельцы сайтов и спецслужбы получают возможность манипулировать людьми, используя полученную информацию в своих целях, например для рекламы товаров. Очень часто социальные сети используются для распространения вредоносных программ и рекламных сообщений (*спама*).

## Электронная почта

Электронная почта — один из первых сервисов (служб) Интернета, но и сейчас она играет в сети огромную роль. Для того чтобы отправлять и принимать сообщения, пользователь должен зарегистрировать почтовый ящик на одном из почтовых серверов в Интернете.

Электронный адрес состоит из двух частей — названия почтового ящика и имени сервера; они разделяются символом @, который в России называют «собакой» (его официальное название — «*коммерческое at*»). Адрес *vasya@mail.ru* означает почтовый ящик[1] *vasya* на сервере *mail.ru*.

---

[1] Строго говоря, здесь *vasya* — это учётная запись (аккаунт) на сервере *mail.ru*, с которой связан почтовый ящик.

Используя дополнительные источники, выясните, кто и когда предложил использовать знак @ в адресах электронной почты. Как называют этот знак в других странах? Какие другие значения он имеет?

Для отправки сообщения ваш компьютер должен обменяться данными с почтовым сервером по протоколу **SMTP**. Затем электронное письмо передаётся на сервер, где зарегистрирован почтовый ящик адресата (на рис. 1.13 — это сервер **gmail.com**). Письмо сохраняется на сервере до тех пор, пока адресат (Джон) со своего компьютера не примет пришедшую ему почту, используя протокол **POP3** или протокол **IMAP**.

Используя дополнительные источники, выясните, как образованы сокращения SMTP, POP3 и IMAP.

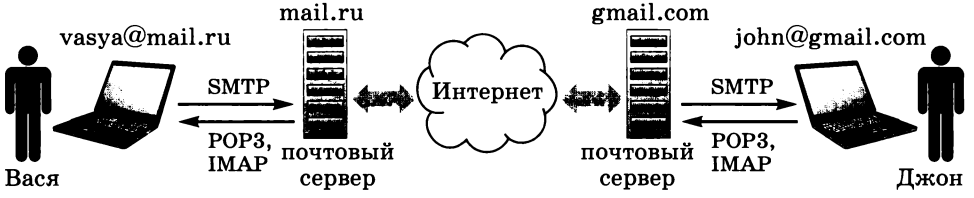



**Рис. 1.13**

Работа электронной почты очень напоминает работу обычной (бумажной) почты. Чтобы отправить письмо (по протоколу SMTP), вы идёте в почтовое отделение (на ваш почтовый сервер). Затем письмо передаётся в почтовое отделение, где живёт адресат, причём вы чаще всего не знаете, как именно это происходит. Наконец, почтальон (по протоколу POP3 или IMAP) приносит письмо адресату домой.

Сообщение электронной почты состоит из заголовка, текста письма и вложенных файлов.

**Заголовок** содержит служебную информацию, необходимую для пересылки. Вот пример информации в заголовке:

| | |
|---|---|
| Кому: | john@gmail.com |
| От кого: | vasya@mail.ru |
| Ответить: | vasya-home@mail.ru |
| Копия: | boss@mail.ru |
| Тема: | О покупке слона |

Поле «Ответить» используется тогда, когда сообщение посылается с одного адреса (например, с рабочего), а отвечать нужно на

другой (домашний). По всем адресам, указанным в поле «Копия», отправляются копии письма.

Считается плохим тоном не заполнять поле «Тема» осмысленным текстом. Во-первых, часто сообщения без темы удаляются сразу как спам. Во-вторых, многие люди получают десятки сообщений в день, и им удобно сразу сортировать письма по темам, а потом уже читать. В-третьих, искать нужное сообщение среди десятков других тоже удобнее всего по полю «Тема».

Используя дополнительные источники, выясните, зачем применяют поле «Скрытая копия» в заголовке письма.

**Основная часть** письма тоже строится по правилам, похожим на правила составления бумажных писем. Сначала идёт приветствие, затем суть сообщения, в конце — фамилия и имя автора, а если это официальное письмо — его должность и сведения об организации. Например:

Здравствуй, Джон!
Нет ли у тебя желания купить слона?
С уважением, Василий Рогов,
генеральный директор ООО «Слонопотам»,
г. Солнечный, ул. Слоновья, 2
тел. +7 (1812) 111-22-33, факс +7 (1812) 111-22-34
http://slonopotam.ru

Укажите все ошибки в письме, которое приведено в рабочей тетради.

Вместе с письмом можно отправить файлы. Многие почтовые серверы запрещают пересылку исполняемых файлов (с расширением *exe*), потому что так могут распространяться вирусы и вредоносные программы.

Для обмена файлами объёмом больше нескольких мегабайт можно использовать облачные хранилища, т. е. дисковую память на файловых серверах в Интернете (например, **cloud.mail.ru** и **disk.yandex.ru**).

Существуют несложные **правила этикета при работе с электронной почтой:**

- всегда точно формулируйте тему письма;
- начинайте письмо с приветствия, заканчивайте подписью (вашим именем);
- пишите грамотно, правильно расставляйте знаки препинания, проверяйте текст с помощью систем автоматической проверки орфографии;

- не используйте сокращения и жаргонные выражения, которые могут быть непонятны адресату;
- даже если основное содержание письма — это файл в приложении, обязательно пишите комментарий в теле письма (пустое письмо многие воспринимают как признак неуважения), например:

  Привет, Сеня!
  Посылаю эссе в приложении. Что ты об этом думаешь?
  Серафим.

- не пишите текст всеми прописными буквами, это воспринимается как крик;
- прежде чем отправить письмо, внимательно перечитайте его; отправленное письмо уже не вернуть;
- старайтесь сразу отвечать на полученные письма или хотя бы подтверждать их получение, например:

  Привет, Серафим!
  Эссе получил. Подробнее отвечу завтра, когда прочитаю.
  Сеня.

Работать с электронной почтой можно прямо в браузере (такая возможность есть у большинства почтовых серверов) или с помощью специальных программ — **почтовых клиентов.** Если вы используете почтовую программу, ваша почта хранится на вашем компьютере и будет всегда доступна, даже когда нет связи с Интернетом.

Почтовые программы «умеют» создавать, отправлять и принимать сообщения, проверять почту через заданный интервал времени, раскладывать сообщения по папкам. Часто используемые адреса можно хранить в адресной книге.

В состав современных версий операционной системы *Windows* входит почтовая программа **Почта Windows.** Несколько бóльшими возможностями обладают профессиональные программы **Microsoft Outlook** (входящая в пакет *Microsoft Office*) и **TheBat.** На компьютерах фирмы *Apple* устанавливается почтовый клиент **Apple Mail.**

www Используя дополнительные источники, выясните, какие существуют бесплатные почтовые программы, работающие в операционных системах *Windows*, *Linux* и *mac OS*.

## Файловые архивы

Для обмена файлами существуют файловые архивы — «склады» файлов. Если зайти на такой сайт в браузере, вместо красочных

веб-страниц вы увидите просто список файлов и папок. Файл начинает скачиваться, если щёлкнуть на его имени.

Для обмена файлами используется протокол **FTP**. На компьютере-сервере работает специальная программа — **FTP-сервер**, которая принимает запросы клиентов и отвечает на них по протоколу FTP. Поэтому URL-адреса файлов начинаются с символов ftp://, например:

*ftp://files.example.com/pub/firebird.zip*

FTP-сервер позволяет **скачивать** файлы на компьютер пользователя (англ. *download*) и **загружать** файлы на удалённый компьютер (англ. *upload*).

Используя дополнительные источники, выясните, как образовано сокращение FTP. www

FTP-серверы используются для распространения бесплатного программного обеспечения, обновлений антивирусных баз, а также для загрузки файлов на веб-сайт.

Для работы с FTP-сервером необходимо зарегистрироваться под своим кодовым именем — *логином (*от англ. *log in*) и ввести пароль (англ. *password*). Многие FTP-серверы разрешают анонимный вход: вместо имени нужно ввести слово *anonymous* (анонимный), а вместо пароля — любую последовательность символов.

Если вы точно знаете имя файла, с помощью специальных поисковых систем (например, **www.filesearch.ru**) можно найти FTP-сервер, где он находится.

Используя поисковую систему **www.filesearch.ru**, найдите на FTP-серверах файлы *syntrace.zip* и *minimax.ps*, определите их адреса и размер. www

## Форумы

**Форумы** — это специальные веб-сайты, предназначенные для общения посетителей в форме обмена сообщениями. Сообщения хранятся на серверах в Интернете и поэтому доступны всем участникам в любой момент.

Администратор создаёт несколько разделов форума, отличающихся по тематике. Пользователи создают в этих разделах **темы** для обсуждения (иногда тему называют **топик** или **тред**).

Используя дополнительные источники, выясните, от каких иностранных слов произошли слова «топик» и «тред». www

Участники могут отвечать на любые сообщения в теме, комментировать их (рис. 1.14). Для изучения общественного мнения автор первого сообщения темы (**топик-стартер**, от англ. *topic starter* — тот, кто начал тему) может добавить к ней опрос (голосование).

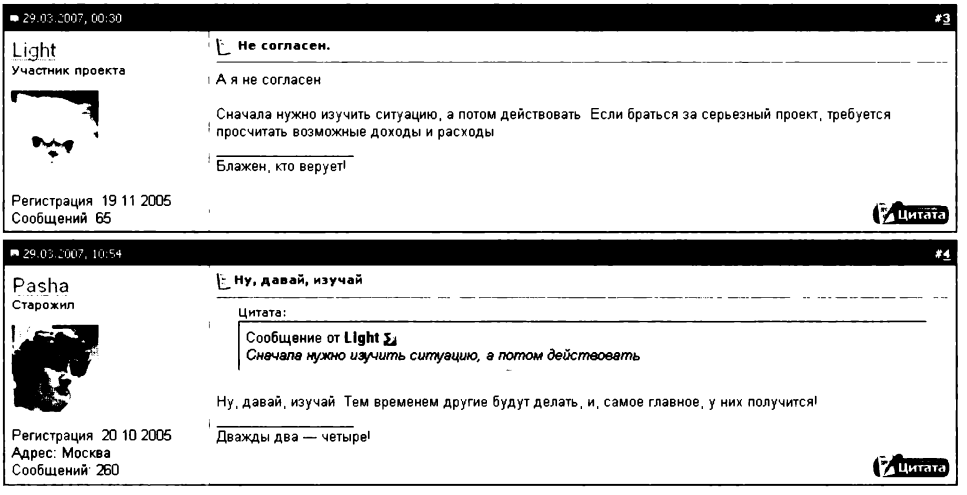



**Рис. 1.14**

Большинство форумов могут просматривать все желающие. Для того чтобы отправить своё сообщение, обычно требуется регистрация. Могут быть и закрытые разделы, для доступа в которые кроме регистрации нужно специальное разрешение администратора.

www Используя дополнительные источники, выясните происхождение и значение слов **ник**, **аватар**, **юзерпик**.

На большинстве форумов работает система личных сообщений — внутренняя электронная почта форума.

За порядком следят ответственные участники — **администратор форума** и назначенные им **модераторы**. Они могут изменять, перемещать и удалять любые сообщения, удалять профили пользователей и ограничивать им доступ — **банить**, т. е. запрещать отправлять сообщения. Обычно в форумах наказывается:

- отклонение от темы — **оффтопик**;
- оскорбление участников, нецензурная брань;
- реклама.

www Используя дополнительные источники, выясните происхождение и значение слов «модератор», «бан», «оффтопик».

На некоторых форумах есть список часто задаваемых вопросов и ответов на них. В английском языке для такого списка используется сокращение **FAQ** *(Frequently Asked Questions)*, а в русском — **ЧаВо**. Перед тем как создать новую тему, нужно попытаться найти ответ на вопрос самостоятельно, прочитав этот документ (иначе вас могут забанить).

Фирмы, которые продают аппаратуру и программное обеспечение, часто создают на своих сайтах форумы, где их представители помогают пользователям решать возникающие вопросы.

## Онлайн-общение

Для общения **в реальном времени** (это значит, что собеседники одновременно находятся за компьютерами) используют программы для обмена мгновенными сообщениями — **мессенджеры**. Самые известные мессенджеры для настольных компьютеров — **Mail.ru Агент**, **Kopete** (для *Linux*), **iChat** (для компьютеров фирмы *Apple*). Для смартфонов и других мобильных устройств популярны программы **WhatsApp**, **Google Hangouts** и **Telegram**.

С помощью программы **Skype** можно установить через Интернет голосовую и видеосвязь между компьютерами. Для этого на каждом из компьютеров должен быть микрофон и веб-камера (без них можно работать в режиме чата).

Используя материалы Интернета, постройте в тетради круговую диаграмму, которая сравнивает количество пользователей различных программ-мессенджеров для мобильных устройств.

## Облачные сервисы

Когда не было компьютерных сетей, все программы и данные хранились на компьютере пользователя. Каждому нужно было покупать и устанавливать программное обеспечение, при отказе жёсткого диска все данные могли быть потеряны.

Сейчас скорость доступа к Интернету постоянно увеличивается, поэтому появилась идея хранить данные и даже программы на удалённых серверах, в облаке. Здесь слово «облако» означает сложную систему, внутреннее устройство которой мы не знаем. Например, в облачном хранилище данных можно сохранять свои файлы. При этом неизвестно, где именно они хранятся, какое программное обеспечение используется и т. д. Но к этим дан-

ным всегда можно обратиться из любого места, где есть доступ к Интернету.

Используя материалы Интернета, составьте список наиболее популярных облачных сервисов для хранения данных и сравните объёмы хранилищ, которые предоставляются бесплатно.

Многие облачные хранилища могут выполнять **синхронизацию данных**. Предположим, что вы используете настольный компьютер, ноутбук и смартфон, где работаете с одними и теми же файлами. Конечно, хочется, чтобы при обновлении файла на одном компьютере он автоматически обновлялся на другом. Это и есть синхронизация, она выполняется через Интернет, с помощью облачного хранилища.

Предположим, что вы изменили файл на домашнем компьютере. Программа-клиент сразу же автоматически обновит его в облачном хранилище. Затем, когда вы включите ноутбук, запущенная на нём программа-клиент обнаружит, что файл в облаке более новый, и обновит файл на диске ноутбука.

В облаке можно размещать не только данные, но и программы. Вы уже знакомы с сервисом **Google Docs**, с помощью которого можно в браузере редактировать документы, не устанавливая на свой компьютер никаких программ. Это пример сервиса типа «программное обеспечение как услуга».

Используя дополнительные источники, выясните, что обозначает сокращение **SaaS**.

Фирмы могут даже взять в аренду компьютеры-серверы со всем необходимым программным обеспечением и разместить на них все данные, которые используют сотрудники.

У облачных сервисов есть и недостатки. Во-первых, для их использования нужно иметь скоростной канал доступа к Интернету. Во-вторых, ваши данные размещаются непонятно где, и непонятно, кто имеет к ним доступ. Поэтому в облаке нельзя хранить секретную и личную информацию, которую вы не хотите распространять.

## Информационные системы

**Информационная система** — это аппаратные и программные средства, предназначенные для того, чтобы своевременно обеспечить пользователей нужной информацией.

Информационная система содержит базу данных и программное обеспечение для поиска информации.

На многих сайтах доступны прогнозы погоды на разные сроки (**pogoda.ru**, **gismeteo.ru**, **pogoda.yandex.ru**). С помощью сервиса **rasp.yandex.ru** можно узнать расписание электричек, поездов дальнего следования и самолётов по всей России. На сайтах аэропортов постоянно обновляется табло фактического прибытия и отправления самолетов с учётом задержки рейсов.

Заказывать билеты на поезда и самолёты удобно на специальных сайтах, которые связаны с системой продажи билетов железной дороги и авиакомпаний. Здесь же можно получить полную информацию о расписании, возможных вариантах проезда (перелёта), стоимости и наличии билетов. Оплатить билеты можно прямо на сайте с помощью банковской карты или платёжных систем, а также через терминалы приёма платежей.

Широко используются **электронные билеты** (англ. *e-ticket*) на поезда и самолёты. Электронный билет — это информация о заказе, сделанном через веб-сайт, которая внесена в базу данных. Для регистрации по электронному билету в аэропорту или в поезде достаточно просто предъявить паспорт.

Сервисы веб-картографии **Google Maps** (**maps.google.ru**), **Яндекс.Карты** (**maps.yandex.ru**), **Карты@Mail.ru** (**maps.mail.ru**) позволяют найти на карте любой адрес, проложить маршрут и оценить его длину. На этих сайтах доступны не только карты (хранящиеся в векторном формате), но также снимки многих районов из космоса.

Практически во всех организациях и государственных органах документы хранятся в базах данных в цифровом виде. На портале **www.gosuslugi.ru** граждане могут подать через Интернет заявление на оказание различных государственных услуг, например на оформление заграничного паспорта, представление налоговой декларации или регистрацию предприятия.

## Выводы

- Всемирная паутина — это служба для доступа к гипертекстовым документам.
- Веб-сайт — это группа веб-страниц, которые расположены на одном сервере, объединены общей идеей и связаны с помощью гиперссылок. Чтобы сайт стал доступен другим компьютерам, на сервере должна быть запущена специальная программа — веб-сервер.
- Для того чтобы отправлять и принимать электронные сообщения, пользователь должен зарегистрировать почтовый ящик на одном из почтовых серверов в Интернете.
- Файловые архивы предназначены для распространения файлов.
- Форумы — это специальные веб-сайты, предназначенные для общения посетителей в форме обмена сообщениями.

- Для общения в реальном времени (онлайн-общения) используют программы для обмена мгновенными сообщениями (мессенджеры).
- Облачный сервис — это возможность пользоваться различными программами, дисковым пространством, и аппаратурой удалённо, через Интернет.
- Информационная система — это аппаратные и программные средства, предназначенные для того, чтобы своевременно обеспечить пользователей нужной информацией.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Почему при совершении денежных операций через Интернет не используется протокол HTTP?
2. Как вы думаете, можно ли передавать по Интернету секретную информацию? Если да, какие правила нужно соблюдать?
3. Может ли работать сайт без веб-сервера? Почему?
4. Вася не хочет регистрироваться на почтовом сервере. Сможет ли он отправить сообщение по электронной почте?
5. Какие информационные системы вы использовали? Какие используют ваши родственники?
6. Для вашего места жительства сравните прогнозы погоды на неделю, полученные с помощью разных сайтов. Как вы думаете, почему они различаются?
7. Используя информационные системы Интернета, оцените время, за которое вы можете быстрее всего добраться в выбранный город. Какой транспорт вы можете использовать? Сколько это будет стоить?
8. Варфоломей планирует лететь на самолёте из Нижнего Новгорода в Анапу 25 июля, так чтобы 26 июля начать культурный отдых на море. Используя информационные системы в Интернете, предложите ему возможные варианты перелёта.
9. Используя сервисы для работы с картами, проложите маршрут из школы домой и определите его длину.
10. Выполните по указанию учителя задания в рабочей тетради.

## Подготовьте сообщение

а) «Веб 2.0 — "за»" и "против"»
б) «Что такое Веб 3.0?»
в) «Что такое вики?»
г) «Облачные сервисы»
д) «Геоинформационные системы»

## Интересные сайты

**mail.ru** — портал с бесплатной электронной почтой
**mail.yandex.ru** — бесплатная Яндекс-почта
**ritlabs.com** — почтовая программа *TheBat*
**mozilla-russia.org** — бесплатная почтовая программа *Mozilla Thunderbird*
**drive.google.com** — облачное хранилище данных *Google Drive*
**disk.yandex.ru** — облачное хранилище данных *Яндекс Диск*
**cloud.mail.ru** — облачное хранилище данных *Облако@Mail.Ru*

## Практические работы

Выполните практические работы:
№ 2 «Службы Интернета»;
№ 3 «Информационные системы».

# § 6
# Веб-сайты

***Ключевые слова:***

- статические веб-страницы
- динамические веб-страницы
- язык HTML
- веб-программирование
- веб-приложение
- скрипт
- система управления содержимым (CMS)
- хостинг

## Веб-страницы

Веб-сайт состоит из веб-страниц. **Веб-страницы** — это обычные текстовые файлы (в формате «только текст», англ. *plain text*). Для того чтобы определить структуру документа (заголовки, абзацы, списки и др.), используют язык **HTML**. Это именно язык разметки, а не полноценный язык программирования: в нём нет переменных, циклов, ветвлений, процедур и функций.

Используя дополнительные источники, выясните, от каких слов образовано сокращение HTML.

www

В языке HTML используются команды особого типа — **тэги**. Существуют тэги для выделения заголовков, абзацев, списков. С помощью тэгов в веб-страницы добавляют рисунки, звуки, анимацию, видео, которые хранятся на сервере в виде отдельных файлов. Часто для дополнительных данных на сайте создаются специальные ката-

логи, например рисунки могут быть размещены в каталоге *images*, звуковые и видеофайлы — в каталоге *media* (рис. 1.15).

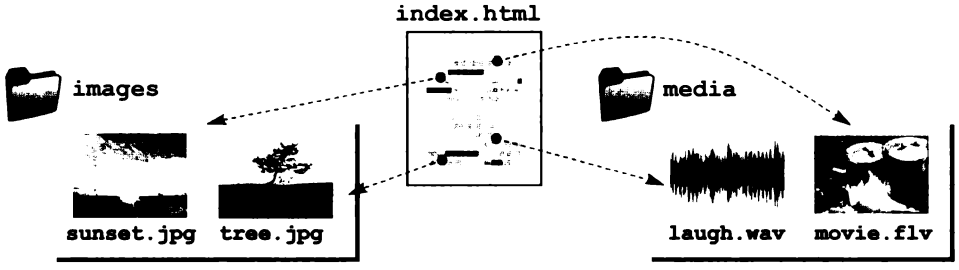



**Рис. 1.15**

Браузер, получив от сервера запрошенную веб-страницу, обрабатывает её текст и выводит информацию на экран в удобной для человека форме. Встретив команды для вставки дополнительных данных (например, рисунков), браузер запрашивает их с сервера. Таким образом, для полной загрузки веб-страницы может потребоваться несколько обращений (*запросов*) к серверу.

На веб-странице размещены 5 рисунков (каждый в виде отдельного файла), 2 звуковых файла и 4 ссылки на видео на сервере **youtube.com**. Сколько запросов направит браузер веб-серверу?

Несмотря на существующие стандарты языка *HTML*, разные браузеры могут по-разному показывать одну и ту же веб-страницу. Поэтому профессиональные разработчики обязательно проверяют, чтобы сайт выглядел, по возможности, одинаково в разных браузерах (это свойство называют *кроссбраузерностью сайта*).

Веб-страницы можно разделить на два типа:

- *статические* веб-страницы (они обычно имеют расширения *htm* или *html*) хранятся на сервере в готовом виде;
- *динамические* веб-страницы (с расширениями *php*, *asp* и др.) — полностью или частично создаются на сервере в момент запроса.

Используя дополнительные источники, выясните, на каких языках программирования написаны динамические веб-страницы с расширениями *php*, *asp*, *py*, *pl*.

**Статические страницы** меньше нагружают сервер и быстрее загружаются, потому что их текст полностью готов, а серверу остается просто переслать его по сети. Однако они не позволяют работать с изменяющимися данными: выбрать информацию из

базы данных, добавить комментарии к фотографиям, построить гостевую книгу и т. п. Кроме того, для того чтобы поддерживать сайт (вносить изменения в его содержание и дизайн), нужен квалифицированный работник, знающий язык *HTML* и способный исправлять код страниц. Статические веб-страницы можно использовать на небольших сайтах-визитках, содержимое которых изменяется только автором.

**Динамические страницы** — это файлы, которые содержат программы. В результате работы этих программ строится веб-страница на языке HTML. Когда сервер получает запрос на такую страницу, он выполняет содержащуюся в ней программу. При этом можно загружать информацию из базы данных, хранящейся на сервере, и добавлять в базу данных информацию пользователя — рисунки, видео, комментарии.

Практически все крупные сайты состоят из динамических веб-страниц. Однако создание динамического веб-сайта — достаточно сложная задача, для решения которой нужно (кроме знания языка *HTML*) уметь программировать на одном из серверных языков (*PHP*, *Python*, *ASP*, *Perl*).

Как правило, динамические сайты работают значительно медленнее, чем статические. Это связано с тем, что серверу при получении запроса необходимо обратиться к базе данных, построить запрошенную страницу в памяти и только потом переслать её по сети на компьютер клиента.

Постройте в тетради таблицу, в которой сравниваются достоинства и недостатки статических и динамических веб-страниц.

## Веб-программирование

**Веб-программирование** — это программирование динамических сайтов в Интернете.

Результат работы веб-программиста — это **веб-приложение**, т. е. программа, обеспечивающая работу сайта.

Веб-программисты разрабатывают два типа программ:

- серверные, которые работают на веб-сервере;
- клиентские, которые выполняются в браузере на компьютере пользователя.

Для создания серверных программ используют языки **PHP**, **Python**, **ASP**, **Perl**. Их изучение выходит за рамки школьного курса.

Клиентские программы, которые внедрены в веб-страницы, пишут на языке **JavaScript**. Такой подход часто называют **динамическим HTML** (англ. **DHTML**: *Dynamic HTML*). Его основная цель — обеспечить *интерактивность*, т. е. сделать так, чтобы веб-страница реагировала на действия пользователя.

Программа на языке *JavaScript* называется сценарием или скриптом.

**Скрипт**, или **сценарий** (англ. *script*), — это программный код для автоматизации какой-то операции пользователя.

С помощью скрипта можно изменять содержимое и оформление веб-страницы в ответ на действия пользователя, например:

- заменять текст, оформление, рисунки;
- строить многоуровневые выпадающие меню;
- скрывать и открывать части страницы;
- проверять данные, введённые пользователем;
- выполнять вычисления и т. д.

Используя дополнительные источники, выясните, на каком ещё языке кроме *Javascript* можно писать скрипты на веб-страницах. В чём недостаток этого языка?

## Системы управления сайтом

Для управления большими сайтами применяют **системы управления сайтом (CMS)**. Так называются информационные системы, позволяющие редактировать содержание сайта (добавлять, изменять и удалять материалы) без знания языка HTML.

Используя дополнительные источники, выясните, от каких слов образовано сокращение *CMS*.

**CMS** — это набор программ, написанных на серверном языке, например на *PHP*. На сайте, который использует *CMS*, есть панель администратора. С помощью этой панели администратор сайта может добавлять, редактировать и удалять материалы на сайте без привлечения специалиста.

Использование *CMS* значительно ускоряет создание и обслуживание сайта. Разработчику не нужно думать о том, как выбирать информацию из базы данных или сделать форму для отправки писем, потому что система это уже умеет делать. Вместо этого он может сразу заниматься содержанием и оформлением сайта. Внешний вид сайта изменяется с помощью шаблонов, в которых задано расположение и оформление элементов веб-страниц.

Найдите в Интернете названия бесплатных *CMS* и создайте в тетради таблицу адресов сайтов, откуда эти *CMS* можно загрузить.

## Размещение сайта

Обычно веб-сайты создаются для того, чтобы их можно было просматривать с любого компьютера, имеющего выход в Интернет. Поэтому сайт нужно размещать на компьютере, который подключен к Интернету круглые сутки.

Конечно, можно хранить сайт на диске своего домашнего компьютера, но этот вариант имеет много недостатков:

- нестабильность канала связи с Интернетом, например при сбоях питания;
- компьютер должен быть постоянно включён;
- на компьютере нужно установить и настроить веб-сервер — программу, которая принимает запросы браузеров с других компьютеров и возвращает им нужные веб-страницы;
- вам придётся самостоятельно организовывать защиту сайта от взломщиков, вредоносных программ и сетевых атак.

Поэтому чаще всего сайты находятся на серверах компаний, которые оказывают услуги **хостинга**, т. е. размещают сайты, занимаются их обслуживанием и отвечают за сохранность данных.

Используя дополнительные источники, выясните, что такое **DDoS-атака** на сайт.

Как правило, хостинг — это платная услуга, её стоимость зависит от выбранного тарифного плана. Тарифный план определяет максимально допустимый размер сайта, возможность создания динамических страниц, поддержку работы с базами данных, количество адресов электронной почты и т. п.

Существуют бесплатные хостинги (например, **ucoz.ru**), где можно размещать свои сайты. «Платой» за размещение сайтов служит реклама, которая автоматически встраивается в каждую веб-страницу.

Чаще всего файлы загружаются на сайт по протоколу *FTP*. Владельцу сайта выдается имя пользователя (логин) и пароль для входа на FTP-сервер хостинговой компании. С помощью специальных программ — FTP-клиентов — можно работать с файлами и папками на удалённом сервере так же, как на своём компьютере: создавать и удалять папки, скачивать файлы на свой компьютер, загружать файлы на сервер, переименовывать и удалять их.

Сравните тарифные планы на нескольких разных хостингах (по указанию учителя).

Найдите в Интернете статистику использования трёх самых популярных веб-серверов. Постройте в тетради по этим данным круговую диаграмму. Не забудьте, что кроме этих трёх серверов есть ещё и другие.

## Выводы

- Веб-страница — это текстовый файл, написанный на языке HTML.
- Различают статические и динамические веб-страницы.
- Статические страницы хранятся на сервере в готовом виде и сразу передаются клиенту по его запросу.
- Динамические страницы создаются в памяти сервера после получения запроса; их можно использовать для работы с базами данных. Динамические сайты больше нагружают сервер и работают медленнее, чем статические.
- Веб-программирование — это программирование динамических сайтов в Интернете. Результат работы веб-программиста — это веб-приложение, т. е. программа, обеспечивающая работу сайта.
- Скрипт, или сценарий, — это программный код для автоматизации какой-то операции пользователя.
- Система управления сайтом (CMS) — это программа, которая содержит инструменты для добавления, редактирования и удаления информации на сайте.
- Хостинг — это услуга по размещению сайта на сервере, который все время подключён к сети, и его обслуживанию.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Как вы думаете, почему рисунки на веб-сайте обычно сохраняют в отдельной папке?
2. Можно ли просматривать веб-страницу без браузера?
3. Как вы думаете, почему разные браузеры могут по-разному показывать одну и ту же веб-страницу?
4. Почему большие сайты делают на основе динамических страниц?
5. Как вы думаете, какие недостатки имеют системы управления сайтом?

6. За счёт чего зарабатывают бесплатные хостинги?
7. Найдите в Интернете информацию о том, люди каких специальностей участвуют в создании веб-сайтов.
8. Обсудите с учителем и одноклассниками вопрос «Дизайн сайта — цель или средство?».
9. Создайте сайт вашего класса на бесплатном хостинге, который использует *CMS* (например, на **ucoz.ru**).
10. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «Профессия — веб-программист»
б) «Системы управления содержимым сайта (*CMS*)»
в) «Хостинг веб-сайтов»

### Практическая работа

Выполните практическую работу № 4 «Веб-сайты».

# § 7
# Язык HTML

***Ключевые слова:***

- тэг
- контейнер
- атрибут
- гиперссылка
- список

## Простейшая страница

Веб-страница — это текстовый файл в формате «только текст», который можно редактировать в любом текстовом редакторе, таком как *Блокнот* в *Windows*. Этот файл должен иметь расширение *htm* или *html*. Двойной щелчок на значке файла открывает его в браузере.

Как вы уже знаете, язык *HTML* — это язык разметки документа. Команды языка *HTML* — *тэги* — заключаются в угловые скобки, все символы внутри угловых скобок считаются командами и на экран не выводятся.

Используя дополнительные источники, выясните, что означает английское слово *tag*.

Простейшая веб-страница состоит из двух тэгов: начинается с открывающего тэга <html> и заканчивается закрывающим тэгом </html>:

```
<html>
</html>
```

Такие пары тэгов образуют **контейнеры**: закрывающий тэг ограничивает область действия открывающего. Закрывающий тэг всегда начинается знаком «/» (прямой слеш, от англ. *slash*).

В записанной выше странице, кроме тэгов, нет никаких данных, поэтому, если открыть её в браузере, мы увидим пустое поле.

Разберём теперь более сложный пример. Ниже мы видим код веб-страницы, а на рис. 1.16 — её вид в браузере.

```
<html>
<head>
<title>Первая страница</title>
</head>
<body>
Привет!
</body>
</html>
```

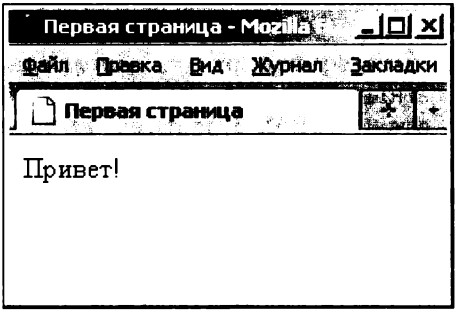



**Рис. 1.16**

Здесь два блока содержательной информации — надписи «Первая страница» и «Привет!». Посмотрев на код, можно заметить, что он разбит на две части контейнерами <head> и <body>.

Первая часть (контейнер <head>) — это головная часть страницы. Там размещается служебная информация, например ключевые слова и описание страницы для поисковых систем, кодировка символов и т. п. В нашем примере здесь всего один элемент — <title>, в нём записывают название страницы.

Найдите на рис. 1.16, в каком месте веб-страницы выводится текст из контейнера <title>.

Содержимое элемента <title> — очень важная информация, на которую поисковые системы смотрят в первую очередь. Поэтому нужно, чтобы этот текст как можно точнее отражал содержимое веб-страницы.

Вторая (основная) часть страницы расположена внутри контейнера <body>. В нашем случае там находится строка «Привет!», которую мы видим в окне браузера. В примерах, которые будут приводиться далее, мы будем писать только то, что содержится в контейнере <body>.

Используя дополнительные источники, переведите на русский язык английские слова *head*, *body* и *title*.

## Заголовки

В языке *HTML* есть специальные тэги для выделения заголовков разных уровней:

**<h1>** — заголовок документа;
**<h2>** — заголовок раздела;
**<h3>** — заголовок подраздела;
**<h4>** — заголовок параграфа.

Первая буква «h» в названии тэга связана с английским словом *header* — заголовок. Вот как выглядит текст с заголовками двух уровней (рис. 1.17).

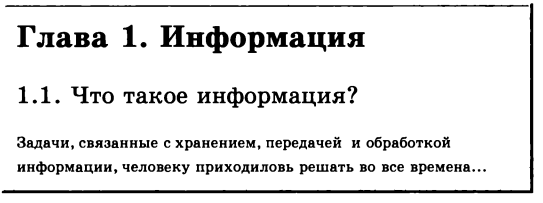



**Рис. 1.17**

Для этого требуется написать такие команды:

```
<h1>Глава 1. Информация</h1>
<h2>1.1 Что такое информация?</h2>
Задачи, связанные с хранением, передачей
и обработкой информации, человеку приходилось
решать во все времена...
```

Обратите внимание, что мы нигде не указали, как именно оформить заголовки (какой шрифт и отступы использовать и т. п.). Браузер применяет то оформление, которое установлено в его настройках *по умолчанию* (т. е. для случая, когда оформление не задано явно).

Как мы видим, заголовки выделяются жирным шрифтом увеличенного размера и выравниваются по левому краю. Если, например, заголовок документа нужно выровнять по центру, используют такую запись:

```
<h1 align="center">Глава 1. Информация</h1>
```

Здесь тэг <h1> содержит ещё дополнительное свойство, или атрибут align. Атрибуты тэгов записывают в открывающем тэге внутри угловых скобок. Значение атрибута указывают после знака равенства в кавычках. Для заголовков используют три типа выравнивания: left (по умолчанию), center и right.

www Используя дополнительные источники, переведите на русский язык английские слова *align*, *left*, *center* и *right*.

## Абзацы

Абзац в языке HTML выделяется контейнером <p> (от англ. *paragraph* — абзац) — рис. 1.18:

```
<p>
В школе учится 70 человек.
Остальные 430 учеников валяют дурака.
</p>
<p>
Сколько всего учеников в этой школе?
</p>
```

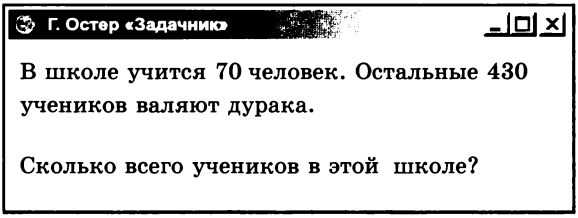



**Рис. 1.18**

По умолчанию абзацы отделяются друг от друга дополнительным интервалом, однако абзацного отступа («красной строки») браузер не делает.

Для выравнивания строк абзаца в тэг <p> можно добавлять атрибут align, как для заголовков. Кроме уже упомянутых вариантов выравнивания (left, center, right) для широких текстовых колонок часто используется значение justify — выравнивание по ширине.

Как нужно оформить для веб-страницы абзац с текстом «*Pacta sunt servanda*» с выравниванием по правому краю? Найдите перевод этого выражения на русский язык.

Для того чтобы просто перейти на новую строку, не начиная новый абзац (как при записи стихотворения), нужно использовать непарный тэг <br> (рис. 1.19).

```
И вечный бой! Покой нам
только снится <br>Сквозь
кровь и пыль...<br>Летит,
летит степная кобылица
<br>И мнёт ковыль...
```

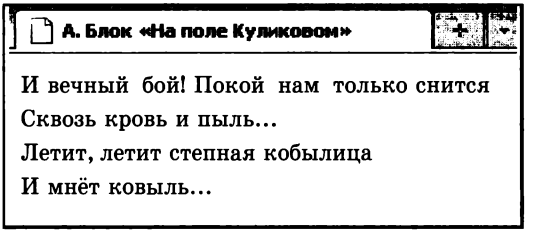



**Рис. 1.19**

Как нужно оформить на веб-странице трёхстрочное стихотворение японского поэта Р. Коно (перевод В. Маркова)?

```
Сквозь урагана рев,
Когда дрожит вся кровля,
Цикады слышен звон...
```

Используя дополнительные источники, выясните, как называется этот жанр поэзии.

## Гиперссылки

Гиперссылка связывает элемент веб-страницы с другим документом. Гиперссылки по умолчанию выделяются синим цветом и подчёркиваются. Цвет посещённых гиперссылок изменяется на фиолетовый (чтобы пользователь видел, где он уже был).

Для создания гиперссылки в языке *HTML* служит тэг <a> (от англ. *anchor* — якорь). Его атрибут href (от англ. *hyper reference* — гиперссылка) указывает на документ, который должен быть загружен после щелчка на гиперссылке. Например (рис. 1.20):

```
Переход на
<a href="newpage.html">
новую страницу</a>.
```

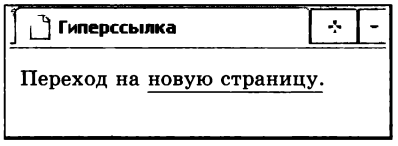



**Рис. 1.20**

Обратите внимание, что адрес связанного документа (`newpage.html`) в окне браузера не виден[1], а на экране появляется выделенное содержимое контейнера (текст «новую страницу»).

Поскольку в атрибуте `href` указано только имя файла, а не его полный адрес, браузер будет искать документ в той же папке, где находится страница со ссылкой.

Ссылка на файл `info.htm` в папке `news` запишется в виде:

```
<a href="news/info.htm">Информация</a>.
```

Две точки означают выход из каталога «наверх», в родительскую папку. Например:

```
<a href="../../news/info.htm">Информация</a>.
```

В этом случае для поиска файла `info.htm` нужно выйти на два уровня вверх по дереву папок и затем войти в папку `news`.

Все рассмотренные выше гиперссылки — *локальные* (местные), т. е. они ведут к файлам, находящимся на том же сервере. Внешняя ссылка представляет собой полный адрес документа — *URL*, который включает протокол доступа к данным, имя сервера, папки и имя самого документа. Например:

```
<a href="http://example.net/news/info.htm">Информация</a>.
```

Если имя папки и файла не указано, ссылка ведёт на главную страницу сайта:

```
<a href="http://example.net/">Главная страница</a>.
```

Оформите на языке *HTML* гиперссылку с текстом «Информация», которая ссылается на:

а) документ *moreinfo.htm* в текущей папке;
б) рисунок *more.png* в папке *images*;
в) главную страницу сайта *moreinfo.ru*;
г) архив *songs.zip* в папке *download* на сайте *moreinfo.ru*.

---

[1] Его обычно можно увидеть в строке состояния в нижней части окна браузера, если навести указатель мыши на гиперссылку.

## Списки

В языке HTML можно использовать два вида списков — *маркированные* (каждый элемент отмечен маркером) и *нумерованные* (с числовой или буквенной нумерацией).

**Маркированный список** применяется для перечисления элементов множества, когда их порядок не важен. В языке *HTML* он строится с помощью тэга `<ul>` (от англ. *Unordered List*). Каждый элемент списка вложен в контейнер `<li>` (от англ. *List Item*) — рис. 1.21:

```
<ul>
<li>Вася</li>
<li>Петя</li>
<li>Коля</li>
</ul>
```

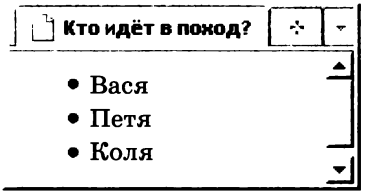



**Рис. 1.21**

Если порядок перечисления элементов важен, применяется **нумерованный список**. В языке *HTML* для этой цели используют тэг `<ol>` (от англ. *Ordered List*). В остальном оформление очень похоже на маркированный список (рис. 1.22):

```
<ul>
<li>Детский сад</li>
<li>Школа</li>
<li>Университет</li>
</ol>
```

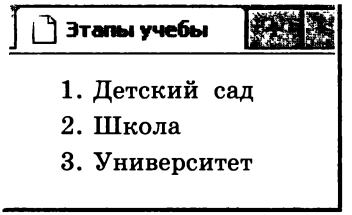



**Рис. 1.22**

Используя дополнительные источники, переведите на русский язык английские выражения *unordered list*, *ordered list*, *list item*.

Найдите ошибки в оформлении списка на веб-странице:

```
<li>Васнецов</li>
<ul>Суриков
<li>Шишкин</li>
</ol>
```

## Рисунки

Рисунки хранятся на веб-серверах в виде отдельных файлов. Браузеры умеют работать с тремя форматами растровых рисунков: *GIF*, *JPEG* и *PNG*.

Рисунки в **формате GIF** применяются для кодирования чётких изображений (схем, чертежей) или мелких деталей, имеющих не более 256 цветов. Рисунки в **формате JPEG** используют для хранения изображений с размытыми границами объектов, например фотографий. **Формат PNG** был создан для замены формата GIF, его можно применять для изображений с полупрозрачными областями.

Для вставки изображения в код веб-страницы используется непарный тэг `<img>` (от англ. *image*). Его атрибут `src` (от англ. *source*) задаёт имя файла, например:

```
<img src="night.jpg">
```

Этот файл браузер будет искать в той же папке, где находится веб-страница. Можно, как и при создании гиперссылок, указывать относительный путь к файлу:

```
<img src="../../images/night.jpg">
```

или даже загружать картинку с другого веб-сервера:

```
<img src="http://example.com/images/night.jpg">
```

По умолчанию рисунок вставляется прямо в текст как одна «большая буква» (рис. 1.23, *а*). Для того чтобы установить обтекание текстом, используют атрибут `align`:

```
<img src="night.jpg" align="left">
```

Для этого атрибута чаще всего используют значения `left` и `right` (рис. 1.23, *б* и *в*).

*а*) *б*) *в*)

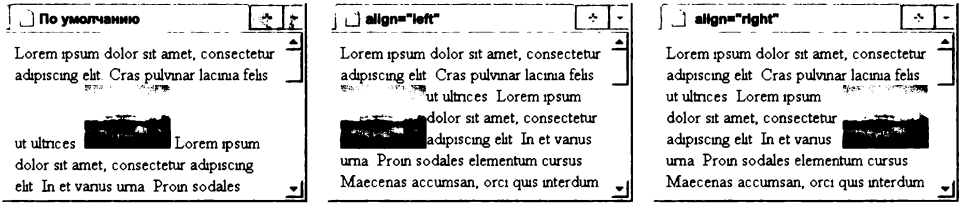



**Рис. 1.23**

Вариант с выравниванием по левому краю смотрится плохо, потому что фотография «стучится» о соседний текст, подходит к нему вплот-

ную. Чтобы этого не происходило, можно добавить отступы с помощью атрибутов `hspace` (от англ. *horizontal space*) и `vspace` (от англ. *vertical space*):

```
<img src="night.jpg" align="left" hspace="10" vspace="10">
```

Значения отступов указаны в пикселях. Вот что получается (рис. 1.24).

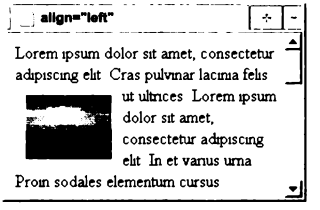



**Рис. 1.24**

Браузер может определить размеры рисунка только тогда, когда он загрузит его с сервера. Поэтому сначала вместо рисунка на веб-странице появляется небольшой прямоугольник, который потом заменяется изображением. При этом размеры поля, отведённого рисунку, меняются, остальной материал на экране тоже сдвигается, что неудобно для пользователя. Однако есть простое решение этой проблемы — заранее указать браузеру размеры рисунка (атрибуты `width` — ширина и `height` — высота):

```
<img src="night.jpg" width="800" height="600"
alt="Ночь на Ладоге">
```

Атрибут `alt` (англ. *alternative*) задаёт текст, который показывается на месте рисунка, пока тот ещё не загружен. Кроме того, значение атрибута `alt` учитывают поисковые системы.

Рисунки очень часто служат гиперссылками. Для этого тэг `<img>` нужно поместить внутрь контейнера-ссылки `<a>`, например:

```
<a href="gallery.htm"><img src="night.jpg" border="0"></a>
```

Атрибут `border` (англ. *border*) установлен равным нулю для того, чтобы убрать синюю рамку, которая появляется по умолчанию вокруг рисунка-ссылки.

Используя дополнительные источники, переведите на русский язык английские слова и выражения *image*, *source*, *align*, *vertical space*, *horizontal space*, *alternative*, *border*.

Запишите в тетради тэг, с помощью которого можно добавить в веб-страницу:
а) рисунок *dawn.png* из текущей папки;
б) рисунок *sunny-day.jpg* из папки *images* с выравниванием по правому краю;
в) рисунок *beatles.jpg* из папки *img* с сайта *mysongs.ru*, размеры рисунка — 450 × 338 пикселей, выравнивание по правой границе, замещающий текст — «Группа Битлз».

## Выводы

- Веб-страница — это текстовый файл в формате «только текст», который можно редактировать в любом текстовом редакторе.
- Для разметки веб-страниц используется язык HTML. Команды языка HTML (тэги) заключаются в угловые скобки. Тэги могут содержать дополнительные свойства — атрибуты.
- Парные тэги — контейнеры — состоят из двух частей: открывающей и закрывающей. Закрывающая часть начинается знаком / (прямой слэш).
- Существуют специальные тэги для выделения заголовков, абзацев, списков, гиперссылок, вставки рисунков.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Из каких двух блоков состоит веб-страница?
2. Почему нужно внимательно выбирать название страницы в контейнере `<title>`?
3. Объясните, что неправильно в такой записи:

```
<h1><h2>Важный момент</h2></h1>
```

4. Откуда браузер «узнаёт», как оформлять заголовки и гиперссылки?
5. Как отличить в тексте веб-страницы локальную ссылку от внешней (на другой сайт)?
6. В каких случаях лучше использовать нумерованные списки, а в каких — маркированные?
7. Выполните по указанию учителя задания в рабочей тетради.

## Подготовьте сообщение

а) «Таблицы в языке HTML»
б) «Оформление страницы с помощью стилей»
в) «Примеры использования *Javascript*»

## Интересные сайты

**htmlbook.ru** — сайт для тех, кто делает сайты
**javascript.ru** — учебник *Javascript*

## Практические работы

Выполните практические работы:
№ 5 «Простая веб-страница»;
№ 6 «Гиперссылки, списки и рисунки».

---

## ЭОР к главе 1 из Единой коллекции цифровых образовательных ресурсов (school-collection.edu.ru)

Физическая топология сети
Гарантированная доставка. Принцип гарантированной доставки
Демонстрация IP-адресации
IP-адресация. Маски подсети
Основные концепции. Числовые и символьные имена
Организация пространства имён
Службы сети Интернет
История Интернет

# Глава 2
# МАТЕМАТИЧЕСКАЯ ЛОГИКА

## § 8
## Логика и компьютеры

***Ключевые слова:***

- логика
- формальная логика
- логическое высказывание
- алгебра логики
- логические переменные
- логическая операция
- операция «НЕ»
- операция «И»
- операция «ИЛИ»
- логическая функция

### Что такое высказывание?

В быту мы часто используем слова «логика», «логично». **Логика** (от древнегреческого λογικος — наука о рассуждении) — это наука о том, как правильно рассуждать, делать выводы, доказывать утверждения.

В естественном языке рассуждения связаны с самыми разными предметами и понятиями, и поэтому изучать их достаточно сложно. Древнегреческий философ **Аристотель** начал изучать общие правила построения правильных выводов из известной информации, которая считается истинной. Такая логика называется **формальной**, она изучает истинность и ложность *логических высказываний*, отвлекаясь от их содержания.

**Аристотель (384–322 до н.э.)**

**Логическое высказывание** — это повествовательное предложение, про которое можно однозначно сказать, истинно оно или ложно.

Используя это определение, проверим, можно ли считать логическими высказываниями следующие предложения:

1) Сейчас идёт дождь.
2) Вчера жирафы улетели на север.

3) Беги сюда!
4) Который час?
5) В городе $N$ живёт более 2 миллионов человек.
6) У квадрата 10 сторон, и все разные.
7) История — интересный предмет.

Здесь высказываниями являются только предложения 1, 2 и 6, остальные не подходят под определение. Предложения 3 и 4 не повествовательные (призыв к действию и вопрос). Предложение 5 станет высказыванием только в том случае, если $N$ заменить на название конкретного города. Утверждение 7 кто-то считает истинным, а кто-то — ложным (нет однозначности), его можно более строго сформулировать в виде «По мнению $N$, история — интересный предмет». Для того чтобы оно стало высказыванием, нужно заменить $N$ на имя человека.

Используя определение, проверьте, являются ли логическими высказываниями предложения:

1) Лошади едят сено.
2) Карету мне, карету!
3) Где расположен Канин Нос?
4) Восемью три – двенадцать.
5) Программировать очень просто.

Логика и компьютеры тесно связаны. В классической формальной логике высказывание может быть истинно или ложно, третий вариант исключается[1]. Если обозначить истинное значение единицей, а ложное — нулём, то получится, что формальная логика изучает правила выполнения операций с нулями и единицами, т. е. с двоичными кодами. Оказалось, что всю обработку двоичных данных можно свести к выполнению логических операций.

**Дж. Буль (1815–1864)**

Важный шаг в этом направлении сделал английский математик **Джордж Буль**. Буль впервые ввёл в науку двоичные переменные, принимающие только два значения — «истина» и «ложь», — и три основные логические операции: НЕ, И, ИЛИ. Кроме того, он предложил применить для исследования логичес-

[1] Существуют неклассические логические системы, например трёхзначная логика, где кроме «истинно» и «ложно» есть ещё состояние «не определено» (или «возможно»).

ких высказываний методы алгебры. Позже этот раздел математики получил название *алгебры логики*, или *алгебры высказываний*. Ещё его называют *булевой алгеброй* (по имени Дж. Буля).

**Алгебра логики** — это математический аппарат, с помощью которого записывают, упрощают и преобразуют логические высказывания, вычисляют их значения.

Алгебра логики определяет правила выполнения операций с логическими значениями «ложь» и «истина». Если обозначить эти значения как 0 и 1, то получается, что с помощью алгебры логики можно описать алгоритмы работы с двоичными данными. Например, так можно построить запоминающие элементы и выполнять арифметические действия.

Используя дополнительные источники, выясните, как называлась основная научная работа Дж. Буля и в каком году она была написана. Сколько лет было тогда учёному?

## Простые и сложные высказывания

Высказывания бывают простые и сложные (составные). Простые высказывания нельзя разделить на более мелкие высказывания, например: «Сейчас идёт дождь» или «Форточка открыта». Сложные (составные) высказывания строятся из простых с помощью логических связок — **логических операций** НЕ, И, ИЛИ.

В алгебре логики высказывания обычно обозначаются латинскими буквами. Таким образом, мы уходим от конкретного содержания высказываний, нас интересует только их истинность или ложность. Например, можно обозначить буквой $A$ высказывание «Сейчас идёт дождь», а буквой $B$ — высказывание «Форточка открыта».

Так как высказывания могут быть истинными или ложными, введённые символы $A$ и $B$ можно рассматривать как **логические переменные**, которые могут принимать два возможных значения: «ложь» (0) и «истина» (1). Из них строятся сложные высказывания:

**не** $A$ = Неверно, что сейчас идёт дождь.
$A$ **и** $B$ = Сейчас идёт дождь и открыта форточка.
$A$ **или** $B$ = Сейчас идёт дождь или открыта форточка.

Если и другие логические операции, но НЕ, И и ИЛИ используются чаще всего. Оказывается, с их помощью можно выразить любую логическую операцию, поэтому эти три операции можно считать основными, **базовыми**, и говорят, что они составляют базис.

Различные устройства компьютера строятся на основе элементов, выполняющих логические операции НЕ, И, ИЛИ.

При введённых выше обозначениях $A$ и $B$ запишите на русском языке высказывания:
а) **не** $B$; б) (**не** $A$) **и** $B$; в) $A$ **или** (**не** $B$).

При тех же обозначениях запишите в символьном виде высказывания:
а) «Неверно, что сейчас идёт дождь и открыта форточка».
б) «Неверно, что сейчас идёт дождь или закрыта форточка».

## Операция НЕ

**Операция НЕ** часто называется **отрицанием** или **инверсией.** В алгебре логики всего два возможных значения (0 и 1), поэтому логическое отрицание — это переход от одного значения к другому: от 1 к 0 или наоборот. Если высказывание $A$ истинно, то НЕ $A$ ложно, и наоборот.

Используя дополнительные источники, переведите на русский язык слово *inverse*, от которого произошло слово «инверсия».

Операцию НЕ обозначают чертой сверху, например: $\overline{A}$. В школьном алгоритмическом языке[1] эта операция обозначается словом **не**, а в языке программирования Паскаль — английским словом **not**.

Используя дополнительные источники, найдите другие обозначения операции НЕ.

Операцию НЕ можно задать в виде таблицы (рис. 2.1).
Эта таблица состоит из двух частей: слева перечисляются все возможные значения исходной величины (их всего два — 0 и 1), а в последнем столбце записывают результат выполнения логической операции для каждого из этих вариантов. Такая таблица называется **таблицей истинности** логической операции. Таблица истинности задаёт *логическую функцию.*

| $A$ | не $A$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

**Рис. 2.1**

[1] Вспомним, что мы договорились называть его просто «алгоритмический язык».

**Логическая функция** — это правило преобразования входных логических значений в логическое значение-результат.

Используя таблицу истинности на рис. 2.1, определите, как можно упростить выражение **не**(**не** *A*). Рассмотрите оба варианта: когда *A* = 0 и когда *A* = 1. Сделайте вывод.

## Операция И

Из двух простых высказываний *A* и *B* (например, *A* = Сейчас идёт дождь, *B* = Форточка открыта) можно составить сложное высказывание *A* **и** *B*. Высказывание *A* **и** *B* истинно в том и только в том случае, когда оба высказывания, *A* и *B*, истинны одновременно.

Для понимания операции И можно представить себе простую схему, в которой для включения лампочки используются два выключателя, соединённых последовательно (рис. 2.2). Чтобы лампочка загорелась, нужно обязательно включить оба выключателя. Вместе с тем, чтобы выключить лампочку, достаточно выключить любой из них.

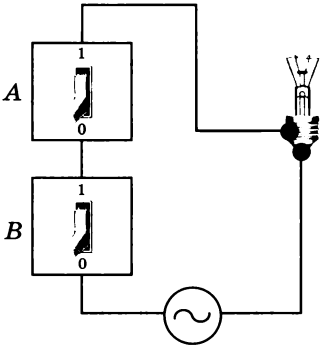



**Рис. 2.2**

Операция И (в отличие от НЕ) выполняется с двумя логическими значениями. В алгоритмическом языке системы КуМир операция И обозначается словом **и**, а в Паскале — словом **and**.

Используя дополнительные источники, найдите другие обозначения операции И.

В таблице истинности операции И будет уже не один столбец с исходными данными, а два, мы обозначим исходные данные как *A* и *B*. Число строк также выросло, с 2 до 4, поскольку с помощью 2 бит можно записать 4 разных комбинации значений двух переменных: 00, 01, 10 и 11. Как следует из определения операции И, в последнем столбце будет всего одна единица, для варианта *A* = *B* = 1 (оба высказывания, *A* и *B*, истинны одновременно) — рис. 2.3.

| *A* | *B* | *A* и *B* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Рис. 2.3**

Из значений $A$ и $B$ в каждой строке этой таблицы составьте двоичное число и запишите его в десятичной системе счисления. Почему строки в таблице расположены именно так?

Легко проверить, что этот результат можно получить «обычным» умножением $A$ на $B$, поэтому операцию И называют **логическим умножением**. Она часто обозначается знаком умножения (точкой): $A \cdot B$.

С точки зрения обычной математики, эта операция выбирает *наименьшее* из исходных значений. Математики используют ещё одно название операции И — **конъюнкция**.

Используя дополнительные источники, выясните, от какого слова произошло слово «конъюнкция» и что оно обозначает.

С помощью таблицы истинности можно упрощать логические выражения. Например, рассмотрим выражение $A$ **и** 1. По таблице истинности на рис. 2.3 получаем:

при $A = 0$: $A$ **и** 1 = 0 **и** 1 = 0

при $A = 1$: $A$ **и** 1 = 1 **и** 1 = 1.

Можно заметить, что в любом случае результат совпадает с $A$, поэтому $A$ **и** 1 = $A$.

Используя таблицу истинности операции И, упростите выражения:

*а*) $A$ **и** 0; *б*) $A$ **и** $A$; *в*) $A$ **и** (**не** $A$).

## Операция ИЛИ

Высказывание $A$ **или** $B$ (например, «Сейчас идет дождь или форточка открыта») истинно тогда, когда истинно хотя бы одно из входящих в него высказываний или оба одновременно.

В алгоритмическом языке операция ИЛИ обозначается словом **или**, а в языке Паскаль — английским словом **or**.

Используя дополнительные источники, найдите другие обозначения операции ИЛИ.

Для понимания операции ИЛИ можно представить себе схему с двумя выключателями, соединёнными параллельно (рис. 2.4).

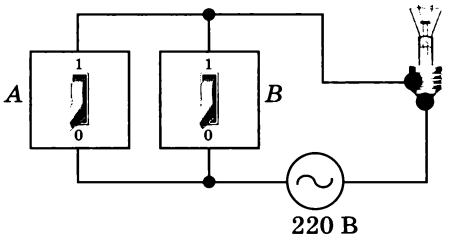



**Рис. 2.4**

Чтобы лампочка загорелась, достаточно включить хотя бы один из выключателей. Чтобы выключить лампочку, необходимо обязательно выключить оба. В таблице истинности будет только один ноль — для варианта $A = B = 0$ (рис. 2.5).

| *A* | *B* | *A* **или** *B* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Рис. 2.5**

Операцию ИЛИ называют **логическим сложением,** потому что она похожа на обычное математическое сложение. Поэтому она часто обозначается знаком сложения: $A+B$. Единственное отличие — в последней строке таблицы истинности: в математике 1+1 равно 2, а в алгебре логики — единице.

Можно считать, что в результате применения операции ИЛИ из исходных значений выбирается *наибольшее*. Другое название этой операции — **дизъюнкция.**

Используя дополнительные источники, выясните, от какого слова произошло слово «дизъюнкция» и что оно обозначает.

Запишите в тетради ответы на следующие вопросы.
- Сколько строк в таблице истинности функции с двумя переменными?
- Сколько существует возможных вариантов распределения нулей и единиц в последнем столбце?
- Сколько можно придумать различных логических функций с двумя переменными?

Используя таблицу истинности операции И, упростите выражения:
*а*) *A* **или** 0; *б*) *A* **или** 1; *в*) *A* **или** *A*; *г*) *A* **или** (**не** *A*).

## Выводы

- Логическое высказывание — это повествовательное предложение, про которое можно однозначно сказать, истинно оно или ложно.
- Алгебра логики — это математический аппарат, с помощью которого записывают, упрощают и преобразуют логические высказывания, вычисляют их значения.
- Логическая функция — это правило преобразования входных логических значений в логическое значение-результат.
- Если высказывание $A$ истинно, то **не** $A$ ложно, и наоборот.
- Высказывание $A$ **и** $B$ истинно тогда и только тогда, когда оба высказывания, $A$ **и** $B$, истинны одновременно.
- Высказывание $A$ **или** $B$ истинно тогда, когда истинно хотя бы одно из высказываний, $A$ или $B$, или оба они истинны одновременно.
- С помощью операций НЕ, И, ИЛИ можно выразить любую логическую операцию, поэтому говорят, что эти три операции составляют базис.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Объясните значения слов «логика», «формальная логика», «алгебра логики».
2. Можно ли считать высказываниями эти предложения?
   1) Не плачь, девчонка!
   2) Почему я водовоз?
   3) Купите слоника!
   4) Клубника очень вкусная.
   5) Сумма $X$ и $Y$ равна 36.
3. Как вы думаете, зачем в курсе информатики изучается логика?
4. Почему в таблице истинности для операции НЕ две строки, а для других изученных операций — четыре?
5. Сколько строк в таблице истинности выражения с тремя переменными? С четырьмя? С пятью?
6. В каком порядке обычно записываются значения переменных в таблице истинности? Зачем это нужно?
7. В чём различие арифметического и логического сложения?
8. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «Информатика и логика»
б) «Логическая операция исключающее ИЛИ»
в) «Логическая операция импликация»
г) «Логическая операция эквиваленция»

# § 9
# Логические элементы

***Ключевые слова:***

- логический элемент
- логическая схема

## Условные обозначения

Все узлы компьютера построены на логических элементах — электронных схемах, которые выполняют логические операции. В принципе, для того чтобы собрать компьютер, достаточно только базовых логических элементов — НЕ, И и ИЛИ.

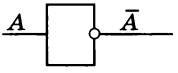



**Рис. 2.6**

Чтобы все инженеры-компьютерщики однозначно понимали схемы различных устройств, разработали государственный стандарт (ГОСТ 2.743-91), в котором определены условные обозначения логических элементов. **Элемент НЕ** обозначается прямоугольником с кружком на выходе (рис. 2.6).

Определите, какой сигнал будет на выходе в схемах, в которых последовательно соединяются: (а) два элемента НЕ; (б) три элемента НЕ (рис. 2.7).

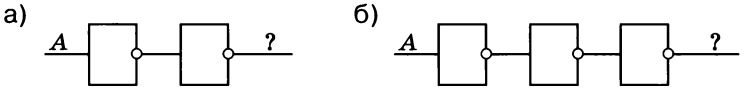



**Рис. 2.7**

Составьте таблицы истинности для обеих функций.

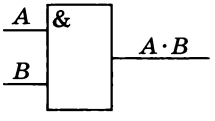



**Рис. 2.8**

**Элемент И** обозначается прямоугольником со знаком & в верхней части. У такого элемента два входа и один выход (рис. 2.8).

В иностранной литературе приняты другие условные обозначения для логических элементов. Используя дополнительные источники, выясните, как обозначаются за рубежом логические элементы НЕ, И и ИЛИ.

## Исследование логических элементов

Используя тренажёр (веб-страницу **element.htm**), постройте таблицу истинности элемента с кодовым именем NAND. Сравните её с таблицей истинности операции И. Как с помощью известных вам логических элементов собрать элемент NAND? Как бы вы его назвали?

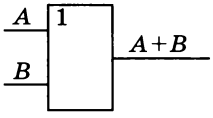



**Рис. 2.9**

**Элемент ИЛИ** обозначается прямоугольником с единицей в верхней части (рис. 2.9).

Используя тренажёр (веб-страницу **element.htm** на сайте поддержки учебника **http://kpolyakov.spb.ru/school/osnbook.htm**), постройте таблицу истинности элемента с кодовым именем NOR. Сравните её с таблицей истинности операции ИЛИ. Как с помощью известных вам логических элементов собрать элемент NOR? Как бы вы его назвали?

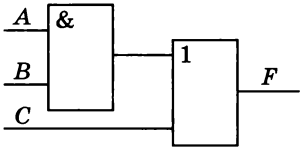
Используя тренажёр, постройте таблицы истинности элементов с кодовыми именами XOR и EQV. Как бы вы их назвали?

Как, используя элемент XOR и другие известные вам элементы, построить элемент EQV? Нарисуйте схему в тетради.

Сколько существует различных комбинаций значений трёх логических переменных? Сколько строк будет в таблице истинности логической схемы с тремя входами? В каком порядке вы их запишете?

Постройте таблицу истинности и запишите логическую функцию для каждой логической схемы (рис. 2.10).

*а)*




*б)*

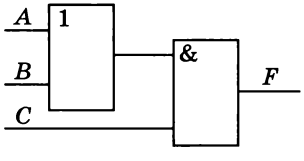



**Рис. 2.10**

Используя тренажёр (веб-страницу **element2.htm** на сайте поддержки учебника **http://kpolyakov.spb.ru/school/osnbook.htm**), постройте таблицы истинности схем с кодовыми именами KD и DK. Сравните их с таблицами истинности, полученными в предыдущем задании.

Используют также элементы, у которых некоторые входы или выходы отмечены кружками. Эти кружки обозначают дополнительные элементы НЕ, которые установлены на входе или на выходе. Например, схема на рис. 2.11, *а* может быть нарисована в подробной форме — так, как на рис. 2.11, *б*.

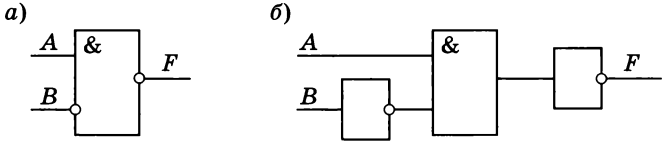



**Рис. 2.11**

Для каждой схемы (рис. 2.12) постройте таблицу истинности и запишите логическую функцию. Выполните работу в парах — один выполняет задания для первых трёх схем, другой — для последних трёх.

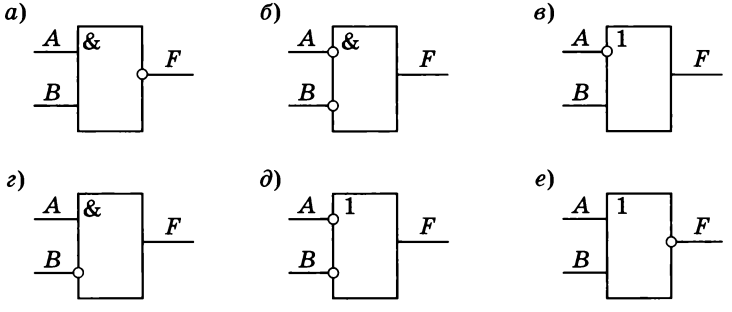
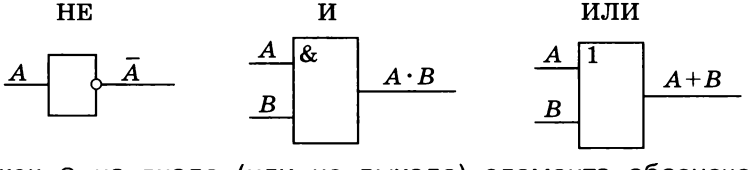



**Рис. 2.12**

Определите, какие логические схемы выполняют одну и ту же функцию. Запишите равенства, которые вы только что обнаружили.

## Выводы

- Все узлы компьютера построены на логических элементах — электронных схемах, которые выполняют логические операции.
- Для того чтобы собрать компьютер, достаточно только базовых логических элементов — НЕ, И, ИЛИ.
- Элементы, которые выполняют логические операции НЕ, И и ИЛИ, обозначаются на схемах логических устройств следующим образом:




- Кружок ○ на входе (или на выходе) элемента обозначает операцию НЕ, применённую к входному (или выходному) сигналу.

Нарисуйте в тетради интеллект-карту этого параграфа.

### Вопросы и задания

Выполните по указанию учителя задания в рабочей тетради.

### Практическая работа

Выполните практическую работу № 7 «Логические элементы».

# § 10
# Другие логические операции

***Ключевые слова:***

- импликация
- исключающее ИЛИ
- эквиваленция

Мы изучили две логические операции с двумя переменными — И и ИЛИ. Существуют ли другие? И сколько их?

В таблице истинности любой логической операции с двумя переменными всего четыре строки, и эти таблицы отличаются только четырьмя значениями функции в последнем столбце. Поэтому можно придумать всего $16 = 2^4$ различных логических операций с двумя переменными.

На уроках математики вы доказывали теоремы, в которых встречались выражения «если ..., то...», «тогда и только тогда, когда...». Эти связки тоже обозначают логические операции, с которыми мы сейчас и познакомимся.

## Импликация

Слово «**импликация**» означает «**следование**» — из одного утверждения следует другое. Если истинно первое, то истинно и второе. Например, возьмём высказывание

$X$ = Если идёт дождь, то Лена раскрывает зонтик.

Если ввести обозначения для простых высказываний:

$A$ = Идёт дождь,

$B$ = Лена раскрывает зонтик,

то можно записать высказывание $X$ в символьном виде:

$X = A \to B$.

Стрелка вправо обозначает логическую операцию «импликация».

Используя дополнительные источники, выясните, от какого иностранного слова произошло слово «импликация». Что оно означает?

Теперь давайте разберёмся, когда такое высказывание будет истинно, а когда — ложно. Если дождь идёт ($A = 1$) и Лена раскрывает зонтик ($B = 1$), то импликация, очевидно, истинна. Если же при дожде Лена зонтик не раскрыла, импликация ложна (из $A$ не следует $B$!).

| $A$ | $B$ | $A \to B$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Рис. 2.13**

А если дождя нет ($A = 0$)? Тогда о состоянии зонтика Лены мы ничего сказать не можем, потому что он может быть как закрыт, так и открыт. И в обоих случаях импликация будет истинна, потому что *не исключено*, что из $A$ следует $B$ (возможно, что, когда пойдёт дождь, Лена откроет зонтик). Говорят, что «из истины следует истина, а из лжи — что угодно».

Таблица истинности операции импликация показана на рис. 2.13. Как видим, эта функция равна нулю только при одном значении исходных данных: $1 \to 0 = 0$, во всех остальных случаях она равна 1.

Обычно, говоря «если..., то...», мы имеем в виду причинно-следственную связь, когда одно вызывает другое. Например, «Если светит солнце, то лужи высыхают» (именно потому, что светит солнце!). Импликация не говорит о причине и следствии, а показывает *возможность* такой связи. Например, может быть истинной импликация «если сегодня вторник, то Эльбрус покрыт снегом».

Импликация часто используется при решении логических задач. Например, формулировку вида «если $A$, то $B$» можно записать как $A \to B = 1$.

Постройте таблицу истинности логической функции $B \to A$[1]. Сравните её с таблицей истинности функции $A \to B$. Выполняется ли для импликации переместительный закон (если поменять местами $A$ и $B$, то результат не изменяется)?

Постройте таблицу истинности логической функции $\overline{A} + B$. Сравните её с таблицами истинности известных вам функций с двумя переменными. Какую формулу вы сейчас доказали?

---

[1] В этой и следующих трёх задачах сохраняйте порядок столбцов в таблице истинности: в первом столбце записывайте значение $A$, во втором — значение $B$.

Постройте таблицу истинности логической функции $\overline{B} \to \overline{A}$. Сравните её с таблицами истинности известных вам функций с двумя переменными. Какую формулу вы сейчас доказали?

Постройте таблицу истинности логической функции $(A \to B)$ **и** $(B \to A)$. Как бы вы назвали эту функцию?

## Эквиваленция

Таблица истинности, которую вы построили в последнем задании, определяет логическую операцию «эквиваленция» (её также называют «равнозначность» или «логическое равенство»), которая обозначается знаком $\leftrightarrow$. Эта операция соответствует связке «тогда и только тогда». Высказывание $A \leftrightarrow B$ истинно в том и только в том случае, когда $A$ и $B$ равны (рис. 2.14).

| $A$ | $B$ | $A \leftrightarrow B$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Рис. 2.14**

Используя дополнительные источники, выясните, какие знаки кроме $\leftrightarrow$ также используются для обозначения эквиваленции.

Постройте таблицу истинности логической функции $A \cdot B + \overline{A} \cdot \overline{B}$. Сравните её с таблицами истинности известных вам функций с двумя переменными. Какую формулу вы только что доказали?

Постройте таблицу истинности логической функции $(A + \overline{B}) \cdot (\overline{A} + B)$. Сравните её с таблицами истинности известных вам функций с двумя переменными. Какую формулу вы только что доказали?

Постройте таблицу истинности логической функции $A \cdot \overline{B} + \overline{A} \cdot B$. Сравните её с таблицами истинности известных вам функций с двумя переменными. Какую формулу вы только что доказали?

## Исключающее ИЛИ

Функция, которую вы исследовали в последнем задании, называется **исключающее ИЛИ**. Её результат равен 1, если значения входных сигналов не равны (рис. 2.15).

Исключающее ИЛИ обозначается знаком $\oplus$. Смысл этой операции хорошо передаёт поговорка «либо пан, либо пропал»: возможен только один вариант из двух, но не оба одновременно.

| $A$ | $B$ | $A \oplus B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Рис. 2.16**

Сравните таблицы истинности обычной операции «ИЛИ» и «исключающего ИЛИ».

Сравните таблицы истинности логических функций $A \oplus B$ и $A \leftrightarrow B$. Какая формула связывает две эти операции?

Операция исключающее ИЛИ иначе называется **разделительной дизъюнкцией** (это значит «один или другой, но не оба вместе») или **сложением по модулю два**. Второе название связано с тем, что её результат равен остатку от деления арифметической суммы $A + B$ на 2:

$$A \oplus B = (A + B) \bmod 2.$$

Здесь mod обозначает операцию взятия остатка от деления.

Составьте таблицы истинности логических функций $A \oplus 0$, $A \oplus 1$ и $A \oplus A$. Сравните значения каждой функции со столбцом $A$. Как можно упростить эти формулы?

Сравните таблицу истинности логической функции $A \cdot \overline{B} + \overline{A} \cdot B$ (см. задание выше) с таблицей истинности операции исключающее ИЛИ. Какую формулу вы только что доказали?

Постройте таблицу истинности логической функции $(A + B) \cdot (\overline{A} + \overline{B})$. Сравните её с таблицами истинности известных вам функций с двумя переменными. Какую формулу вы только что доказали?

Составьте таблицу истинности логической функции $(A \oplus B) \oplus B$. Сравните столбец значений функции со столбцом $A$. Какую формулу можно записать в результате сравнения?

Из результатов выполнения последнего задания следует важный вывод: если два раза применить к значению $A$ операцию исключающее ИЛИ с одним и тем же значением $B$, то мы восстановим исходное значение $A$. В этом смысле исключающее ИЛИ — *обратимая операция*.

Какие ещё обратимые логические операции вы знаете?

Используя дополнительные источники, выясните, в каких языках программирования есть логическая операция «исключающее ИЛИ» и как она обозначается.

Запишите в тетради формулы, с помощью которых можно представить операции импликацию, эквиваленцию и исключающее ИЛИ через базовые логические операции: НЕ, И и ИЛИ. Используйте результаты выполнения заданий в рабочей тетради.

## Шифрование

Формулу $(A \oplus B) \oplus B = A$ можно использовать для шифрования данных. Пусть $A$ и $B$ — это двоичные коды одинаковой длины. Чтобы зашифровать данные $A$ с использованием ключа $B$, надо применить операцию исключающее ИЛИ отдельно для каждого двоичного разряда $A$ и $B$. Для расшифровки ещё раз применяется исключающее ИЛИ с тем же ключом $B$. Нужно отметить, что такой метод шифрования очень нестойкий: для достаточно длинных текстов его легко взломать.

Например, пусть Алиса хочет секретно передать Борису число 9 в виде четырёхбитной двоичной цепочки 1001. Для шифрования они заранее договорились использовать ключ — двоичную цепочку 0101. Алиса шифрует каждый бит отдельно, выполняя операцию исключающее ИЛИ с соответствующим битом ключа, в результате получается цепочка 1100 (рис. 2.16).

| | | | | |
|---|---|---|---|---|
| Данные | 1 | 0 | 0 | 1 |
| Ключ | 0 | 1 | 0 | 1 |
| Результат | 1 | 1 | 0 | 0 |

**Рис. 2.16**

Борис, получив цепочку 1100, применяет для расшифровки тот же ключ 0101 и получает исходное сообщение — 1001 (рис. 2.17).

| | | | | |
|---|---|---|---|---|
| Зашифрованные данные | 1 | 1 | 0 | 0 |
| Ключ | 0 | 1 | 0 | 1 |
| Результат | 1 | 0 | 0 | 1 |

**Рис. 2.17**

*Работа в парах*. Договоритесь с напарником, какой четырёхбитный ключ вы будете использовать для шифровки. Зашифруйте год, когда произошло какое-нибудь известное историческое событие (четырёхбитный код каждой цифры отдельно с тем же ключом), и передайте напарнику зашифрованное сообщение. Попросите его назвать это событие.

## Выводы

- Импликация $A \to B$ истинна во всех случаях, кроме $A = 1$ и $B = 0$.
- Эквиваленция $A \leftrightarrow B$ истинна тогда и только тогда, когда $A$ и $B$ имеют одинаковые значения.
- Исключающее ИЛИ $A \oplus B$ принимает значение «истина» тогда и только тогда, когда значения $A$ и $B$ различны.

- Операция исключающее ИЛИ обратима: если взять любое значение $A$ и выполнить дважды исключающее ИЛИ с любой постоянной, то получится исходное значение $A$. Эту особенность можно использовать для простого шифрования.
- Логические функции $A \to B$, $A \leftrightarrow B$ и $A \oplus B$ можно выразить через базовые логические операции НЕ, И и ИЛИ:

$A \to B = \overline{A} + B$;
$A \leftrightarrow B = A \cdot \overline{B} + \overline{A} \cdot \overline{B} = (A + \overline{B}) \cdot (\overline{A} + \overline{B})$;
$A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B = (A + B) \cdot (\overline{A} + \overline{B})$.

Нарисуйте в тетради интеллект-карту этого параграфа.

### Вопросы и задания

1. Известно, что высказывание «Если $X$ делится на 6, то $X$ делится на 15» истинно. Будет ли верно это высказывание для чисел 7, 15, 18, 90?
2. Известно, что высказывание «Если Алиса ходила в поход, то Борис тоже ходил» ложно. Кто ходил в поход?
3. В лесу живут два медведя. Известно, что в берлоге всегда находится один медведь, а два медведя в берлогу не помещаются. С помощью какой логической операции можно сформулировать это высказывание?
4. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «Логическая операция штрих Шеффера»
б) «Логическая операция стрелка Пирса»
в) «Частотный анализ текста»

### Практическая работа

Выполните практическую работу № 8 «Шифрование».

## § 11
## Логические выражения

***Ключевые слова:***

- формализация
- логическое выражение
- таблица истинности
- вычислимое выражение
- тождественно истинное выражение
- тождественно ложное выражение
- равносильные выражения
- логическая схема

## Формализация

Обозначив простые высказывания буквами — логическими переменными и используя логические операции, можно записать любое высказывание в виде логического выражения.

**Логическое выражение** — это выражение, результат вычисления которого — логическое значение (истина или ложь).

Например, пусть система сигнализации должна дать аварийный сигнал, если вышли из строя два из трёх двигателей самолёта. Обозначим высказывания:

$A$ = Первый двигатель вышел из строя;
$B$ = Второй двигатель вышел из строя;
$C$ = Третий двигатель вышел из строя;
$X$ = Аварийная ситуация.

Тогда логическое высказывание $X$ можно записать в виде логического выражения (**логической формулы**):

$$X = (A \text{ и } B) \text{ или } (A \text{ и } C) \text{ или } (B \text{ и } C).$$

Это выражение может быть записано с помощью других обозначений:

$$X = (A \cdot B) + (A \cdot C) + (B \cdot C). \qquad (*)$$

Таким образом, мы выполнили *формализацию*.

**Формализация** — это переход от конкретного содержания высказываний к записи с помощью формального языка.

При вычислении логических выражений установлен такой порядок выполнения операций:
1) действия в скобках;
2) отрицание (НЕ);
3) логическое умножение (И), слева направо;
4) логическое сложение (ИЛИ), слева направо.

Можно ли убрать скобки в выражении (*)? Почему?

Уберите лишние скобки в логических выражениях:

*а*) $X = (A + (B \cdot C) \cdot (A + C))$;

*б*) $X = (A + \overline{B}) \cdot (\overline{C} \cdot A) \cdot (A + (\overline{B} + \overline{C})))$.

Вычислите значение логического выражения $X = (A \cdot B + C) \cdot (\overline{A} + \overline{C})$ при:
*а*) $A = 0$, $B = 0$, $C = 1$;
*б*) $A = 0$, $B = 1$, $C = 1$;
*в*) $A = 1$, $B = 1$, $C = 0$.

## Таблицы истинности

Любую логическую функцию можно задать с помощью *таблицы истинности*, которая показывает, чему равно значение логического выражения при всех возможных комбинациях значений исходных переменных. Построим таблицу истинности для выражения

$$X = A \text{ и не } B \text{ или не } A \text{ и } B,$$

которое можно также записать в виде

$$X = A \cdot \overline{B} + \overline{A} \cdot \text{B}.$$

Сколько строк в таблице истинности выражения с двумя переменными?

Будем вычислять выражение по частям: добавим в таблицу истинности дополнительные столбцы $A \cdot \overline{B}$ и $\overline{A} \cdot$ B, а потом выполним операцию ИЛИ с этими значениями (рис. 2.18).

| $A$ | $B$ | $A \cdot \overline{B}$ | $\overline{A} \cdot B$ | $X$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 |

**Рис. 2.18**

Из этой таблицы истинности видно, что при некоторых значениях переменных значение $X$ истинно, а при некоторых — ложно. Такие выражения называют **вычислимыми**.

Высказывание «Вася — школьник или он не учится в школе» всегда истинно (для любого Васи). Выражение, истинное при любых значениях переменных, называется **тождественно истинным** или **тавтологией**.

Высказывание «Сегодня безветрие, и дует сильный ветер» никогда не может быть истинным. Соответствующее логическое

выражение всегда ложно, оно называется **тождественно ложным** или **противоречием.**

Выполните формализацию высказываний, о которых шла речь в последних двух абзацах. Запишите формулы для упрощения каждого из них.

Постройте самостоятельно таблицу истинности логического выражения $X = (A + B) \cdot (\overline{A} + \overline{B})$. Сравните её с таблицей на рис. 2.18. Истинно ли высказывание «Разные логические выражения могут определять одну и ту же логическую функцию»?

**Если два выражения принимают одинаковые значения при всех значениях переменных, они называются равносильными или тождественно равными. Равносильные выражения определяют одну и ту же логическую функцию, т. е. при одинаковых исходных данных приводят к одинаковым результатам.**

Сколько строк в таблице истинности выражения с тремя переменными?

Постройте таблицу истинности логического выражения $X = A \cdot B + A \cdot C + B \cdot C$.

---

Предположим, что нам известна только часть таблицы истинности для функции трёх переменных (рис. 2.19).

| *A* | *B* | *C* | *F* |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 |

**Рис. 2.19**

Как вы думаете, можно ли по такой части таблицы истинности определить логическую функцию? Почему?

Всего в таблице истинности функции от трёх переменных $2^3 = 8$ строк, для каждой из них нужно знать, чему равно значение функции. В нашем примере пять значений функции неизвестны, причём каждое из них может быть равно 0 или 1, т. е. у нас есть 5 свободных бит.

Сколько различных значений можно закодировать с помощью 5 бит? Как вы нашли это значение?

Итак, таблице на рис. 2.19 могут соответствовать 32 различные логические функции. Проверим, подходят ли следующие варианты:

а) $A + B + \overline{C}$; б) $\overline{A} \cdot \overline{C} + \overline{B}$; в) $A + \overline{B} \cdot C$;
г) $A \cdot B \cdot \overline{C}$; д) $\overline{A} \cdot \overline{B} + \overline{C}$.

Прежде всего, заметим, что в столбце значений функции два нуля и одна единица. Следовательно, вариант а) не подходит, потому что цепочка операций ИЛИ со всеми переменными (или их инверсиями, обратными значениями) даст только один ноль — в случае, когда все слагаемые равны нулю.

Проверяем вариант б), подставляя значения переменных $A$, $B$ и $C$ сначала из первой строки таблицы, потом из второй и третьей. В первой строке получаем $\overline{A} \cdot C + B = 1 \cdot 1 + 0 = 1$, этот результат совпадает со значением функции в этой строке. Для второй строки $\overline{A} \cdot C + B = 1 \cdot 1 + 1 = 1$, значит, эта функция тоже не подходит.

Проверьте самостоятельно, подходят ли остальные варианты.

Скажите без вычислений, сколько нулей и сколько единиц должно быть в последнем столбце таблицы истинности функций с тремя переменными:

а) $A + \overline{B} + \overline{C}$; б) $\overline{A} \cdot C \cdot B$; в) $A \cdot \overline{B} \cdot \overline{C}$;
г) $\overline{A} + B + C$; д) $\overline{A} + \overline{B} + \overline{C}$.

Теперь предположим, что нам известна часть таблицы какой-то логической функции, причём с пропусками (рис. 2.20).

| *A* | *B* | *C* | *F* |
|---|---|---|---|
| 0 | | 1 | 0 |
| 1 | 0 | | 1 |
| | | 1 | 1 |

**Рис. 2.20**

Выясним, какие из следующих функций могут соответствовать этой таблице:

а) $\overline{A} + B + \overline{C}$; б) $\overline{A} \cdot C \cdot B$; в) $A + \overline{B} + \overline{C}$;
г) $A \cdot \overline{B} \cdot C$; д) $A + B + \overline{C}$.

Во-первых, обратим внимание, что в столбце значений функции две единицы, поэтому сразу делаем вывод, что это не могут быть цепочки двух операций И (ответы б) и г) неверные). Остались три цепочки из операций ИЛИ, причём для верхней строки (при $A = 0$ и $C = 1$) мы должны получить 0 при каком-то выборе неизвестного значения $B$. Подставляем $A = 0$ и $C = 1$ в формулы-кандидаты:

а) $1 + B + 0$; в) $0 + \overline{B} + 0$; д) $0 + B + 0$.

Видим, что в случае а) сумма не может быть равна нулю, это неверный ответ. А варианты в) и д) подходят: в первом из них нужно в первой строке поставить в пустую ячейку 1, а во втором — 0.

Кратко решение можно записать так:

1) поскольку в столбце значений функции один ноль, это не может быть цепочка операций И; остаются только цепочки операций ИЛИ;
2) для того чтобы получить нулевое значение функции в первой строке таблицы, нужно, чтобы переменная $A$ (равная 0 в этой строке) входила в логическую сумму без инверсии (к ней не должна применяться операция НЕ) а переменная $C$ (равная 1) — с инверсией;
3) этим условиям удовлетворяют функции в) и д).

Известна часть таблицы какой-то логической функции с пропусками (рис. 2.21).

| $A$ | $B$ | $C$ | $F$ |
|---|---|---|---|
| 0 | | 1 | 0 |
| 1 | 0 | | 1 |
| | | 1 | 0 |

**Рис. 2.21**

Выясните, какие из следующих функций могут соответствовать этой таблице:

а) $A \cdot \overline{B} + \overline{C}$; б) $\overline{A} \cdot C \cdot B$; в) $A + \overline{B} + \overline{C}$;
г) $A \cdot \overline{B} \cdot C$; д) $A + B + \overline{C}$.

## Составление условий

Логические выражения часто используются при решении математических задач с помощью компьютеров.

Построим условия (логические выражения), соответствующие заштрихованным областям на числовой оси (рис. 2.22).

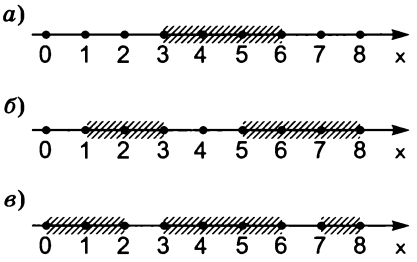



**Рис. 2.22**

На рисунке 2.22, *а* выделен отрезок [3; 6]. Для того чтобы определить такую область, нужно ограничить значение $x$ с двух сторон: оно должно быть больше или равно трём и меньше или равно шести, причем эти два условия должны выполняться *одновременно*, т. е. их нужно связать с помощью операции И:

$$(3 \le x) \textbf{ и } (x \le 6).$$

Область на рис. 2.22, *б* — это объединение двух отрезков. Мы можем отдельно записать условия для каждого отрезка и связать их с помощью операции ИЛИ:

$$(1 \le x) \textbf{ и } (x \le 3) \quad \textbf{или} \quad (5 \le x) \textbf{ и } (x \le 8).$$

Запишите условие, которое определяет область на рис. 2.22, *в*.

Теперь рассмотрим задачу с областью на плоскости. Запишем условие, соответствующее заштрихованной части (рис. 2.23).

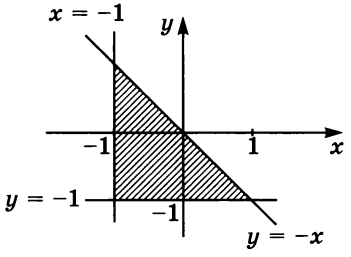



**Рис. 2.23**

Требуется составить логическое выражение, зависящее от переменных $x$ и $y$, которое будет равно 1 (истинно) внутри заштрихованной области и равно 0 вне её.

Выделенная область находится справа от вертикальной линии $x = -1$, поэтому должно выполняться условие $x \geq -1$, которое определяет полуплоскость (рис. 2.24).

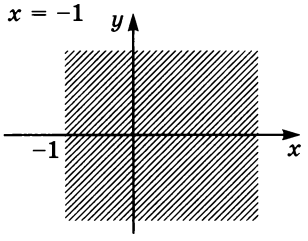



**Рис. 2.24**

Кроме того, все точки области находятся выше горизонтальной линии $y = -1$ и ниже наклонной прямой $y = -x$, что даёт ещё два условия: $y \geq -1$ и $y \leq -x$. Все три простых условия должны выполняться одновременно, поэтому их нужно связать с помощью двух операций И:

$$(x \geq -1) \textbf{ и } (y \geq -1) \textbf{ и } (y \leq -\text{x}).$$

Запишите условие, которое определяет области на рисунках (рис. 2.25).

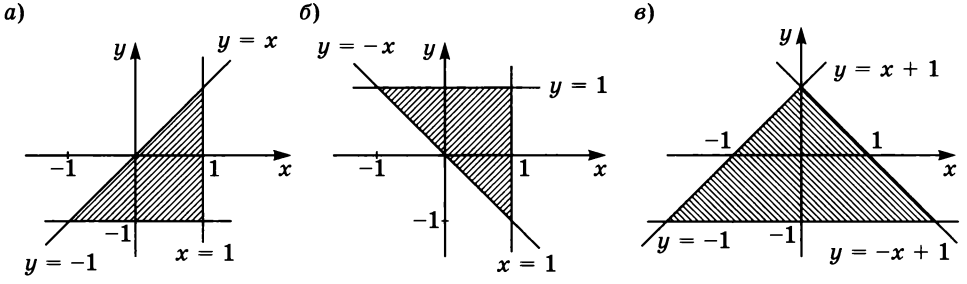



**Рис. 2.25**

Построим логическое выражение для области на рис. 2.26.

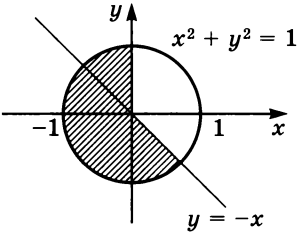



**Рис. 2.26**

Можно разделить заштрихованную область на две части (рис. 2.27).

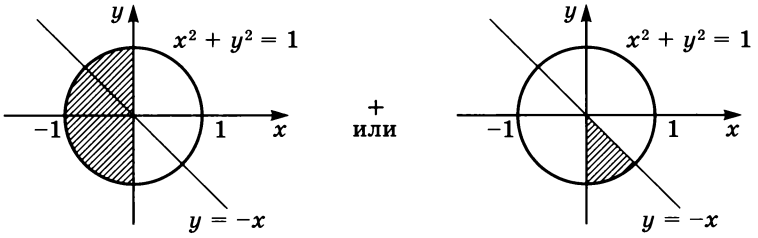



**Рис. 2.27**

Обе области находятся внутри круга радиуса 1 с центром в начале координат, т. е. в области $x^2 + y^2 \le 1$. Первая область описывается выражением

$$(x^2 + y^2 \le 1) \textbf{ и } (x \le 0),$$

а вторая — выражением

$$(x^2 + y^2 \le 1) \textbf{ и } (x \ge 0) \textbf{ и } (y \le -x).$$

Поскольку нам нужно «сложить» две области, эти выражения нужно объединить с помощью операции логического сложения (ИЛИ):

$$(x^2 + y^2 \le 1) \textbf{ и } (x \le 0) \textbf{ или } (x^2 + y^2 \le 1) \textbf{ и } (x \ge 0) \textbf{ и } (y \le -x).$$

Общее условие $x^2 + y^2 \le 1$ можно вынести за скобки:

$$(x^2 + y^2 \le 1) \textbf{ и } ((x \le 0) \quad \textbf{или} \quad (x \ge 0) \textbf{ и } (y \le -x)).$$

Попробуйте ещё упростить полученное условие.

Запишите условие, которое определяет области на рисунках (рис. 2.28).

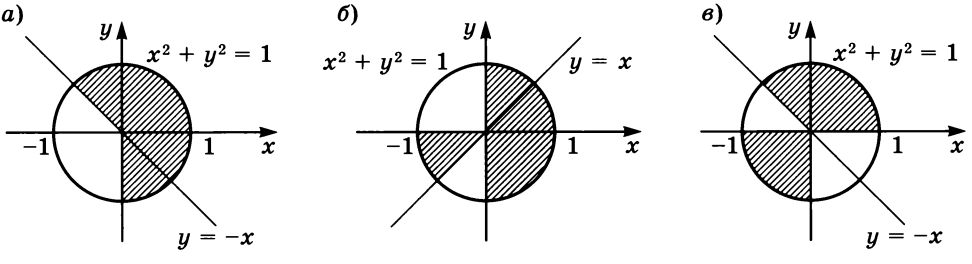



**Рис. 2.28**

## Логические схемы

Вернёмся снова к примеру с системой аварийной сигнализации самолёта. Предположим, что на каждом двигателе установлен логический датчик, который выдаёт условный сигнал 1 (например, высокий уровень напряжения), если двигатель неисправен, и условный сигнал 0, если двигатель исправен. Требуется построить **логическую схему** — схему логического устройства, — которая при аварии выдаёт условный сигнал 1, а в режиме нормальной работы — сигнал 0.

Сигналы от трёх двигателей назовём $A$, $B$ и $C$. В начале параграфа мы уже составили логическое выражение для запуска аварийной сигнализации:

$$X = A \cdot B + A \cdot C + B \cdot C.$$

Здесь три логических умножения и два логических сложения. Сначала выполняются все операции умножения (слева направо), а затем — все операции сложения (тоже слева направо). Расставим номера операций:

$$\overset{1}{X} = \overset{4}{A \cdot B} + \overset{2}{A \cdot C} + \overset{5}{B \cdot C}.$$

Последней выполняется вторая операция сложения. Поэтому последний элемент в схеме — это элемент логического сложения ИЛИ (рис. 2.29).

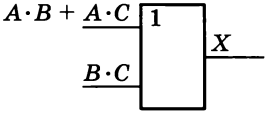



**Рис. 2.29**

На первый вход этого элемента ИЛИ подаётся сигнал $A \cdot B + A \cdot C$, в этом выражении последняя операция — логическое сложение, добавляем ещё один элемент ИЛИ (рис. 2.30).

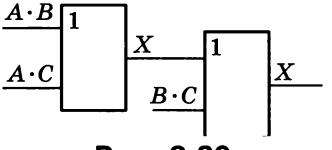



**Рис. 2.30**

Три операции логического умножения добавляют в схему три элемента И (рис. 2.31).

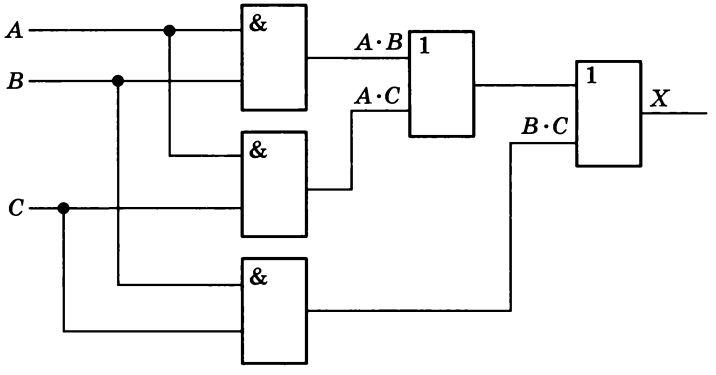



**Рис. 2.31**

Запишите в тетради логическое выражение по логической схеме (рис. 2.32).

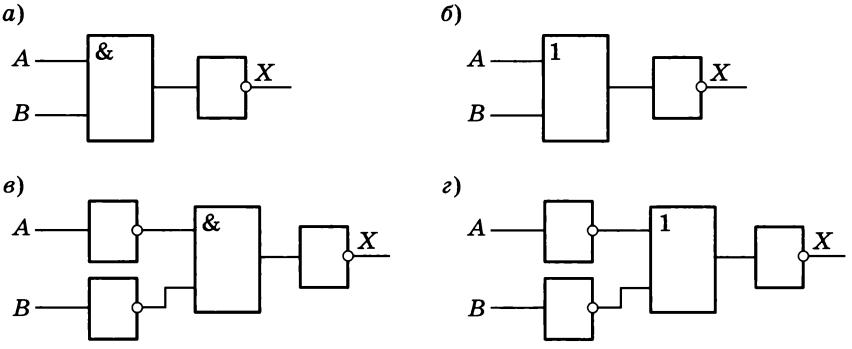



**Рис. 2.32**

Постройте логическую схему, соответствующую выражению:
а) $X = \overline{A} \cdot C + B \cdot \overline{C}$; б) $Y = A \cdot C + \overline{B} \cdot \overline{C}$;
в) $Z = \overline{A} \cdot \overline{B} + A \cdot B \cdot \overline{C}$.

*Работа в парах*. Один из вас пусть нарисует логическую схему для выражения $X = \overline{A + B \cdot C}$ (черта сверху обозначает, что операция отрицания применяется ко всему выражению), а второй — для выражения $Y = (A + B) \cdot (A + C)$. Постройте таблицы истинности для своих выражений и сравните их. Какую формулу вы сейчас вместе доказали?

*Работа в парах*. Запишите логическое выражение, включающее 5–6 операций, и предложите соседу нарисовать логическую схему. Проверьте и обсудите с ним его решение.

Путешествуя по Зазеркалью, Алиса увидела дверь с тремя кнопками, которые были отмечены буквами $A$, $B$ и $C$. На двери висела схема (рис. 2.33).

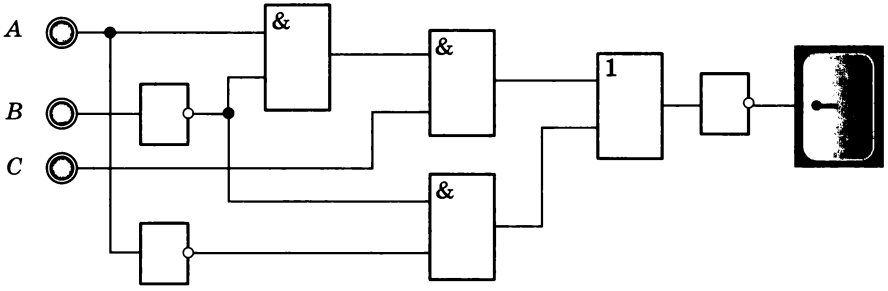



**Рис. 2.33**

На какую (одну!) из трёх кнопок нужно нажать Алисе, чтобы открыть дверь?

## Выводы

- Формализация — это переход от конкретного содержания высказываний к записи с помощью формального языка.
- В логических выражениях операции выполняются в следующем порядке:
  1) действия в скобках;
  2) отрицание (НЕ);
  3) логическое умножение (И), слева направо;
  4) логическое сложение (ИЛИ), слева направо.

  Для изменения порядка действий используются скобки.
- Таблица истинности логического выражения показывает, чему равно значение выражения при всех возможных комбинациях значений исходных переменных.
- Логические выражения, истинность которых зависит от значений исходных переменных, называют вычислимыми.
- Логическое выражение, которое всегда истинно, называется тождественно истинным или тавтологией. Выражение, которое всегда ложно, называют тождественно ложным или противоречием. Пример тождественно истинного выражения: $A + \overline{A}$, пример тождественно ложного: $A \cdot \overline{A}$.
- Два выражения, принимающие одинаковые значения при всех значениях переменных, называются равносильными или тождественно равными. Равносильные выражения определяют одну и ту же логическую функцию.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Что можно сделать для того, чтобы изменить порядок выполнения действий в логических выражениях?
2. Поясните разницу между терминами «логическое выражение» и «логическая функция».
3. Как можно доказать (или опровергнуть) логическую формулу?
4. Можно ли сказать, что таблица истинности однозначно определяет:
   а) логическое выражение;
   б) логическую функцию?
5. Что такое вычислимое логическое выражение?
6. Что такое равносильные выражения?
7. Выполните по указанию учителя задания в рабочей тетради.

# § 12
# Множества и логика

***Ключевые слова:***

- множество
- дополнение
- пересечение
- объединение
- диаграмма Эйлера–Венна
- поисковый запрос

## Множества

**Множество** — некоторый набор элементов, каждый из которых отличается от остальных. Множество может состоять из конечного числа элементов (например, множество букв русского алфавита), бесконечного числа элементов (например, множество натуральных чисел) или вообще быть пустым (например, множество слонов, живущих на Северном полюсе). Пустое множество обозначается символом ∅. Множества, с которыми работает компьютер, не могут быть бесконечными, потому что его память конечна.

Чтобы определить множество, мы можем перечислить все его элементы. Например, множество, состоящее из Васи, Пети и Коли, можно записать так: {Вася, Петя, Коля}.

Запишите в виде перечисления элементов:
а) множество натуральных чисел на отрезке [–5; 5];
б) множество чётных однозначных чисел;
в) множество целых чисел, делящихся на 4, на отрезке [0; 22];
г) множество простых чисел на отрезке [5; 20].

Можно задать множество иначе: определить условие (*логическое выражение*), которое должно быть истинным для всех элементов множества и ложным для всех элементов, не входящих во множество. Например, можно ввести множество драконов с пятью зелёными хвостами или множество чисел, делящихся на 11.

Запишите (словами или в символьном виде) условие, которое определяет множество:

а) {1, 3, 5, 7, 9};
б) {5, 6, 7};
в) {а, е, ё, и, о, у, ы, э, ю, я};
г) {17, 34, 51, 68, 85};
д) {00, 01, 10, 11};
е) {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C};
ж) отрезок [0; 1].

Хотя множество — это математическое понятие, с множествами мы имеем дело каждый раз, когда обращаемся к поисковой системе в Интернете: ведь нас интересует множество страниц, на которых есть нужная нам информация. Задать такое множество перечислением элементов невозможно: во-первых, мы не знаем адресов этих страниц; во-вторых, их очень много. Поэтому для того, чтобы задать нужное нам множество, требуется написать **поисковый запрос** — логическое выражение, которое его определяет.

## Диаграммы Эйлера–Венна

Множества удобно изображать графически, в виде диаграмм. Их называют **диаграммами Эйлера–Венна** в честь авторов этой идеи — математика Леонарда Эйлера и логика Джона Венна. На такой диаграмме каждому множеству соответствует какая-то область (круг, прямоугольник и др.) — рис. 2.34. Все элементы внутри этой области принадлежат множеству, все элементы вне области — не принадлежат.

**Рис. 2.34**

Вы уже знаете, что множество можно задать условием (логическим выражением), которое выполняется для всех элементов множества и не выполняется для всех элементов, не входящих в него. Дальше для сокращения записи мы будем вместо слов «множество, для которого выполняется условие $A$» писать просто «множество $A$». Тогда множество «НЕ $A$» на диаграмме — это все точки за границами круга (рис. 2.35).

**Рис. 2.35**

Такое множество называется **дополнением** множества $A$ до **универсального множества** $U$, включающего все элементы некоторого класса. Например, если мы рассматриваем только целые числа и $A$ — это множество чётных целых чисел, то $\overline{A}$ — множество нечётных целых чисел.

Можно считать, что дополнение $\overline{A}$ — это «разность» между универсальным множеством $U$ и множеством $A$, т. е. все элементы из $U$, которые не входят в $A$.

Для каждого из следующих множеств выберите универсальное множество и запишите дополнение $\overline{A}$:

а) $A$ = {1, 3, 5, 7, 9};
б) $A$ = {а, е, ё, и, о, у, ы, э, ю, я};
в) $A$ = {17, 34, 51, 68, 85};
г) $A$ = {00, 10};
д) $A$ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C};
е) $A$ = отрезок [0; 1].

$A \cdot B$

На диаграмме можно изображать несколько множеств, каждому из них соответствует своя область (круг). Круги на диаграмме могут пересекаться. Элементы, расположенные в общей части кругов $A$ и $B$, — это **пересечение** множеств $A$ и $B$. Для этих элементов выполняется как условие $A$, так и условие $B$, т. е. выполняется условие $A$ **и** $B$ ($A \cdot B$) — рис. 2.36.

**Рис. 2.36**

Если круги не пересекаются (множества не содержат общих элементов), их пересечение — это пустое множество $\varnothing$.

Для пары множеств определите пересечение $A \cdot B$.

а) $A$ = {1, 3, 5, 7, 9}, $B$ = {1, 5, 6, 9, 12};
б) $A$ = {а, б, в, г, д, е, ё, ж}, $B$ = {а, е, ё, и, о, у, ы, э, ю, я};
в) $A$ = {17, 34, 51, 68, 85}, $B$ = {17, 34, 51, 68, 85};
г) $A$ = {00, 10}, $B$ = {01, 11};
д) $A$ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C}, $B$ = {A, B, C, D, E, F, G, H};
е) $A$ = [5; 15], $B$ = [10; 20];
ж) $A$ = [5; 15], $B$ = [0; 20];
з) $A$ = [5; 15], $B$ = [10; 12];
и) $A$ = [5; 15], $B$ = [20; 30].

$A + B$

Элементы, входящие хотя бы в одно из множеств: в $A$ или в $B$, образуют новое множество, которое называется **объединением** множеств $A$ и $B$. Для всех элементов этого множества выполняется условие $A$ **или** $B$ ($A + B$) — рис. 2.37.

**Рис. 2.37**

Для пары множеств определите объединение $A + B$:

а) $A$ = {1, 3, 5, 7, 9}, $B$ = {1, 5, 6, 9, 12};
б) $A$ = {а, б, в, г, д, е, ё, ж }, $B$ = {а, е, ё, и, о, у, ы, э, ю, я};
в) $A$ = {17, 34, 51, 68, 85}, $B$ = {17, 34, 51, 68, 85};
г) $A$ = {00, 10}, $B$ = {01, 11};
д) $A$ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C}, $B$ = {A, B, C, D, E, F, G, H};
е) $A$ = [5; 15], $B$ = [10; 20];
ж) $A$ = [5; 15], $B$ = [0; 20];
з) $A$ = [5; 15], $B$ = [10; 12];
и) $A$ = [5; 15], $B$ = [20; 30].

Подобные диаграммы можно нарисовать для любого логического выражения, ведь каждое из них определяет некоторое множество. Например, возьмём выражение $\overline{A} + B$. Оно равно 0 только при $A = 1$ и $B = 0$, поэтому на диаграмме незакрашенной

останется только область, которая входит в круг $A$ и не входит в круг $B$ (рис. 2.38).

$\overline{A} + B$

В тетради постройте диаграммы для логических выражений:

а) $A + \overline{B}$;

б) $A \cdot \overline{B} + \overline{A} \cdot B$;

в) $A \cdot B + \overline{A} \cdot \overline{B}$.

**Рис. 2.38**

Диаграмма для трёх переменных содержит три круга, каждый из которых (в общем случае) пересекается с двумя другими (рис. 2.39).

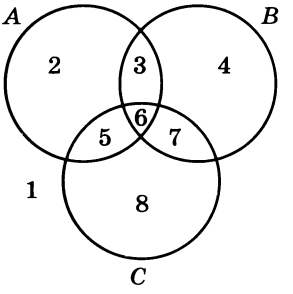



**Рис. 2.39**

Для удобства на рис. 2.39 области пронумерованы. Запишем, для примера, логическое выражение для области 3. Эта область находится внутри кругов $A$ и $B$ (следовательно, выражения $A$ и $B$ истинны), но вне круга $C$, поэтому выражение $C$ ложно. Получается условие $A$ **и** $B$ **и** (**не** $C$), или, в других обозначениях, $A \cdot B \cdot \overline{C}$.

Запишите в тетради логические выражения для остальных областей на рис. 2.39.

Для того чтобы найти выражение для объединения двух или нескольких областей, надо сложить (используя логическое сложение — операцию ИЛИ) выражения для всех составляющих. Например, выражение для объединения областей 3 и 4 на рис. 2.39 имеет вид:

$$3 + 4\text{: } A \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C}.$$

Вместе с тем точки в этих областях отличаются от других тем, что они входят в область $B$ и не входят в область $C$. Поэтому справедлива более простая формула:

$$3 + 4\text{: } B \cdot \overline{C}.$$

Это означает, что логические выражения в некоторых случаях можно упростить.

## Количество элементов во множестве

Предположим, что множество $A$ содержит 10 элементов, множество $B$ — 15 элементов, а их пересечение (множество $A \cdot B$) —

два элемента. Как определить, сколько элементов содержится во множестве $A + B$?

Попробуем рассмотреть задачу в общем виде и вывести формулу для её решения. Обозначим через $N_X$ число элементов в области $X$. Далее операцию И будет обозначать символом &, а операцию ИЛИ — символом | (именно эти символы используются в поисковых запросах в Интернете).

Построим диаграмму с двумя областями — $A$ и $B$. Эти области могут быть разделены (рис. 2.40, *а*) или пересекаться (рис. 2.40, *б*).

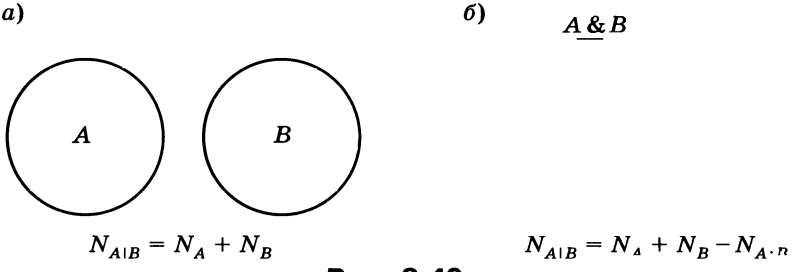



**Рис. 2.40**

В первом случае (рис. 2.40, *а*), когда области не пересекаются, получаем очевидную формулу:

$$N_{A\,|\,B} = N_A + N_B.$$

Во втором случае (рис. 2.40, *б*) в сумму $N_A+N_B$ общие элементы (элементы множества $N_{A\&B}$) входят дважды. Поэтому, чтобы получить количество элементов в объединении множеств, нужно из этой суммы вычесть число общих элементов:

$$N_{A|B} = N_A + N_B - N_{A\&B}. \qquad (*)$$

Эта формула, которую называют формулой включений и исключений, справедлива и для рис. 2.40, *а*, где $N_{A\&B} = 0$.

Используя формулу (*), постройте выражения для вычисления $N_A$ и $N_{A\&B}$.

Рыбаки в посёлке ловят только лещей и судаков. 25 рыбаков ловят лещей, 12 рыбаков — судаков, причём 5 рыбаков ловят и лещей, и судаков. Сколько всего рыбаков в посёлке? Выполните формализацию задачи и решите её.

У дяди Вани живёт 30 животных: овцы и кролики. Все кролики белые, а у овец разный цвет шерсти. Известно, что у дяди Вани живёт 18 овец и 25 животных с белой шерстью. Сколько белых овец у дяди Вани? Выполните формализацию задачи и решите её.

В физико-математическом классе 27 учеников. Среди них нет таких, которые не программируют и не ходят в турпоходы. Известно, что 20 человек ходят в турпоходы, среди них 5 программистов. Сколько в классе программистов? Выполните формализацию задачи и решите её.

## Сложные запросы в поисковых системах

Для решения задач, в которых используются множества, например множества страниц, полученных от поисковой системы в ответ на какой-то запрос, удобно применять диаграммы Эйлера–Венна.

**Задача 1.** Известно количество страниц, которые находит поисковый сервер по следующим запросам (здесь символ «&» обозначает операцию И, а «|» — операцию ИЛИ):

| | |
|---|---|
| собаки \| кошки | 770 |
| кошки | 550 |
| собаки & кошки | 100 |

Сколько страниц будет найдено по запросу собаки?

Введём два множества: $A$ — множество страниц, где есть слово «собаки», $B$ — множество страниц со словом «кошки». По формуле, которая получена в предыдущем пункте, получаем:

$$N_A = N_{A|B} - N_B + N_{A\&B} = 770 - 550 + 100 = 320.$$

Известно количество страниц, которые находит поисковый сервер по следующим запросам:

| | |
|---|---|
| незабудка | 220 |
| лилия & незабудка | 100 |
| лилия \| незабудка | 450 |

Сколько страниц найдёт этот сервер по запросу лилия?

Известно количество страниц, которые находит поисковый сервер по следующим запросам:

| | |
|---|---|
| енот | 200 |
| кашалот | 300 |
| кашалот \| енот | 450 |

Сколько страниц найдет этот сервер по запросу

кашалот & енот?

Известно количество страниц, которые находит поисковый сервер по следующим запросам:

| | |
|---|---|
| Италия | 320 |
| Франция | 450 |
| Франция & Италия | 80 |

Сколько страниц найдёт этот сервер по запросу

Франция | Италия?

Рассмотрим теперь более сложную задачу с тремя областями.

**Задача 2.** Известно количество страниц, которые находит поисковый сервер по следующим запросам:

| | |
|---|---|
| собаки & лемуры | 320 |
| кошки & лемуры | 280 |
| (кошки \| собаки) & лемуры | 430 |

Сколько страниц будет найдено по запросу

кошки & собаки & лемуры?

Заметим, что во всех запросах есть часть & лемуры. Это означает, что область поиска во всех случаях ограничена страницами, на которых встречается слово «лемуры».

Обозначим буквами *С*, *К* и *Л* области (группы страниц), содержащие ключевые слова «собаки», «кошки» и «лемуры» соответственно. Нас интересует только область, выделенная фоном на рис. 2.41, *а*.

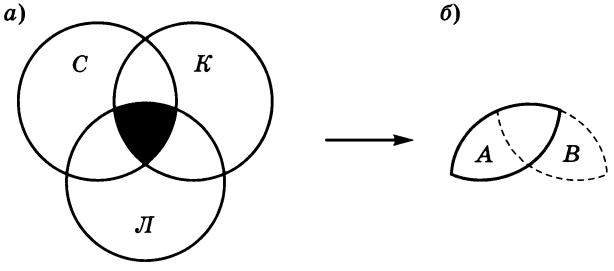



**Рис. 2.41**

Эта область образована в результате пересечения двух областей (рис. 2.41, *б*):

*А* = собаки & лемуры
*В* = кошки & лемуры

Поэтому задачу можно свести к задаче с двумя областями.

Известно количество страниц, которые находит поисковый сервер по следующим запросам:

| | |
|---|---|
| A | 320 |
| B | 280 |
| A \| B | 430 |

Сколько страниц будет выдано по запросу A & B?

Используя формулу включений и исключений, полученную в предыдущем пункте, находим:

$$N_{A\&B} = N_A + N_B - N_{A|B} = 320 + 280 - 430 = 170.$$

Известно количество страниц, которые находит поисковый сервер по следующим запросам:

| | |
|---|---|
| берёза & сирень | 220 |
| берёза & сирень & арбуз | 30 |
| сирень & (берёза \| арбуз) | 340 |

Сколько страниц найдёт этот сервер по запросу

арбуз & сирень?

Известно количество страниц, которые находит поисковый сервер по следующим запросам:

| | |
|---|---|
| яхта & диван | 270 |
| диван & пирог | 350 |
| яхта & диван & пирог | 80 |

Сколько страниц найдёт этот сервер по запросу

(пирог | яхта) & диван?

Задачу с тремя областями не всегда удаётся свести к более простой задаче с двумя областями. Серьёзным упрощением может стать то, что какие-то два множества не имеют общих элементов.

Если два множества не имеют общих элементов, что можно сказать об их изображении на диаграмме Эйлера–Венна?

**Задача 3.** Известно количество страниц, которые находит поисковый сервер по следующим запросам:

| | |
|---|---|
| собаки | 200 |
| кошки | 250 |
| лемуры | 450 |
| кошки \| собаки | 450 |
| кошки & лемуры | 40 |
| собаки & лемуры | 50 |

**Сколько страниц найдёт этот сервер по запросу**

```
(кошки | собаки) & лемуры?
```

**Здесь часть** `& лемуры` **встречается не во всех запросах, поэтому свести задачу к задаче с двумя областями не удаётся. Используя те же обозначения, что и в задаче 2, построим диаграмму с тремя переменными и выделим интересующую область, которая соответствует запросу**

```
(кошки | собаки) & лемуры.
```

**На рисунке 2.42 эта область выделена фоном.**

**Рис. 2.42**

**В общем виде задача с тремя областями очень сложна. Попробуем найти какое-нибудь упрощающее условие. Например, выделим три условия:**

| | |
|---|---|
| `собаки` | **200** |
| `кошки` | **250** |
| `кошки \| собаки` | **450** |

**Это означает, что область `кошки | собаки` равна сумме областей `кошки` и `собаки`, т. е. эти области *не пересекаются*! Таким образом, в нашем случае диаграмма выглядит так (рис. 2.43).**

**Рис. 2.43**

**Размеры областей 1 (собаки & лемуры) и 2 (кошки & лемуры) нам известны, они составляют соответственно 40 и 50 страниц, поэтому по запросу**

```
(кошки | собаки) & лемуры
```

**поисковый сервер найдёт 40 + 50 = 90 страниц.**

Известно количество страниц, которые находит поисковый сервер по следующим запросам:

| | |
|---|---|
| солнце | 230 |
| крабы | 220 |
| лето | 100 |
| крабы \| солнце | 450 |
| крабы & лето | 60 |
| солнце & лето | 20 |

Сколько страниц найдёт этот сервер по запросу

```
крабы | солнце | лето?
```

## Выводы

- Множество — это набор неповторяющихся элементов.
- Множество может состоять из конечного числа элементов, бесконечного числа элементов или быть пустым. Множества, с которыми работает компьютер, не могут быть бесконечными, потому что его память конечна.
- Чтобы определить множество, можно перечислить все его элементы или задать условие, которое определяет элементы множества. Для всех элементов множества это условие должно быть истинным, для элементов, не входящих во множество, — ложным.
- Дополнение множества $A$ до универсального множества $U$, включающего все элементы некоторого класса, — это все элементы из $U$, которые не входят в $A$.
- Пересечение двух множеств — это множество, составленное из элементов, входящих в оба исходных множества.
- Объединение двух множеств — это множество, составленное из элементов, которые входят хотя бы в одно из этих множеств.
- Для наглядного изображения множеств используют диаграммы Эйлера–Венна, на которых каждое множество обозначается кругом или другой фигурой.
- На диаграмме Эйлера–Венна дополнение множества $A$ — это все точки за пределами области $A$; пересечение множеств $A$ и $B$ — это общая часть областей $A$ и $B$, а объединение множеств $A$ и $B$ — это все точки, входящие в область $A$ или в область $B$.

- Количество элементов в объединении двух множеств вычисляется по формуле включений и исключений:

$$N_{A|B} = N_A + N_B - N_{A\&B},$$

где $N_A$ и $N_B$ — число элементов соответственно в множествах $A$ и $B$, а $N_{A\&B}$ — число их общих элементов.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

Выполните по указанию учителя задания в рабочей тетради.

## ЭОР к главе 2 из Единой коллекции цифровых образовательных ресурсов (school-collection.edu.ru)

Элементарные логические операции
С чего начинается логика
Беседа 5а. Логические связки
Беседа 5б. Логические формулы
Логические задачи и алгебра высказываний
Схема множеств-1
Схема множеств-2
Схема множеств-3

# Глава 3
# МОДЕЛИРОВАНИЕ

## § 13
## Модели и моделирование

***Ключевые слова***:

- модель
- моделирование
- анализ
- синтез
- материальные модели
- информационные модели
- имитационные модели
- игровые модели
- адекватность

### Что такое модель?

При слове «модель» у многих, наверное, появляется мысль о моделях самолётов, кораблей, танков и другой техники, которые стоят на полках магазинов. Однако слово «модель» имеет более широкое значение. Например, игрушки, в которые играют дети всех возрастов, — это модели реальных объектов, с которыми они сталкиваются в жизни (или столкнутся в будущем).

Говоря о модели, мы всегда указываем на какой-то другой объект (процесс, явление), например: «Глобус — это модель Земли». Здесь другой объект — это Земля, он называется **оригиналом**. Объект становится моделью только тогда, когда есть оригинал, модели без оригинала не существует.

Зачем вообще нужны модели? Они появляются тогда, когда мы хотим решить какую-то **задачу**, связанную с оригиналом, а изучать оригинал трудно или даже невозможно:

- оригинал **не существует**; например, учебники истории — это модели общества, которого уже нет; возможные последствия ядерной войны учёные изучали на моделях, потому что ставить реальный эксперимент было бы безумием;
- исследование оригинала **дорого или опасно** для жизни, например, при управлении ядерным реактором, испытании скафандра для космонавтов, создании нового самолёта или корабля;
- оригинал **сложно или невозможно исследовать** непосредственно, например Солнечную систему, молекулы и атомы, очень быстрые процессы в двигателях внутреннего сгорания, очень медленные движения материков;

- нас интересуют только **некоторые свойства** оригинала; например, чтобы испытать новую краску для самолёта, не нужно строить самолёт.

Итак, модель всегда связана не только с оригиналом, но и с конкретной задачей, которую мы хотим решить с помощью модели.

Для любого оригинала можно построить множество разных моделей. Например, моделью человека может служить его фотография, паспорт, генетический код, манекен, рентгеновский снимок, биография. Зачем столько? Дело в том, что каждая из этих моделей отражает только те *свойства, которые важны при решении конкретной задачи*. Такие свойства в теории моделирования называют **существенными**.

Вместе с тем одна и та же модель может описывать множество самых разных оригиналов. Например, в различных задачах атом, муха, человек, автомобиль, высотное здание, даже планета Земля могут быть представлены как материальные точки (если размеры соседних объектов и расстояния между ними значительно больше).

---

**Модель** — это объект, который обладает существенными свойствами другого объекта или процесса (*оригинала*) и используется вместо него.

---

Назовите свойства самолёта, существенные с точки зрения:
а) конструктора;
б) дизайнера;
в) экономиста;
г) лётчика;
д) бортпроводника;
е) пассажира.

Практически всё, что мы делаем с помощью компьютеров, — это моделирование. Например, база данных библиотеки — это модель реального хранилища книг, компьютерный чертёж — это модель детали и т.д.

---

**Моделирование** — это создание и исследование моделей для изучения оригиналов.

---

С помощью моделирования можно решать **задачи четырёх типов:**

- *изучение* оригинала (в научных и учебных целях);
- *анализ* («что будет, если ...») — прогнозирование влияния различных воздействий на оригинал;
- *синтез* («как сделать, чтобы ...») — управление оригиналом;

- *оптимизация* («как сделать лучше всего ...») — выбор наилучшего решения в данных условиях.

Назовите задачи, которые решаются в каждом случае.

а) Даниил считает, как купить новый планшетный компьютер по минимальной цене.
б) Кирилл выясняет, будет ли плавать в воде кусок пластика.
в) Константин проверяет, выдержит ли верёвка вес альпиниста.
г) Василий хочет сделать такой стол, который выдержит нагрузку в 200 кг.
д) Алёна изучает строение молекулы воды.

## Какие бывают модели?

Существует множество классификации моделей, каждая из которых отражает какое-то одно свойство. Универсальной классификации моделей нет.

**По своей природе** модели делятся на *материальные* (физические, предметные) и *информационные*.

**Материальные модели** «можно потрогать» — это игрушки, уменьшенные копии самолётов и кораблей, чучела животных, учебные модели молекул и т. п.

**Информационные модели** — это информация о свойствах оригинала и его связях с внешним миром. Среди них выделяют *вербальные модели* (словесные, мысленные) и *знаковые модели*, записанные с помощью какого-то формального языка:

- *графические* (схемы, карты, фотографии, чертежи);
- *табличные*;
- *математические* (формулы);
- *логические* (варианты выбора на основе анализа условий);
- *специальные* (ноты, химические формулы и т. п.).

Различают *статические* и *динамические* модели.

В **статических моделях** предполагается, что интересующие нас свойства оригинала не изменяются во времени.

**Динамические модели** описывают движение, развитие, изменение.

Какие из этих моделей статические, а какие — динамические:

а) модель полёта шарика;
б) фотография;
в) видеозапись;
г) история болезни;
д) анализ крови;
е) модель молекулы воды;
ж) модель развития землетрясения;
з) модель вращения Луны вокруг Земли?

Динамические модели могут быть дискретными и непрерывными.

**Модель** называется **дискретной**, если она описывает поведение оригинала только в отдельные моменты времени. Например, модель колонии животных определяет их численность один раз в год.

**Непрерывная модель** описывает поведение оригинала для всех моментов времени из некоторого временно́го промежутка. Например, формула $y = sin\ x$ и график этой функции — это непрерывные модели. Так как компьютер работает только с дискретными данными, все компьютерные модели — дискретные.

**По характеру связей** модели делятся на *детерминированные* и *вероятностные*.

В **детерминированных моделях** связи между исходными данными и результатами жёстко заданы, при одинаковых исходных данных всегда получается один и тот же результат (например, при расчёте по известным формулам).

**Вероятностные модели** учитывают случайность событий в реальном мире, поэтому при одних и тех же условиях результаты нескольких испытаний модели могут отличаться. К вероятностным относятся модели броуновского движения частиц, полёта самолёта с учётом ветра, движения корабля при морском волнении, поведения человека. В результате эксперимента с такими моделями определяют некоторые средние величины по результатам серии испытаний, например среднюю скорость движения частиц, среднее отклонение корабля от курса и т. п. Несмотря на случайность, эти результаты достаточно стабильны, т. е. мало меняются при повторных испытаниях.

Используя дополнительные источники, выясните, от каких иностранных слов произошли слова «вербальный», «статический», «динамический», «детерминированный».

По материалам параграфа составьте в тетради схемы различных классификаций моделей.

**Имитационные модели** используются в тех случаях, когда поведение сложной системы нельзя (или крайне трудно) предсказать теоретически, но можно *смоделировать* её реакцию на внешние условия. Для того чтобы найти оптимальное (самое лучшее) решение задачи, нужно выполнить моделирование при многих возможных вариантах и выбрать наилучший из них. Такой метод часто называют *методом проб и ошибок*.

Имитационные модели позволяют очень точно описать поведение оригинала, но полученные результаты справедливы только

для тех случаев, которые мы моделировали (что случится в других условиях — непонятно). Примеры использования имитационных моделей:

- испытание лекарств на мышах, обезьянах, группах добровольцев;
- модели биологических систем;
- экономические модели управления производством;
- модели систем массового обслуживания (банки, магазины и т. п.).

Для понимания работы процессора можно использовать его имитационную модель, которая позволяет вводить команды в определённом формате и выполнять их, и показывает изменение значений *регистров* (ячеек памяти) процессора. Подобные модели применяют в том случае, когда нужно написать программу для системы, на которой её невозможно отлаживать (например, для микропроцессора, встроенного в бытовую технику). Такой подход называют *кросс-программированием*: программа пишется и отлаживается в одной системе, а работать будет в другой. В этом случае другую систему приходится моделировать с помощью имитационной модели.

**Игровые модели** позволяют учитывать действия противника, например, при моделировании военных действий, соревнований, конкуренции в бизнесе. Задача игрового моделирования — найти *лучшую стратегию в игре* — план действий, который даёт наилучшие результаты даже в том случае, когда противник играет безошибочно. Этими вопросами занимается *теория игр* — раздел математики, одним из создателей которого был американский учёный Джон фон Нейман.

## Адекватность моделей

Итак, при моделировании мы заменяем один объект (объект-оригинал) другим. Поэтому всегда возникает вопрос, можно ли верить полученным результатам. Иначе говоря, будет ли оригинал вести себя так же, как и модель?

---

**Адекватность модели** — это совпадение существенных свойств модели и оригинала в рассматриваемой задаче.

---

Используя дополнительные источники, выясните, от какого иностранного слова произошло слово «адекватный».

Адекватность означает, что результаты моделирования:

- не противоречат выводам теории, например законам сохранения (вещества, энергии и т. п.);
- подтверждаются экспериментом с оригиналом.

Таким образом, адекватность модели окончательно можно доказать только экспериментом: если результаты нашего моделирования близки к наблюдаемым на практике, это означает, что модель адекватна.

Для того чтобы вычислить ошибку моделирования, нужно модуль разности между результатом моделирования $X$ и результатом эксперимента $X_*$ разделить на результат эксперимента и умножить на 100%:

$$\delta X = \frac{|X - X_*|}{X_*} \cdot 100\%.$$

Величина $\delta X$ (читается «дельта икс») называется **относительной ошибкой**. На практике модель обычно считается адекватной, если относительная ошибка не превышает 10%.

Феофан построил математическую модель, которая позволяет прогнозировать изменение веса кошки. Для какого периода времени модель Феофана адекватна?

| **Возраст кошки, мес.** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Вес, кг (модель)** | 0,3 | 0,6 | 0,9 | 1,2 | 1,4 | 1,6 | 1,8 | 1,9 | 2,1 | 2,2 | 2,3 | 2,4 |
| **Вес, кг (фактически)** | 0,3 | 0,6 | 0,9 | 1,1 | 1,3 | 1,5 | 1,7 | 1,7 | 1,8 | 1,9 | 2,0 | 2,1 |

Нужно понимать, что любая модель отличается от оригинала, поэтому она может быть адекватна только при определённых условиях — в той задаче, для решения которой она создавалась. Например, модель деления амёб (через некоторый интервал времени каждая амёба делится надвое) адекватна только при малом количестве амёб и небольших интервалах наблюдения, иначе амёбы заполнили бы всё пространство.

Во многих случаях результаты моделирования — это некоторые числа, измеренные или рассчитанные по результатам эксперимента с моделью. Это могут быть, например, сила, расстояние, скорость, ускорение, давление и др. Чаще всего эти величины для модели и оригинала будут различаться, поэтому нужно уметь пересчитывать «модельные» данные в соответствующие значения для оригинала. Этими вопросами занимается *теория подобия*. Простейший пример — работа с картой. Расстояние, измеренное по карте, нужно умножить на масштабный множитель, тогда получится соответствующее расстояние на реальной местности.

## Выводы

- Модель — это объект, который обладает существенными свойствами другого объекта, процесса или явления (оригинала) и используется вместо него.
- Моделирование — это создание и исследование моделей для изучения оригиналов.
- С помощью моделирования можно решать задачи четырёх типов:
  1) изучение оригинала;
  2) анализ — прогнозирование влияния различных воздействий на оригинал;
  3) синтез — управление оригиналом;
  4) оптимизация — выбор наилучшего решения в данных условиях.
- Универсальной классификации моделей нет. По своей природе модели делятся на материальные и информационные.
- Адекватность модели — это совпадение существенных свойств модели и оригинала в рассматриваемой задаче. Проверить адекватность можно только путём эксперимента с оригиналом.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Что вы думаете по поводу другого определения модели: «Модель — это упрощённое представление реального объекта, процесса или явления»?
2. Приведите примеры разных моделей человека. Для решения каких задач они предназначены?
3. Приведите примеры моделей, с которыми мы работаем на компьютерах.
4. К какому типу (типам) можно отнести следующие модели?
   а) Каляка — это маляка с тремя грымзиками.
   б) $a^2 + b^2 = c^2$.
   в) Если горит красный свет, то стой. Если горит зелёный свет — иди.
   г) $2H_2 + O_2 = 2H_2O$.
   Используйте разные классификации.
5. Какую модель — вероятностную или детерминированную — вы рекомендуете выбрать для исследования движения судна в шторм? Почему?
6. Сравните достоинства и недостатки имитационных и теоретических моделей (например, записанных в виде формул).

7. Верно ли, что модели, используемые при создании компьютерных игр, это игровые модели? Обоснуйте вашу точку зрения.
8. Как можно доказать, что модель неадекватна?
9. Почему ни одна модель не может быть полностью адекватна оригиналу?
10. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «Анализ и синтез»
б) «Детерминированные и вероятностные модели»
в) «Игровые модели»
г) «Адекватность моделей»

### Практическая работа

Выполните практическую работу № 9 «Броуновское движение».

## § 14
## Математическое моделирование

***Ключевые слова:***

- математическая модель
- хорошо поставленные задачи
- существенные данные
- тестирование модели
- компьютерная модель
- эксперимент
- анализ результатов моделирования

Для того чтобы исследовать модель с помощью компьютера, нужно записать её на каком-то формальном языке. Условия многих задач точнее всего записываются с помощью языка математики — в виде формул. Такие модели называются **математическими.**

Когда формулы написаны, для исследования модели с помощью компьютера нужно написать программу — составить **компьютерную модель.** В этом параграфе мы на примерах рассмотрим основные этапы разработки и исследования математических моделей.

### Постановка задачи

Этап постановки задачи — самый важный при моделировании. Если здесь допущена ошибка, то фактически решается совсем не та задача, которую нужно решить, и после завершения моделирования всё придётся начать заново.

Для того чтобы задачу можно было решить, она должна быть **хорошо поставлена (корректна)**. Это значит, что:

- заданы все связи между исходными данными и результатом;
- известны все исходные данные;
- решение существует;
- решение единственно.

К сожалению, в реальных задачах бывает сложно строго доказать существование и единственность решения; более того, задача может иметь множество решений. В таких случаях формулировку задачи можно уточнить, например, так:

- найти любое решение, если оно существует;
- найти все решения в некоторой области (например, все решения уравнения на отрезке [0, 1]);
- найти всё множество решений (например, для уравнения $\sin x = 1$).

Приведём примеры **плохо поставленных (некорректных)** задач.

**Задача 1.** Уроки в школе начинаются в 8-30. В 10-00 к школе подъехал красный автомобиль. Определите, когда Шурик выйдет играть в футбол.

**Задача 2.** Мальчик Вася в синей кепке бросает белый мяч со скоростью 12 м/с. Через какое время мяч впервые ударится о земную поверхность?

**Задача 3.** Решите уравнение $\sin x = 4$.

**Задача 4.** Найдите функцию, график которой проходит через точки (0, 0) и (1, 1).

Для каждой из этих задач определите, почему их нельзя считать хорошо поставленными. ?

Что делать, если полученная задача плохо поставлена? Решить её нельзя, поэтому остаётся уточнять условия и исходные данные. Если и это невозможно, нужно вводить **допущения** – упрощающие предположения, которые позволят сделать задачу хорошо поставленной.

Все дальнейшие рассуждения мы будем проводить для задачи 2. Она плохо поставлена, потому что неизвестно, из какой точки и под каким углом брошен мяч. Дополним условие, чтобы сделать задачу корректной, например, так: *Вася бросает мяч вертикально вверх. В момент броска мяч находится на высоте 1,5 м.*

Всегда ли существует решение задачи 2? Да, всегда. По закону всемирного тяготения мяч притягивается к земной поверхности и когда-нибудь упадёт на поверхность.

Единственно ли решение? Да, в этой задаче решение единственно.

## Разработка математической модели

На этапе разработки математической модели нужно:

1) выделить исходные данные, *существенные* для решения данной задачи;
2) построить математическую модель, отражающую только существенные свойства оригинала.

? Какие данные в формулировке задачи 2, на ваш взгляд, существенные, а какие — нет?

Введём некоторые допущения:

- мяч — материальная точка (его размеры малы в сравнении с высотой полёта);
- сопротивление воздуха не учитывается.

При решении задачи могут использоваться несколько моделей разных типов. Например, для лучшего понимания полезно построить *графическую модель* задачи (рис. 3.1).

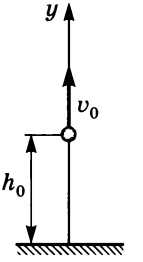


**Рис. 3.1**

За начало координат удобно принять точку, откуда вылетает мяч. Обозначим через $v_0$ начальную скорость мяча, через $h_0$ — начальную высоту мяча ($h_0$ = 1,5 м) — это исходные данные. Нужный результат — это время полёта мяча $t_п$ (рис. 3.2).

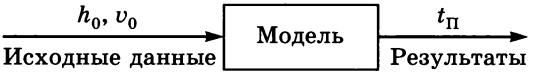


**Рис. 3.2**

Графическая модель не даёт ответа на поставленный вопрос, а только позволяет лучше понять задачу. Поэтому для численных расчётов нужно построить математическую модель — связать с помощью законов физики известные данные и результат.

Координата $y$ при движении тела, брошенного вертикально вверх, вычисляется по формуле:

$$y = h_0 + v_0 \cdot t - \frac{g \cdot t^2}{2},$$

где $g \approx 9{,}81$ м/с$^2$ — ускорение свободного падения. Эта формула и представляет собой математическую модель задачи. В ней нет

упоминания о Васе, мяче, и т. п., есть только условные обозначения. Мы выполнили **формализацию** — построили формальную модель на языке математических формул.

По условию задачи нужно найти время $t_п$, при котором мяч упадёт на земную поверхность. Запишите условие «мяч упадёт на землю» в виде формулы и затем запишите уравнение, которое нужно решить.

## Тестирование модели

После построения модели её обязательно нужно протестировать (проверить).

**Тестирование** — это проверка модели на наборе исходных данных с известным результатом.

Например, при моделировании накопления денег в банке сумма не должна меняться при нулевой ставке. Тестирование модели движения судна тоже начинается с простых задач: если штурвал поворачивают влево, судно должно уходить влево, и наоборот.

Удачное тестирование модели не гарантирует, что она правильна; тестирование может только установить ошибочность модели. Чтобы доказать её правильность, нужно проверить модель при всех допустимых исходных данных (в том числе и для тех, для которых правильный ответ неизвестен), а это практически невозможно.

Выполним тестирование математической модели, построенной для нашей задачи:

$$y = h_0 + v_0 \cdot t - \frac{g \cdot t^2}{2}.$$

Используя эту модель, определите:
а) чему равна координата $y$ мяча при $t = 0$?
б) станет ли в какой-то момент координата $y$ равной нулю? Почему?
в) что произойдёт при нулевой начальной скорости?

Все результаты не противоречат теории, поэтому можно считать, что тестирование успешно.

Можно ли считать, что мы доказали правильность модели? Почему?

Руслан построил свою модель движения мяча, брошенного вертикально вверх:

$$y = h_0 + v_0 \cdot t + \frac{g \cdot t^2}{2}.$$

Выполните тестирование модели по тем же критериям, сделайте выводы.

У Марии для той же задачи получилась ещё одна модель:

$$y = h_0 + v_0 \cdot t - \frac{g \cdot t^2}{3}.$$

Выполните тестирование модели по тем же критериям, сделайте выводы.

## Построение компьютерной модели

Мы только что получили модель задачи, которая свелась к уравнению:

$$0 = h_0 + v_0 \cdot t - \frac{g \cdot t^2}{2}.$$

Это квадратное уравнение можно решить аналитически, получить точный ответ в виде формул. Эти формулы нужно заложить в программу. Приведём программы на алгоритмическом языке и на языке Паскаль, которые находят решение по формулам, известным вам из курса математики:

```
алг Полёт
нач
  вещ h0=1.5, v0=12,
      g=9.81
  вещ a, b, c, D, t1, t2
  a:=-g/2
  b:=v0
  c:=h0
  D:=b*b-4*a*c
  t1:=(-b+sqrt(D))/(2*a)
  t2:=(-b-sqrt(D))/(2*a)
  вывод t1, нс, t2
кон
```

```
program Polet;
var h0, v0, g: real;
    a, b, c, D, t1, t2: real;
begin
  h0:= 1.5; v0:= 12; g:= 9.81;
  a:=-g/2;
  b:=v0;
  c:=h0;
  D:=b*b-4*a*c;
  t1:=(-b+sqrt(D))/(2*a);
  t2:=(-b-sqrt(D))/(2*a);
  writeln(t1);
  writeln(t2);
end.
```

Изучите программу и ответьте на вопросы.

- Что обозначают переменные a, b, c, D, t1 и t2?
- Какое из двух решений квадратного уравнения нужно выбрать?
- Какой результат возвращает стандартная функция sqrt?

– Может ли случиться так, что уравнение не будет иметь вещественных корней?
– Как изменить начальную высоту шарика и начальную скорость?

В более сложных случаях точное решение найти не удаётся, и приходится строить *компьютерную имитационную модель*. Для этого можно применить табличный процессор (*OpenOffice Calc*, *Microsoft Excel* и т.п.), написать собственную программу на одном из языков программирования или использовать готовую среду для моделирования (например, *Simulink* или *VisSim*). К сожалению, такие модели чаще всего дают только приближённое (неточное) решение задачи.

---

Попробуем написать свою программу для имитационного моделирования. Как вы знаете, компьютер — это дискретное устройство, он работает с дискретными данными. Поэтому нужно выполнить **дискретизацию** задачи. Мы будем вычислять скорость и высоту шарика только в отдельные моменты времени $0$, $\Delta t$, $2\Delta t$, $3\Delta t$, ..., где $\Delta t$ — это небольшой интервал времени, который называют **интервалом дискретизации**. Таким образом, рассматриваются только значения времени $t_0$, $t_1$, $t_2$, ..., где $t_i = i \cdot \Delta t$.

Пусть в начале отрезка $[t_i, t_{i+1}]$ (при $t = t_i$) мы знаем координату (высоту) шарика $y_i$ и значение его скорости $v_i$. Для простоты будем считать, что в течение времени $t_i \le t < t_{i+1}$ скорость не изменяется, а в конце отрезка изменяется скачком. Тогда координату и скорость в конце отрезка (в момент $t_{i+1}$) можно вычислить по формулам:

$$y_{i+1} = y_i + v_i \cdot \Delta t;$$
$$v_{i+1} = v_i - g \cdot \Delta t.$$

Здесь $g$ — это ускорение свободного падения, благодаря которому изменяется скорость.

Запишите значения координаты и скорости в конце первого и второго отрезков времени (в моменты $t_1$ и $t_2$), выразив их через известные величины: $v_0$, $h_0$, $g$ и $\Delta t$.

Так как в каждый конкретный момент нас интересуют только текущие значения координаты и скорости, в программе будем использовать всего одну переменную `y` для хранения текущей координаты (т. е. координаты в данный момент времени) и одну переменную `v` для хранения текущей скорости. Получается такая программа:

```
алг Полёт-2
нач
  вещ h0=1.5, v0=12, g=9.81
  вещ y, v, t, dt=0.01
  y:=h0; v:=v0; t:=0
  нц пока y>=0
    y:=y+v*dt
    v:=v-g*dt
    t:=t+dt
  кц
  вывод t
кон
```

```
program Polet_2;
var h0, v0, g: real;
    y, v, t, dt: real;
begin
  h0:=1.5; v0:=12; g:=9.81;
  dt:=0.01;
  y:=h0; v:=v0; t:=0;
  while y>=0 do begin
    y:=y+v*dt;
    v:=v-g*dt;
    t:=t+dt;
  end;
  writeln(t);
end.
```

Изучите программу и ответьте на вопросы.

- Что обозначают переменные y, v и t?
- Какие начальные значения присваиваются переменным y, v, и t? Почему именно такие?
- Когда работа цикла завершится?
- Может ли случиться так, что цикл будет работать бесконечно?
- Что произойдёт, если в условии работы цикла использовать строгое неравенство $y > 0$?
- Значение какой переменной выводится в качестве результата? Почему?
- Как изменятся при уменьшении интервала дискретизации $\Delta t$ длина текста программы и количество операций, выполняемых компьютером?

Для тестирования этой программы нужно провести контрольные расчёты для простых случаев с известным результатом, например для тех, которые мы ранее использовали при тестировании математической модели. Если проверка прошла удачно (противоречий не обнаружено), можно переходить к компьютерному эксперименту.

## Эксперимент с моделью

Эксперимент — это испытание модели в тех условиях, которые нас интересуют (результатов мы заранее не знаем). Например, для модели накопления денег в банке задаётся ненулевая ставка (процент ежегодного увеличения); движение судна моделируется с учётом случайных помех — морского волнения и ветра и т. п.

Объём данных, получаемых при математическом моделировании сложных физических процессов, часто настолько велик (мега-

байты и даже гигабайты!), что эти данные сначала сохраняют в виде файлов, а потом обрабатывают специальными программами.

Можно ли слепо верить результатам моделирования? Конечно же нет, ведь мы не проверяли (и не могли проверить!) работу модели в этих условиях. Поэтому необходим анализ результатов.

## Анализ результатов

Во-первых, нужно убедиться, что результаты моделирования не противоречат известным из теории фактам, например не нарушаются законы сохранения вещества и энергии. Например, если оказалось, что при наличии трения объект движется быстрее, чем без трения, то модель, скорее всего, ошибочна.

Во-вторых, необходимо проверить результаты моделирования на реальном объекте — провести эксперимент с оригиналом. Если нам удалось решить поставленную задачу (поведение оригинала соответствует[1)] поведению модели), можно считать модель адекватной и работу законченной.

Если же результаты нас не устраивают (поведение объекта и оригинала значительно различаются), требуется вернуться к одному из предыдущих этапов и повторить моделирование, например:

- изменить алгоритм или условия моделирования;
- изменить модель: учесть дополнительные свойства, которые ранее считались несущественными (например, учесть сопротивление воздуха);
- изменить постановку задачи (если выяснилось, что решили не ту задачу, которую нужно было решать).

Возможно, что несоответствие вызвано принятыми допущениями. Так, в задаче с полётом мяча полезно ответить на следующие вопросы, которые помогут выяснить причину неудачи:

— Что изменится, если начальная скорость будет отличаться от заданной?
— Что изменится, если мяч не считать материальной точкой?
— Насколько сильно сопротивление воздуха влияет на результат? И т. д. ...

## Выводы

- Математическая модель — это модель, записанная в виде математических формул.
- Основные этапы математического моделирования — это постановка задачи, разработка модели, тестирование модели,

[1)] Часто считается, что допустимо расхождение результатов эксперимента и моделирования не более, чем на 10%.

разработка компьютерной модели, эксперимент с моделью и анализ результатов.

- Задача называется хорошо поставленной, если:
  — заданы все связи между исходными данными и результатом;
  — известны все исходные данные;
  — решение существует;
  — решение единственно.
- При построении модели нужно выделить существенные данные, влияющие на результат.
- Тестирование — это проверка модели на простых исходных данных с известным результатом.
- Для проведения компьютерных экспериментов нужно построить компьютерную модель на основе математической модели. Для этого можно написать свою программу, использовать табличный процессор или специальную среду моделирования.
- Компьютерный эксперимент — это испытание модели в тех условиях, для которых мы не знаем результата.
- После выполнения компьютерного эксперимента выполняется анализ результатов.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Что делать, если задача плохо поставлена?
2. Как выделить существенные свойства, которые нужно учесть в модели?
3. В чём различие между математической моделью и словесным описанием?
4. Как вы думаете, перечень этапов моделирования при использовании моделей других типов (не математических) будет таким же или другим?
5. Задача сформулирована так:

   Электрик Семён в зелёном комбинезоне едет на красном автомобиле «Лада Калина» из Москвы в Воронеж. Его средняя скорость равна 90 км/ч, причём каждые 2 часа Семён отдыхает по 15 минут. Когда он приедет в Воронеж?

   Какие данные здесь существенны, а какие — нет? Каких данных не хватает?
6. Можно ли доказать правильность (или ошибочность) модели с помощью тестирования?
7. Чем эксперимент с моделью отличается от тестирования?

8. Что делать, если после анализа результатов моделирования обнаружилось, что поведение оригинала существенно отличается от поведения модели?
9. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «Математическое моделирование»
б) «Программные средства для моделирования»

### Практические работы

Выполните практические работы:
№ 10 «Полёт шарика»;
№ 11 «Полёт шарика-2».

# § 15
# Табличные модели. Диаграммы

***Ключевые слова:***

- таблица «объект – свойство»
- таблица «объект – объект»
- оптимальный маршрут
- диаграмма

## Таблицы «объект – свойство»

Табличные модели удобно использовать тогда, когда нужно в наглядной форме представить **информацию об объектах,** имеющих одинаковый набор свойств (таблицы **«объект – свойство»**).

Например, в следующей таблице элементы в каждой строке связаны между собой — это свойства некоторого объекта (человека) — табл. 3.1.

Таблица 3.1

| Фамилия | Имя | Рост, см | Вес, кг | Год рождения |
|---|---|---|---|---|
| Иванов | Иван | 175 | 67 | 1996 |
| Петров | Пётр | 164 | 70 | 1998 |
| Сидоров | Сидор | 168 | 63 | 2000 |

Именно так хранится информация в базах данных.

Возможен и другой вариант таблицы, когда роли строк и столбцов меняются. В первом столбце записываются названия свойств, а данные в каждом из следующих столбцов описывают свойства какого-то объекта. Например, вот таблица с характеристиками разных марок автомашин (табл. 3.2).

Таблица 3.2

| **Марка** | Лада Приора | Лада Калина | ВАЗ 2110 | ВАЗ 21099 |
|---|---|---|---|---|
| **Мощность двигателя, л.с.** | 98 | 89 | 79 | 70 |
| **Максимальная скорость, км/ч** | 183 | 165 | 165 | 156 |
| **Время разгона до 100 км/ч, с** | 11,5 | 12,5 | 14 | 15 |

В виде таблиц оформляются расписания (уроков, поездов, самолётов), статистические данные (например, сколько произведено чугуна и стали на душу населения в разных странах). Функция тоже может быть задана в виде таблицы. С помощью таблицы Д. И. Менделеева устанавливается связь между свойствами химического элемента и зарядом атомного ядра.

Запишите в виде таблицы значения функции $y = x^2$ на отрезке [0; 5] с шагом 1. Как вы думаете, это непрерывная модель или дискретная? Почему?

## Таблицы «объект – объект»

Таблица может определять **отношения между объектами** (таблица **«объект–объект»**). Например, в табл. 3.3 показано, сколько и каких автомобилей сумела продать компания в разных городах.

Таблица 3.3

| | **Лада** | **УАЗ** | **Тойота** | **Форд** |
|---|---|---|---|---|
| **Москва** | 520 | 210 | 805 | 370 |
| **Санкт-Петербург** | 430 | 350 | 260 | 410 |
| **Пермь** | 120 | 200 | 150 | 230 |

Для обработки табличных данных предназначены специальные программы — *табличные процессоры* (электронные таблицы), с которыми вы уже знакомы.

## Оптимальный маршрут

Рассмотрим задачу, которая требует анализа табличных данных: определение оптимального (самого лучшего) маршрута поездки.

**Задача.** Путешественник прибыл в посёлок Берёзовое в 8 утра по местному времени и увидел следующее расписание автобусов (табл. 3.4).

Таблица 3.4

| Отправление из | Прибытие в | Время отправления | Время прибытия |
|---|---|---|---|
| Берёзовое | Лесное | 07:30 | 10:00 |
| Берёзовое | Осиновое | 11:50 | 14:10 |
| Лесное | Берёзовое | 12:50 | 15:20 |
| Полевое | Лесное | 13:20 | 14:40 |
| Осиновое | Полевое | 14:00 | 17:15 |
| Лесное | Осиновое | 14:20 | 15:30 |
| Осиновое | Лесное | 14:40 | 15:50 |
| Берёзовое | Полевое | 16:00 | 17:50 |
| Лесное | Полевое | 16:10 | 17:30 |
| Полевое | Осиновое | 17:40 | 19:55 |

Определите самое раннее время, когда он может попасть в посёлок Полевое, и как ему нужно ехать.

Из расписания видно, что автобусы ходят между четырьмя населёнными пунктами. Нарисуем схему, показывающую все возможные способы переезда из посёлка Березовое в посёлок Полевое. Буквы в кружках обозначают посёлки (Б — Берёзовое, П — Полевое, Л — Лесное и О — Осиновое), а слева и справа от них записано время отправления и прибытия автобусов согласно расписанию (рис. 3.3).

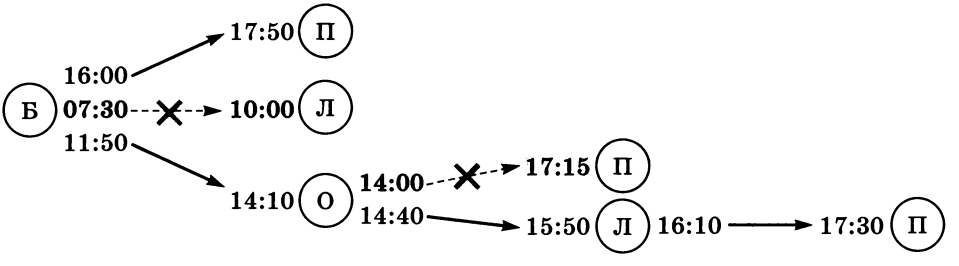



**Рис. 3.3**

Сравните таблицу 3.1 и схему на рис. 3.3. Ответьте на вопросы.
- Что означают буквы Б, П, Л и О в кружках?
- Что обозначают числа справа и слева от кружков?
- Что обозначают штриховые линии, перечёркнутые крестиками?
- Какие варианты маршрута есть у путешественника? Сколько их?
- Какой из этих вариантов позволяет быстрее добраться до Полевого?

Обратите внимание, что при поиске оптимального решения мы построили графическую модель задачи в виде *дерева* — многоуровневой структуры. С деревьями вы знакомы из курса 7 класса, скоро мы снова с ними встретимся.

## Анализ диаграмм

Данные таблиц можно наглядно представить в виде диаграмм. Посмотрим, как можно обрабатывать информацию, «закодированную» в форме диаграмм.

Вспомните, какие типы диаграмм вы знаете. В каких случаях используется каждый из них?

**Задача 1.** Биологи пересчитали лосей, белок и зайцев на трёх участках заповедника и построили диаграмму (рис. 3.4).

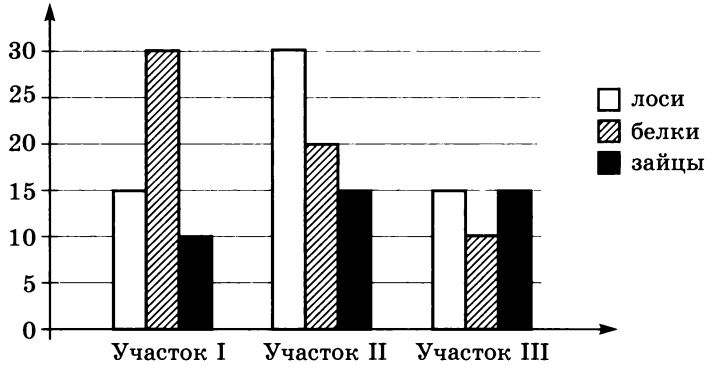



**Рис. 3.4**

Какая из диаграмм на рис. 3.5 правильно отражает соотношение общего числа животных разных видов по всему заповеднику?

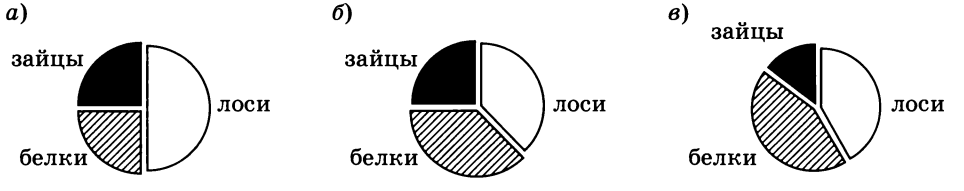



**Рис. 3.5**

Сначала нужно «снять» данные со столбчатой диаграммы и записать их в таблицу (табл. 3.5).

Таблица 3.5

| | Участок I | Участок II | Участок III |
|---|---|---|---|
| **Лоси** | 15 | 30 | 15 |
| **Белки** | 30 | 20 | 10 |
| **Зайцы** | 10 | 15 | 15 |

Теперь добавим справа один столбец и сосчитаем, сколько было всего животных каждого вида и их общее количество (табл. 3.6).

Таблица 3.6

| | Участок I | Участок II | Участок III | Всего |
|---|---|---|---|---|
| **Лоси** | 15 | 30 | 15 | 60 |
| **Белки** | 30 | 20 | 10 | 60 |
| **Зайцы** | 10 | 15 | 15 | 40 |
| **Всего** | | | | 160 |

Изучите последний столбец таблицы на табл. 3.6. Ответьте на вопросы.

- Для каких животных соответствующие секторы на круговой диаграмме должны быть равны по площади?
- Есть ли ещё какие-то простые соотношения между секторами (например, какой-то сектор составляет 1/2, 1/3, 1/4 от общего количества или от площади другого сектора)?

*Ответ*: правильная диаграмма показана на рис. 3.5, *б*.

**Задача 2.** В фирме «Слонопотам» работают менеджеры, рабочие и охрана. Они ездят на машинах четырёх марок: «Лада», «Форд», «Тойота» и «Ауди», каждый имеет только одну машину. На диаграмме 1 (рис. 3.6) показано количество работников, имеющих машины определённой марки, а на диаграмме 2 — соотношение менеджеров, рабочих и охраны.

Какие из этих утверждений следуют из анализа диаграмм?

а) Все «форды» могут принадлежать менеджерам.
б) Все охранники могут ездить на «ауди».
в) Все «тойоты» могут принадлежать рабочим.
г) Все рабочие могут ездить на «фордах».

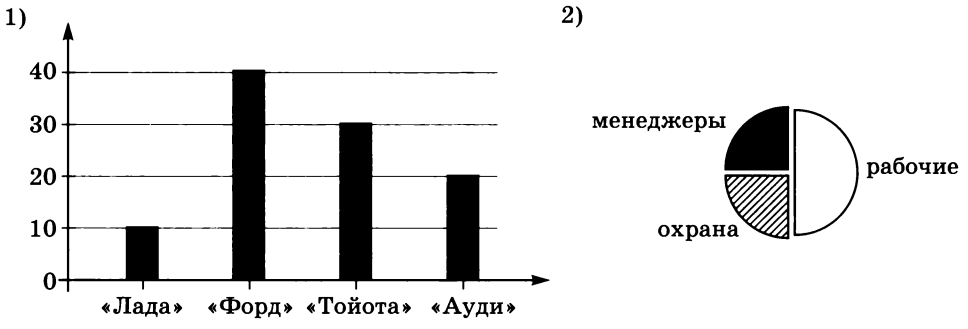



**Рис. 3.6**

Изучите диаграммы на рис. 3.6. Определите общее количество работников фирмы. Затем определите, какую часть составляют от общего количества менеджеры, охрана и рабочие. Потом вычислите, сколько в фирме менеджеров, охранников и рабочих.

Теперь рассмотрим предложенные утверждения.

а) Все «форды» (40 штук) не могут принадлежать менеджерам, так как менеджеров только 25 и каждый имеет одну машину.

б) Все охранники (25 человек) не могут ездить на «ауди», потому что этих машин всего 20.

в) Все «тойоты» (30 штук) **могут** принадлежать рабочим (их 50 человек).

г) Все рабочие (50 человек) не могут ездить на «фордах» (их всего 40).

Таким образом, верно только утверждение в).

## Выводы

- В таблице типа «объект – свойство» представляется информация о группе объектов, имеющих одинаковый набор свойств. В таком виде хранится информация в базах данных.
- Таблица типа «объект – объект» задаёт связь между двумя группами объектов.
- Для наглядного представления данных таблиц используют диаграммы.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

Выполните по указанию учителя задания в рабочей тетради.

## Подготовьте сообщение

а) «Диаграммы Ганта»

б) «Использование ленты времени»

# § 16
# Списки и деревья

***Ключевые слова:***

- список
- дерево
- иерархия
- корень
- лист
- двоичное дерево

## Списки

**Список** — это последовательность элементов, в которой важен порядок их расположения. Говорят, что список — это **упорядоченная** последовательность.

Как можно назвать неупорядоченную последовательность, в которой элементы не повторяются?

Для каждого элемента в списке, кроме первого, можно назвать предыдущий элемент; для каждого элемента, кроме последнего, — следующий (рис. 3.7).

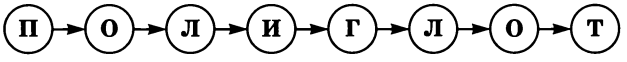



**Рис. 3.7**

Для списка на рис. 3.7 назовите:
а) первый и последний элементы;
б) предыдущий элемент для «И»;
в) предыдущий элемент для «П»;
г) следующий элемент для «Г»;
д) следующий элемент для «Т».

Списки могут служить моделями для каких-то объектов. Например, список на рис. 3.7 — это модель слова «полиглот». Предложение можно представить в виде списка слов, абзац — в виде списка предложений, текст — в виде списка абзацев.

Используя дополнительные источники, выясните, от какого иностранного слова произошло слово «полиглот» и что оно обозначает.

Структура данных «список» есть во многих языках программирования. Списки обычно записывают в квадратных скобках, перечисляя через запятую их элементы по порядку, с первого до последнего. Например, так:

```
['Amicus', 'Socrates', 'sed', 'magis', 'amica', 'veritas']
```

Используя дополнительные источники, выясните значение известной фразы, которую задаёт этот список слов.

Используя дополнительные источники, выясните, название какого языка программирования образовано от слов «обработка списков».

В список можно добавлять новые элементы (до или после заданного элемента), можно также заменять и удалять элементы. Если применять эти операции к словам (спискам букв), то мы можем из одного слова получить другое. Например, слово КОРОНА можно получить из слова КРАН так:

КРАН → КОАН → КОРН → КОРО → КОРОН → КОРОНА

Назовите операции, которые были выполнены на каждом шаге.

Но это не единственный вариант такого преобразования.

Постарайтесь найти самый короткий вариант получения слова КОРОНА из слова КРАН.

---

Для того чтобы оценить «расстояние» между «словами» при таком преобразовании в виде числа, каждой операции преобразования присваивается своя «цена», например, так (табл. 3.7).

Таблица 3.7

| Операция | Цена |
|---|---|
| Замена гласной буквы на гласную или согласной на согласную | 1 |
| Замена гласной на согласную или согласной на гласную | 2 |
| Вставка или удаление буквы | 5 |

Как можно преобразовать слово СКАНЕР в слово ПРИНТЕР? Определите «стоимость» такого преобразования — «расстояние» между словами.

Подобная задача возникает при исследовании белко́в человека и животных, этим занимается наука *биоинформатика*. Белки́ — это цепочки аминокислот, каждая аминокислота обозначается латинской буквой. Сравнить два белка — это значит определить, как проще всего из одного белка получить другой.

---

## Что такое дерево?

Предположим, в некоторой фирме есть директор, ему подчиняются главный инженер и главный бухгалтер, у каждого из них есть свои подчинённые. Если мы захотим нарисовать схему управления этой фирмы, она получится **многоуровневой** (рис. 3.8).

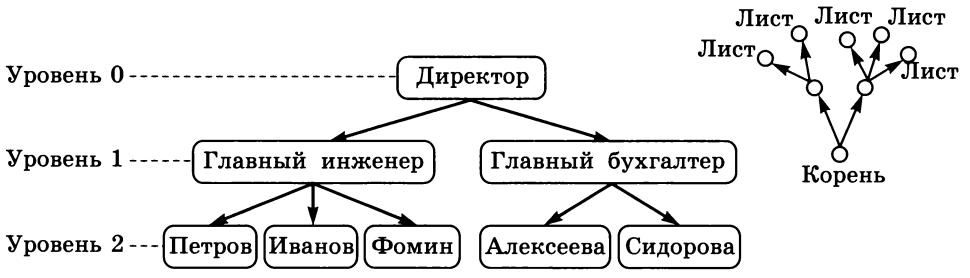



**Рис. 3.8**

Такая структура, в которой одни элементы «подчиняются» другим, называется *иерархической*. В информатике её называют **деревом**. Дело в том, что, если перевернуть эту схему вверх ногами, она становится похожа на дерево.

---

**Дерево** — это структура данных, которая служит моделью многоуровневой структуры (иерархии).

---

Несколько деревьев образуют *лес*.

Используя дополнительные источники, выясните, от какого иностранного слова произошло слово «иерархия».

## Из чего состоит дерево?

Дерево состоит из узлов, связанных между собой. Самый первый узел, расположенный на верхнем уровне (в него не входит ни одна стрелка), — это **корень** дерева, от него отходят **ветви** дерева (рис. 3.9). Конечные узлы, из которых не выходит ни одна

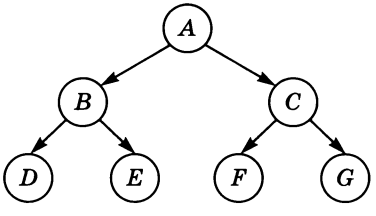



**Рис. 3.9**

ветвь, называются **листьями**. Все остальные узлы, кроме корня и листьев, — это внутренние узлы.

Назовите корень, листья и внутренние узлы дерева, изображённого на рис. 3.9.

Как вы думаете, можно ли считать список частным случаем дерева? Почему?

**Путь** — это последовательность узлов, где каждый следующий связан с предыдущим.

Можно ли считать, что путь — это список? Почему?

**Высота дерева** — это наибольшая длина пути от корня дерева к листу.

Определите высоту дерева на рис. 3.9.

**Поддерево** — это часть дерева, которая тоже представляет собой дерево.

Например, в дереве на рис. 3.9 узлы $B$, $D$ и $E$ вместе с ветвями $BD$ и $BE$ составляют поддерево.

Назовите другие поддеревья дерева на рис. 3.9.

Из двух связанных узлов тот, который находится на более высоком уровне, называется **родителем**, а другой — **сыном**. Корень — это единственный узел, у которого нет родителя; у листьев нет сыновей.

Используются также понятия «предок» и «потомок». **Потомок** какого-то узла — это узел, в который можно перейти по стрелкам от узла-предка. Соответственно, **предок** какого-то узла — это узел, из которого можно перейти по стрелкам в данный узел.

Дерево часто используют для изображения родственных связей семьи: такое дерево называется **генеалогическим деревом**. В корне записывают самого дальнего известного предка, остальные узлы — это его потомки: на первом уровне — дети, на втором — внуки и т. д. (рис. 3.10).

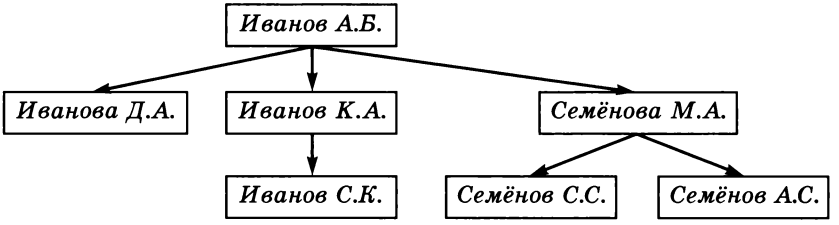



**Рис. 3.10**

Для дерева на рис. 3.9 определите родителей, предков и потомков узлов *E*, *C* и *A*.

Составьте в тетради генеалогическое дерево вашей семьи.

## Где используются деревья?

Типичный пример иерархии — различные **классификации** (животных, растений, минералов, химических соединений). Например, отряд Хищные делится на два подотряда: Псообразные и Кошкообразные. В каждом из них выделяют несколько семейств (рис. 3.11).

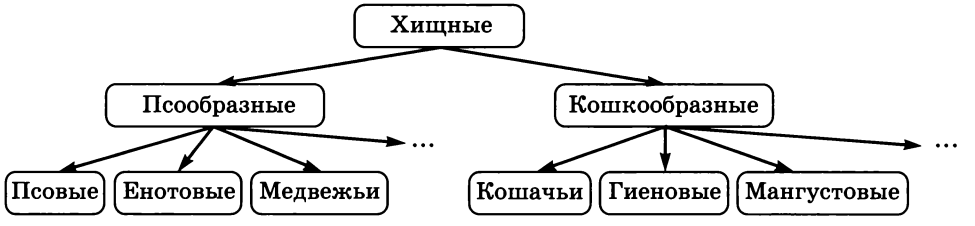



**Рис. 3.11**

Конечно, на этой схеме показаны не все семейства, остальные обозначены многоточием.

В текстах иерархию часто представляют в виде многоуровневого списка. Например, оглавление книги о хищниках может выглядеть так:

Глава 1. Псообразные
- Псовые
- Енотовые
- Медвежьи

Глава 2. Кошкоообразные
2.1 Кошачьи
2.2 Гиеновые
2.3 Мангустовые
...

Работая с файлами и папками, мы тоже встречаемся с иерархией: классическая файловая система имеет древовидную структуру[1]. Вход в папку — это переход на следующий (более низкий) уровень иерархии (рис. 3.12).

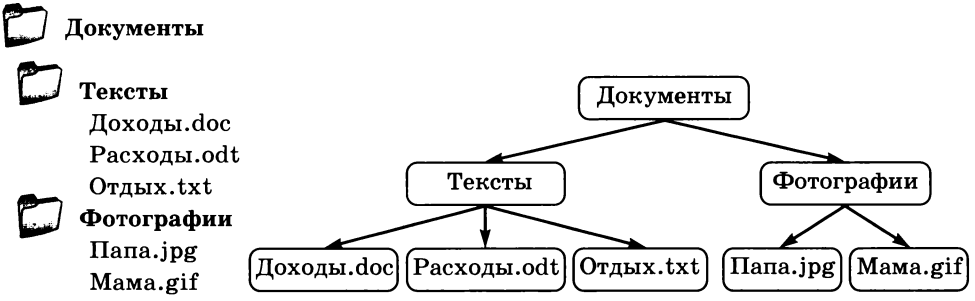



**Рис. 3.12**

Запишите по схеме на рис. 3.12 полный адрес файла *Расходы.odt*. Если вы работаете в операционной системе Windows, считайте, что папка *Документы* находится в корневой папке диска C:, для системы *Linux* — в папке */home/sonya*.

Алгоритм вычисления арифметического выражения может быть представлен в виде модели-дерева (рис. 3.13).

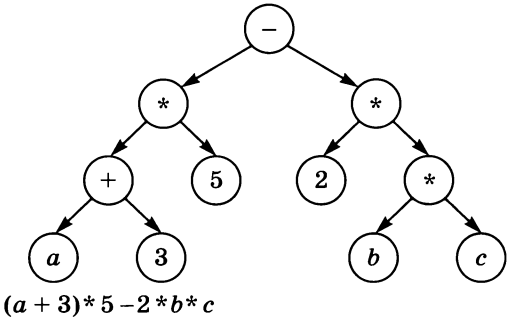



**Рис. 3.13**

[1] В некоторых файловых системах файл может «принадлежать» нескольким папкам одновременно. При этом древовидная структура, строго говоря, нарушается.

Здесь листья — это числа и переменные, а корень и промежуточные узлы — знаки операций. Вычисления выполняются «снизу вверх» — от листьев — к корню.

Представление арифметического выражения в виде дерева позволяет выделить независимые друга от друга операции, которые современные процессоры могут выполнять одновременно (или, как говорят, *параллельно*). Например, на рис. 3.14 видно, что умножение $b \cdot c$ никак не зависит от сложения $a + 3$. Если считать, что время выполнения всех операций одинаково и равно $T$, то процессор, имеющий несколько устройств для выполнения арифметических операций, вычислит это выражение за время $3T$ вместо $5T$.

В дереве для вычисления арифметического выражения (см. рис. 3.13) каждый узел может иметь не более двух сыновей, поэтому оно называется **двоичным** (или **бинарным**). От расположения сыновей зависит результат вычитания и деления, поэтому используют термины «левый сын» и «правый сын», «левое поддерево» и «правое поддерево».

Используя дополнительные источники, выясните, от какого иностранного слова произошло слово «бинарный».

Постройте дерево, соответствующее арифметическому выражению (5*b+a)/(2*a+3*b+6).

## Перебор вариантов

С помощью дерева можно изобразить многошаговый процесс, в котором на каждом шаге делается выбор из нескольких вариантов. Например, русские богатыри в сказках часто выбирали на развилке, куда поехать — налево, прямо или направо. В игре «крестики-нолики» каждый игрок на каждом ходу тоже выбирает один из всех возможных вариантов.

Например, построим все двухбуквенные слова, которые можно записать с помощью алфавита {А, Б, В}. Начнём с пустого слова, в котором нет ни одной буквы (на рис. 3.14 оно изобра-

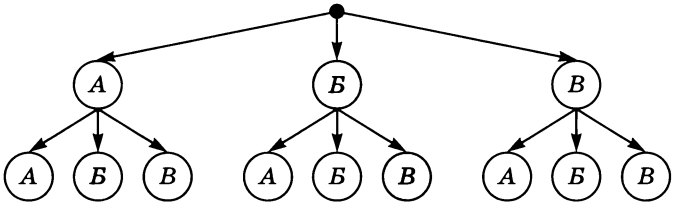



**Рис. 3.14**

жено чёрным кружком в корне дерева), и будем на каждом шаге добавлять в конец слова одну из трёх возможных букв (см. рис. 3.14).

Чтобы найти само слово, нужно выписать метки всех узлов на пути от корня до листа. Например, узлы, выделенные серым фоном на рис. 3.14, соответствуют слову БВ.

Можно рисовать это дерево иначе, повернув его на бок. В § 15 такая запись использовалась при выборе оптимального маршрута.

Разведчик выяснил, что ключ к замку от сейфа состоит из трёх символов, причём могут использоваться буквы из алфавита {A, B, C, D}. Две одинаковые буквы не могут стоять рядом. Рядом с буквой D обязательно должна стоять буква A. Если в ключе есть буква B, то там не может быть буквы C. Постройте дерево перебора вариантов и подсчитайте, сколько различных ключей удовлетворяют этим условиям.

## Дерево для двоичного кода

Для двоичного кода тоже можно построить дерево. Например, для кода

| А | Б | В | Г | Д |
|---|---|---|---|---|
| 0 | 11 | 101 | 110 | 111 |

дерево выглядит так (рис. 3.15).

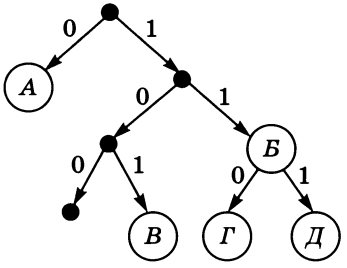



**Рис. 3.15**

Из каждого узла выходят две ветви — при движении по левой ветви мы выбираем 0, при движении по правой — 1. В некоторых узлах записаны буквы, остальные обозначены точками. Участки ветвей, соединяющие два узла, называются **рёбрами**. Чтобы получить код буквы,

нужно пройти по ветвям от корня к узлу, в котором записана эта буква, и выписать коды всех пройденных рёбер.

Проверьте, выполняется ли для этого кода *условие Фано*: ни одно из кодовых слов не совпадёт с началом другого кодового слова. Как сразу определить это по дереву? Где должны располагаться узлы с буквами, чтобы условие Фано выполнялось?

Постройте дерево для следующего кода:

| А | Б | В | Г |
|---|---|---|---|
| 00 | 100 | 101 | 01 |

Выполняется ли для него условие Фано? Сколько букв можно добавить в кодовую таблицу, чтобы все коды были не длиннее трёх символов и выполнялось условие Фано?

## Выводы

- Список — это упорядоченная последовательность элементов. Для каждого элемента в списке, кроме первого, можно назвать предыдущий элемент; для каждого элемента, кроме последнего, — следующий.
- В список можно добавлять новые элементы (до или после заданного элемента), можно также заменять и удалять элементы.
- Дерево — это структура данных, которая служит моделью многоуровневой структуры (иерархии). Несколько деревьев образуют лес.
- Дерево состоит из узлов, связанных между собой. Самый первый узел, расположенный на верхнем уровне, — это корень дерева. От корня отходят ветви дерева. Участок ветви, соединяющий два узла, называется ребром. Конечные узлы, из которых не выходит ни одна ветвь, называются листьями.
- Путь — это последовательность узлов, где каждый следующий связан с предыдущим.
- Высота дерева — это наибольшая длина пути от корня дерева к листу.
- Поддерево — это часть дерева, которая тоже представляет собой дерево.
- Деревья можно использовать для описания классификации, иерархической файловой системы, записи арифметических выражений, перебора вариантов, анализа и построения кодов.

## Интеллект-карта

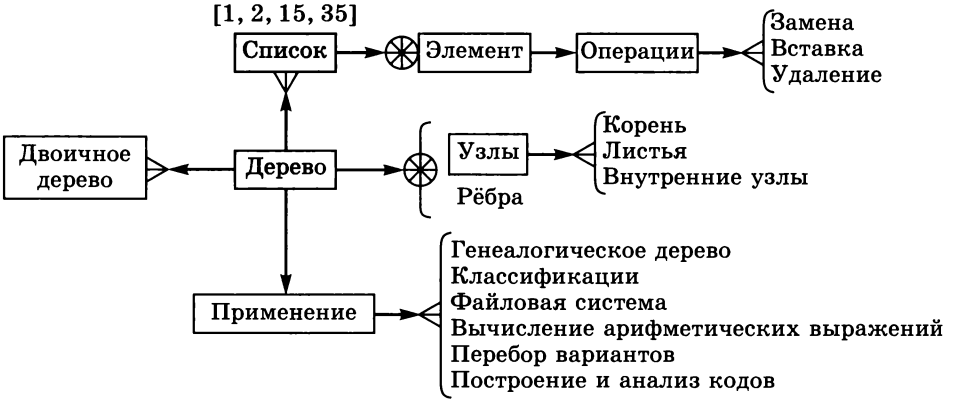



**Рис. 3.16**

## Вопросы и задания

1. Чем отличается список от множества?
2. Можно ли сказать, что список — это частный случай двоичного дерева? Почему?
3. Может ли количество листьев дерева совпадать с количеством его узлов?
4. Сколько узлов может быть в двоичном дереве высотой 2? Высотой 3? Для каждого случая назовите наибольшее и наименьшее количество узлов.
5. Сколько рёбер может быть в двоичном дереве высотой 2? Высотой 3? Для каждого случая назовите наибольшее и наименьшее количество рёбер.
6. Может ли двоичное дерево высоты 3 содержать больше узлов, чем дерево высоты 5?
7. Если для кода выполняется обратное условие Фано (ни одно кодовое слово не совпадает с окончанием другого кодового слова), то сообщение можно декодировать однозначно. Какое дерево нужно построить, чтобы убедиться в выполнении обратного условия Фано?
8. Выполните по указанию учителя задания в рабочей тетради.

# § 17
# Графы

***Ключевые слова:***

- граф
- вершина
- ребро
- матрица смежности
- степень вершины
- связный граф
- взвешенный граф
- весовая матрица
- ориентированный граф
- оптимальный путь
- количество путей

## Что такое граф?

Давайте подумаем, как можно наглядно представить такую информацию:

> От пос. Васюки три дороги идут в Солнцево, Грибное и Ягодное. Между Солнцевом и Грибным и между Грибным и Ягодным также есть дороги. Кроме того, есть дорога, которая идет из Грибного в лес и возвращается обратно в Грибное.

Нарисуйте в тетради схему дорог по этому описанию.

Можно, например, нарисовать такую схему (рис. 3.17, *а*).

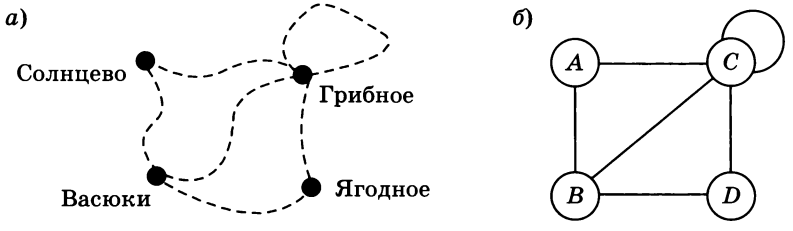



**Рис. 3.17**

В информатике для исследования таких схем используют *графы*.

---

**Граф** — это набор вершин (узлов) и связей между ними — рёбер.

---

Граф, соответствующий нашей схеме дорог, показан на рис. 3.17, *б*, для краткости населённые пункты обозначены латинскими буквами.

## Матрица смежности графа

Для хранения информации об узлах и связях показанного выше графа можно использовать таблицу такого вида (рис. 3.18).

| | *A* | *B* | *C* | *D* |
|---|---|---|---|---|
| *A* | **0** | 1 | 1 | 0 |
| *B* | 1 | **0** | 1 | 1 |
| *C* | 1 | 1 | **1** | 1 |
| *D* | 0 | 1 | 1 | **0** |

**Рис. 3.18**

Единица на пересечении строки *A* и столбца *B* означает, что между вершинами *A* и *B* есть связь. Ноль указывает на то, что связи нет. Такая таблица называется **матрицей смежности**. Она симметрична относительно главной диагонали (см. закрашенные клетки в таблице).

Исследуйте матрицу смежности и сравните её с графом на рис. 3.17, *б*. Что означает единица на пересечении столбца *C* и строки *C*?

В этом графе есть **петля** — ребро, которое начинается и заканчивается в одной и той же вершине.

**Степенью вершины** называют количество рёбер, с которыми связана вершина. При этом петля считается дважды (с вершиной связаны оба конца ребра!).

Подсчитайте по матрице смежности, сколько рёбер в графе. Определите степени всех вершин. Как вы рассуждали?

Строго говоря, *граф — это математический объект, а не рисунок*. Конечно, его можно нарисовать на плоскости (например, как на рис. 3.17, *б*), но матрица смежности не даёт никакой информации о том, как именно располагать вершины друг относительно друга. Для таблицы, приведённой выше, возможны, например, такие варианты (рис. 3.19).

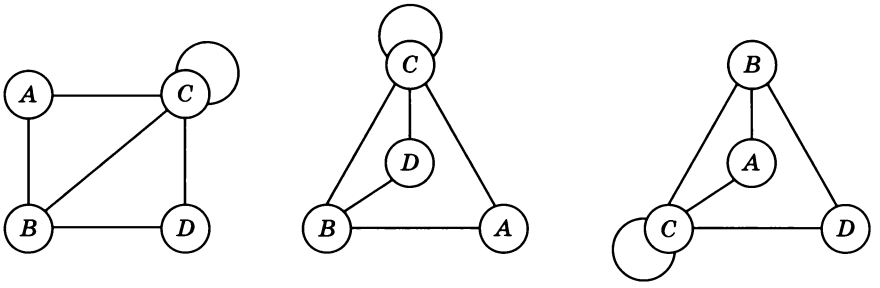



**Рис. 3.19**

Нарисуйте по матрице смежности (рис. 3.20) два разных изображения графа.

| | *A* | *B* | *C* | *D* |
|---|---|---|---|---|
| *A* | 1 | 1 | 1 | 1 |
| *B* | 1 | 0 | 0 | 0 |
| *C* | 1 | 0 | 0 | 1 |
| *D* | 1 | 0 | 1 | 0 |

**Рис. 3.20**

Граф имеет 4 вершины, причём каждая вершина связана рёбрами со всеми остальными. Нарисуйте этот граф. Сколько всего рёбер в этом графе?

Граф имеет *N* вершин, причём каждая вершина связана рёбрами со всеми остальными. Сколько всего рёбер в этом графе?

Граф имеет 4 ребра. Чему равна сумма степеней вершин в этом графе? Зависит ли она от количества вершин?

Граф имеет *N* рёбер. Чему равна сумма степеней вершин в этом графе?

Попробуйте нарисовать граф с пятью вершинами, где все вершины имеют степень 3. Получилось ли у вас? Почему?

## Связный граф

В графе на рис. 3.17, *б* все вершины связаны: между любой парой вершин существует **путь** — последовательность вершин, в которой каждая следующая связана ребром с предыдущей. Такой граф называется *связным*.

**Связный граф** — это граф, между любыми вершинами которого существует путь.

Теперь представьте себе, что дороги Васюки–Солнцево, Васюки–Грибное и Грибное–Ягодное завалило снегом (или размыло дождём) так, что по ним ни пройти, ни проехать (рис. 3.21).

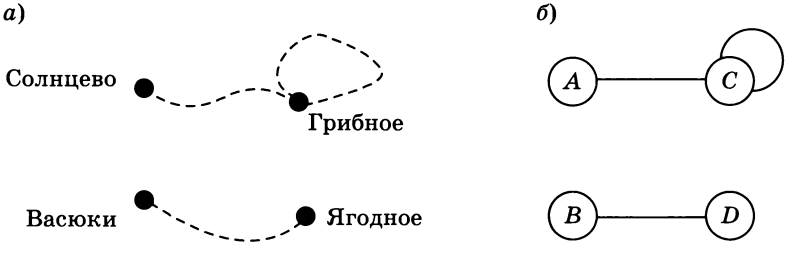



**Рис. 3.21**

Эту схему тоже можно считать графом (она соответствует определению), но в таком графе есть две несвязанные части, каждая из которых — связный граф. Такие части называют **компонентами связности.**

Постройте матрицу смежности графа, изображённого на рис. 3.21.

Граф имеет 4 вершины и две компоненты связности. Какое наибольшее количество рёбер может быть в этом графе, если в нём нет петель? Нарисуйте этот граф.

Вспоминая материал предыдущего параграфа, можно сделать вывод, что дерево — это частный случай связного графа. Но у него есть одно важное свойство — в дереве нет замкнутых путей — **циклов**, т. е. путей, которые начинаются и заканчиваются в одной и той же вершине.

Найдите все циклы в графе на рис. 3.17.

**Дерево** — это связный граф, в котором нет циклов.

## Взвешенный граф

Если в нашем примере нас заинтересует не только наличие дорог между посёлками, но ещё и расстояния между ними, каждой связи нужно сопоставить число — **вес ребра** (рис. 3.22).

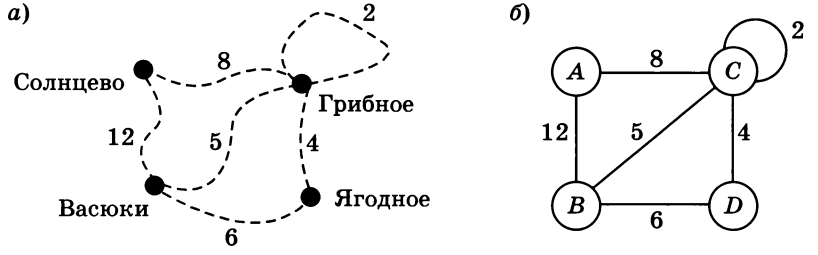



**Рис. 3.22**

**Взвешенный граф** — это граф, с каждым ребром которого связано некоторое число — вес ребра.

Весом может быть не только расстояние, но и, например, стоимость проезда или другая величина.

Как хранить информацию о таком графе? Ответ напрашивается сам собой — нужно в таблицу записывать не 1 или 0, а вес ребра. Если связи между двумя узлами нет, на бумаге можно оставить ячейку таблицы пустой, а при хранении в памяти компьютера записывать в неё условный код, например, число −1 или очень большое число. Такая таблица называется **весовой матрицей**, потому что содержит веса рёбер. В данном случае она выглядит так (рис. 3.23).

| | *A* | *B* | *C* | *D* |
|---|---|---|---|---|
| *A* | | 12 | 8 | |
| *B* | 12 | | 5 | 6 |
| *C* | 8 | 5 | 2 | 4 |
| D | | 6 | 4 | |

**Рис. 3.23**

Так же как и матрица смежности, весовая матрица симметрична относительно диагонали.

Что означают пустые ячейки в весовой матрице?

Как по весовой матрице сразу определить, сколько рёбер в графе?

Определите по весовой матрице (рис. 3.24) длины путей $ADBEC$, $ABDCE$, $DEBAC$. Как вы рассуждали?

| | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| $A$ | | 3 | 4 | 5 | |
| $B$ | 3 | | | 6 | 7 |
| $C$ | 4 | | | 2 | 5 |
| $D$ | 5 | 6 | 2 | | 4 |
| $E$ | | 7 | 5 | 4 | |

**Рис. 3.24**

## Оптимальный путь в графе

Для того чтобы определить **оптимальный** (наилучший) **путь** между двумя вершинами графа, нужно ввести какой-то числовой показатель, по которому можно сравнивать пути — определять, какой из них лучше. Обычно для оценки пути используют сумму весов ребёр, входящих в этот путь. Например, при поиске кратчайшего пути чем меньше это значение, тем лучше.

Какие показатели вы используете в жизни для определения оптимального пути? Всегда ли самый короткий путь — самый лучший?

Если в графе немного узлов, по весовой матрице можно легко определить оптимальный путь из одной вершины в другую простым перебором вариантов. Рассмотрим граф, заданный весовой матрицей на рис. 3.25 (числа определяют стоимость поездки между соседними пунктами).

| | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| $A$ | | | 3 | 1 | |
| $B$ | | | 4 | 5 | 1 |
| $C$ | 3 | 4 | | | 2 |
| $D$ | 1 | 5 | | | 1 |
| $E$ | | 1 | 2 | 1 | |

**Рис. 3.25**

Найдём наилучший путь из $A$ в $B$ — такой, при котором общая стоимость поездки минимальная.

Для решения задачи будем строить дерево перебора вариантов. Видим, что из пункта $A$ напрямую в $B$ ехать нельзя, а можно ехать только в $C$ и $D$ (рис. 3.26).

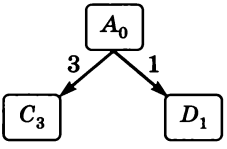



**Рис. 3.26**

Числа около рёбер обозначают стоимость поездки по этому участку, а индексы у названий узлов показывают общую стоимость проезда в данный узел из узла $A$.

Теперь разберём варианты дальнейшего движения из узла $C$ (рис. 3.27).

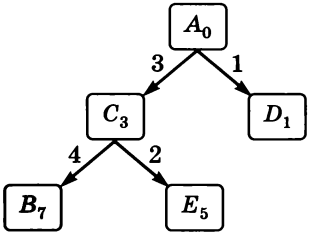



**Рис. 3.27**

Почему, на ваш взгляд, на схеме не показана возможность движения из $C$ в $A$?

Видим, что из $C$ сразу можно попасть в $B$, стоимость проезда в этом случае равна 7.

Почему нельзя на этом остановиться, ведь путь из $A$ в $B$ найден?

Проверяя пути через узел $E$, выясняем, что можно сократить стоимость до 6 (рис. 3.28).

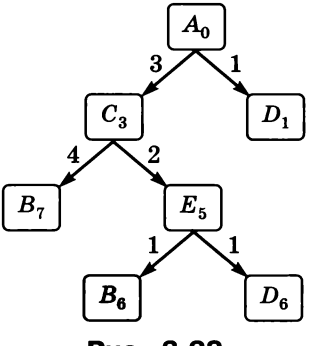



**Рис. 3.28**

Нужно ли исследовать дальше путь, содержащий цепочку $ACED$? Сравните стоимость этого пути и стоимость уже найденного пути из $A$ в $B$.

Аналогично строим вторую часть схемы (рис. 3.29).

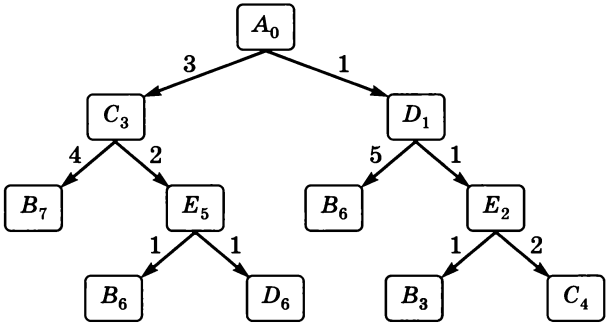



**Рис. 3.29**

Таким образом, **оптимальный (наилучший) путь** — $ADEB$, его стоимость — 3.

Нужно ли проверять пути $ACED$ и $ADEC$, не дошедшие до узла $B$? Могут ли они улучшить результат?

Конечно, для более сложных графов метод перебора работает очень долго, поэтому используются более совершенные (но значительно более сложные) методы.

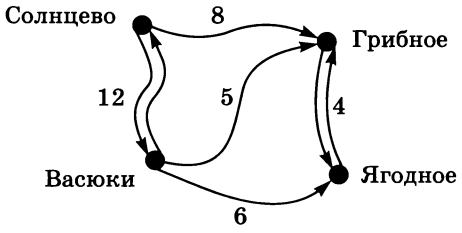
## Ориентированный граф

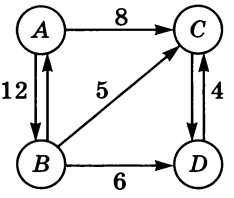
Наверное, вы заметили, что при изображении деревьев, которые описывают иерархию (подчинение), мы ставили стрелки от верхних уровней к нижним. Это означает, что для каждого ребра указывается направление, и двигаться можно только по стрелкам, но не наоборот.

**Ориентированный граф (орграф)** — это граф, в котором каждое ребро имеет направление.

Орграф может служить, например, моделью системы дорог с односторонним движением. Матрица смежности и весовая матрица для орграфа уже не обязательно будут симметричными.

На схеме на рис. 3.30 всего две дороги с двусторонним движением, по остальным можно ехать только в одну сторону.




| | A | B | C | D |
|---|---|---|---|---|
| A | | 12 | 8 | |
| B | 12 | | 5 | 6 |
| C | | | | 4 |
| D | | | 4 | |

**Рис. 3.30**

Рёбра в орграфе называют **дугами**. Дуга, в отличие от ребра, имеет начало и конец.

Нарисуйте граф по весовой матрице, показанной на рис. 3.31. С помощью дерева перебора найдите все возможные пути из вершины $A$ в вершину $E$, не проходящие дважды через одну и ту же вершину, и стоимости проезда по каждому из этих путей. Определите оптимальный путь из вершины $A$ в вершину $E$.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | | 5 | 3 | | |
| B | | | | 4 | |
| C | | 3 | | 6 | 2 |
| D | | | 4 | | 7 |
| E | | | | | |

**Рис. 3.31**

## Количество путей

Определим количество возможных путей из вершины $A$ в вершину $K$ для ориентированного графа, показанного на рис. 3.32.

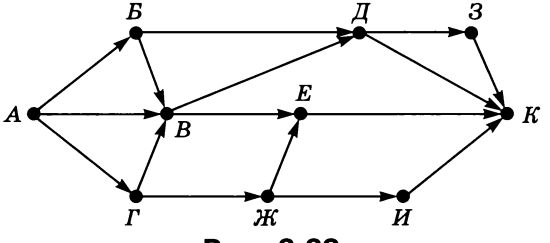



**Рис. 3.32**

Так как граф ориентированный, переходить в другую вершину можно только по стрелкам.

В графе на рис. 3.32 есть одна начальная вершина *А*, из которой дуги только выходят. Такая вершина называется **истоком**. Вершина, в которую дуги только входят (и ни одна не выходит), называется конечной вершиной или **стоком**. В нашем графе сток — это вершина *К*.

По весовой матрице на рис. 3.31 найдите исток и сток в графе. Как вы рассуждали?

Будем двигаться по стрелкам от начальной вершины *А*. Около каждой вершины запишем количество возможных путей из вершины *А* в эту вершину. В вершину *А* существует единственный путь — пустой (никуда не ехать). Найдём все вершины, в которые можно приехать только из *А*: это вершины *Б* и *Г*, записываем около них количество путей 1 (рис. 3.33).

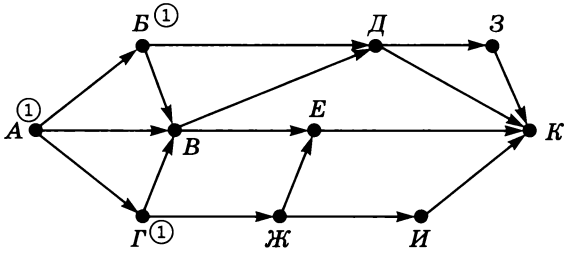



**Рис. 3.33**

Теперь ищем вершины, в которые можно попасть только из уже отмеченных вершин. Например, в вершину *В* есть один путь из *А* напрямую, а также по одному пути через *Б* и *Г* (так как эти вершины отмечены числом 1). Общее количество путей из *А* в *В* равно сумме отметок предыдущих вершин: для вершины *В* получаем 3 пути. В вершину *Ж* можно попасть только из *Г*, поэтому число путей в *Г* и *Ж* совпадает (рис. 3.34).

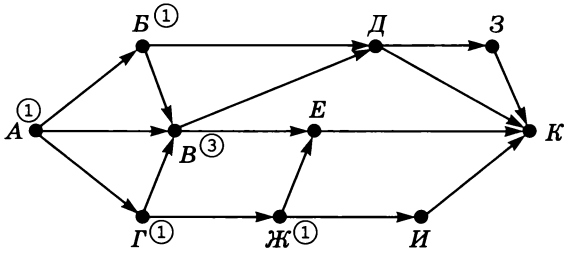



**Рис. 3.34**

В вершину *Д* идёт один путь через *Б* и три пути через *В*, поэтому общее число путей — 4. Аналогично получаем 4 пути в вершину *Е*: 3 пути через *В* и один через *Ж* (рис. 3.35).

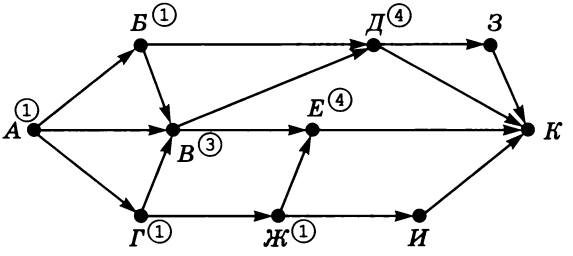



**Рис. 3.35**

Далее находим один путь из *А* в *И* (только через *Ж*) и 4 пути из *А* в *З* (все через *Д*). В конечную вершину *К* можно приехать через вершины *Д* (4 пути), *З* (4 пути), *Е* (4 пути) и *И* (1 путь), таким образом, общее количество различных путей равно 13 (рис. 3.36).

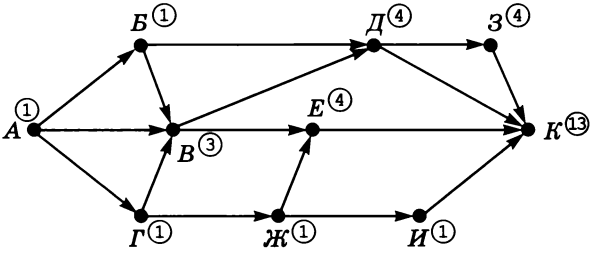



**Рис. 3.36**

## Выводы

- Граф — это набор вершин (узлов) и связей между ними — рёбер.
- Матрица смежности — это таблица, в которой единица на пересечении строки и столбца обозначает ребро между соответствующими вершинами, а ноль — отсутствие ребра.
- Связный граф — это граф, между любыми вершинами которого существует путь.
- Цикл — это замкнутый путь в графе.
- Дерево — это связный граф, в котором нет циклов.
- Взвешенный граф — это граф, с каждым ребром которого связано некоторое число — вес ребра. Взвешенный граф описывается весовой матрицей.

- Ориентированный граф (орграф) — это граф, в котором каждое ребро имеет направление. Рёбра орграфа называют дугами. Матрица смежности и весовая матрица орграфа могут быть несимметричными.

Нарисуйте в тетради интеллект-карту этого параграфа.

### Вопросы и задания

1. Можно ли сказать, что лес (множество деревьев) — это граф? Почему?
2. Как по матрице смежности определить, есть ли петли в графе?
3. Как по весовой матрице определить длину пути в графе?
4. Когда для представления данных используются орграфы? Приведите примеры.
5. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «Задача о Кёнигсбергских мостах»
б) «Решение логических задач с помощью графов»

# § 18
# Игровые стратегии

***Ключевые слова:***

- стратегия
- выигрышная позиция
- проигрышная позиция
- дерево игры
- неполное дерево игры

## Выигрышные и проигрышные позиции

Как вы уже знаете из § 16, игровые модели — это модели, которые описывают соперничество двух (или более) сторон, каждая из которых стремится к выигрышу, т. е. преследует свою цель. Часто цели участников противоречивы — выигрыш одного означает проигрыш других.

Построением и изучением игровых моделей занимается **теория игр** — раздел прикладной математики. Задача состоит в том, чтобы найти *стратегию* (алгоритм игры), который позволит тому или другому участнику получить наибольший выигрыш (или, по крайней мере,

наименьший проигрыш) в предположении, что соперники играют безошибочно.

**Стратегия** — это алгоритм игры, который позволяет добиться цели в игре в предположении, что соперники играют безошибочно.

Мы будем изучать **игры с полной информацией**, в которых результат не зависит от случая (говорят, что в таких играх нет неопределённости). Игроки делают ходы по очереди, в любой момент им известна позиция и все возможные дальнейшие ходы.

Можно ли считать играми с полной информацией «крестики-нолики», карточные игры, шахматы, шашки, «морской бой»?

Теоретически в играх с полной информацией можно построить последовательность ходов из заданной начальной позиции, которая приведёт одного из игроков к выигрышу или, по крайней мере, к ничьей. Нас будут интересовать простые игры, где не так много вариантов развития событий и перебор ходов возможен за приемлемое время. В большинстве игр, например в шахматах, количество вариантов настолько велико, что их полный перебор требует огромного времени и поэтому нереален.

Подсчитайте, сколько различных ходов могут сделать крестики в начале игры «крестики-нолики» на поле 3 × 3. Сколько различных позиций может возникнуть после ответного хода ноликов? После второго хода крестиков? После второго хода ноликов? Как можно сократить количество рассматриваемых вариантов в этой игре?

Подсчитайте, сколько различных ходов могут сделать белые в начале шахматной игры.

Все *позиции* (игровые ситуации) делятся на выигрышные и проигрышные.

**Выигрышная позиция** — это такая позиция, в которой игрок, делающий первый ход, может гарантированно выиграть при любой игре соперника, если не сделает ошибку.

При этом говорят, что у него есть **выигрышная стратегия** — алгоритм выбора очередного хода, позволяющий ему выиграть.

**Проигрышная позиция** — это такая позиция, в которой игрок, делающий первый ход, обязательно проиграет, если его соперник не сделает ошибку.

В этом случае говорят, что у него нет выигрышной стратегии. Таким образом, общая стратегия игры состоит в том, чтобы своим ходом создать проигрышную позицию для соперника.

Найдите среди позиций игры в «крестики-нолики», показанных на рис. 3.37, выигрышные и проигрышные. Во всех случаях ходят нолики.

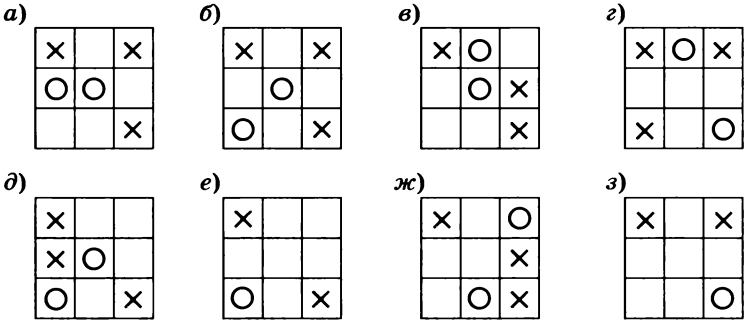



**Рис. 3.37**

Найдите все возможные ходы белых и чёрных фигур в позиции, показанной на рис. 3.38.

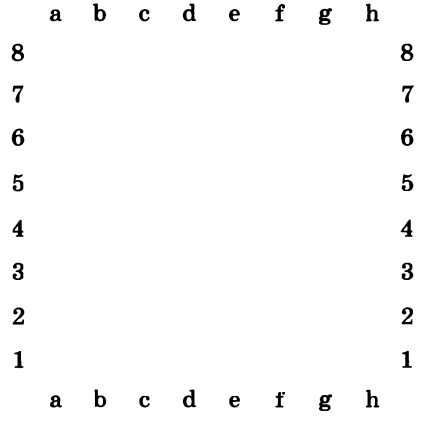



**Рис. 3.38**

Найдите выигрышный ход белых (они начинают).

Выигрышные и проигрышные позиции можно охарактеризовать так:

- позиция, из которой все возможные ходы ведут в выигрышные позиции, — *проигрышная*;
- позиция, из которой хотя бы один из возможных ходов ведёт в проигрышную позицию, — *выигрышная*, при этом выигрышная стратегия игрока состоит в том, чтобы перевести игру в эту проигрышную (для соперника) позицию.

В позициях, показанных на рис. 3.39, найдите выигрышный ход ноликов, который создаёт проигрышную (для крестиков) позицию.

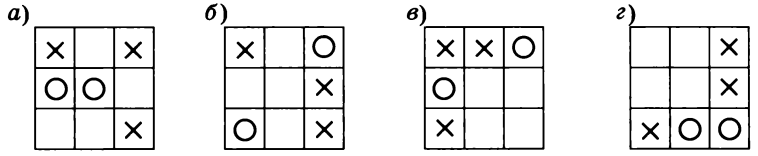



**Рис. 3.39**

Докажите, что все позиции, показанные на рис. 3.40, проигрышные для ноликов. Рассмотрите все возможные ходы и для каждого из них укажите выигрывающий ход крестиков.

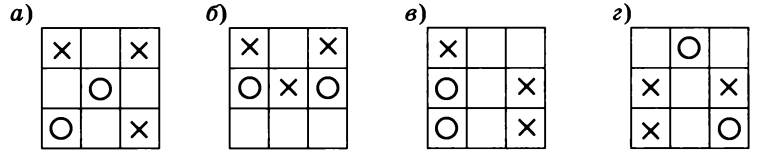



**Рис. 3.40**

## Дерево перебора вариантов

Одна из простых игр, где можно перебрать все варианты, — это игра с камнями. Перед двумя игроками лежит куча из некоторого количества камней (обозначим его $S$) или других одинаковых предметов (монет, бусин, пуговиц и т. п.) За один ход игрок может взять один или два камня. Тот, кто возьмёт последний камень, проигрывает.

Сыграйте в эту игру с соседом несколько раз для $S = 6$. Начинайте игру по очереди.

Для небольших значений $S$ легко построить дерево перебора вариантов. Для $S = 4$ такое дерево показано на рис. 3.41.

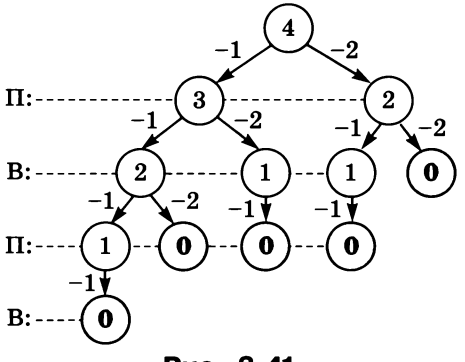



**Рис. 3.41**

Буквой П обозначены ходы первого игрока (пусть его зовут Петя), а буквой В — ходы второго игрока (будем называть его Ваней). Числа в узлах дерева показывают количество камней в куче после очередного хода, а «–1» и «–2» — взятие из кучи соответственно одного или двух камней.

Очевидно, что позиция $S = 1$ — проигрышная: тот, кто ходит в этой позиции, проигрывает, потому что берёт последний (единственный) камень. Тогда выигрышными будут все позиции, из которых можно получить $S = 1$ каким-либо ходом — это позиции $S = 2$ и $S = 3$.

Из позиции $S = 4$ все возможные ходы ведут в выигрышные позиции $S = 2$ и $S = 3$, поэтому эта позиция проигрышная. Если Ваня не сделает ошибку, то Петя в любом случае проиграет.

Используя дерево перебора вариантов, выясните, какой ход должен сделать Ваня, чтобы выиграть в позиции $S = 2$? В позиции $S = 3$?

Выигрышную стратегию можно показать с помощью *неполного* дерева игры. Дело в том, что для выигрывающего игрока нам достаточно указать на каждом ходу один-единственный вариант, переводящий игру в проигрышную (для его соперника) позицию. А вот для проигрывающего игрока на каждом ходу нужно рассматривать все варианты, чтобы доказать, что он никак не может избежать поражения. Построим неполное дерево для начальной позиции $S = 4$, в которой, как мы показали, выигрышная стратегия есть у второго игрока (рис. 3.42).

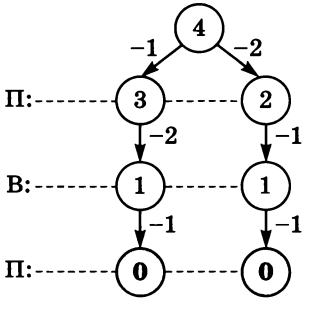



**Рис. 3.42**

По этому дереву видно, что при любом своём первом ходе Петя обязательно проигрывает на втором ходу, взяв последний камень (если, конечно, Ваня не ошибётся).

Постройте дерево перебора вариантов для игры с камнями при S = 6. Выясните, какова начальная позиция — выигрышная или проигрышная. Почему? Постройте неполное дерево игры с доказательством стратегии выигрывающего игрока.

По результатам исследований определите, при каких значениях $S$ начальные позиции будут выигрышными, а при каких — проигрышными. Если позиция выигрышная, какой стратегии должен следовать Петя?

## Решение без дерева

Для того чтобы при заданном значении $S$ определить, какова начальная позиция — выигрышная или проигрышная, не обязательно строить дерево. Составим таблицу, в которой для каждой позиции (для каждого значения $S$) будем указывать, какая это позиция и на каком ходу завершится игра. Как мы знаем, $S = 1$ — это проигрышная позиция, Петя проигрывает за один ход. Отмечаем её в таблице как $П_1$, где буква «П» означает проигрыш, а индекс «1» — количество ходов (рис. 3.43).

| S | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | $П_1$ |

**Рис. 3.43**

Выигрышными будут все позиции, из которых можно каким-либо ходом получить проигрышную позицию $S = 1$, т. е. позиции $S = 2$ и $S = 3$. Отмечаем их как $В_1$ (выигрыш за 1 ход) — рис. 3.44.

| S | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | $В_1$ | $В_1$ | $П_1$ |

**Рис. 3.44**

Из позиции $S = 4$ все возможные ходы ведут в выигрышные позиции, поэтому эта позиция — проигрышная (проигрыш за 2 хода) — рис. 3.45.

| S | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | $П_2$ | $В_1$ | $В_1$ | $П_1$ |

**Рис. 3.45**

Рассуждая таким же образом, заполните таблицу до конца.

## Исследование игры

Исследуйте самостоятельно следующую игру.

В игре участвуют два игрока, назовём их Петя (он ходит первый) и Ваня (второй). Вначале перед игроками лежит куча из некоторого количества камней (обозначим его $S$). За один ход игрок может доба-

вить в кучу один камень (ход «+1») или увеличить количество камней в куче в два раза (ход «*2»). Например, имея кучу из 5 камней, за один ход можно получить кучу из 6 или 10 камней. У каждого игрока есть неограниченное количество камней. Победителем считается игрок, первым получивший кучу, в которой 14 камней или больше.

а) Определите, при каких значениях $S$ Петя может выиграть первым ходом.

б) Определите, при каких значениях $S$ Петя не может выиграть первым ходом, а Ваня выигрывает своим первым ходом. Заполните таблицу выигрышных и проигрышных позиций для всех значений $S$ от 1 до 13.

в) Определите, при каких значениях $S$ Ваня не всегда может выиграть своим первым ходом, но у него есть стратегия, следуя которой он выигрывает своим первым или вторым ходом.

г) Постройте неполное дерево игры с доказательством стратегии выигрывающего игрока при $S = 4$.

## Выводы

- Стратегия — это алгоритм, который позволяет добиться цели в игре в предположении, что соперники играют безошибочно.
- Выигрышная позиция — это такая позиция, в которой игрок, делающий первый ход, может гарантированно выиграть при любой игре соперника, если не сделает ошибку.
- Проигрышная позиция — это такая позиция, в которой игрок, делающий первый ход, обязательно проиграет, если его соперник не сделает ошибку.
- Позиция, из которой все возможные ходы ведут в выигрышные позиции, — проигрышная.
- Позиция, из которой хотя бы один из возможных ходов ведёт в проигрышную позицию, — выигрышная, при этом выигрышная стратегия игрока состоит в том, чтобы перевести игру в эту проигрышную (для соперника) позицию.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Что такое выигрышная стратегия в игре?
2. Как доказать, что заданная позиция в игре является выигрышной (или проигрышной)? Как вы думаете, в каких случаях это сделать не удаётся?
3. Почему для того, чтобы доказать выигрыш какого-то игрока в заданной начальной позиции, не нужно строить полное дерево игры?
4. Выполните по указанию учителя задания в рабочей тетради.

## ЭОР к главе 3 из Единой коллекции цифровых образовательных ресурсов (school-collection.edu.ru)

Кибернетическая модель «Мышь в лабиринте» (видеофрагмент)
Кибернетическая модель «Черепаха» (видеофрагмент)
Демонстрационная математическая модель
Демонстрационная имитационная модель
Как быстрее спуститься на лифте в час пик
Модель «Зависимость скорости роста колонии от скорости деления клеток»
Полет шарика без сопротивления воздуха
Модель «Диффузия газов»
Компьютерное моделирование движения тела с учётом изменения $g$ («гора Ньютона»)
Видеоролик-анимация «Модель атома гелия»
Географическая модель Земли
Архангельский собор. 3D-модель
Графы — 1, 2, 3, ...

# Глава 4
# ПРОГРАММИРОВАНИЕ

## § 19
## Символьные строки

***Ключевые слова:***

- символьная строка
- длина строки
- сцепление строк
- срез строки (подстрока)
- удаление символов
- вставка символов
- поиск подстроки

### Что такое символьная строка?

В середине XX века первые компьютеры создавались, прежде всего, для выполнения сложных математических расчётов, а сейчас они чаще всего обрабатывают текстовую (символьную) информацию.

**Символьная строка** — это последовательность символов.

В алгоритмическом языке и в Паскале для работы со строками используются специальные типы данных, которые позволяют:

- работать с целой символьной строкой как с единым объектом;
- использовать строки переменной длины.

Такой тип данных в алгоритмическом языке называется **литерным** и обозначается **лит**, а в Паскале называется **строковым** и обозначается `string`.

Используя дополнительные источники, выясните, что означают слова «литера» и «литерный».

Используя дополнительные источники, выясните значение английского слова *string*.

Вот пример объявления строки:

```
лит s                          var s: string;
```

Новое значение записывается в строку с помощью оператора присваивания:

```
s:='Вася пошёл гулять'         s:='Вася пошёл гулять';
```

или оператора ввода с клавиатуры:

```
ввод s                         readln(s);
```

Обратите внимание, что при вводе строк в Паскале нужно использовать оператор readln (англ. *read line* — читать до конца строки) вместо read.

Существуют стандартные функции, которые определяют длину строки (количество символов в ней). В алгоритмическом языке такая функция называется длин, а в Паскале — length (в переводе с англ. — длина). В этом примере в целочисленную переменную *n* записывается длина строки s:

```
n:=длин(s)                     n:=length(s);
```

Напишите полную программу, которая вводит строку с клавиатуры и выводит на экран её длину. Проверьте, как эта программа реагирует на строку с пробелами.

## Сравнение строк

Строки можно сравнивать между собой так же, как числа. Например, можно проверить равенство двух строк:

```
ввод s
если s='sEzAm' то
  вывод 'Слушаюсь и повинуюсь!'
иначе
  вывод 'Пароль неправильный'
все
```

Та же программа на языке Паскаль:

```
readln(s);
if s='sEzAm' then
  write('Слушаюсь и повинуюсь!')
else
  write('Пароль неправильный');
```

Запишите в тетради, как нужно объявить в этой программе переменную s.

Можно также определить, какая из двух строк больше, какая — меньше. Если строки состоят только из русских или толь-

ко из латинских букв, то меньше будет та строка, которая идёт раньше в алфавитном порядке. Например, слово «паровоз» будет меньше, чем слово «пароход»: они отличаются в пятой букве и «в» < «х». Это можно проверить экспериментально, например, с помощью такой программы:

```
s1:='паровоз'
s2:='пароход'
если s1<s2 то
  вывод s1, '<', s2
иначе
  если s1=s2 то
    вывод s1, '=', s2
  иначе
    вывод s1, '>', s2
  все
все
```

```
s1:='паровоз';
s2:='пароход';
if s1<s2 then
  write(s1, '<', s2)
else
  if s1=s2 then
    write(s1, '=', s2)
  else
    write(s1, '>', s2);
```

Но откуда компьютер «знает», что такое алфавитный порядок? Оказывается, при сравнении используются коды символов (вспомните материал учебника для 8 класса). В современных кодировках и русские, и английские буквы расположены в алфавитном порядке, т. е. код буквы «в» меньше, чем код буквы «х».

С помощью программы сравните пары слов и сделайте выводы:

пар – парк  Пар – пар  steam – Пар
Steam – steam  5Steam – Steam

Не используя программу, сравните пары слов:

парта – парк  ПАрта – Парк  СПАМ – Spam
ПОЧТА – spam  ПО4та – ПОЧта  почТА – Post
55 – 66  9 – 128

## Посимвольная обработка строк

Для того чтобы работать с отдельными символами строки, к ним обращаются так же, как к элементам массива: в квадратных скобках записывают номер нужного символа. Например, так можно изменить четвёртый символ строки на «а» (конечно, длина строки должна быть не менее четырёх символов):

```
s[4]:= 'a'
```

```
s[4]:='a';
```

Приведём программу, которая вводит строку с клавиатуры, заменяет в ней все буквы «э» на буквы «е» и выводит полученную строку на экран:

```
алг Замена э на е
нач
  лит s
  вывод 'Введите строку: '
  ввод s
  цел i
  нц для i от 1 до длин(s)
    если s[i]='э'
      то s[i]:='е'
    все
  кц
  вывод s
кон
```

```
program Replace;
var s: string;
    i: integer;
begin
   writeln('Введите строку');
   readln(s);
   for i:=1 to length(s) do
      if s[i]='э' then
         s[i]:='е';
   writeln(s)
end.
```

В цикле

```
нц для i от 1 до длин(s)
  ...
кц
```

```
for i:=1 to length(s) do
  ...
```

мы перебираем все символы строки с первого до последнего, и если очередной символ — буква «э», делаем замену:

```
если s[i]='э'
  то s[i]:='е'
все
```

```
if s[i]='э' then
   s[i]:='е';
```

Вспомните, чем отличается запись `s='е'` от записи `s:='е'`.

Запишите решение этой задачи, используя цикл **пока** (**while**).

## Операции со строками

Оператор **+** используется для «сложения» (**объединения, сцепления**) строк, эта операция иногда называется **конкатенацией**. Например:

```
s1:='Привет'
s2:='Вася'
s:=s1 + ', ' + s2 + '!'
```

Здесь и далее считаем, что в программе объявлены строковые (литерные) переменные `s`, `s1` и `s2`.

Запишите в тетради, какое значение будет иметь переменная *s* после выполнения этого фрагмента программы. Проверьте ответ с помощью компьютера.

Для обработки строк обычно используют готовые вспомогательные алгоритмы из библиотеки языка программирования — процедуры и функции. Различие между ними состоит в том, что процедура изменяет переданную ей строку, а функция возвращает результат — новое значение, — не изменяя исходную строку.

Для того чтобы **выделить часть строки (подстроку)**, в алгоритмическом языке применяется операция получения среза (англ. *slicing*). Например, `s[3:7]` означает «символы строки *s* с 3-го по 7-й включительно». В Паскале для этого используется функция `copy`, она принимает три параметра: имя строки, номер начального символа и *количество символов*. Оба следующих фрагмента копируют в строку `s1` символы строки `s` с 3-го по 7-й (всего 5 символов):

```
s:='123456789'
s1:=s[3:7]
```

```
s:='123456789';
s1:=copy(s,3,5);
```

Запишите в тетради, какое значение будет иметь переменная `s1` после выполнения этого фрагмента программы. Проверьте ответ с помощью компьютера.

Для **удаления части строки** нужно вызвать стандартную процедуру, указав имя строки, номер начального символа и число удаляемых символов:

```
s:='123456789'
удалить(s, 3, 6)
```

```
s:='123456789';
delete(s, 3, 6);
```

Запишите в тетради, какое значение будет иметь переменная `s` после выполнения этого фрагмента программы. Проверьте ответ с помощью компьютера.

При вставке символов процедуре передают вставляемый фрагмент, имя исходной строки и номер символа, с которого начинается вставка:

```
s:= '123456789'
вставить('ABC', s, 3)
```

```
s:= '123456789';
insert('ABC', s, 3);
```

Запишите в тетради, какое значение будет иметь переменная `s` после выполнения этого фрагмента программы. Проверьте ответ с помощью компьютера.

Используя только операции выделения подстроки и «сложения» строк, постройте из строки

```
s:='информатика'
```

как можно больше слов русского языка. Постарайтесь использовать наименьшее возможное число операций. Проверьте ваши решения с помощью программы.

Приведите несколько способов построения строки

```
'А. Семёнов'
```

из строки

```
s:='Семёнов Андрей'
```

Какой из них лучше? Как вы сравнивали эти способы?

## Поиск в символьных строках

Существуют функции для **поиска подстроки** (и отдельного символа) в строке. Им нужно передать образец для поиска и строку, в которой надо искать:

```
s:='Здесь был Вася.'
n:=позиция('с', s)
если n>0 то
  вывод 'Номер символа ', n
иначе
  вывод 'Символ не найден.'
все
```

```
s:='Здесь был Вася.';
n:=pos('с', s);
if n>0 then
  write('Номер символа ', n)
else
  write('Символ не найден.');
```

Функция `позиция` возвращает целое число — номер символа, с которого начинается образец (буква «с») в строке `s`. Если образец встречается в строке несколько раз, функция находит первый из них. В языке Паскаль функция `pos` (от англ. *position* — позиция, расположение) работает точно так же.

Выясните экспериментально, какое значение возвращает функция `позиция (pos)`, если образец для поиска не найден в строке.

Как можно найти вторую букву «с» с начала строки?

Вводится строка, в которой сначала записана фамилия человека, а затем через пробел — его имя, например `'Семёнов Андрей'`. Запишите операторы, которые позволяют:

а) найти номер пробела, разделяющего фамилию и имя, и записать его в переменную `p`;
б) выделить из строки фамилию и записать её в переменную `fam`;
в) выделить из строки имя и записать его в переменную `name`;
г) приписать перед фамилией первую букву имени, точку и пробел.

## Преобразования «строка ↔ число»

Иногда символьная строка, которая передаётся программе, содержит запись числа. С таким значением нельзя выполнять арифметические операции, потому что это символы, а не число.

Чему будут равны значения переменных n и s после выполнения этих команд? Как нужно объявить эти переменные в программе?

```
n:= 12 + 34;
s:='12' + '34';
```

Для того чтобы с данными можно было выполнять вычисления, нужно преобразовать число, записанное в виде цепочки символов, в числовое значение. Для этого в алгоритмическом языке есть стандартные функции:

**лит_в_цел** — переводит строку в целое число;
**лит_в_вещ** — переводит строку в вещественное число.

Разберём такой пример:

```
лит s, цел N, лог OK
s:='123'
N:= лит_в_цел(s, OK)      | N = 123
если не OK то
  вывод 'Ошибка!'
все
```

Строку не всегда можно преобразовать в число (например, если в ней содержатся буквы). Поэтому функция лит_в_цел использует второй параметр — логическую переменную OK. Функция записывает в эту переменную логическое значение да («истина»), если операция закончилась успешно, и нет (ложь), если произошла ошибка.

Изучите приведённый фрагмент программы и выясните, как объявляется логическая переменная.

А вот пример использования функции лит_в_вещ:

```
лит s, вещ X, лог OK
s:='123.456';
X:=лит_в_вещ(s, OK)      | X = 123.456
если не OK то
  вывод 'Ошибка!'
все
```

Какие из этих строк можно преобразовать в целое число, какие — в вещественное?

а) '45'; б) '5 р.'; в) '14.5'; г) '14;5'; д) 'tu154'; е) '543.0'; ж) '(30)'.

Обратное преобразование (из числа в строку) возможно всегда:

```
N:=123
s:=цел_в_лит(N)    | s='123'
X:=123.456
s:=вещ_в_лит(X)    | s='123.456'
```

Изучите приведённый фрагмент программы и выясните, как называются функции для преобразования целого числа и вещественного числа в символьную строку.

В языке Паскаль строка преобразуется в число (целое или вещественное) с помощью процедуры `val`:

```
var r: integer;
...
s:='123';
val(s, N, r); {N=123}
s:='123.456';
val(s, X, r); {X=123.456}
```

Третий параметр `r` служит для того, чтобы определить, была ли ошибка. Если после вызова процедуры `val` значение `r` равно нулю, то ошибки не было, иначе в переменную `r` записывается номер первого ошибочного символа.

Преобразование числа в строку выполняет процедура `str`:

```
n:=123;
str(N, s);       {s='123'}
x:=123.456;
str(X, s);       {s='1.234560E+002'}
str(X:10:3, s); {s='   123.456'}
```

По умолчанию вещественные числа записываются в научном формате (**'1.234560E+002'** означает $1{,}23456 \cdot 10^2$). В последней строке примера используется форматный вывод: запись **X:10:3** означает «вывести число в 10 позициях с тремя знаками в дробной части».

## Выводы

- Символьная строка — это последовательность символов.
- Длина строки — это количество символов в строке.
- Подстрока — это часть символьной строки.
- При обращении к отдельному символу строки его номер записывают в квадратных скобках.
- Знак «+» при работе со строками означает объединение строк.
- Для обработки символьных строк используют вспомогательные алгоритмы стандартной библиотеки — процедуры и функции. Процедура изменяет переданную ей строку, а функция возвращает результат — новое значение, не изменяя исходную строку.
- Функции поиска подстроки возвращают номер символа, с которого начинается подстрока, или 0 в случае неудачи.
- Строку можно преобразовать в число для того, чтобы затем выполнять с ним вычисления. Число можно преобразовать в символьную строку.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Во многих языках программирования можно использовать массивы символов, т. е. массивы, каждый элемент которых — один символ. Чем отличается строка от массива символов?
2. Чем отличается действие оператора + для чисел и для символьных строк?
3. Можно ли обойтись без стандартной функции для вставки подстроки? Если да, то чем её можно заменить?
4. Как определить, что при поиске в строке образец не найден?
5. Как бы вы искали первый символ «с» с конца строки?
6. Выполните по указанию учителя задания в рабочей тетради.

## Практические работы

Выполните практические работы:

№ 12 «Посимвольная обработка строк»;

№ 13 «Обработка строк. Функции»;

№ 14 «Преобразования "строка ↔ число"».

# § 20
# Обработка массивов

***Ключевые слова:***

- массив
- перестановка элементов
- вспомогательная переменная
- выход за границы массива
- линейный поиск
- сортировка

Современные компьютеры работают с огромными массивами данных. При этом размер программы не зависит от размера массива: можно написать очень короткую программу, которая обрабатывает миллиарды чисел.

Из курса 8 класса вы уже знаете, как найти сумму элементов массива и его максимальный (минимальный) элемент, определить количество элементов, удовлетворяющих условию. В этом параграфе мы изучим некоторые более сложные алгоритмы.

Вспомните и запишите в тетрадь, как:

- объявить целочисленный массив из *N* элементов;
- заполнить массив нулями;
- заполнить массив натуральными числами от 1 до *N*;
- заполнить массив случайными числами в диапазоне [50, 100];
- найти сумму значений всех элементов массива;
- найти сумму значений чётных элементов массива;
- найти количество отрицательных элементов массива;
- найти максимальный элемент массива.

## Перестановка элементов массива

Во многих задачах нужно **переставлять элементы массива**, т. е. требуется менять местами значения двух ячеек памяти.

Представьте себе, что в кофейной чашке налит сок, а в стакане — кофе, и вы хотите, чтобы было наоборот. Что вы сделаете?

Вернёмся к программированию. Чтобы поменять местами значения двух переменных — `a` и `b`, нужно использовать третью переменную того же типа:

```
c:=a
a:=b
b:=c
```

```
c:=a;
a:=b;
b:=c;
```

Перестановка двух элементов массива, например A[i] и A[k], выполняется так же:

```
c:=A[i]                    c:=A[i];
A[i]:=A[k]                 A[i]:=A[k];
A[k]:=c                    A[k]:=c;
```

Теперь рассмотрим несколько задач, в которых требуется переставить элементы массива в другом порядке.

**Задача 1.** Массив A содержит чётное количество элементов N. Нужно поменять местами пары соседних элементов: первый со вторым, третий — с четвёртым и т. д. (рис. 4.1).

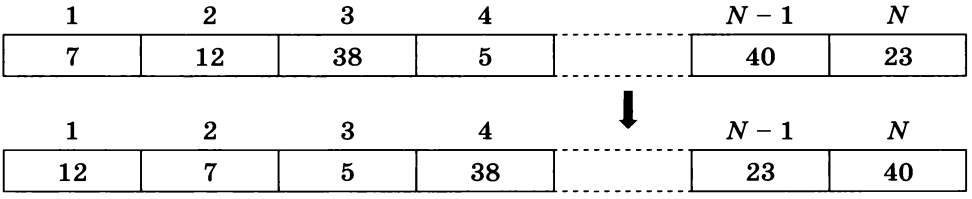



**Рис. 4.1**

С каким элементом нужно поменять местами элемент A[i]? Зависит ли ваш ответ от свойств числа i?

Сергей написал такой алгоритм для решения задачи 1:

```
нц для i от 1 до N
  поменять местами A[i] и A[i+1]
кц
```

Выполните его вручную для массива {1, 2, 3, 4} (N = 4). Объясните результат. Найдите ошибки в этом алгоритме.

Обратите внимание, что на последнем шаге цикла мы будем менять местами элемент A[N] с несуществующим элементом A[N+1]. Это вызовет ошибку, которая называется **выход за границы массива**. Но даже если изменить наибольшее значение переменной цикла на N−1, программа всё равно будет работать неправильно, т. е. приведёт к неверному решению задачи 1. Получится, что первый элемент просто «переедет» на место последнего.

Для того чтобы исправить программу, нужно изменять переменную i с шагом 2:

```
нц для i от 1 до N-1 шаг 2
  c:=A[i]
  A[i]:=A[i+1]
  A[i+1]:=c
кц
```

```
i:=1;
while i<N do begin
  c:=A[i];
  A[i]:=A[i+1];
  A[i+1]:=c;
  i:=i+2
end;
```

Решение на языке Паскаль получилось немного более сложным, потому что в этом языке можно изменять переменную цикла только с шагом 1 или −1, а нам нужен шаг 2. В этой ситуации можно, например, использовать цикл с условием (цикл **while**).

Предложите другое решение этой задачи, записав нужные операторы в теле цикла.

```
нц для i от 2 до N шаг 2
  ...
кц
```

```
i:=2;
while i<=N do begin
  ...
end;
```

## Реверс массива

**Задача 2.** Требуется выполнить **реверс массива**, т. е. переставить элементы массива в обратном порядке, так чтобы первый элемент стал последним, а последний — первым (рис. 4.2).

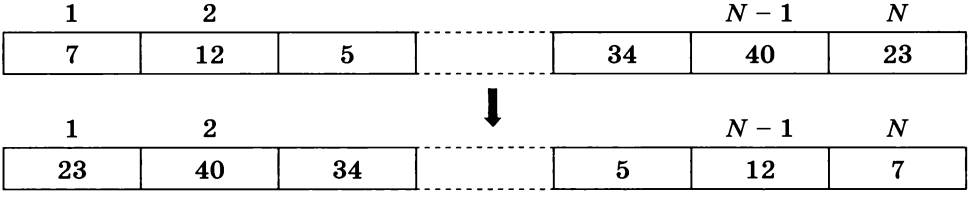



**Рис. 4.2**

С каким элементом нужно поменять местами элемент A[1]? Элемент A[2]? Элемент A[i]?

Если индекс первого элемента в паре увеличивается на 1, а индекс второго элемента уменьшается на 1, что можно сказать о сумме индексов этих элементов?

**В этой задаче сумма индексов элементов, участвующих в обмене, для всех пар равна N+1, поэтому элемент с номером *i* должен меняться местами с (N+1–i)-м элементом.**

Выполните вручную следующий алгоритм для массива {1, 2, 3, 4} (N = 4). Объясните результат. Найдите ошибки в этом алгоритме (почему он не решает задачу 2?).

```
нц для i от 1 до N
  поменять местами A[i] и A[N+1-i]
кц
```

**Для того чтобы не выполнять реверс дважды, нужно остановить цикл на середине массива:**

```
нц для i от 1 до  div(N, 2)
  поменять местами A[i] и A[N+1-i]
кц
```

Запишите в тетради операторы, которые нужно добавить в тело цикла. Для обмена используйте вспомогательную переменную **c**.

```
нц для i от 1 до div(N, 2)
  ...
кц
```

```
for i:=1 to N div 2 do begin
  ...
end;
```

Запишите в тетради другое решение задачи реверса массива, которое использует цикл с условием (**пока**, **while**).

## Линейный поиск в массиве

Очень часто требуется найти в массиве заданное значение или определить, что его там нет. Для этого нужно просмотреть все элементы массива с первого до последнего. Как только будет найден элемент, равный заданному значению X, надо завершить цикл и вывести результат. Такой **поиск** называется **линейным**.

Сколько операций сравнения придётся выполнить, если в массиве N элементов? В каком случае количество сравнений будет наименьшим? Наибольшим?

Катя торопилась и написала такой алгоритм линейного поиска:

```
i:=1
нц пока A[i]<>X
  i:=i+1
кц
вывод 'A[',i,']=',X
```

Проверьте: правильно ли сработает алгоритм, если искать в массиве {1, 2, 3} число 2? Число 4?

Этот алгоритм хорошо работает, если нужный элемент в массиве есть, однако приведёт к ошибке, если такого элемента нет, — получится зацикливание и выход за границы массива. Чтобы этого не произошло, в условие нужно добавить ещё одно ограничение: `i <= N`. Если после окончания цикла это условие нарушено, это означает, что поиск был неудачным — элемента нет:

```
i:=1
нц пока i<=N и A[i]<>X
  i:=i+1
кц
если i<=N то
  вывод 'A[',i,']=',X
иначе
  вывод 'Не нашли!'
все
```

```
i:=1;
while (i<=N) and (A[i]<>X) do
  i:=i+1;
if i<=N then
  write('A[', i, ']=', X)
else
  write('Не нашли!');
```

В сложном условии `i <= N и A[i]<> X` первым должно проверяться именно отношение `i <= N`. Если первая часть условия, соединённого с помощью операции И, ложна, то вторая часть не вычисляется — уже понятно, что всё условие ложно. Дело в том, что, если `i > N`, проверка условия `A[i] <> X` приводит к **выходу за границы массива**, и программа может завершиться аварийно.

Возможен ещё один поход к решению этой задачи: используя цикл с переменной, перебрать все элементы массива и досрочно завершить цикл, если найдено требуемое значение:

```
nX:=0
нц для i от 1 до N
  если A[i]=X то
    nX:=i
    выход
  все
кц
если nX>0 то
  вывод 'A[', nX, ']=', X
иначе
  вывод 'Не нашли!'
все
```

```
nX:=0;
for i:=1 to N do
  if A[i]=X then begin
    nX:=i;
    break
  end;
if nX>0 then
  write('A[', nX, ']=', X)
else
  write('Не нашли!');
```

Для выхода из цикла используется оператор `выход` (в Паскале — `break`), номер найденного элемента сохраняется в переменной *nX*. Если её значение осталось равным нулю (не изменилось в ходе выполнения цикла), то в массиве нет элемента, равного *X*.

## Сортировка массивов

**Сортировка** — это расстановка элементов списка (массива) в заданном порядке.

Возникает естественный вопрос: зачем сортировать данные? На него легко ответить, вспомнив, например, работу со словарями: сортировка слов по алфавиту облегчает поиск нужной информации.

Для чисел обычно используют сортировку по **возрастанию** (каждый следующий элемент больше предыдущего) или **убыванию** (следующий элемент меньше предыдущего). Если в массиве есть одинаковые элементы, их можно расставить по **неубыванию** (каждый следующий элемент не меньше предыдущего) или **невозрастанию**.

Математики и программисты изобрели множество способов сортировки. В целом их можно разделить на две группы: 1) простые, но медленно работающие (на больших массивах), и 2) сложные, но быстрые.

Мы изучим один из простых методов, который называется **методом выбора**. Для примера будем рассматривать сортировку по возрастанию (неубыванию).

Предположим, что мы нашли минимальный элемент массива. Где он должен размещаться в отсортированном массиве?

Запишите в тетради фрагмент программы для поиска номера минимального элемента массива.

Для того чтобы поставить на место минимальный элемент, мы просто поменяем его местами с первым элементом массива.

Запишите в тетради фрагмент программы, который меняет местами элементы `A[1]` и `A[nMin]`. Используйте вспомогательную переменную `c`.

После того как мы установили на место первый (минимальный) элемент, находим минимальный из оставшихся и ставим его

на второе место и т. д. Полный алгоритм записывается в виде вложенного цикла:

```
нц для i от 1 до N-1
    | ищем минимальный среди A[i]..A[N]
  nMin:=i
  нц для j от i+1 до N
    если A[j]<A[nMin] то
      nMin:=j
    все
  кц
    | переставляем A[i] и A[nMin]
  c:=A[i]
  A[i]:=A[nMin]
  A[nMin]:=c
кц
```

На первом шаге внешнего цикла значение i равно 1, и мы ставим на первое место минимальный элемент. На следующем шаге i = 2, мы ищем минимальный элемент среди всех, кроме первого, и т. д.

Обратите внимание, что внешний цикл выполняется N-1 раз (а не N). Этого достаточно, потому что если N-1 элементов стоят на своих местах, то последний тоже стоит на своём месте (другого свободного места для него нет).

Решение на языке Паскаль выглядит так:

```
for i:=1 to N-1 do begin
  nMin:=i;
  for j:=i+1 to N do
    if A[j]<A[nMin] then nMin:=j;
  c:=A[i];
  A[i]:=A[nMin];
  A[nMin]:=c;
end;
```

Что получится, если внешний цикл (по переменной i) выполнить только 1 раз? Только 3 раза?

## Выводы

- Поменять местами значения двух элементов массива можно с помощью вспомогательной переменной.
- Выход за границы массива — это обращение к элементу массива с несуществующим индексом.

- Линейный поиск — это перебор всех элементов массива до тех пор, пока не будет найден нужный элемент или не закончится массив.
- Сортировка — это расстановка элементов списка (массива) в заданном порядке. Для чисел обычно используют сортировку по возрастанию (неубыванию) или убыванию (невозрастанию). Данные сортируют для того, чтобы ускорить последующий поиск.

Нарисуйте в тетради интеллект-карту этого параграфа.

### Вопросы и задания

1. Что такое выход за границы массива? Что опаснее — чтение или запись данных за границами массива?
2. На какой идее основан метод сортировки выбором?
3. Объясните, зачем нужен вложенный цикл в алгоритме сортировки.
4. Как нужно изменить программу сортировки, чтобы элементы массива были отсортированы по убыванию?
5. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «Сортировка методом пузырька»
б) «Сортировка методом вставки»

### Практические работы

Выполните практические работы:
№ 15 «Перестановка элементов массива»;
№ 16 «Линейный поиск в массиве»;
№ 17 «Сортировка».

## § 21
## Матрицы (двумерные массивы)

***Ключевые слова:***

- матрица
- строка
- столбец
- главная диагональ
- побочная диагональ

## Что такое матрицы?

Многие программы работают с данными, организованными в виде таблиц. Например, при составлении программы для игры в крестики-нолики нужно запоминать состояние каждой клетки квадратной доски Можно поступить так: пустым клеткам присвоить код –1, клетке, где стоит нолик, — код 0, а клетке с крестиком — код 1. Тогда информация о состоянии поля может быть записана в виде таблицы (рис. 4.3).

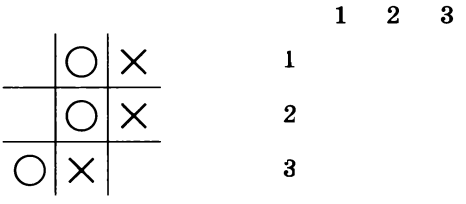



**Рис. 4.3**

Такие таблицы называются матрицами или **двумерными массивами**.

---

**Матрица** — это прямоугольная таблица, составленная из элементов одного типа (чисел, строк и т. д.).

---

Каждый элемент матрицы, в отличие от линейного массива, имеет два индекса — номер строки и номер столбца. На рисунке 4.3 серым фоном выделен элемент, находящийся на пересечении второй строки и третьего столбца — `A[2,3]`.

Определите значения элементов `A[1,2]`, `A[2,1]` и `A[3,3]` на рис. 4.3.

Матрицу часто называют двумерным массивом, потому что каждый элемент матрицы имеет два индекса — номера строки и столбца.

При объявлении матриц указывают два диапазона индексов (для строк и столбцов):

```
цел N=3, M=4
целтаб A[1:N,1:M]
```

```
const N=3; M=4;
var A: array[1..N,1..M]
       of integer;
```

Каждому элементу матрицы можно присвоить любое значение, допустимое для выбранного типа данных.

Поскольку индексов два, для заполнения матрицы нужно использовать вложенный цикл. Далее в примерах будем считать, что объявлена матрица из `N` строк и `M` столбцов, а `i` и `j` — целочисленные переменные, обозначающие индексы строки и столбца. В этом примере матрица заполняется случайными числами и выводится на экран:

```
нц для i от 1 до N
  нц для j от 1 до M
    A[i,j]:=irand(20, 80)
    вывод A[i,j]:3
  кц
  вывод нс
кц
```

```
for i:=1 to N do begin
  for j:=1 to M do begin
    A[i,j]:=random(61)+20;
    write(A[i,j]:3)
  end;
  writeln
end;
```

Такой же двойной цикл нужно использовать для перебора всех элементов матрицы.

Запишите в тетради фрагмент программы, который вычисляет сумму всех элементов матрицы в переменной s.

Запишите в тетради фрагмент программы, который вычисляет количество ненулевых элементов матрицы в переменной k.

## Обработка элементов матрицы

Покажем, как можно обработать (например, сложить) некоторые элементы квадратной матрицы A, содержащей N строк и N столбцов.

Для квадратной матрицы используют понятия **«главная диагональ»** (серые клетки на рис. 4.4, *а*) и **«побочная диагональ»** (рис. 4.4, *б*). На рис. 4.4, *в* выделены главная диагональ и все элементы под ней.

*а)* *б)* *в)*

**Рис. 4.4**

Главная диагональ — это элементы A[1,1], A[2,2], ..., A[N,N], т. е. элементы, у которых номер строки равен номеру столбца. Для перебора этих элементов нужен один цикл:

```
нц для i от 1 до N
  | работаем с
кц
```

```
for i:=1 to N do
  {работаем с        }
end;
```

Элементы побочной диагонали — это A[1,N], A[2,N-1], ..., A[N,1]. Заметим, что сумма номеров строки и столбца для каждого элемента равна N+1, поэтому получаем такой цикл перебора:

```
нц для i от 1 до N
  | работаем с  A[i,N+1-i]
кц
```

```
for i:=1 to N do begin
  {работаем с  A[i,N+1-i]}
end;
```

Запишите в тетради фрагмент программы, который вычисляет сумму всех элементов главной диагонали квадратной матрицы в переменной *s*.

Запишите в тетради фрагмент программы, который вычисляет количество ненулевых элементов побочной диагонали матрицы в переменной *k*.

В случае *в*) (обработка всех элементов на главной диагонали и под ней) нужен вложенный цикл: номер строки будет меняться от 1 до *N*, а номер столбца для каждой строки *i* — от 1 до *i*:

```
нц для i от 1 до N
  нц для j от 1 до  i
    | работаем с  A[i,j]
  кц
кц
```

```
for i:=1 to N do
  for j:=1 to  i  do begin
    {работаем с  A[i,j]}
  end
```

Запишите в тетради фрагмент программы, который вычисляет среднее арифметическое элементов матрицы, находящихся на главной диагонали и под ней.

Чтобы переставить строки или столбцы, достаточно одного цикла. Например, переставим строки 2 и 4, используя вспомогательную целую переменную c:

```
нц для j от 1 до M
  c:=A[2,j]
  A[2,j]:=A[4,j]
  A[4,j]:=c
кц
```

```
for j:=1 to M do
  c:=A[2,j];
  A[2,j]:=A[4,j];
  A[4,j]:=c;
end;
```

Запишите в тетради фрагмент программы, который переставляет столбцы матрицы с индексами 3 и 4.

## Выводы

- Матрица — это прямоугольная таблица, составленная из элементов одного типа (чисел, строк и т. д.).
- Каждый элемент матрицы имеет два индекса — номера строки и столбца.

- Главная диагональ квадратной матрицы — это элементы, у которых индекс строки равен индексу столбца.

Нарисуйте в тетради интеллект-карту этого параграфа.

### Вопросы и задания

1. Сравните понятия «массив» и «матрица».
2. Как вы думаете, можно ли считать, что первый индекс элемента матрицы — это номер столбца, а второй — номер строки?
3. Почему суммирование элементов главной диагонали требует одиночного цикла, а суммирование элементов под главной диагональю — вложенного?
4. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

«Игра "Жизнь"»

### Практическая работа

Выполните практическую работу № 18 «Матрицы».

## § 22
## Сложность алгоритмов

***Ключевые слова:***

- временна́я сложность
- пространственная сложность
- время работы алгоритма
- асимптотическая сложность
- линейная сложность
- квадратичная сложность

### Как сравнивать алгоритмы?

Большинство задач могут быть решены по-разному, с помощью разных алгоритмов. Поэтому возникает вопрос: как выбрать лучший? И ещё один важный вопрос — способен ли современный компьютер за приемлемое время найти решение задачи? Например, в игре в шахматы возможно лишь конечное количество позиций и, следовательно, только конечное количество различных партий. Значит, теоретически можно перебрать все возможные партии и выяснить, кто побеждает при правильной игре — белые или чёрные. Однако количество вариантов настолько велико, что

современные компьютеры не могут выполнить такой перебор за приемлемое время.

Что мы хотим от алгоритма? Во-первых, чтобы он работал как можно быстрее. Во-вторых, чтобы объём необходимой памяти был как можно меньше. В-третьих, чтобы он был как можно более прост и понятен, что позволяет легче отлаживать программу. К сожалению, эти требования противоречивы, и в серьёзных задачах редко удаётся найти алгоритм, который был бы лучше остальных по всем показателям.

Часто говорят о временно́й сложности алгоритма (быстродействии) и пространственной сложности, которая определяется объёмом необходимой памяти.

---

**Временем работы алгоритма** называется количество элементарных операций $T$, выполненных исполнителем.

---

Такой подход позволяет оценивать именно качество алгоритма, а не свойства исполнителя (например, быстродействие компьютера, на котором выполняется алгоритм). При этом мы считаем, что время выполнения всех элементарных операций одинаково.

Как правило, величина $T$ будет существенно зависеть от объёма исходных данных: поиск в списке из 10 элементов завершится гораздо быстрее, чем в списке из 10000 элементов. Поэтому сложность алгоритма обычно связывают с размером входных данных $N$ и определяют как функцию $T(N)$. Например, для алгоритмов обработки массивов в качестве размера $N$ используют длину массива. Функция $T(N)$ называется **временно́й сложностью алгоритма**.

Временна́я сложность агоритма определяется функцией $T(N) = 2N^3$. Во сколько раз увеличится время работы алгоритма, если размер данных $N$ увеличится в 10 раз?

**Пространственная сложность** — это зависимость объёма занимаемой памяти от размера данных $N$. Для ускорения работы некоторых алгоритмов нужно использовать дополнительную память, которая может быть намного больше, чем память для хранения исходных данных.

Для быстрой сортировки массива из $N$ элементов с помощью алгоритма Семёна требуется $N$ вспомогательных массивов, каждый из которых содержит $N$ элементов. Как изменится объём нужной дополнительной памяти, если $N$ увеличится в 10 раз?

Поскольку память постоянно дешевеет, а быстродействие компьютеров растёт медленно, более важна временна́я сложность алгоритмов. Пространственную сложность мы дальше рассматривать не будем.

## Примеры вычисления сложности

Рассмотрим алгоритмы выполнения различных операций с массивом A длины N, который может быть объявлен в программе так:

```
целтаб A[1:N]
```

```
var A: array[1..N] of integer;
```

**Пример 1.** Вычислить сумму значений первых трёх элементов массива (при $N \geq 3$).

Решение этой задачи содержит всего один оператор:

```
Sum:=A[1]+A[2]+A[3];
```

Этот алгоритм включает две операции сложения и одну операцию записи значения в память, поэтому его сложность $T(N) = 3$ не зависит от размера массива вообще.

Вычислите количество операций (считая сравнения и присваивание значений переменным) при выполнении фрагмента программы:

```
a:=8
b:=15
если a<b то
  c:=2*a+b
иначе
  c:=2*b+a
все
```

```
a:=8;
b:=15;
if a<b then
  c:=2*a+b
else
  c:=2*b+a;
```

**Пример 2.** Вычислить сумму значений всех элементов массива.

В этой задаче уже не обойтись без цикла:

```
Sum:=0
нц для i от 1 до N
  Sum:=Sum+A[i]
кц
```

```
Sum:=0;
for i:=1 to N do
  Sum:=Sum+A[i];
```

Здесь выполняется $N$ операций сложения и $N + 1$ операция записи в память, поэтому его сложность $T(N) = 2N + 1$ возрастает линейно с увеличением длины массива.

Вычислите количество операций при выполнении фрагмента программы:

```
a:=0; b:=1
нц для i от 1 до N
  a:=a+b*b
  b:=b+1
кц
```

```
a:=0; b:= 1;
for i:=1 to N do begin
  a:=a+b*b;
  b:=b+1
end;
```

**Пример 3.** Отсортировать все элементы массива по возрастанию методом выбора. Напомним, что метод выбора предполагает поиск на каждом шаге минимального из оставшихся неупорядоченных значений (здесь i, j, nMin и c — целочисленные переменные):

```
нц для i от 1 до N-1
  nMin:=i
  нц для j от i+1 до N
    если A[j]<A[nMin] то
      nMin:=j
    все
  кц
  c:= A[i]
  A[i]:= A[nMin]
  A[nMin]:=c
кц
```

```
for i:=1 to N-1 do begin
  nMin:=i;
  for j:=i+1 to N do
    if A[j]<A[nMin] then
      nMin:=j;
  c:=A[i];
  A[i]:=A[nMin];
  A[nMin]:=c;
end;
```

Подсчитаем отдельно количество сравнений и количество перестановок. Количество сравнений элементов массива не зависит от данных и определяется числом шагов внутреннего цикла:

$$T_c(N)=(N-1)+(N-2)+\ldots+2+1=\frac{N(N-1)}{2}=\frac{1}{2}N^2-\frac{1}{2}N.$$

Здесь использована формула для суммы первых $N-1$ членов арифметической прогрессии.

На каждом шаге внешнего цикла происходит перестановка двух элементов, общее количество перестановок равно $T_n(N)=N-1$, т. е. сложность по перестановкам — линейная.

Определите количество операций при вычислении суммы значений элементов квадратной матрицы A размером N × N (здесь i, j и Sum — целочисленные переменные):

```
Sum:=0
нц для i от 1 до N
  нц для j от 1 до N
    Sum:=Sum+A[i,j]
  кц
кц
```

```
Sum:=0;
for i:=1 to N do
  for j:=1 to N do
    Sum:=Sum+A[i,j];
```

По результатам этих примеров можно сделать выводы:

- простой цикл, в котором количество шагов пропорционально $N$, — это алгоритм линейной сложности;
- вложенный цикл, в котором количество шагов внешнего и внутреннего цикла пропорционально $N$, — это алгоритм квадратичной сложности.

## Что такое асимптотическая сложность?

Допустим, что нужно сделать выбор между несколькими алгоритмами, которые имеют разную сложность. Какой из них лучше (работает быстрее)? Оказывается, для этого необходимо знать размер массива данных, которые нужно обрабатывать. Сравним, например, три алгоритма, сложность которых

$$T_1(N) = 10000 \cdot N, \quad T_2(N) = 100 \cdot N^2 \quad \text{и} \quad T_3(N) = N^3.$$

Построим эти зависимости на графике (рис. 4.5). При $N \le 100$ получаем $T_3(N) < T_2(N) < T_1(N)$, при $N = 100$ количество операций для всех трёх алгоритмов совпадёт, а при бóльших $N$ имеем $T_3(N) > T_2(N) > T_1(N)$.

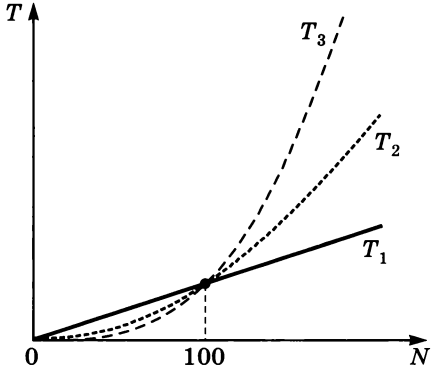



**Рис. 4.5**

Обычно в теоретической информатике при сравнении алгоритмов используется их **асимптотическая сложность**, т. е. скорость роста количества операций при больших значениях $N$.

Запись $O(N)$ (читается **«О большое от $N$»**) обозначает линейную сложность алгоритма. Это значит, что, начиная с некоторого значения $N = N_0$, количество операций будет меньше, чем $c \cdot N$, где $c$ — некоторая постоянная:

$$T(N) \le c \cdot N \text{ для } N \ge N_0.$$

При увеличении размера данных в 10 раз объём вычислений алгоритма с линейной сложностью увеличивается тоже примерно в 10 раз.

Пусть, например, $T(N) = 2N - 1$, как в алгоритме поиска суммы значений элементов массива. Очевидно, что при этом $T(N) \le 2N$ для всех $N \ge 1$, поэтому алгоритм имеет линейную сложность.

Определите любые подходящие значения $c$ и $N_0$, такие что $T(N) \le c \cdot N$ для $N \ge N_0$, для алгоритмов с линейной асимптотической сложностью:

а) $T(N) = 12N - 8$; б) $T(N) = 7N + 5$.

Многие известные алгоритм имеют **квадратичную сложность $O(N^2)$**. Это значит, что сложность алгоритма при больших $N$ меньше, чем $c \cdot N^2$:

$$T(N) \le c \cdot N^2 \text{ для } N \ge N_0.$$

Если размер данных увеличивается в 10 раз, то количество операций (и время выполнения такого алгоритма) увеличивается примерно в 100 раз. Пример алгоритма с квадратичной сложностью — сортировка методом выбора, для которой число сравнений

$$T_c(N) = \frac{1}{2}N^2 - \frac{1}{2}N \le \frac{1}{2}N^2 \text{ для всех } N \ge 0.$$

Определите любые подходящие значения $c$ и $N_0$, такие что $T(N) \le c \cdot N^2$ для $N \ge N_0$, для алгоритмов с квадратичной асимптотической сложностью:

а) $T(N) = 5N^2 - 3N - 2$; б) $T(N) = 7N^2 - 3N - 5$.

---

Алгоритм имеет **асимптотическую сложность** $O(f(N))$, если найдется такая постоянная $c$, что, начиная с некоторого $N = N_0$, выполняется условие $T(N) \le c \cdot f(N)$.

---

Это значит, что график функции $c \cdot f(N)$ проходит выше, чем график функции $T(N)$, по крайней мере, при $N \geq N_0$ (рис. 4.6).

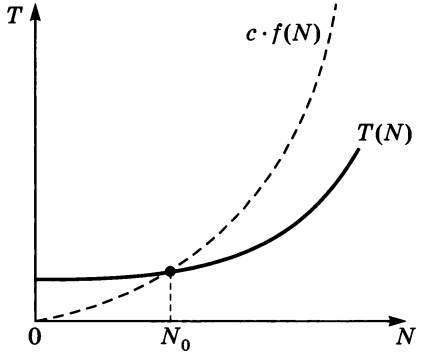



**Рис. 4.6**

Если количество операций не зависит от размера данных, то говорят, что асимптотическая сложность алгоритма $O(1)$, т. е. количество операций меньше некоторой постоянной при любых $N$.

Существует также немало алгоритмов с кубической сложностью $O(N^3)$. При больших значениях $N$ алгоритм с кубической сложностью требует бóльшего количества вычислений, чем алгоритм со сложностью $O(N^2)$, а тот, в свою очередь, работает дольше, чем алгоритм с линейной сложностью. Однако при небольших значениях $N$ всё может быть наоборот, это зависит от постоянной $c$ для каждого из алгоритмов.

Известны и алгоритмы, для которых количество операций растёт быстрее, чем любой многочлен, например как $O(2^N)$ или $O(N!)$, где $N!$ обозначает факториал числа $N$ — произведение всех натуральных чисел от 1 до $N$ : $N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$. Такие алгоритмы встречаются чаще всего в задачах поиска оптимального (наилучшего) варианта, которые решаются только методом полного перебора. Самая известная задача такого типа — это *задача коммивояжёра* (бродячего торговца), который должен посетить по одному разу каждый из указанных городов и вернуться в начальную точку. Для него нужно выбрать маршрут, при котором стоимость поездки (или общая длина пути) будет минимальной.

В таблице 4.1 показано примерное время работы алгоритмов, имеющих разную временну́ю сложность, при $N = 100$ на компьютере с быстродействием 1 миллиард операций в секунду.

Таблица 4.1

| $T(N)$ | Время выполнения |
|---|---|
| $N$ | 100 нс |
| $N^2$ | 10 мс |
| $N^3$ | 0,001 с |
| $2^N$ | $10^{13}$ лет |

Юный программист Григорий поспорил с учителем, что сможет к завтрашнему уроку с помощью компьютера решить сложную задачу перебора вариантов. Дома он определил временну́ю сложность алгоритма: $T(N) = 2^N$. Для какого наибольшего значения $N$ сможет Григорий решить задачу за сутки, если его компьютер выполняет 1 миллиард операций в секунду?

## Выводы

- Временем работы алгоритма называется количество элементарных операций $T$, выполненных исполнителем.
- Временна́я сложность алгоритма обычно зависит от объёма исходных данных $N$, например от размера массива.
- Пространственная сложность — это объём памяти, необходимой для работы алгоритма.
- Простой цикл, в котором количество шагов пропорционально $N$, — это алгоритм линейной сложности.
- Вложенный цикл, в котором количество шагов внешнего и внутреннего цикла пропорционально $N$, — это алгоритм квадратичной сложности.
- Алгоритм имеет асимптотическую сложность $O(f(N))$, если найдётся такая постоянная $c$, что, начиная с некоторого $N = N_0$, выполняется условие $T(N) \le c \cdot f(N)$.
- Линейная сложность означает, что при увеличении размера массива в $K$ раз количество операций увеличивается примерно в $K$ раз.
- Квадратичная сложность означает, что при увеличении размера массива в $K$ раз количество операций увеличивается примерно в $K^2$ раз.

Нарисуйте в тетради интеллект-карту этого параграфа.

### Вопросы и задания

1. Какие критерии используются для оценки алгоритмов?
2. Почему скорость работы алгоритма оценивается не временем выполнения, а количеством элементарных операций?
3. Как учитывается размер данных при оценке быстродействия алгоритма?
4. В каких случаях алгоритм, имеющий асимптотическую сложность $O(N^2)$, может работать быстрее, чем алгоритм с асимптотической сложностью $O(N)$?
5. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

«Задача коммивояжёра»

# § 23
# Как разрабатывают программы?

***Ключевые слова:***

- постановка задачи
- построение модели
- разработка алгоритма и способа представления данных
- кодирование
- отладка
- тестирование
- документирование
- внедрение и сопровождение

Как вы знаете, новые программы для компьютеров пишут программисты. Но это не совсем точно: любая достаточно сложная программа проходит несколько этапов от рождения идеи до выпуска готового продукта, и в этом участвует множество специалистов.

## Этапы разработки программ

**1. Постановка задачи.** Сначала определяют задачи, которые должна решать программа, и записывают все требования к ней в виде документа — *технического задания*. Это очень важный этап, потому что ошибка в самом начале разработки приведёт к тому, что будет решена совершенно другая задача.

**2. Построение модели.** Когда задача поставлена, нужно выполнить **формализацию** — записать все требования на формальном

языке, например на языке математических формул. В результате строится модель исходной задачи, в которой чётко определяются все связи между исходными данными и желаемым результатом.

**3. Разработка алгоритма и способа представления данных.** Любая компьютерная программа служит для обработки данных. Поэтому очень важно определить, как будут представлены данные в памяти компьютера (например, в виде отдельных переменных или массивов).

Способ хранения данных определяет и алгоритмы работы с ними: если выбрана неудачная структура данных, очень сложно написать хороший алгоритм обработки. Известная книга швейцарского специалиста Никлауса Вирта, автора языка Паскаль, так и называется «Алгоритмы + структуры данных = программы».

**4. Кодирование.** Только теперь, когда выбран способ хранения данных и готовы алгоритмы для работы с ними, программисты приступают к написанию программы. Эта работа называется *кодированием*, потому что программист кодирует алгоритм — записывает его на языке программирования. Результат его работы — текст программы — часто называют *программным кодом*.

**5. Отладка.** Ни один человек не может написать достаточно большую программу без ошибок. Поэтому программисту приходится искать и устранять ошибки в программах. Этот процесс называется *отладкой программы*.

Все ошибки можно разделить на две группы: синтаксические ошибки и логические ошибки. *Синтаксические ошибки* — несоответствие правилам языка программирования — обнаруживаются транслятором, поэтому найти и исправить их достаточно просто.

Сложнее исправлять *логические ошибки* — ошибки в составлении алгоритма. Из-за логических ошибок программа работает не так, как требуется. Чтобы исправить такую ошибку, программисту приходится внимательно изучить работу программы, иногда даже выполнить многие вычисления вручную, без компьютера, и сравнить результаты каждого шага с теми результатами, которые даёт программа.

Логические ошибки могут привести к *отказу* — аварийной ситуации, например к делению на ноль. Часто при отказе операционная система завершает работу программы, и данные могут быть потеряны. Отказы часто называют *ошибками времени выполнения* (англ. *runtime error*).

**6. Тестирование.** Когда программист исправил все обнаруженные им ошибки, он передаёт программу на тестирование — тщательную проверку в различных режимах. Обычно эту работу выполняют специально обученные люди — *тестировщики*.

Тестирование в компании, которая разрабатывает программу, называется альфа-тестированием. Когда оно завершено, начина-

ется бета-тестирование (внешнее тестирование). Программа (бета-версия) рассылается некоторым клиентам или даже распространяется свободно. Цель этого этапа — привлечь к тестированию множество людей, чтобы они смогли найти как можно больше ошибок в программе.

**7. Документирование** — это разработка документации на программу. Этим занимаются *технические писатели*. Техническая документация описывает, как работает программа, а руководство пользователя содержит инструкцию по использованию программы.

**8. Внедрение и сопровождение**. Когда программа отлажена и документация по ней готова, её нужно передать заказчику. Компания берёт на себя *сопровождение* программы — обучение пользователей, исправление найденных ими ошибок, техническую поддержку (ответы на вопросы). Часто компании выпускают новые версии программ, в которых исправляются ошибки и добавляются новые возможности.

## Методы проектирования программ

Современные программы очень сложны, они могут состоять из сотен тысяч и миллионов строк. Написать такую программу в одиночку невозможно, поэтому над проектом работают большие команды программистов. Нужно как-то разделить работу между ними, чтобы каждый мог выполнять свою часть независимо от других. Для этого необходимо разбить задачу на подзадачи (рис. 4.7).

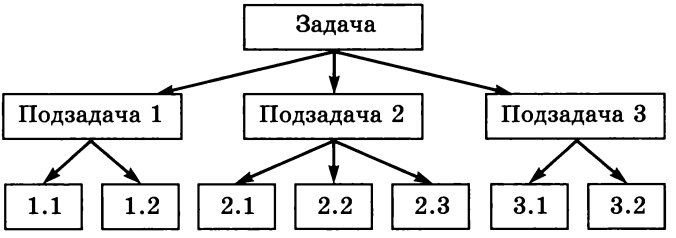



**Рис. 4.7**

Решение каждой подзадачи оформляется в виде подпрограммы — вспомогательного алгоритма (вспомните материал 7 класса). Программист получает персональное задание — написать одну или несколько подпрограмм. Он может работать независимо от других, важно только соблюдать правила обмена данными между «его» подпрограммой и остальными.

Если нужно, подзадачи разбиваются на более мелкие подзадачи (см. третий уровень дерева на рис. 4.7) и т. д., пока все подпрограммы не будут записаны полностью на языке програм-

мирования. Обычно подпрограмма не должна быть длиннее, чем 30—40 строк.

Такой приём называется **последовательным уточнением** или **проектированием «сверху вниз»**, от основной задачи к мелким подзадачам.

Существует и другой подход — **проектирование «снизу вверх»**: сначала разработать подпрограммы для решения самых простых задач, а потом собирать из них подпрограммы для более крупных задач, как из кубиков. При этом мы строим дерево, показанное на рис. 4.7, снизу вверх, с нижнего уровня. На практике программисты обычно сочетают оба подхода.

## Отладка программы

Простейший метод отладки программы — это *вывод отладочной информации*. Рассмотрим этот способ на примере.

Программисту нужно было написать программу, которая вычисляет корни квадратного уравнения $ax^2+bx+c=0$. Он поспешил и написал программу так:

```
алг КвУр
нач
  вещ a, b, c, D, x1, x2
  вывод 'Введите a, b, c: '
  ввод a, b, c
  D:=b*b-4*a*a;
  x1:=(-b+sqrt(D))/2*a;
  x2:=(-b-sqrt(D))/2*a;
  вывод 'x1=', x1, ' x2=', x2
кон
```

```
program SqEq;
var a, b, c, D,
    x1, x2: real;
begin
  write('Введите a, b, c: ');
  readln(a, b, c);
  D:= b*b-4*a*a;
  x1:=(-b+sqrt(D))/2*a;
  x2:=(-b-sqrt(D))/2*a;
  writeln('x1=', x1,
          ' x2=', x2)
end.
```

Для вычисления квадратного корня здесь используется стандартная функция `sqrt`. Оказалось, что программа в некоторых случаях работает верно (например, при $a = 1$, $b = 2$ и $c = 1$), а в других случаях — неверно (например, при $a = 1$, $b = -5$ и $c = 6$).

Для того чтобы найти ошибку, нужно определить её **возможные причины**. В нашем случае есть три варианта:

1) неверно вводятся данные;
2) неверно вычисляется дискриминант $D=b^2-4ac$;
3) неверно вычисляются корни $x_1=\frac{-b+\sqrt{D}}{2a}$, $x_2=\frac{-b-\sqrt{D}}{2a}$.

Добавим в программу две дополнительные команды для вывода отладочной информации:

1) выведем значения коэффициентов $a$, $b$ и $c$ сразу после ввода;

2) выведем вычисленное значение дискриминанта.

Значения корней уравнения уже и так выводятся в конце работы программы.

```
ввод a, b, c                    readln(a, b, c);
вывод a, ' ', b,' ', c, нс      writeln(a, ' ', b, ' ', c);
D:= b*b-4*a*a;                  D:= b*b - 4*a*a;
вывод 'D=', D, нс               writeln('D=', D);
...                             ...
```

При вводе коэффициентов 1, −5 и 6 программа (на алгоритмическом языке) выводит:

```
1.0 -5.0 6.0
D=21.0
x1=4.791288 x2=0.208712
```

По первой строке видим, что ввод выполнен правильно — именно такие числа мы вводили. А вот значение дискриминанта, вычисленного программой, отличается от того, что мы ожидаем получить: $D = (-5)^2 - 4 \cdot 1 \cdot 6 = 1$. Поэтому нужно искать ошибку в выражении для вычисления $D$.

Если исправить эту ошибку (сделайте это самостоятельно), мы увидим, что дискриминант считается правильно, а корни уравнения — нет (при $a = 1$, $b = -5$ и $c = 6$ мы должны получить $x_1 = 3$ и $x_2 = 2$). Поэтому останется исправить ошибки в строках, где вычисляются корни.

Современные среды программирования, в том числе *КуМир*, *АЛГО* и *PascalABC.NET*, содержат встроенный отладчик, который позволяет:

- выполнять программу в пошаговом режиме;
- после выполнения очередной команды просматривать значения переменных в памяти;
- устанавливать *точки останова*, где программа должна остановиться и перейти в пошаговый режим.

Доработайте программу так, чтобы учесть случай, когда уравнение не имеет вещественных корней.

## Документирование программы

К выпуску программы компания-разработчик должна подготовить **документацию** на программу. Руководство пользователя (это наиболее важная часть документации) содержит всю информацию, необходимую для использования программы:

- назначение программы;
- формат входных данных;
- формат выходных данных;
- примеры использования программы.

Для примера составим документацию на простую программу, отладкой которой мы только что занимались.

*Назначение программы*: вычисление корней квадратного уравнения $ax^2 + bx + c = 0$.

*Формат входных данных*: значения коэффициентов $a$, $b$ и $c$ вводятся с клавиатуры через пробел в одной строке.

*Формат выходных данных*: значения корней уравнения выводятся на экран через пробел в одной строке; перед значением первого корня выводится текст `x1=`, перед значением второго корня — текст `x2=`.

*Пример использования программы* (решение уравнения $x^2 - 5x + 6 = 0$):

```
Введите a, b, c: 1 -5 6
x1=4.791288 x2=0.208712
```

## Выводы

- Этапы разработки программного обеспечения:
  - — постановка задачи;
  - — построение модели;
  - — разработка алгоритма и способа представления данных;
  - — кодирование;
  - — отладка;
  - — тестирование;
  - — документирование;
  - — внедрение и сопровождение.
- При использовании метода проектирования «сверху вниз» (метода последовательного уточнения) задача разбивается на подзадачи, каждая из подзадач оформляется в виде вспомогательного алгоритма. Сначала составляется основная программа, а затем все вспомогательные алгоритмы.
- При использовании метода проектирования «снизу вверх» разработка программы начинается с наиболее мелких подзадач, из которых основная программа затем собирается, как из кубиков.

Нарисуйте в тетради интеллект-карту этого параграфа.

### Вопросы и задания

1. Почему алгоритмы и способы хранения данных разрабатываются одновременно?
2. Чем отличается тестирование от отладки?
3. Можно ли считать, что программа, успешно прошедшая тестирование, не содержит ошибок?
4. Может ли произойти отказ в программе, в которой нет логических ошибок?
5. Если программа плохо документирована, к каким последствиям это может привести?
6. Как вы думаете, почему важно сопровождение программы после её сдачи заказчику?
7. Чем отличаются два подхода к проектированию программ: «сверху вниз» и «снизу вверх»?
8. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «Структурное программирование»
б) «Парадигмы (стили) программирования»

### Практическая работа

Выполните практическую работу № 19 «Отладка программы».

# § 24
# Процедуры

***Ключевые слова:***

- процедура
- параметр
- локальная переменная
- рекурсивная процедура

## Что такое подпрограмма?

Когда мы в 7 классе работали с исполнителем Робот, мы уже использовали вспомогательные алгоритмы (подпрограммы, процедуры). Каждая процедура решала одну подзадачу, из них строилась программа для решения основной задачи.

Подпрограммы полезны в первую очередь потому, что готовые алгоритмы можно использовать много раз при решении более

сложных задач, не «изобретая велосипед». Из подпрограмм составляются библиотеки, некоторые из них входят в состав языков программирования. Мы просто используем их, не задумываясь о том, как они работают. Это экономит время программистов, освобождая их от повторного выполнения работы, которая уже была кем-то сделана раньше.

Подпрограммы бывают двух типов — процедуры и функции. Подпрограммы-процедуры выполняют какие-то действия. Например, `writeln` в языке Паскаль — это стандартная подпрограмма-процедура, которая выводит данные на экран. Подпрограммы-функции возвращают результат (число, строку). Подпрограмма `sqrt`, вычисляющая квадратный корень числа, — это функция.

В этом параграфе мы научимся писать свои подпрограммы-процедуры, а в следующем займёмся функциями.

Определите тип подпрограммы (процедура или функция), которая:

а) рисует окружность на экране;
б) определяет площадь круга;
в) вычисляет значение синуса угла;
г) изменяет режим работы программы;
д) возводит число $x$ в степень $y$;
е) включает двигатель автомобиля;
ж) проверяет оставшееся количество бензина в баке;
з) измеряет высоту полёта самолёта.

## Простая процедура

Предположим, что в нескольких местах программы требуется выводить на экран строку из 10 знаков '-' (например, для того чтобы отделить два блока результатов друг от друга). Это можно сделать, например, так:

```
вывод '----------', нс        writeln('----------');
```

Конечно, можно вставить этот оператор вывода везде, где нужно вывести такую строку. Но тут есть две сложности. Во-первых, строка из минусов хранится в памяти много раз. Во-вторых, если мы задумаем как-то изменить эту строку (например, заменить знак '-' на '='), нужно будет искать эти операторы вывода по всей программе.

Для таких случаев в языках программирования предусмотрены **процедуры** — вспомогательные алгоритмы, которые выполняют некоторые действия:

```
алг С процедурой
нач
  ...
  printLine
  ...
кон
алг printLine
нач
  вывод '----------', нс
кон
```

```
program withProc;
procedure printLine;
begin
  writeln('----------')
end;
begin
  ...
  printLine;
  ...
end.
```

Многоточием в текстах программ будем обозначать некоторые операторы.

В алгоритмическом языке процедура оформляется точно так же, как и основной алгоритм, но размещается после основной программы.

В языке Паскаль процедура начинается с ключевого слова **procedure**, тело процедуры начинается с **begin** и заканчивается ключевым словом **end** с точкой с запятой. Процедура расположена выше основной программы, до момента её первого использования.

Фактически мы ввели в язык программирования новую команду printLine, которая была расшифрована прямо в тексте программы. Для того чтобы процедура заработала, в основной программе (или в другой процедуре) необходимо её вызвать по имени.

Что произойдёт, если вызвать процедуру, но не включить её текст в программу? Проверьте этот вариант с помощью компьютера.

Что произойдёт, если включить текст процедуры в программу, но не вызывать её? Проверьте этот вариант с помощью компьютера.

Использование процедур сокращает код, если какие-то операции выполняются несколько раз в разных местах программы. Когда процедура написана и тщательно протестирована, можно передать её другим программистам для использования в этом же или другом проекте.

Большую программу всегда разбивают на несколько частей, оформляя в виде процедур отдельные этапы сложного алгоритма. Такой подход делает всю программу более понятной и позволяет разделить работу между программистами.

## Процедура с параметром

Теперь представьте себе, что нужно выводить строки из знаков «минус» разной длины (5, 10 и др). Конечно, можно сделать несколько процедур, например:

```
алг printLine5
нач
  вывод '-----', нс
кон
алг printLine10
нач
  вывод '----------', нс
кон
```

```
procedure printLine5;
begin
  writeln('-----')
end;
procedure printLine10;
begin
  writeln('----------')
end;
```

Но так делать не нужно. Дело в том, что обе процедуры выводят цепочки знаков «минус» (т. е. выполняют одни и те же действия!), только разной длины. Поэтому хочется использовать всего одну процедуру, передавая ей нужную длину цепочки.

Заметим, что процедуру printLine10 можно переписать с помощью цикла:

```
алг printLine10
нач
  цел i
  нц для i от 1 до 10
    вывод '-'
  кц
  вывод нс
кон
```

```
procedure printLine10;
var i: integer;
begin
  for i:=1 to 10 do
    write('-');
  writeln
end;
```

Эта процедура делает то же самое, что и первый вариант, — выводит строку из 10 минусов и переходит на новую строку.

Видим, что процедура стала длиннее и усложнилась, появился цикл. Внутри процедуры объявлена переменная *i*. Эта переменная принадлежит процедуре, она называется локальной. Другие процедуры и основная программа не могут обращаться к «чужой» локальной переменной.

Где вы уже встречались со словом «локальный» в курсе информатики? Вспомните, от какого иностранного слова оно произошло.

**Локальная переменная** — это переменная, объявленная внутри подпрограммы. Другие подпрограммы и основная программа не могут к ней обращаться.

Локальная переменная существует только тогда, когда работает процедура. Как только работа процедуры закончена, все локальные переменные удаляются из памяти.

Чем будет отличаться процедура, рисующая 5 знаков «минус», от последнего варианта процедуры printLine10?

Если мы хотим, чтобы число повторений цикла можно было менять, в процедуре вместо числа нужно использовать *переменную*. И значение этой переменной нужно как-то передать процедуре. Оформляется это так:

```
алг printLine( цел n )
нач
  цел i
  нц для i от 1 до  n
    вывод '-'
  кц
  вывод нс
кон
```

```
procedure printLine( n: integer );
var i: integer;
begin
  for i:=1 to  n  do
    write('-');
  writeln
end;
```

Величина *n* называется параметром процедуры. В заголовке процедуры добавились круглые скобки, где записано имя параметра и его тип.

**Параметр** — это величина, от которой зависит работа процедуры. Параметр имеет имя и тип, с ним можно работать так же, как с локальной переменной.

Наша процедура printLine имеет один параметр, обозначенный именем n, — длину строки из знаков «минус».

При вызове такой процедуры в скобках нужно передать фактическое значение, которое должна принять переменная *n* внутри процедуры. Такое значение называется аргументом (или фактическим параметром).

**Аргумент** — это значение параметра, которое передаётся процедуре.

При вызове процедуры аргумент передаётся в скобках:

```
printLine(10);
```

Что будет выведено на экран при выполнении фрагмента программы?

```
printLine(7);
printLine(5);
printLine(3);
```

Для тестирования процедуры printLine Иван хочет написать небольшую программу, в которой длина линии вводится с клавиатуры. Где нужно поместить оператор ввода — в процедуре или в основной программе?

## Несколько параметров

Давайте немного улучшим процедуру: сделаем так, чтобы можно было изменять не только длину строки, но и символы, из которых она строится. Для этого в процедуру нужно добавить ещё один параметр символьного типа (**лит** в алгоритмическом языке и string в языке Паскаль):

```
алг printLine(лит c, цел n)
...
```

```
procedure printLine
  (c: string; n: integer);
...
```

В алгоритмическом языке параметры в заголовке процедуры отделяются запятой, а в Паскале – точкой с запятой.

Запишите в тетради полный текст процедуры printLine.

Что будет выведено на экран при выполнении фрагмента программы?

```
printLine('-', 10);
printLine('=', 7);
printLine('o', 5);
```

## Процедуры в других языках программирования

Процедура printLine с одним параметром на языках *Python* и *C* может быть записана так:

```
def printLine (n):
  print ('-'*n)
```

```
void printLine (int n)
{
  int i;
  for (i=1; i<=n; i++)
    putchar('-');
}
```

В языке *Python* объявление процедуры начинается ключевым словом **def**. Тип параметра указывать не нужно, он определяется автоматически. Тело процедуры записывается с отступом вправо, так же, как тело цикла и условного оператора. Вывести n одинаковых символов можно без цикла, построив сразу нужную строку с помощью «умножения» символа '-' на n.

Программа на языке *C* очень похожа на программу на Паскале. Признак процедуры — слово **void** в заголовке (вместо **procedure** на Паскале). Тело процедуры заключено в фигурные скобки. Стандартная функция putchar выводит на экран один символ.

## Рекурсия

Составим процедуру, которая выводит на экран двоичную запись натурального числа. Поскольку число будет меняться, это должна быть процедура с параметром:

```
алг printBin(цел n)
нач
  ...
кон
```

```
procedure printBin(n: integer);
begin
  ...
end;
```

Вспомним алгоритм перевода числа в двоичную систему: нужно делить число на 2, каждый раз выписывая остаток от деления, пока не получится 0. На алгоритмическом языке алгоритм можно записать так:

```
нц пока n<>0
  вывод mod(n,2)
  n:= div(n,2)
кц
```

Проверьте вручную работу этого алгоритма для числа 6. Удалось ли вам получить правильный ответ? Почему?

Проблема только в том, что первой мы получаем последнюю цифру двоичной записи, поэтому остатки выводятся в обратном порядке (не так, как нужно).

Есть разные способы решения этой задачи, которые сводятся к тому, чтобы запоминать остатки от деления (например, в символьной строке) и затем, когда результат будет полностью получен, вывести его на экран. Однако можно применить ещё один красивый подход. Идея такова: *чтобы вывести двоичную запись числа* n, *нужно сначала вывести двоичную запись числа* div(n,2), а затем — его последнюю двоичную цифру, которая вычисляется как mod(n,2).

Что же получилось? Прочитайте ещё раз фразу, выделенную курсивом в предыдущем абзаце. Выходит, что для того, чтобы решить задачу для исходного числа, нужно предварительно решить *ту же самую задачу* для меньшего числа div(n,2).

Такой алгоритм очень просто программируется:

```
алг printBin(цел n)
нач
  printBin(div(n,2))
  вывод mod(n,2)
кон
```

```
procedure printBin(n: integer);
begin
  printBin(n div 2);
  write(n mod 2)
end;
```

У нас получилось, что процедура printBin вызывает сама себя! Такой приём в программировании называется рекурсией, а процедура — рекурсивной.

**Рекурсивная процедура** — это процедура, которая вызывает сама себя.

Проверьте с помощью отладчика[1] в пошаговом режиме, что произойдёт при вызове этой процедуры.

Приведённая процедура printBin ошибочна, вернее, она не доделана. Представим себе, что мы передали процедуре число 2. Сначала она вызывает сама себя для значения div(2,2) = 1, затем — для значения div(1,2) = 0, и потом ещё бесконечно много раз для нуля. Такие вызовы никогда не закончатся, и программа зациклится. Чтобы этого не произошло, нужно выйти из процедуры (и закончить эти вложенные вызовы), когда значение параметра станет равно нулю:

```
алг printBin(цел n)
нач
  если n=0 то выход все
  printBin(div(n,2))
  вывод mod(n,2)
кон
```

```
procedure printBin(n:
integer);
begin
  if n=0 then exit;
  printBin(n div 2);
  write(n mod 2)
end;
```

В алгоритмическом языке для выхода из процедуры используется оператор выход, а в языке Паскаль — оператор exit.

Убедимся, что процедура остановится при любом заданном натуральном числе. Действительно, при каждом вложенном вызове значение параметра уменьшается (делится нацело на 2). В результате когда-нибудь оно обязательно станет равно нулю, и вложенные вызовы закончатся.

[1] Для входа в процедуру в пошаговом режиме отладчика используйте клавишу F7.

Сформулируйте алгоритм вывода цепочки одинаковых символов, используя рекурсию. Напишите рекурсивную процедуру. Попробуйте придумать два варианта решения.

## Выводы

- Процедура — это вспомогательный алгоритм (подпрограмма), решающий самостоятельную задачу, который может использоваться несколько раз.
- Локальная переменная — это переменная, объявленная внутри подпрограммы. Другие подпрограммы и основная программа не могут к ней обращаться.
- Параметр — это величина, от которой зависит работа процедуры. Параметр имеет имя, с ним можно работать так же, как с локальной переменной.
- Аргумент — это значение параметра, которое передаётся процедуре.
- Рекурсивная процедура — это процедура, которая вызывает сама себя.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Зачем нужны процедуры?
2. Достаточно ли включить процедуру в текст программы, чтобы она «сработала»?
3. Какие возможности появляются, когда в процедуру добавляются параметры?
4. Как определить, что переменная — локальная?
5. Выполните по указанию учителя задания в рабочей тетради.

## Подготовьте сообщение

а) «Рекурсия в природе и искусстве»
б) «Ханойские башни»

## Практические работы

Выполните практические работы:
№ 20 «Процедуры»;
№ 21 «Рекурсивные процедуры».

# § 25
# Функции

***Ключевые слова:***

- функция
- вызов функции
- параметры
- рекурсивная функция

## Что такое функция?

Представьте себе, что вы заказываете товар с доставкой по телефону или в интернет-магазине. Если говорить на языке программистов, вы вызываете вспомогательный алгоритм. Но, в отличие от процедуры, исполнитель этого алгоритма не только выполняет какие-то действия, но и *возвращает результат* — товар, который вам привозит курьер. Это второй тип вспомогательных алгоритмов (подпрограмм). Такие подпрограммы называются функциями.

---

**Функция** — это вспомогательный алгоритм, который возвращает результат (число, строку символов и др.).

---

Построим функцию, которая возвращает среднее арифметическое двух целых чисел.

Какой тип данных нужно использовать для хранения среднего арифметического двух целых чисел?

Функция принимает два параметра — исходные целые числа и возвращает результат — вещественное число:

```
алг вещ Avg(цел a, b)
нач
  знач:=(a+b)/2
кон
```

```
function Avg(a, b: integer): real;
begin
  Avg:=(a+b)/2
end.
```

В заголовке функции указывают тип результата — перед именем функции в алгоритмическом языке записано ключевое слово **вещ**. В языке Паскаль заголовок функции начинается словом **function**, а после скобок, в которых перечисляются параметры, через двоеточие записывают тип результата — `real` (вещественное число).

Используя дополнительные источники, выясните, что означает английское слово *average*, от которого образовано название функции `Avg`.

Результат функции (возвращаемое значение) нужно записать в специальную переменную. В алгоритмическом языке она называется знач, а в Паскале совпадает с именем функции.

```
знач:=(a+b)/2
```

```
Avg:=(a+b)/2;
```

При вызове функции нужно сказать, где сохранить полученный от неё результат. Например, можно записать его в отдельную переменную того типа, который возвращает функция:

```
вещ sr
...
sr:=Avg(2,3)
```

```
var sr: real;
...
sr:=Avg(2,3);
```

Результат функции можно сразу вывести на экран:

```
вывод Avg(4,8)
```

```
writeln(Avg(4,8));
```

Что будет выведено на экран в результате работы этого фрагмента программы?

```
sr:=Avg(3,5)
вывод sr+Avg(7,11)
```

```
sr:=Avg(3,5);
write(sr+Avg(7,11));
```

Функции можно передавать не только аргументы-константы, но также имена переменных и арифметические выражения:

```
цел a, b
вещ sr
...
a:=5
b:=7
sr:=Avg(a, b+8)
```

```
var a, b: integer;
    sr: real;
...
a:=5;
b:=7;
sr:=Avg(a,b+8);
```

Наша функция Avg возвращает вещественное число, поэтому вызовы этой функции можно применять везде, где разрешено использовать вещественное число, в том числе в арифметических выражениях, условных операторах и циклах. Например:

```
c:=2*Avg(x,y)+z
если Avg(a,b)>4 то
  ...
все
нц пока Avg(a,b)<x
  ...
кц
```

```
c:=2*Avg(x,y)+z;
if Avg(a,b)>4 then begin
  ...
end;
while Avg(a,b)<x do begin
  ...
end;
```

Найдите значения переменных a, b и x, при которых в результате работы этого фрагмента программы будет выведено сообщение «Да!»:

```
если Avg(a,b)>x то
  вывод 'Да!'
все
```

```
if Avg(a,b)>x then
  writeln('Да!');
```

Найдите начальные значения переменных a, b и x, при которых этот цикл выполнится ровно четыре раза:

```
нц пока Avg(a,b)<x-1
  b:=b+1
кц
```

```
while Avg(a,b)<x-1 do begin
  b:= b+1
end;
```

## Функции в других языках программирования

Функция Avg на языках *Python* и *C* может быть записана так:

```
def Avg(a, b):
  return(a+b)/2
```

```
float Avg(int a, int b)
{
  return(a+b)/2.0;
}
```

В обоих этих языках для того, чтобы определить результат работы функции, используют оператор **return** («вернуть»). Тело функции на языке *Python* записывается с отступом, а на языке *C* заключается в фигурные скобки. По умолчанию (если не указано иначе) в языке *C* при делении целого числа на целое получается целое число (остаток отбрасывается). Чтобы получить вещественный результат, мы разделили сумму a+b на вещественное число 2.0.

Изучите текст программы на языке *C*, сравните его с программой на Паскале и выясните, как в языке *C* указывается, что результат работы фукции — вещественная величина.

## Примеры функций

**Задача 1.** Составить функцию, которая определяет наибольшее из двух целых чисел.

Алгоритм определения наибольшего из двух чисел вы уже знаете из курса 8 класса. Остаётся только «завернуть» его в функцию. Например, так:

```
алг цел Max(цел a, b)
нач
  если a>b то
        знач:=a
  иначе знач:=b
  все
кон
```

```
function Max(a, b: integer):
                    integer;
begin
  if a>b then
       Max:=a
  else Max:=b
end;
```

Одна функция может вызывать другую. Например, можно составить функцию Max3, которая возвращает наибольшее из трёх чисел, используя готовую функцию Max:

```
алг цел Max3(цел a,b,c)
нач
  знач:=Max(Max(a,b),c)
кон
```

```
function Max3(a, b, c: integer):
                    integer;
begin
  Max3:=Max(Max(a,b),c)
end;
```

Постройте функцию Max4, которая вычисляет наибольшее из четырёх чисел, используя функцию Max. Приведите два варианта решения задачи.

**Задача 2.** Составить функцию, которая вычисляет сумму цифр натурального числа.

Последняя цифра — это остаток от деления числа на 10 (операция mod). Для того чтобы удалить последнюю цифру числа, можно разделить его на 10 без остатка (операция div). Поэтому для решения этой задачи на каждом шаге цикла «отрезаем» от числа последнюю цифру, добавляем её значение к сумме, и затем удаляем её из числа. Цикл заканчивается, когда все цифры удалены и осталось нулевое значение:

```
алг цел sumDigits(цел N0)
нач
  цел N, d, sum
  N:=N0
  sum:=0
  нц пока N<>0
    d:=mod(N,10)
    sum:=sum+d
    N:=div(N,10)
  кц
  знач:=sum
кон
```

```
function sumDigits(N: integer):
                    integer;
var d, sum: integer;
begin
  sum:=0;
  while N<>0 do begin
    d:=N mod 10;
    sum:=sum+d;
    N:=N div 10
  end;
  sumDigits:=sum
end;
```

В алгоритмическом языке подпрограмма не может изменять значение параметра-аргумента. Поэтому мы вынуждены ввести дополнительную локальную переменную $N$, которая будет изменяться в цикле.

Как нужно изменить функцию, чтобы она вычисляла количество цифр числа?

Как нужно изменить функцию, чтобы она вычисляла количество единиц в двоичной записи числа?

---

**Задача 3.** Составить функцию, которая удаляет все двойные пробелы в символьной строке, заменяя их на одиночные.

Ответьте на вопросы по условию задачи.
— Какие исходные данные (параметры) принимает функция?
— Какой тип данных нужно использовать для хранения исходных данных?
— Что будет результатом работы этой функции?
— Какой тип данных нужно использовать для хранения результата?
— Нужно ли внутри функции использовать цикл?
— Если цикл нужен, то какого типа должен быть цикл (с известным числом повторений или с условием)?

Функция принимает один параметр — символьную строку, и возвращает тоже символьную строку, поэтому её заголовок выглядит так:

```
алг лит нетДвПроб(лит s0)
нач
  ...
кон
```

```
function noDSp(s: string):
                  string;
begin
  ...
end;
```

Поскольку двойных пробелов может быть много, в программе нужен цикл. Так как неизвестно, сколько их, это будет цикл с условием (**пока**, **while**). Он должен остановиться, когда двойных пробелов больше не останется.

С помощью какой функции можно определить, что в символьной строке больше нет двойных пробелов? Как её нужно использовать?

Запишите в тетради операторы языка программирования, с помощью которых:

а) в переменную p записывается номер символа в строке s, с которого начинается двойной пробел;

б) из строки s удаляется один символ в позиции p.

Приведём полный текст функции:

```
алг лит нетДвПроб(лит s0)
нач
  лит s, цел p
  s:=s0
  p:=позиция('  ', s)
  нц пока p>0
    удалить(s, p, 1)
    p:=позиция('  ', s)
  кц
  знач:=s
кон
```

```
function noDSp(s: string):
          string;
var p: integer;
begin
  p:=pos('  ', s);
  while p>0 do begin
    delete(s, p, 1);
    p:=pos('  ', s)
  end;
  noDSp:=s
end;
```

Изучите текст функции и ответьте на вопросы.

— Что означает условие p>0 в заголовке цикла?

— Что произойдёт, если удалить строку с вызовом функции позиция (pos) перед циклом? Если удалить такую же строку внутри цикла?

## Логические функции

Программисты часто используют логические функции, возвращающие логические значения («да»/«нет», «истина»/«ложь», True/False). Такие функции полезны для того, чтобы определять, успешно ли выполнена задача или обладают ли данные каким-то свойством.

Мы напишем простую функцию, которая определяет чётность числа — возвращает значение да (в Паскале – True), если число-параметр чётное и нет (False), если нечётное:

```
алг лог Чётное(цел N)
нач
  знач:=(mod(N,2)=0)
кон
```

```
function Even(N: integer):
          boolean;
begin
  Even:=(N mod 2=0);
end;
```

В алгоритмическом языке логические данные относятся к типу **лог**, а в Паскале — к типу boolean. Такой тип указан в заголовке функции для её результата.

В правой части оператора присваивания записано условие: результатом функции будет да, если условие истинно, и нет, если оно ложно. Можно было записать то же самое иначе:

```
если mod(N,2)=0 то
  знач:=да
иначе
  знач:=нет
все
```

```
if N mod 2=0 then
  Even:=True
else
  Even:=False;
```

но эта запись более длинная и её не используют.

Запишите в тетради в развёрнутой форме присваивание значения логической переменной:

```
res:=(a>b+c);
```

Запишите в тетради в краткой форме присваивание значения логической переменной:

```
если a+b>10 то
  res:=нет
иначе
  res:=да
все
```

```
if a+b>10 then
  res:=False
else
  res:=True;
```

Результат, который возвращает логическая функция, можно использовать во всех условиях как обычное логическое значение. Например, так:

```
если Чётное(a) то
  ...
все
```

```
if Even(a) then begin
  ...
end;
```

или так:

```
нц пока Чётное(x) то
  ...
кц
```

```
while Even(x) do begin
  ...
end;
```

Найдите значения переменных a и b, при которых в результате работы этого фрагмента программы будет выведено сообщение Да!:

```
если Чётное(a+3*b) то
  вывод 'Да!'
все
```

```
if Even(a+3*b)then
  writeln('Да!');
```

Найдите значение переменной a, при котором этот цикл выполнится ровно четыре раза:

```
нц пока Чётное(a) и a>5
  a:=div(a,2)
кц
```

```
while Even(a) and (a>5) do begin
  a:=a div 2
end;
```

## Рекурсия

Вы уже знакомы с рекурсивными процедурами, которые вызывают сами себя. Функции тоже могут быть рекурсивными, в некоторых случаях это позволяет записать решение задачи намного проще.

**Рекурсивная функция** — это функция, которая вызывает сама себя.

Вернёмся к задаче вычисления суммы цифр числа. Можно сформулировать алгоритм её решения так: сумма цифр числа N равна значению последней цифры плюс сумма цифр числа, полученного отбрасыванием последней цифры.

*Вход*: натуральное число N.
*Шаг 1*: d := mod(N,10)
*Шаг 2*: M := div(N,10)
*Шаг 3*: s := **сумма цифр числа** M
*Шаг 4*: sum := s + d
*Результат*: sum.

Итак, для того чтобы найти сумму цифр числа, нужно сложить его последнюю цифру и *сумму цифр другого числа*, т. е. выполнить тот же самый алгоритм, только с другими исходными данными (эта строка в записи алгоритма выделена фоном). Получился *рекурсивный алгоритм*, в программе его можно записать в виде рекурсивной функции:

```
алг цел sumDigRec(цел N)
нач
  цел d, sum
  если N=0 то
    знач:=0
  иначе
    d:=mod(N,10)
    sum:=sumDigRec(div(N,10))
    знач:=sum+d
  все
кон
```

```
function sumDigRec(N: integer):
                              integer;
var d, sum: integer;
begin
  if N=0 then
    sumDigRec:=0
  else begin
    d:=N mod 10;
    sum:=sumDigRec(N div 10);
    sumDigRec:=sum+d
  end
end;
```

Изучите текст функции и ответьте на вопросы.
— Зачем добавлен условный оператор с условием N=0?
— Что произойдёт, если удалить этот условный оператор?
— Как можно доказать, что для любого целого числа рекурсия обязательно закончится?

В рекурсивном варианте функции исчез цикл, поэтому можно сделать вывод: рекурсия может заменить цикл. Верно и обратное: любую рекурсивную функцию можно записать без рекурсии, с помощью циклов. Решение с помощью цикла (оно называется **итерационным**) обычно работает быстрее, чем рекурсивное, и требует меньше памяти. Однако рекурсивное решение очень часто короче и проще для понимания.

Используя дополнительные источники, выясните, что означает слово «итерация». От какого слова оно произошло?

## Выводы

- Функция — это подпрограмма, которая возвращает результат (число, строку символов и др.).
- Вызов функции можно использовать в арифметических выражениях и условиях так же, как и переменную того типа, который возвращает функция.
- В теле функции можно вызывать другие функции и процедуры.
- Логическая функция возвращает логическое значение («да»/«нет», `True`/`False`).
- Рекурсивная функция — это функция, которая вызывает сама себя.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Чем функция отличается от процедуры?
2. Определите, какие распоряжения начальника можно считать вызовом процедуры, а какие – вызовом функции:
   а) «Проводите Ивана Ивановича!»
   б) «Принесите, пожалуйста, кофе!»
   в) «Подготовьте годовой отчёт!»
   г) «Постройте конуру для собаки!»
3. Как по тексту программы определить, значение какого типа возвращает функция?
4. Сравните рекурсивное решение задачи о сумме цифр числа и решение с помощью цикла. Какое из них вам больше нравится? Обсудите этот вопрос в классе.
5. Выполните по указанию учителя задания в рабочей тетради.

## Практические работы

Выполните практические работы:
№ 22 «Функции»;
№ 23 «Функции-2».

## ЭОР к главе 4 из Единой коллекции цифровых образовательных ресурсов (school-collection.edu.ru)

Алгоритм поиска наибольшего и наименьшего элементов массива
Алгоритм сортировки массива методом пузырька
Цикл с параметром в алгоритме обработки массива
Обработка массивов на языке Паскаль
Алгоритм поиска числа в массиве
Поиск наибольшего и наименьшего элементов массива в программе на Паскале

# Глава 5
# ЭЛЕКТРОННЫЕ ТАБЛИЦЫ

В курсе информатики 8 класса мы познакомились с электронными таблицами и научились решать простые задачи, использовать формулы и стандартные функции, строить диаграммы. Однако во многих практических задачах этого недостаточно. В этой главе вы узнаете о новых возможностях электронных таблиц, которые позволяют выполнять сложные вычисления, обрабатывать большие массивы данных, решать уравнения, искать оптимальные решения.

## § 26
## Условные вычисления

***Ключевые слова:***

- условные вычисления
- функция IF (ЕСЛИ)
- функция NOT (НЕ)
- функция AND (И)
- функция OR (ИЛИ)

### Функция ЕСЛИ

Как вы знаете, в программировании важную роль играют условные операторы (ветвления), позволяющие выбирать один из двух (или нескольких) вариантов обработки данных. В табличных процессорах тоже возможны **условные вычисления**, при которых в ячейку заносится то или иное значение в зависимости от выполнения какого-то условия.

Предположим, что в книжном интернет-магазине «Бука» доставка покупок бесплатна для тех, кто сделал заказ на сумму более 500 рублей, а для остальных доставка стоит 20% от суммы заказа[1] (рис. 5.1).

---

[1] В этой и следующих таблицах результаты вычислений округляются до целых чисел.

| | A | B | C |
|---|---|---|---|
| 1 | *Заказ* | *Сумма* | *Доставка* |
| 2 | 1234 | 256 р. | 51 р. |
| 3 | 1345 | 128 р. | 26 р. |
| 4 | 1456 | 1024 р. | 0 р. |
| 5 | 1565 | 512 р. | 0 р. |
| 6 | 1576 | 345 р. | 69 р. |

**Рис. 5.1**

Таким образом, есть два варианта вычисления стоимости доставки, поэтому в формулах столбца C нужно использовать ветвление. Алгоритм вычисления значения в ячейке C2 может выглядеть так: «если B2 > 500, то записать в ячейку 0, иначе записать значение B2*0,2». В программе на языке Паскаль мы бы записали:

```
if B2>500 then
     C2:=0
else C2:=B2*0.2;
```

В табличных процессорах для условных вычислений используют функцию IF (ЕСЛИ)[1]:

=IF(B2>500;0;B2*0,2) =ЕСЛИ(B2>500;0;B2*0,2)

У этой функции три аргумента, разделённые точками с запятой:

1) условие (B2>500);
2) значение ячейки в том случае, когда условие истинно (0);
3) значение ячейки в том случае, когда условие ложно (B2*0,2).

Определите значения, которые появятся в ячейках диапазона B2:B6 после ввода формул (рис. 5.2).

| | A | B |
|---|---|---|
| 1 | *Сделано* | *Оплата* |
| 2 | 150 | =IF(A2>200;A2*0,2;A2*0,1) |
| 3 | 240 | =IF(A3>200;A3*0,2;A3*0,1) |
| 4 | 110 | =IF(A4>200;A4*0,2;A4*0,1) |
| 5 | 270 | =IF(A5>200;A5*0,2;A5*0,1) |
| 6 | 200 | =IF(A6>200;A6*0,2;A6*0,1) |

**Рис. 5.2**

[1] В программе *OpenOffice Calc* используются английские названия функций, а в русской версии табличного процессора *Microsoft Excel* — русские. В тексте приводятся два варианта каждой формулы, из которых вам нужно выбрать тот, который соответствует используемой программе.

В условии можно использовать не только числовые, но и символьные данные. Например, фирма «Салют» в этом месяце проводит рекламную акцию: предоставляет скидку 20% на все товары (рис. 5.3).

| | A | B | C | D |
|---|---|---|---|---|
| 1 | *Код товара* | *Фирма* | *Цена* | *Скидка* |
| 2 | 1234 | Салют | 3999 р. | 800 р. |
| 3 | 1345 | Звезда | 2799 р. | |
| 4 | 1456 | Гамбит | 6290 р. | |
| 5 | 1565 | Салют | 3750 р. | 750 р. |
| 6 | 1576 | Гамбит | 1234 р. | |

**Рис. 5.3**

В этом случае в ячейку D2 запишем формулу:

=IF(B2="Салют";C2*20%;"")
=ЕСЛИ(B2="Салют";C2*20%;"")

и скопируем её во все ячейки столбца D. Запись «*20%» означает то же самое, что и «*0,2».

Работник получает премию, составляющую 10% от его зарплаты, только тогда, когда на него не поступает жалоб. Какую формулу нужно записать в ячейку D2 (рис. 5.4)?

| | A | B | C | D |
|---|---|---|---|---|
| 1 | *Фамилия* | *Зарплата* | *Жалобы* | *Премия* |
| 2 | Иванов | 12 000 р. | 0 | 1 200 р. |
| 3 | Петров | 14 000 р. | 2 | 0 р. |

**Рис. 5.4**

## Вложенные вызовы ЕСЛИ

Второй и третий аргументы функции IF могут содержать вложенные вызовы этой функции. Пусть, например, в книжном интернет-магазине «Бука» (см. задачу в начале параграфа) для заказов стоимостью более 200 рублей (но не более 500) стоимость доставки составляет 10% от суммы:

```
if B2>500 then
  C2:=0
else
  if B2>200 then
       C2:=B2*0.1
  else C2:=B2*0.2;
```

В табличном процессоре этот алгоритм запишется в виде

```
=IF(B2>500;0;IF(B2>200;B2*0,1;B2*0,2))
=ЕСЛИ(B2>500;0;ЕСЛИ(B2>200;B2*0,1;B2*0,2))
```

Фирма «Форсаж» занимается доставкой мебели. Если в доме нет лифта, за подъём берут дополнительную плату 100 рублей, а если вес покупки больше 100 кг, то плата за подъём составляет 200 рублей. Какую формулу нужно записать в ячейку D2 (рис. 5.5)?

| | A | B | C | D |
|---|---|---|---|---|
| 1 | *Заказ* | *Вес, кг* | *Лифт* | *Цена подъёма* |
| 2 | 1234 | 150 | да | 0 р. |
| 3 | 1235 | 120 | нет | 200 р. |
| 4 | 1236 | 89 | нет | 100 р. |

**Рис. 5.5**

## Сложные условия (И, ИЛИ, НЕ)

Первый аргумент функции IF (ЕСЛИ) может быть сложным условием, которое строится с помощью функций AND (И) — логическое умножение, OR (ИЛИ) — логическое сложение и NOT (НЕ) — отрицание.

**Задача 1.** Пусть в примере на рис. 5.1 бесплатная доставка распространяется только на заказы, у которых номер меньше 1500 и сумма больше 500 рублей.

В этом случае в ячейку C2 нужно записать такую формулу:

```
=IF(AND(A2<1500;B2>500);0;B2*0,2)
=ЕСЛИ(И(A2<1500;B2>500);0;B2*0,2)
```

Здесь использовано сложное условие AND(A2<1500; B2>500), которое истинно только при одновременном выполнении двух условий: A2<1500 и B2>500.

Фирма «Мираж» занимается доставкой питьевой воды. Если в доме нет лифта и заказано более 5 бутылей, то за подъём берут дополнительную плату 20 рублей с каждой бутыли. Какую формулу нужно записать в ячейку D2 (рис. 5.6)? Предложите несколько вариантов решения задачи.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | *Заказ* | *Заказано, бут.* | *Лифт* | *Цена подъёма* |
| 2 | 1234 | 15 | да | 0 р. |
| 3 | 1235 | 12 | нет | 240 р. |

**Рис. 5.6**

**Задача 2.** На трассе разрешается ехать со скоростью от 40 км/ч до 110 км/ч. Радар записывает скорость проезжающих машин, а видеокамера — их номера. Водителям, которые едут со скоростью, меньшей минимальной или большей максимальной, нужно выписать штраф 500 рублей. Требуется построить электронную таблицу такого вида (рис. 5.7).

| | A | B | C |
|---|---|---|---|
| 1 | *Номер* | *Скорость* | *Штраф* |
| 2 | A134AA | 150 | 500 р. |
| 3 | B235BB | 80 | |
| 4 | A157AB | 90 | |
| 5 | A198CX | 30 | 500 р. |
| 6 | K754MM | 180 | 500 р. |

**Рис. 5.7**

Штраф выписывается, когда верно одно из двух условий (скорость меньше 40 км/ч ИЛИ скорость больше 110 км/ч). Формула в ячейке C2 может быть записана так:

```
=IF(OR(B2<40;B2>110);500;"")
=ЕСЛИ(ИЛИ(B2<40;B2>110);500;"")
```

Компания «Уют» проводит акцию: те, кто купил не меньше 5 стульев или не меньше 2 столов, получают приз. Какую формулу нужно записать в ячейку D2 (рис. 5.8)? Предложите несколько вариантов решения задачи.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | *Заказ* | *Стульев* | *Столов* | *Приз* |
| 2 | 1234 | 6 | 0 | 1 |
| 3 | 1235 | 2 | 3 | 1 |
| 3 | 1236 | 4 | 1 | 0 |

**Рис. 5.8**

Логические функции AND (И) и OR (ИЛИ) могут содержать более двух условий, которые перечисляются через точку с запятой.

**Задача 3.** На III тур соревнований нужно отобрать участников, которые набрали по сумме двух первых туров не менее 180 баллов или получили 100 баллов хотя бы в одном туре (рис. 5.9).

| | A | B | C | D |
|---|---|---|---|---|
| 1 | *Участник* | *I тур* | *II тур* | *III тур* |
| 2 | Иванов И.И. | 100 | 70 | + |
| 3 | Петров П.П. | 80 | 75 | |
| 4 | Сидоров С.С. | 65 | 100 | + |
| 5 | Куницын К.К. | 95 | 90 | + |
| 6 | Васильев В.В. | 80 | 90 | |

**Рис. 5.9**

Формула в ячейке D2 содержит сложное условие, где операция OR (ИЛИ) объединяет три условия:

```
=IF(OR(B2+C2>=180;B2=100;C2=100);"+";"")
=ЕСЛИ(ИЛИ(B2+C2>=180;B2=100;C2=100);"+";"")
```

Функция NOT (НЕ) выполняет логическое отрицание. Она используется на практике довольно редко.

## Выводы

- Условные вычисления заключаются в том, что в ячейку таблицы могут быть записаны различные значения в зависимости от выполнения некоторого условия.

- Функция IF (ЕСЛИ) принимает три аргумента:
  1) условие;
  2) значение в случае истинного условия;
  3) значение в случае ложного условия.
- Функции AND (И), OR (ИЛИ) и NOT (НЕ) выполняют логические операции с аргументами (логическими значениями).

Нарисуйте в тетради интеллект-карту этого параграфа.

### Интересные сайты

**excelworld.ru** — «Мир Excel»
**planetaexcel.ru** — «Планета Excel»
**wiki.openoffice.org/wiki/RU/kb/module/calc** — справка по OpenOffice Calc
**help.libreoffice.org/Calc/Welcome_to_the_Calc_Help/ru** — справка по LibreOffice Calc

### Практические работы

Выполните практические работы:
№ 24 «Стандартные функции»;
№ 26 «Таблицы истинности»;
№ 26 «Условные вычисления»;
№ 26-а «Условные вычисления»;
№ 27 «Сложные условия».

# § 27
# Обработка больших массивов данных

***Ключевые слова***:

- вспомогательный столбец
- функция COUNT (СЧЁТ)
- функция COUNTIF (СЧЁТЕСЛИ)
- функция SUMIF (СУММЕСЛИ)
- функция AVERAGEIF (СРЗНАЧЕСЛИ)
- рабочая книга
- лист
- ссылка на другой лист

Табличные процессоры позволяют обрабатывать большие массивы данных, например содержащие несколько тысяч строк. В таких случаях полезно знать специальные приёмы, с которыми вы познакомитесь в этом параграфе.

## Выделение диапазонов

Представьте себе, что вам нужно выделить для обработки 1000 ячеек, расположенных в одном столбце. Конечно, в этом случае можно использовать и стандартный способ: нажать левую кнопку мыши на первой ячейке диапазона и протащить мышь до последней ячейки, однако это очень неудобно.

Существуют и другие методы. Например, можно выделить щелчком мышью первую ячейку, затем прокрутить таблицу до последней строки и щёлкнуть на последней ячейке при нажатой клавише *Shift*.

Можно обойтись вообще без прокрутки. Выделив первую ячейку, нажмите клавиши *Ctrl*+*Shift*+↓. При этом выделяются все элементы данного столбца вниз до первой пустой ячейки. Аналогично выделяются данные в нескольких столбцах: для этого на первом шаге нужно выделить несколько ячеек в верхней строке диапазона.

Можно выделить несколько диапазонов одновременно: второй и следующий диапазоны выделяются при нажатой клавише *Ctrl*.

## Использование вспомогательных столбцов

**Задача 1.** В большой таблице записаны данные участников спортивных соревнований: фамилия, год рождения и вес. Требуется найти количество спортсменов, которые родились в 2004 году. Выполнять сортировку не разрешается.

Один из способов решения этой задачи — добавить в таблицу *вспомогательный столбец*, в котором напротив каждого участника будет записана единица, если его год рождения равен 2004, или 0, если не равен (рис. 5.10).

| | A | B | C | D |
|---|---|---|---|---|
| 1 | *Участник* | *Год рождения* | *Вес, кг* | |
| 2 | Иванов И.И. | 2004 | 56 | 1 |
| 3 | Петров П.П. | 2003 | 62 | 0 |
| 4 | Сидоров С.С. | 2004 | 58 | 1 |
| 5 | … | … | … | |

**Рис. 5.10**

Используя материал предыдущего параграфа, запишите формулу, которую нужно добавить в ячейку D2. Предложите два варианта решения задачи.

Формулу из D2 нужно скопировать во все ячейки столбца D. Напомним, что для такого копирования достаточно выполнить двойной щелчок мышью на маркере заполнения выделенной верхней ячейки с формулой.

Теперь количество участников 2004 года рождения находится как сумма значений в столбце D (функция SUM или СУММ). Если в таблице 1000 строк (со 2-й по 1001-ю), формула запишется так:

=SUM(D2:D1001) =СУММ(D2:D1001)

**Задача 2.** Для таблицы из задачи 1 требуется найти средний вес тех, кто родился в 2004 году.

Здесь тоже можно использовать вспомогательный столбец. Выведем в нём вес участника, если год рождения равен 2004, или пустую строку (""), если год рождения другой (рис. 5.11).

| | A | B | C | D |
|---|---|---|---|---|
| 1 | *Участник* | *Год рождения* | *Вес, кг* | |
| 2 | Иванов И.И. | 2004 | 56 | 56 |
| 3 | Петров П.П. | 2003 | 62 | |
| 4 | Сидоров С.С. | 2004 | 58 | 58 |
| 5 | ... | ... | ... | |

**Рис. 5.11**

Используя материал предыдущего параграфа, запишите формулу, которую нужно добавить в ячейку D2. Предложите два варианта решения задачи.

После этого остаётся подсчитать среднее значение ячеек столбца D с помощью функции AVERAGE (СРЗНАЧ), например:

=AVERAGE(D2:D1001) =СРЗНАЧ(D2:D1001)

Эта функция вычисляет среднее только по тем ячейкам, в которых содержится числовое значение, пустые ячейки не учитываются.

## Стандартные функции

Во многих случаях можно обойтись без вспомогательных столбцов, если использовать стандартные функции.

Функция COUNT (СЧЁТ) определяет количество числовых ячеек диапазона, при этом пустые и текстовые ячейки не учитываются. Например, для таблицы на рис. 5.11 найти количество участников 2004 года рождения можно было по формуле:

=COUNT(D2:D1001) =СЧЁТ(D2:D1001)

Ещё лучше использовать для этой цели функцию COUNTIF (СЧЁТЕСЛИ) — она считает ячейки диапазона, удовлетворяющие какому-то условию. Например, количество участников 2004 года можно было вычислить даже без использования вспомогательного столбца:

=COUNTIF(B2:B1001;"=2004") =СЧЁТЕСЛИ(B2:B1001;"=2004")

Второй аргумент этой функции — условие, записанное в кавычках.

По формуле

=COUNTIF(C2:C1001;">57") =СЧЁТЕСЛИ(C2:C1001;">57")

мы определяем количество участников, вес которых больше 57 кг.

Если после точки с запятой стоит число, это значит, что подсчитывается количество ячеек, равных этому числу. То есть вместо "=2004" можно записать просто 2004:

=COUNTIF(B2:B1001;2004) =СЧЁТЕСЛИ(B2:B1001;2004)

Функция COUNTIF (СЧЁТЕСЛИ) не может работать со сложными условиями, т. е. в условиях нельзя использовать операции И, ИЛИ, НЕ.

Требуется найти количество учеников 2004 года рождения, которые весят больше 60 кг. Для этого используется вспомогательный столбец D. Какую формулу нужно записать в ячейку D2? Как затем решить задачу?

Используя дополнительные источники, выясните, как работает функция COUNTIFS (СЧЁТЕСЛИМН).

Для того чтобы найти долю участников 2004 года рождения в списке (см. рис. 5.11), нужно разделить их количество на общее число рабочих строк:

=COUNTIF(B2:B1001;2004)/COUNT(B2:B1001)
=СЧЁТЕСЛИ(B2:B1001;2004)/СЧЁТ(B2:B1001)

и установить в ячейке процентный формат с нужным числом знаков в дробной части.

Функции SUMIF (СУММЕСЛИ) и AVERAGEIF (СРЗНАЧЕСЛИ) тоже позволяют решать некоторые задачи без вспомогательных столбцов. Например, найти суммарный вес всех участников 2004 года рождения (см. рис. 5.11) можно с помощью одной формулы:

```
=SUMIF(B2:B1001;2004;C2:C1001)
=СУММЕСЛИ(B2:B1001;2004;C2:C1001)
```

Функция SUMIF принимает три аргумента:

1) диапазон, по которому выполняется проверка условия (B2:B1001);
2) условие, которое проверяется ("=2004");
3) диапазон, по которому вычисляется сумма (C2:C1001).

Приведённая только что формула означает: «если ячейка из диапазона B2:B1001 равна 2004, включить в сумму значение соответствующей ячейки из диапазона C2:C1001».

Аналогично вычисляется средний вес этих же участников:

```
=AVERAGEIF(B2:B1001;2004;C2:C1001)
=СРЗНАЧЕСЛИ(B2:B1001;2004;C2:C1001)
```

## Работа с листами

Когда данных много, неудобно размещать всё на одном листе. Возможно, вы знаете, что файл электронной таблицы — это **рабочая книга**, которая может состоять из многих листов. Каждый лист — это отдельная электронная таблица, причём с одного листа вы можете обращаться к данным других листов.

В левом нижнем углу окна расположены элементы управления листами — кнопки для перехода по листам, ярлычки листов и кнопка для создания нового листа (рис. 5.12).

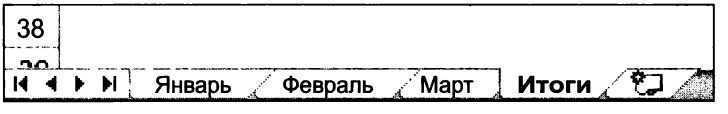



**Рис. 5.12**

Используя контекстное меню ярлычков, которое появляется при нажатии правой кнопки мыши, можно добавлять, удалять, переименовывать листы. Порядок расположения листов изменяется с помощью мыши (перетаскиванием).

Предположите, как можно скопировать лист. Проверьте свою догадку в программе.

Если нужно использовать данные другого листа, ссылка должна содержать имя этого листа и адрес ячейки (или диапазона) на

листе. Эти части ссылки в программе *Calc* отделяются точкой, а в *Excel* — восклицательным знаком, например:

=Январь.B2+Февраль.B2+Март.B2
=Январь!B2+Февраль!B2+Март!B2

Если имя листа содержит пробелы, оно заключается в одиночные апострофы:

=SUM('К оплате'.B2:C4)  =СУММ('К оплате'!B2:C4)

## Выводы

- Для хранения промежуточных результатов можно использовать вспомогательные столбцы таблицы.
- Функция COUNT (СЧЁТ) вычисляет количество ячеек диапазона, содержащих числовые значения.
- Функция COUNTIF (СЧЁТЕСЛИ) вычисляет количество ячеек диапазона, удовлетворяющих условию.
- Функция SUMIF (СУММЕСЛИ) суммирует ячейки одного диапазона, если соответствующие ячейки другого диапазона удовлетворяют условию.
- Функция AVERAGEIF (СРЗНАЧЕСЛИ) находит среднее арифметическое ячеек одного диапазона, если соответствующие ячейки другого диапазона удовлетворяют условию.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

Выполните по указанию учителя задания в рабочей тетради.

## Практические работы

Выполните практические работы:
№ 28 «Обработка больших массивов данных»;
№ 28-а «Обработка больших массивов данных».

# § 28
# Численные методы

***Ключевые слова:***

- аналитическое решение
- численный метод
- приближённое решение
- приближённый метод
- начальное приближение
- итерационный метод
- подбор параметра
- целевая ячейка
- изменяемая ячейка

## Что такое численные методы?

На уроках математики вас учили искать решение уравнения в виде формулы, выражающей неизвестную величину через известные. Например, решение уравнения $ax + b = 1$ при $a \neq 0$ можно записать в виде $x = (1 - b)/a$. Такое решение называется **аналитическим**, оно может быть использовано для теоретического исследования (изучения влияния исходных данных на результат).

Пусть $a > 0$, $b < 1$ и $x = (1 - b)/a$. Как при этом изменится значение $x$, если
а) увеличить $a$, не изменяя $b$;
б) уменьшить $b$, не изменяя $a$?

Однако не все уравнения можно на современном уровне развития математики решить аналитически. Иногда решение есть, но очень сложное. Часто значительно легче получить численное решение, т. е. найти число, которое является решением при конкретных исходных данных (а не формулу!).

**Численное решение** — это решение задачи для конкретных исходных данных.

**Численный метод** — это метод, который применяется для поиска численного решения.

Как правило, численные методы дают не точное, а **приближённое решение**, т. е. решение с некоторой ошибкой. Если эта ошибка будет достаточно мала (например, при вычислении расстояния между городами ошибка составит не более 1 м), то такое решение всех устроит.

**Приближённый метод** — это метод, который позволяет найти решение задачи с некоторой (допустимой) ошибкой (погрешностью).

**Погрешность** — отклонение значения величины, полученного в результате измерений или вычислений, от её истинного (действительного) значения.

Рассмотрим уравнение $x = \cos x$, для которого решение в виде формулы получить нельзя.

Как бы вы решали такое уравнение, если бы его очень нужно было решить?

В этом случае можно использовать, например, **графический метод** решения: построить по точкам графики функций, стоящих в левой и правой частях равенства, и посмотреть, где они пересекаются. В точке пересечения значения функций равны, значит, соответствующее значение $x$ — это решение уравнения (рис. 5.13). Решение можно уточнять, уменьшая шаг построения графика до получения требуемой точности.

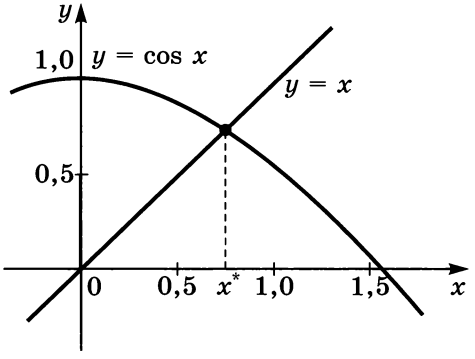



**Рис. 5.13**

Если нужна высокая точность, графический метод требует очень большого объёма вычислений, который можно поручить компьютеру. Однако нужно как-то учесть, что компьютер не способен «посмотреть» на график, и для уточнения решения может использовать только числовые данные. Компьютерный алгоритм решения уравнения может выглядеть примерно так:

1) выбрать отрезок $[a_0; b_0]$ для поиска решения (обычно предполагается, что на этом отрезке есть решение, и притом только одно);
2) с помощью некоторого алгоритма уточнить решение, перейдя к меньшему отрезку $[a, b]$;
3) повторять шаг 2, пока длина отрезка, в который мы «загнали» решение, не станет достаточно малой.

Здесь не совсем ясно, что значит «пока длина отрезка не станет достаточно малой». Обычно задаётся **допустимая погрешность** $\varepsilon$: отклонение полученного решения от истинного $x$ не должно быть больше, чем $\varepsilon$. Когда мы уменьшаем отрезок, внутри которого находится решение, мы увеличиваем точность: при выборе любой точки этого отрезка в качестве решения ошибка не превышает длины отрезка.

Установлено, что корень уравнения находится на отрезке $[a, b]$. Какую точку этого отрезка лучше всего считать решением? Какова будет в этом случае наибольшая возможная погрешность?

Цикл (повторение шага 2 алгоритма) нужно остановить тогда, когда длина отрезка станет меньше, чем $2\varepsilon$. В этом случае при выборе $x = (a + b)/2$ погрешность будет минимальной: не больше, чем $\varepsilon$.

Часто используется другой вариант, когда требуется знать только одну точку вблизи решения — начальное приближение.

**Начальное приближение** — это начальное значение неизвестной величины, которое уточняется с помощью приближённого метода.

Процедуры уточнения (они бывают разные!) работают примерно так:
1) выбрать начальное приближение $x_0$ около решения;
2) с помощью некоторого алгоритма перейти к следующему приближению $x$, которое находится ближе к точному решению $x^*$;
3) повторять шаг 2, пока изменение значения $x$ не станет меньше, чем допустимая погрешность $\varepsilon$.

При каждом уточнении повторяются одинаковые действия, поэтому такие методы называют итерационными.

Вспомните, что означает слово «итерация».

Приближённые методы имеют ряд *недостатков*:
- получается *приближённое* решение, а не точное; это значит, что нельзя написать $x^* = 1{,}2345$, нужно использовать знак приближённого равенства: $x^* \approx 1{,}2345$;
- мы получаем не формулу, а число, по которому невозможно оценить, как меняется решение при изменении исходных данных (сложно выявить зависимость от параметра);
- объём вычислений может быть слишком велик, это не позволяет использовать приближённые методы в системах, где нужно очень быстро получить результат;
- не всегда можно оценить погрешность (ошибку) результата.

Однако в реальных задачах очень часто аналитическое решение получить невозможно или очень трудно, а приближённые методы позволяют быстро решить задачу с заданной точностью.

## Решение уравнений подбором параметра

Для решения уравнений можно использовать табличный процессор, например *OpenOffice Calc* (или *LibreOffice Calc*) или *Microsoft Excel*. Обычно сначала строится график функции, по которому определяют количество корней уравнения и их примерное расположение; затем используется команда *Подбор параметра*. Далее мы будем рассматривать программу *Calc* из пакета *OpenOffice*, указывая на незначительные отличия *Excel*.

Найдём все решения уравнения $x^2 = \cos x$ на отрезке [–2; 2]. Сначала определим, сколько решений имеет это уравнение. Для этого построим графики обеих функций: $f_1(x) = x^2$ и $f_2(x) = \cos x$. Составим таблицу, где значения $x$ расположены в столбце A, значения функций — в столбцах B и C. Выберем шаг изменения $x$, равный 0,25.

Можно ли выбрать другой шаг? Что изменится (улучшится, ухудшится), если увеличивать шаг? Уменьшать шаг?

Введём два первых значения $x$ в столбце A, выделим их и «протащим» маркер заполнения вниз до тех пор, пока последнее значение в столбце A не станет равно 2 (рис. 5.14).

| | A | B | C |
|---|---|---|---|
| 1 | $x$ | $f_1(x)$ | $f_2(x)$ |
| 2 | −2 | | |
| 3 | −1,75 | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |
| 8 | | | |

⇒

| | A | B | C |
|---|---|---|---|
| 1 | $x$ | $f_1(x)$ | $f_2(x)$ |
| 2 | −2 | =A2^2 | =COS(A2) |
| 3 | −1,75 | | |
| 4 | −1,5 | | |
| 5 | −1,25 | | |
| 6 | −1 | | |
| 7 | −0,75 | | |
| 8 | −0,5 | | |

**Рис. 5.14**

Введём в ячейки B2 и C2 формулы, вычисляющие значения обеих функций для значения $x$ в ячейке A2. Затем выделим формулы и скопируем их вниз до конца таблицы (например, сделав двойной щелчок на маркере заполнения).

Теперь выделяем все данные и строим диаграмму типа *Диаграмма X-Y* (*Точечная*) — рис. 5.15.

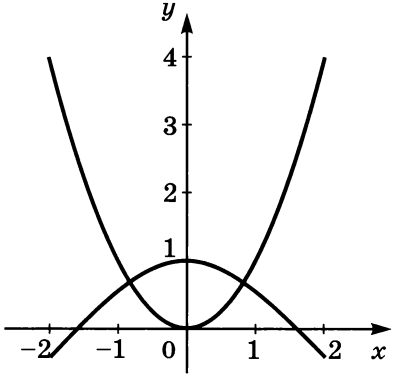



**Рис. 5.15**

Изучите график на рис. 5.15. Сколько решений имеет уравнение на этом отрезке? Найдите начальные приближения для поиска всех решений — значения $x$, расположенные рядом с решениями.

Найдём решение вблизи точки $x = -1$. В отдельных ячейках (чтобы не портить графики!) введём начальное приближение и вычислим значения обеих функций (в ячейках F2 и G2). Кроме того, в H2 запишем формулу для вычисления разности значений функций (рис. 5.16).

| | E | F | G | H |
|---|---|---|---|---|
| 1 | $x$ | $f_1(x)$ | $f_2(x)$ | $f_1(x) - f_2(x)$ |
| 2 | −1 | =E2^2 | =COS(E2) | =F2−G2 |
| 3 | | | | |

**Рис. 5.16**

Чему должна быть равна разность двух функций в точке $x$, которая является решением уравнения?

Табличные процессоры умеют решать задачу подбора параметра, которая формулируется так: *установить в ячейке X значение V, изменяя значение ячейки Y*. Например, в нашем случае нужно установить в ячейке H2 значение 0, изменяя E2. Ячейка H2 называется **целевой**, потому что наша цель — получить в ней определённое значение (ноль). Ячейка E2, значение которой мы хотим подобрать, — это **изменяемая ячейка**.

В главном меню выбираем пункт *Сервис* → *Подбор параметра*[1] и вводим эти данные (рис. 5.17):

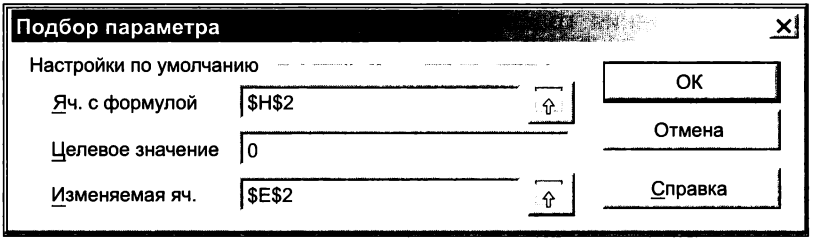



**Рис. 5.17**

После нажатия на кнопку *OK* найденное решение уравнения будет записано в ячейку E2.

Как же найти второе решение? Для этого нужно выбрать другое начальное приближение, например $x = 1$. В остальном порядок действий не меняется.

Можно ли вместо уравнения $x^2 = \cos x$ решать уравнение $x^2 - \cos x = 0$? Чем лучше такой подход?

---

[1] В программе *Excel* — пункт *Подбор параметра* в группе *Анализ что-если* на вкладке *Данные*.

## Выводы

- Аналитическое решение — это решение задачи в виде формулы, которая выражает неизвестную величину через известные.
- Численное решение — это решение задачи для конкретных исходных данных.
- Численный метод — это метод, который применяется для поиска численного решения задачи.
- Приближённый метод — это метод, который позволяет найти решение задачи с некоторой (допустимой) ошибкой.
- Начальное приближение — это начальное значение неизвестной величины, которое уточняется с помощью приближённого метода.
- Для поиска приближённых решений уравнений в электронных таблицах используют подбор параметра.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Сравните численные и аналитические методы решения уравнений. В чем достоинства и недостатки каждого подхода?
2. Какие методы называются приближёнными? В каких случаях они используются?
3. Какие методы называют итерационными?
4. После построения графиков выяснилось, что уравнение имеет три решения. Как вы будете их искать?
5. Найдите все решения уравнения:
   а) $x^2 = 5\cos(x - 1)$;
   б) $x^2 - \sin x = 1$;
   в) $2x^3 - 15\sin x + 0{,}5x - 5 = 0$.
6. Вы хотите положить в банк 20000 рублей. Ежегодно на сумму вклада начисляются проценты. Например, если ставка составляет 10% годовых, каждый год сумма увеличивается в 1,1 раза. При какой ставке сумма вырастет до 30000 рублей за 5 лет?
7. Выполните по указанию учителя задания в рабочей тетради.

## Подготовьте сообщение

а) «Решение уравнений методом деления отрезка пополам»
б) «Решение уравнений методом хорд»

## Практическая работа

Выполните практическую работу № 29 «Решение уравнений».

# § 29
# Оптимизация

***Ключевые слова:***

- оптимальное решение
- оптимизация
- целевая функция
- ограничения
- глобальный минимум
- локальный минимум

## Что такое оптимизация?

**Оптимизация** — это поиск наилучшего (**оптимального**) решения задачи при заданных ограничениях.

С точки зрения математики, цель оптимизации — выбрать значение неизвестной величины $x$ (или нескольких неизвестных величин) наилучшим образом.

Чтобы поставить задачу оптимизации, нужно выбрать **целевую функцию** $f(x)$, которая позволяет сравнивать решения. Оптимальным называется такое решение, при котором целевая функция достигает максимума (если это «доходы», «прибыль») или минимума («расходы», «потери»):

$$f(x) \to \max \quad \text{или} \quad f(x) \to \min.$$

Феофан хочет построить загородный дом, так чтобы его расходы были минимальными. Какое «очевидное» решение есть у этой задачи?

Чтобы задача оптимизации была хорошо поставленной, нужно ввести **ограничения**. Например, человек хочет построить загородный дом за минимальную цену. Здесь целевая функция — это общая цена строительства, нужно сделать её минимальной. Очевидно, что лучшее решение — не строить дом вообще, при этом расходы будут равны нулю. Такое «оптимальное» решение получено потому, что мы не задали ограничения (например, нужен двухэтажный дом с гаражом, фонтаном и садом).

## Локальные и глобальный минимумы

По традиции обычно рассматривают задачу поиска минимума. Если нужно найти максимум, просто меняют знак функции: значение функции $f(x)$ максимально там, где значение функции $-f(x)$ минимально.

В математике различают локальный («местный») и глобальный («общий») минимум (рис. 5.18).

**Локальный минимум** — минимальное значение функции в окрестности некоторой точки.

**Глобальный минимум** — минимальное значение функции при заданных ограничениях.

В точках $x_1$, $x_2$ и $x_3$ функция, график которой показан на рис. 5.18, имеет локальные минимумы. Например, слева и справа от точки $x_1$ значения функции больше, чем $f(x_1)$, т. е. в окрестности точки $x_1$ функция имеет минимальное значение именно при $x = x_1$.

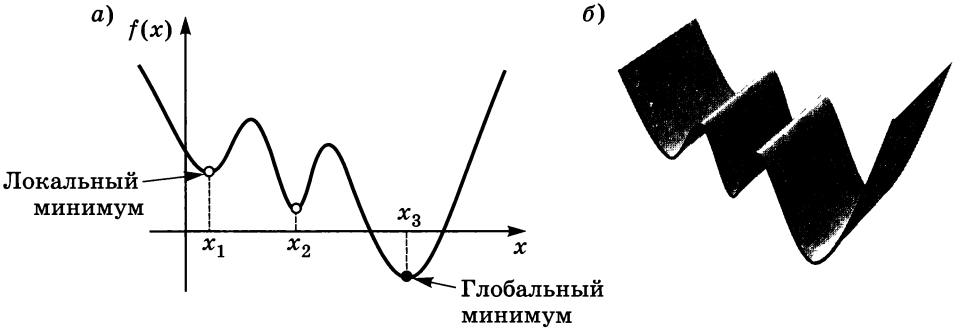



**Рис. 5.18**

Минимум в точке $x_3$ — глобальный, потому что здесь функция имеет наименьшее значение во всей рассматриваемой области.

Очевидно, что нас всегда интересует глобальный минимум. Однако большинство существующих методов оптимизации[1] предназначены именно для поиска локальных минимумов вблизи заданной начальной точки (**начального приближения**).

Можно представить себе, что график функции — это срез поверхности, на которую устанавливается шарик в некоторой начальной точке (см. рис. 5.18, *б*); куда этот шарик скатится, такой минимум и будет найден.

---

Результат поиска локального минимума зависит от начального приближения.

---

## Пример: оптимальная раскройка листа

Рассмотрим пример практической задачи оптимизации. В углах квадратного листа железа, сторона которого равна 1 м, вырезают четыре квадрата со стороной $x$. Затем складывают получившуюся развёртку (по штриховым линиям на рис. 5.19), сваривают швы и таким образом получают бак.

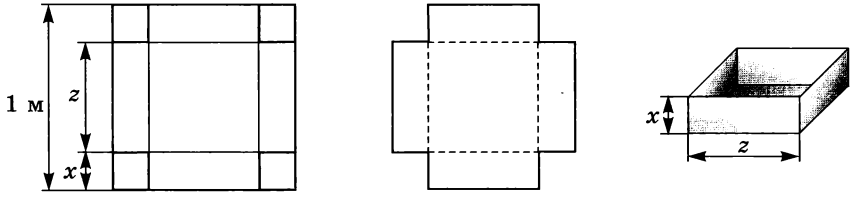



**Рис. 5.19**

[1] Для некоторых типов функций существуют методы глобальной оптимизации, но они сложны и выходят за рамки школьного курса.

Требуется выбрать размер выреза $x$ так, чтобы получился бак наибольшего объёма.

Для того чтобы поставить задачу оптимизации, нужно:

1) определить целевую функцию: в данном случае выразить объём бака через неизвестную величину $x$;
2) задать ограничения на возможные значения $x$.

Легко видеть, что основание получившегося бака — это квадрат со стороной $z$, а его высота равна $x$. Однако величина $z$ зависит от $x$: $z = 1 - 2x$, поэтому объём бака вычисляется по формуле $V(x) = x(1 - 2x)^2$. Это и есть целевая функция, для которой нужно найти максимум.

Какими могут быть минимальное и максимальное значения $x$ в этой задаче?

Заметим, что при $x = 0$ и $x = 0{,}5$ объём бака равен нулю (в первом случае равна нулю высота, во втором — площадь основания). Таким образом, нужно искать максимум целевой функции $V(x) = x(1 - 2x)^2$ на отрезке [0; 0,5].

## Поиск оптимального решения с помощью электронных таблиц

В *OpenOffice Calc* встроенный модуль оптимизации работает только для линейных функций. Чтобы решить рассмотренную выше задачу с баком, где целевая функция нелинейная, нужно использовать расширение *Solver for Nonlinear Programming* (оно входит в стандартную поставку *LibreOffice*). В табличном процессоре *Excel* оптимизация выполняется с помощью стандартной надстройки[1] *Поиск решения*.

Сначала построим график целевой функции $V(x) = x(1 - 2x)^2$, как мы это делали при решении уравнения (рис. 5.20).

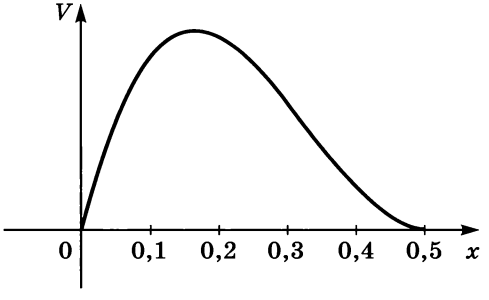



**Рис. 5.20**

По графику на рис. 5.20 определите начальное приближение — значение $x$, рядом с которым находится максимум функции.

[1] Надстройкой в программе *Excel* называют набор функций, который хранится в виде отдельного файла. Эти функции запускаются с помощью кнопок на ленте.

Выделим в таблице две ячейки (например, E2 и F2), в первую запишем начальное приближение, а во вторую — формулу для вычисления объёма для этого значения $x$ (рис. 5.21).

| | E | F |
|---|---|---|
| 1 | $x$ | *Объём* |
| 2 | 0,2 | 0,072 |

**Рис. 5.21**

Задача оптимизации формулируется так: «*найти максимум (или минимум) целевой функции в ячейке X, изменяя значения ячеек Y при ограничениях Z*». В нашей задаче целевая ячейка — F2 (нужно найти её максимум), изменяемая ячейка — E2.

Выбираем в главном меню программы *Calc*[1] пункт *Сервис* → *Поиск решения*, в появившемся окне вводим адреса целевой и изменяемой ячеек (для этого можно установить курсор в поле ввода и щёлкнуть в нужной ячейке) и ограничения (рис. 5.22).


**Рис. 5.22**

[1] В программе Excel используется надстройка *Поиск решения* (вкладка *Данные*).

После щелчка на кнопке *Решить* в ячейке E2 будет записано оптимальное значение $x$, а в целевой ячейке — соответствующее (максимальное) значение объёма.

Надстройка *Поиск решения* позволяет:

- находить максимум или минимум целевой функции;
- решать уравнения, задавая желаемое значение целевой функции;
- использовать несколько изменяемых ячеек и диапазонов; например, запись A2:A6;B15 в списке изменяемых ячеек означает «изменять все ячейки диапазона A2:A6 и ячейку B15»;
- использовать ограничения типа «меньше или равно», «больше или равно», «равно», «целое» и «двоичное» (только 0 или 1);
- устанавливать ограничение на диапазоны; например, ограничение B2:B4>=0 означает, что значения во всех ячейках этого диапазона должны быть больше или равны нулю; ограничение B2:B4>=C2:C4 означает, что значения во всех ячейках диапазона B2:B4 должны быть больше или равны значениям в соответствующих ячейках диапазона C2:C4.

Кнопка *Параметры* позволяет опытным пользователям изменять настройки алгоритма оптимизации.

## Выводы

- Оптимизация — это поиск наилучшего (оптимального) решения задачи при заданных ограничениях.
- Для того чтобы задача оптимизации была хорошо поставленной, нужно определить целевую функцию, позволяющую сравнивать два решения, и ограничения.
- Локальный минимум — это минимально возможное значение функции в некоторой окрестности.
- Глобальный минимум — минимально возможное значение функции при данных ограничениях.
- Большинство численных методов позволяет найти только локальный минимум.
- Результат поиска локального минимума зависит от начального приближения.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Что такое целевая функция?
2. Почему выражение «самый оптимальный» не имеет смысла?
3. Что можно сказать о рекламной фразе «Этот крем обеспечивает оптимальный цвет лица»?
4. Зачем нужны ограничения в задаче оптимизации?

5. В чём разница между понятиями «локальный минимум» и «глобальный минимум»?
6. Почему результат решения задачи оптимизации чаще всего зависит от выбора начального приближения? В каком случае не зависит?
*7. Как задачу решения уравнения можно сформулировать в виде задачи оптимизации?
8. Определите, на какой минимальный срок (целое число лет) необходимо поместить в банк 20 000 рублей, чтобы получить не менее 100 000 рублей при ставке 10% годовых.
9. Найдите стороны прямоугольного треугольника, имеющего наибольшую возможную площадь, если известно, что сумма длин его катетов равна 10.
10. Выполните по указанию учителя задания в рабочей тетради.

## Практическая работа

Выполните практическую работу № 30 «Оптимизация».

## ЭОР к главе 5 из Единой коллекции цифровых образовательных ресурсов (school-collection.edu.ru)

Интерактивный задачник, раздел «Логические формулы в электронных таблицах»

Основные функции MS Excel

Методы вычислений. Модуль 1. Численные методы решения нелинейных уравнений. Нахождение арифметического корня натуральной степени с заданной точностью

# Глава 6
# БАЗЫ ДАННЫХ

## § 30
## Информационные системы

***Ключевые слова:***

- информационная система (ИС)
- база данных (БД)
- система управления базой данных (СУБД)
- локальная ИС
- автономность
- файл-серверная ИС
- клиент-серверная ИС
- распределённая ИС

### Что такое информационная система?

Для современного человека очень важно быстро получать необходимую ему информацию. В этом нам помогают **информационные системы (ИС)**, с помощью которых мы узнаём прогноз погоды и расписание поездов, определяем маршруты путешествий, заказываем билеты на самолеты, бронируем номера в гостиницах и т. п.

Информационные системы работают с огромными объёмами данных (они могут составлять несколько гигабайт и даже терабайт), которые размещаются во внешней памяти компьютера (например, на жёстких дисках) или в облачных хранилищах. Данные хранятся таким образом, чтобы их было легко искать и изменять. Такие наборы данных называются *базами данных*.

---

**База данных (БД)** — это специальным образом организованная совокупность[1)] данных о некоторой предметной области, хранящаяся во внешней памяти компьютера.

---

Данные сами по себе бесполезны, если мы не умеем с ними работать. Поэтому нужны программы, которые позволяют искать и изменять эти данные.

---

1) Термин «совокупность» здесь обозначает множество элементов, обладающих общими свойствами.

**Система управления базами данных (СУБД)** — это программное обеспечение, которое позволяет выполнять все необходимые операции с базой данных.

Вы знаете, что данные в компьютерном формате – это двоичные коды, которые могут обозначать всё, что угодно. Поэтому СУБД должна «знать» формат файлов (что где записано). В первых информационных системах каждая база данных имела свой собственный формат, который придумывал её автор. Это очень неудобно, потому что для каждой ИС нужно разрабатывать специальную программу для работы с базой данных. Более того, если мы изменяем, например, размер данных или порядок их хранения в файле, надо переделывать все программы, работающие с этими данными.

Поэтому постепенно перешли к использованию специальных программ (они и получили название СУБД), которые хранят и обрабатывают любые данные, независимо от их содержания, и могут применяться в самых разных задачах. СУБД:

- выполняют поиск данных;
- позволяют редактировать данные;
- выполняют несложные расчёты;
- обеспечивают *целостность* (правильность, непротиворечивость) данных;
- восстанавливают данные после сбоев.

Как правило, пользователь работает с СУБД не напрямую, а через прикладную программу, в которой предусмотрен удобный ввод данных и оформление результатов (рис. 6.1).

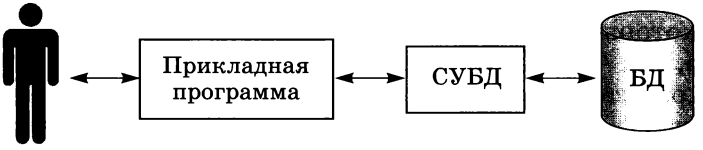



**Рис. 6.1**

Иногда функции СУБД и прикладной программы объединяются в одной программе (например, в *OpenOffice Base* или *Microsoft Access*).

## Локальные и удалённые информационные системы

Проще всего разместить базу данных и СУБД на одном компьютере. Такая информационная система называется **локальной ИС**, с ней работает один пользователь. Преимущество локальных ИС — **автономность**, т. е. независимость от работы компьютерных сетей.

Их недостатки проявляются тогда, когда с базой данных должны работать несколько пользователей:

- базу данных нужно обновлять на каждом компьютере;
- невозможно «стыковать» изменения, вносимые пользователями.

Как правило, в современных информационных системах используют **удалённые базы данных**, расположенные на серверах (специально выделенных компьютерах) локальной или глобальной сети. В этом случае несколько пользователей могут одновременно работать с базой и вносить в неё изменения.

СУБД, работающие с удалёнными базами данных, можно разделить на два типа по способу работы с файлами:

- файл-серверные СУБД;
- клиент-серверные СУБД.

## Файл-серверные информационные системы

**Файл-серверные СУБД** (например, *Microsoft Access*) работают на компьютерах пользователей (они называются **рабочими станциями**). Это значит, что сервер только хранит файлы, но не участвует в обработке данных (рис. 6.2). Когда пользователь вносит изменения в базу, СУБД с его рабочей станции блокирует файлы на сервере, чтобы их в это время не могли изменять другие пользователи.

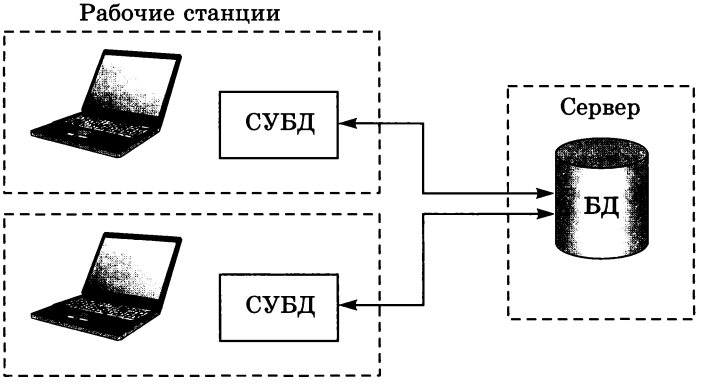



**Рис. 6.2**

Какие высказывания о файл-серверных СУБД верны?

а) Обработку данных выполняет сервер.
б) Рабочие станции могут быть маломощными.
в) Основная нагрузка приходится на сервер.
г) Для увеличения скорости работы системы нужно увеличивать мощность сервера;
д) Права на доступ к данным устанавливаются на рабочих станциях;
е) При поиске данных вся база скачивается на рабочую станцию.

## Клиент-серверные информационные системы

Клиент-серверная СУБД расположена на том же компьютере, где находится база данных. Она полностью берёт на себя всю работу с данными, т. е. читать и изменять данные в базе можно только с помощью этой СУБД (рис. 6.3).

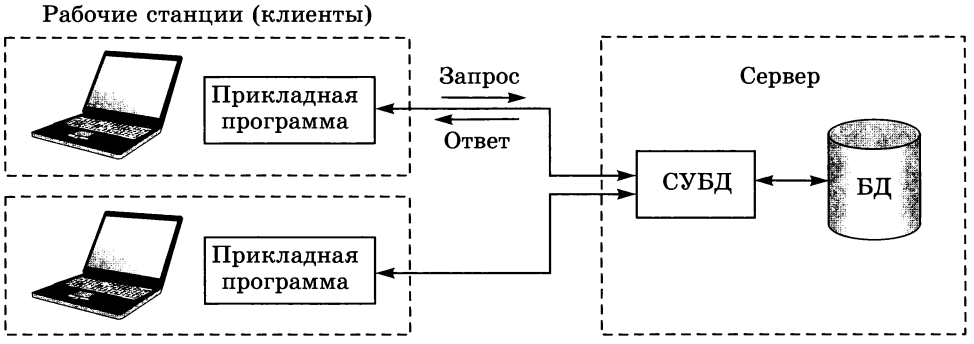



**Рис. 6.3**

На компьютере пользователя работает прикладная программа-клиент. Она отправляет серверу команду (запрос) на специальном языке и, получив ответ сервера, выводит результат на экран. В современных клиент-серверных СУБД для управления данными используют язык *SQL*. СУБД-сервер выполняет запросы в порядке очереди и посылает ответы клиентам. При необходимости серверная и клиентская программы могут быть установлены на одном компьютере.

Какие высказывания о клиент-серверных СУБД верны?

а) Обработку данных выполняет сервер.
б) Рабочие станции могут быть маломощными.
в) Основная нагрузка приходится на сервер.
г) Для увеличения скорости работы системы нужно увеличивать мощность сервера.
д) Права на доступ к данным устанавливаются на рабочих станциях.
е) При поиске данных вся база скачивается на рабочую станцию.

Используя дополнительные источники, найдите названия:

а) двух коммерческих клиент-серверных СУБД;
б) двух бесплатных клиент-серверных СУБД;
в) свободно распространяемой СУБД, которая используется на большинстве веб-сайтов.

Используя дополнительные источники, выясните, от каких слов произошло сокращение *SQL* и как переводится это выражение на русский язык.

## Распределённые информационные системы

Многие современные информационные системы (например, поисковые системы в Интернете) работают с огромными объёмами данных, которые невозможно разместить на одном компьютере. Поэтому появились **распределённые базы данных**, расположенные (по частям) на множестве компьютеров, и СУБД для управления ими. Пользователь работает с распределённой базой данных точно так же, как и с обычной (нераспределённой).

## Выводы

- Информационная система (ИС) предназначена для того, чтобы своевременно обеспечить пользователей нужной информацией.
- База данных (БД) — это специальным образом организованная совокупность данных о некоторой предметной области, хранящаяся во внешней памяти компьютера.
- Система управления базой данных (СУБД) — это программные средства, которые позволяют выполнять все необходимые операции с базой данных.
- Локальная ИС — это ИС, в которой и база данных, и СУБД находятся на компьютере пользователя.
- Удалённые ИС делятся на файл-серверные и клиент-серверные.
- В файл-серверных ИС база данных находится на удалённом компьютере, а СУБД — на компьютере пользователя.
- В клиент-серверных ИС и база данных, и СУБД находятся на удалённом компьютере. Прикладные программы обращаются к СУБД на языке *SQL*.
- Распределённые ИС работают с базами данных, части которых расположены на множестве компьютеров, связанных с помощью сети.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Что входит в состав информационной системы?
2. Является ли базой данных бумажная картотека в библиотеке? Ответ обоснуйте.
3. Почему произошел переход от множества разнообразных форматов хранения данных к использованию универсальных СУБД?

4. В каких случаях лучшим выбором будет локальная ИС? Файл-серверная ИС? Клиент-серверная ИС?
5. Какие достоинства и недостатки есть у распределённых баз данных?
6. Выполните по указанию учителя задания в рабочей тетради.

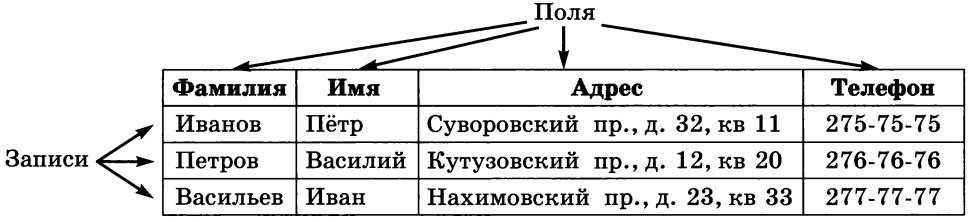
# § 31
# Таблицы

***Ключевые слова:***

- поле
- запись
- тип поля
- ключ
- первичный ключ
- суррогатный ключ
- целостность данных

## Основные понятия

В современных базах данных информация представлена в виде таблиц[1]. Например, так называемый «список контактов» (сведения о друзьях и знакомых) может выглядеть так (рис. 6.4).




| Фамилия | Имя | Адрес | Телефон |
|---|---|---|---|
| Иванов | Пётр | Суворовский пр., д. 32, кв 11 | 275-75-75 |
| Петров | Василий | Кутузовский пр., д. 12, кв 20 | 276-76-76 |
| Васильев | Иван | Нахимовский пр., д. 23, кв 33 | 277-77-77 |

**Рис. 6.4**

Столбцы таблицы называются **полями**, а строки — **записями**. Эта таблица относится к типу «объект — свойство», т. е. запись — это описание некоторого объекта (в данном случае — человека), а поля содержат свойства этого объекта. В таблице на рис. 6.4 четыре поля: *Фамилия, Имя, Адрес* и *Телефон* — и три записи.

Количество и состав полей определяет разработчик базы данных, пользователи могут только изменять записи (добавлять, удалять, редактировать).

[1] В середине XX века применялись базы данных, использующие иерархические (многоуровневые) и сетевые модели данных, но сейчас они очень редко встречаются на практике. Самый известный современный пример иерархической модели данных — *реестр* в операционной системе *Windows*, где хранятся настройки самой системы и программ.

Любое поле должно иметь уникальное (неповторяющееся) имя. Например, нельзя назвать два поля *Фамилия*, но можно одно назвать *Фамилия*, а второе — *Девичья фамилия*.

Каждое поле имеет свой **тип**. Как правило, СУБД поддерживают следующие типы данных:

- целые числа;
- вещественные числа;
- денежные суммы;
- логические значения (битовые поля);
- текстовые данные;
- время, дата;
- произвольные двоичные данные, например закодированный звук, видео и т. д.

Некоторые поля могут быть обязательными для заполнения. Если обязательное поле не заполнено, СУБД не внесёт изменения в базу данных и выдаст сообщение об ошибке.

## Ключ

При чтении и изменении данных в таблице очень важно убедиться, что мы обращаемся именно к нужной записи (например, увеличиваем зарплату именно тому сотруднику, которому её нужно повысить). Поэтому каждая запись должна содержать какое-то уникальное значение, отличающее её от всех остальных.

---

**Ключ** — это поле или комбинация полей, однозначно определяющая запись.

---

Основное свойство ключа — **уникальность**: в таблице не может быть двух записей, у которых одинаковое значение ключа.

Какие из этих данных могут быть ключом, а какие — нет:

а) имя и фамилия;
б) серия и номер паспорта;
в) номер сотового телефона;
г) место работы;
д) марка автомобиля;
е) регистрационный номер автомобиля;
ж) адрес электронной почты;
з) домашний адрес?

Приведите примеры, когда в одном случае какие-то данные (например, марка стиральной машины) могут быть ключом таблицы, а в другом — нет.

Определите возможные ключи в следующих таблицах (здесь перечислены их поля):

а) фамилия, год рождения, номер паспорта;
б) марка автомобиля, регистрационный номер автомобиля, фамилия владельца, год выпуска;
в) фамилия, адрес электронной почты, место работы, номер мобильного телефона;
г) номер заказа, сумма заказа, фамилия, номер мобильного телефона, номер паспорта, адрес электронной почты.

Иногда в таблице есть несколько возможных ключей. В этом случае один из них выбирается как основной и называется **первичным ключом.**

Ключ, состоящий из одного поля, называется **простым**, а соответствующее поле таблицы — **ключевым полем.** Ключ, который состоит из нескольких полей, называется **составным.** Рассмотрим базу данных метеостанции, на которой через каждые 3 часа измеряются температура, влажность воздуха и скорость ветра (рис. 6.5).

| Дата | Время | Температура | Влажность | Скорость ветра |
|---|---|---|---|---|
| 21.07.2012 | 12:00 | 25 | 75 | 4 |
| 21.07.2012 | 15:00 | 23 | 70 | 3 |
| ... | ... | ... | ... | ... |

**Рис. 6.5**

Здесь ни одно поле не может быть ключом, потому что значения в каждом из них могут повторяться. Однако для каждой пары (*Дата*, *Время*) в таблице может быть только одна запись, поэтому комбинация полей *Дата* и *Время* — это ключ.

Второе свойство ключа — **несократимость.** Заметим, что в рассмотренном примере к паре (*Дата, Время*) можно добавить и другие поля таблицы, но такая группа полей уже не будет считаться ключом, потому что она *сократима*, т. е. из неё можно исключить все поля, кроме полей *Дата* и *Время*, сохранив свойство уникальности.

Теперь посмотрим на приведённую на рис. 6.4 таблицу «список контактов». Ни фамилия, ни имя не могут быть ключом, потому что есть много однофамильцев и людей с одинаковыми именами. Составить ключ из полей *Фамилия* и *Имя* тоже не получится (могут быть однофамильцы и тёзки одновременно). Адрес

и домашний телефон также не могут быть ключом, потому что в одной квартире могут жить несколько человек, с которыми вы общаетесь. Может получиться так, что ключом будет комбинация *всех* полей записи.

Работать с составными ключами при выполнении операций с базой данных очень неудобно. В таких случаях часто добавляют в таблицу ещё одно поле — так называемый **суррогатный** (т. е. неестественный) **ключ**, например номер записи или код (*идентификатор*) — рис. 6.6. Во многих СУБД есть возможность заполнять его автоматически при добавлении каждой новой записи. При этом пользователю не нужно задумываться об уникальности такого ключа.

ключ ↓

| Номер | Фамилия | Имя | Адрес | Телефон |
|---|---|---|---|---|
| 1 | Иванов | Пётр | Суворовский просп., д. 32, кв. 11 | 275-75-75 |
| 2 | Петров | Василий | Кутузовский просп., д. 12, кв. 20 | 276-76-76 |
| 3 | Васильев | Иван | Нахимовский просп., д. 23, кв. 33 | 277-77-77 |

**Рис. 6.6**

Используя дополнительные источники, выясните, что означает слово «идентифицировать».

## Целостность

**Целостность базы данных** означает, что она содержит полную и непротиворечивую информацию и удовлетворяет всем заданным ограничениям.

Прежде всего, нужно обеспечить **физическую целостность БД**, т. е. защитить данные в случае отказа оборудования (например, при отключении питания или выходе из строя жёстких дисков). Периодически (например, раз в неделю или даже чаще) администраторы базы данных создают резервные копии всех данных (на магнитных дисках или DVD-дисках) и ведут журнал изменений. При потере рабочей копии восстанавливается самая последняя сохранённая версия базы данных.

Также используют **RAID-массивы** жёстких дисков, где информация дублируется и может быть автоматически восстановлена в случае выхода из строя одного или даже нескольких дисков.

Используя дополнительные источники, выясните, от каких слов образовано сокращение *RAID* и как переводится это выражение на русский язык. Найдите два варианта ответа на этот вопрос.

Теперь представьте себе, что в базе данных отдела кадров по ошибке у работника указан 1698 год рождения, а в поле *Зарплата* введено отрицательное число. В этих случаях нарушается **логическая целостность БД**, т. е. непротиворечивость данных. Чтобы этого не произошло, вводят ограничения на допустимые значения полей (контроль данных):

- каждое поле имеет свой тип; например, СУБД не даст записать в поле целого типа произвольный текст — будет выведено сообщение об ошибке;
- некоторые поля (в первую очередь — первичный ключ) объявляются обязательными для заполнения;
- значения в некоторых полях (например, ключевых) не могут повторяться (при повторении значения выдаётся сообщение об ошибке);
- вводятся условия, которые должны выполняться для значений отдельных полей: например, количество учеников в классе должно быть положительно;
- для сложных данных используются *шаблоны* ввода: например, для ввода семизначного номера телефона можно использовать шаблон ###-##-##, где # означает любую цифру;
- вводятся условия, которые должны выполняться для нескольких полей каждой записи: например, дата увольнения работника не может быть более ранней, чем дата приёма на работу.

Заметим, что целостность БД не гарантирует *достоверность* (истинность) данных, а только означает, что выполнены все установленные ограничения на эти данные и исключены явные противоречия.

Определите, какие нарушения логической целостности допущены в таблице на рис. 6.7. Сколько ошибок вам удалось найти?

| Код сотрудника | Фамилия | Год рождения | Принят | Уволен | Телефон |
|---|---|---|---|---|---|
| 1327 | Иванов | 1811 | 01.03.2011 | | (911) 123-45-67 |
| 1323 | Петров | 1977 | 12.10.2014 | 13.05.2014 | (921) 223-45-67 |
| 1327 | Сидоров | 0 | | 28.09.2015 | 323-45-67 (913) |
| 1329 | Тяпкин | 1989 | да | | iPhone |

**Рис. 6.7**

## Выводы

- В современных базах данных информация представляется для пользователя в виде таблиц типа «объект — свойства».
- Столбцы таблицы (свойства объектов) называются полями, а строки (данные об одном объекте) — записями.
- Ключ — это поле или комбинация полей, однозначно определяющая запись.
- Суррогатный ключ — это дополнительное числовое поле, которое используется как ключ таблицы.
- Целостность базы данных означает, что она содержит полную и непротиворечивую информацию и удовлетворяет всем заданным ограничениям.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Найдите в тексте параграфа и объясните два свойства ключа.
2. Чем отличаются понятия «ключ», «первичный ключ», «простой ключ» и «составной ключ»?
3. В каких случаях в качестве первичного ключа используют номер записи? Можно ли применять такой подход, если в таблице есть другое уникальное поле?
4. Что такое целостность базы данных? Какие виды целостности вы знаете?
5. В таблице четыре поля: *Дата*, *Номер заказа*, *Товар* и *Количество*. Что можно выбрать в качестве первичного ключа?
6. В школьной базе данных хранятся сведения о выданных аттестатах. Таблица включает поля: *Фамилия*, *Имя*, *Отчество*, *Дата рождения*, *Год выпуска*, *Номер паспорта*, *Номер аттестата*. Что можно выбрать в качестве первичного ключа этой таблицы?
7. Выполните по указанию учителя задания в рабочей тетради.

## Подготовьте сообщение

а) «Суррогатные ключи — "за" и "против"»
б) «Поиск с помощью индексов»
в) «Транзакции»

# § 32
# Работа с базой данных

***Ключевые слова:***

- текущая запись
- поиск
- сортировка
- фильтрация
- Конструктор

Далее для иллюстрации мы будем использовать свободно распространяемую СУБД **Base** из пакета *OpenOffice*. Аналогичные приёмы работы применяются и в популярной СУБД **Microsoft Access** (которая обладает ещё бóльшими возможностями), поэтому практические задания можно выполнять в любой из этих программ. В тексте мы будем обращать внимание на некоторые различия между ними.

Вся база данных представляет собой один файл. В нём находятся:

- *таблицы*, в которых хранятся данные;
- *формы* — диалоговые окна, с помощью которых пользователь вводит и изменяет данные;
- *запросы* — обращения к базе данных, в результате которых отбираются нужные данные;
- *отчёты* — шаблоны документов, предназначенных для вывода данных на печать.

## Просмотр таблицы

После открытия файла появляется основное окно базы данных (рис. 6.8).

**Рис. 6.8**

Сейчас в левой части выбрана группа *Таблицы*, а в правой части мы видим список всех таблиц (здесь одна таблица *Учебники*).

После двойного щелчка на названии таблицы она открывается в отдельном окне в режиме редактирования данных (рис. 6.9).

| | Код | Авторы | Название |
|---|---|---|---|
| | 1 | Матвеева Н.В. и др. | Информатика. 2 класс. Ч. 1, 2 |
| | 2 | Матвеева Н.В. и др. | Информатика. 3 класс. Ч. 1, 2 |
| ▷ | 3 | Матвеева Н.В. и др. | Информатика. 4 класс. Ч. 1, 2 |
| | 4 | Плаксин М.А. и др. | Информатика. 3 класс. Ч. 1, 2 |
| | 5 | Плаксин М.А. и др. | Информатика. 4 класс. Ч. 1, 2 |
| | 6 | Могилев А.В. и др. | Информатика. 3 класс. Ч. 1, 2 |
| | 7 | Могилев А.В. и др. | Информатика. 4 класс. Ч. 1, 2 |

Запись 6 из 237 (1)

**Рис. 6.9**

В левой части расположена область выделения, щелчок в ней выделает *текущую* (активную, рабочую) запись, которая обозначается треугольником.

**Текущая запись** — это запись, с которой работает пользователь в настоящий момент.

В нижней строке окна находятся кнопки для перехода по записям |◀|◀|▶|▶| и кнопка |✱| для вставки новой записи. Новая запись всегда добавляется в конец таблицы.

На рисунке 6.9 найдите номер текущей записи и общее количество записей в таблице.

## Поиск и сортировка

При щелчке на кнопке *Найти* на панели инструментов открывается окно, в котором можно задать образец и режимы **поиска** (искать вперёд, назад, в текущем поле или во всех полях и т. д.).

Используя всплывающую подсказку, выясните, как с помощью клавиатуры можно открыть окно поиска.

**Сортировка** — это расстановка данных в определённом порядке.

Простейший вариант сортировки — сортировка по текущему столбцу (по тому, в котором стоит курсор). Для этого служат кнопки  (*Сортировать по возрастанию*) и  *Сортировать по убыванию* на панели инструментов[1], задающие соответственно алфавитный и обратный алфавитный порядок. Кнопка  *Сортировать* позволяет применить несколько уровней сортировки, например отсортировать книги по авторам, а, в свою очередь, книги одного автора отсортировать ещё и по году издания.

Важно понимать, что при сортировке *физическое* расположение записей в базе данных (в файле на диске) не изменяется, они переставляются в списке только при выводе на экран.

Представьте себе, что при каждой сортировке программа изменяла бы расположение данных на диске. Какие достоинства и недостатки есть у такого решения?

## Фильтрация

На уроках химии вы использовали фильтрацию, чтобы отделить осадок от жидкости. В базах данных фильтр служит для того, чтобы оставить нужные записи и временно скрыть ненужные.

---

**Фильтр** — это условие для отбора записей.

---

Самый простой вариант — это **быстрый фильтр** (или *фильтр по выделенному*). Он отбирает все записи, в которых значение текущего поля совпадает со значением активной ячейки таблицы. Например, чтобы оставить только книги издательства «БИНОМ. Лаборатория знаний», нужно сделать активной ячейку, содержащую это название, и щёлкнуть на кнопке  *Быстрый фильтр* на панели инструментов.

Записи, не удовлетворяющие заданному условию, при фильтрации не удаляются из таблицы, а временно скрываются. Чтобы снова показать все записи, нужно отменить фильтр, щёлкнув на кнопке  *Отменить фильтр*. При включённом фильтре эта кнопка выделяется серым фоном (активна).

Сложный фильтр, устанавливающий ограничения на несколько полей, можно задать с помощью кнопки  *Фильтр по умолчанию*. В диалоговом окне задаётся сложное условие, в котором можно использовать логические операции И (AND) и ИЛИ (OR) — рис. 6.10.

---

[1] В русской версии программы *Microsoft Access* на аналогичных кнопках написаны русские буквы «А» и «Я».

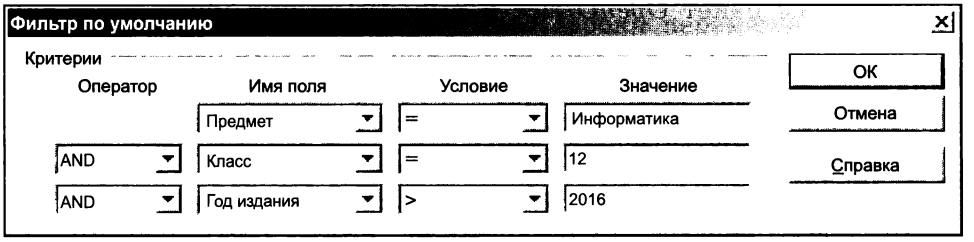



**Рис. 6.10**

Помните, что сначала выполняется операция И, а потом — операция ИЛИ. Кнопка *Удалить фильтр/сортировку* удаляет установленный фильтр и отменяет сортировку.

Приведите пример условия-фильтра, который невозможно задать с помощью диалогового окна, показанного на рис. 6.10.

Фильтры служат для быстрого отбора записей по простым условиям. С каждой таблицей может храниться только один фильтр. Если нужно постоянно использовать несколько разных фильтров или более сложные условия отбора, применяют *запросы*, с которыми мы познакомимся далее.

## Создание таблицы

Работать с готовой базой данных мы уже научились, теперь займёмся созданием новой базы «с нуля». Построим однотабличную базу данных *Футбол*, в которую запишем количество побед, ничьих и поражений нескольких футбольных команд за последний сезон, а также среднюю зарплату футболистов (рис. 6.11).

| | Команда | Победы | Ничьи | Поражения | Зарплата |
|---|---|---|---|---|---|
| | Аметист | 10 | 7 | 3 | 13 290 руб |
| | Бирюза | 5 | 8 | 7 | 12 500 руб |
| | Восход | 13 | 5 | 2 | 22 000 руб |
| | Закат | 7 | 8 | 5 | 18 780 руб |
| | Коллектор | 11 | 6 | 3 | 20 200 руб |
| | Кубань | 6 | 12 | 2 | 14 000 руб |
| | Малахит | 12 | 3 | 5 | 17 340 руб |
| | Ротор | 8 | 12 | 0 | 15 820 руб |
| | Статор | 9 | 10 | 1 | 19 300 руб |
| | Финиш | 12 | 0 | 8 | 12 950 руб |

**Рис. 6.11**

Запустим пакет *OpenOffice* и выберем пункт меню *База данных*. Мастер (программный модуль, который облегчает выполнение стандартных действий) предлагает на выбор три варианта:

- создать новую базу данных;
- открыть существующую базу;
- подключиться к существующей базе, например к базе *Microsoft Access* или к адресной книге почтовой программы.

Выберем первый пункт. После этого нам предлагается *зарегистрировать* базу данных — в этом случае мы сможем использовать данные из неё в других программах *OpenOffice*.

В новой базе ещё нет ни одной таблицы. Создать новую таблицу можно двумя способами, которые перечислены в области *Задачи* (рис. 6.12).

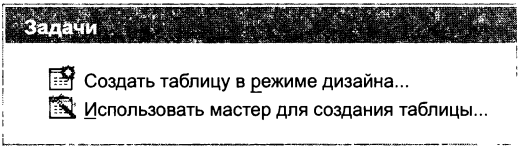



**Рис. 6.12**

Режим дизайна иначе называется *Конструктор*, это означает, что всю работу приходится выполнять вручную. *Мастер* позволяет быстро построить таблицу на основе одного из готовых шаблонов.

Выбираем *режим дизайна*. Появляется окно, где нужно в первом столбце ввести названия всех полей новой таблицы, а во втором — выбрать их типы из выпадающего списка (рис. 6.13).

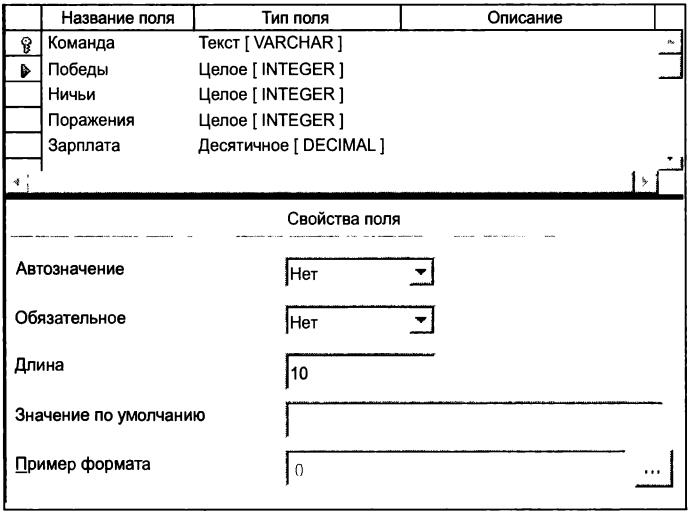



**Рис. 6.13**

Здесь использованы три различных типа данных:

- *Текст* [*VARCHAR*] — текстовая строка переменной длины;
- *Целое* [*INTEGER*] — целое число;
- *Десятичное* [*DECIMAL*] — десятичное число, которое хранится с заданным числом знаков в дробной части; применяется, например, для работы с денежными суммами.

Выясните, какие ещё типы данных используются в вашей СУБД.

Сделать поле *Команда* ключевым можно с помощью контекстного меню. Если выделить несколько полей (щёлкая в области выделения при нажатой клавише *Ctrl*), можно таким же способом создать *составной ключ* таблицы, состоящий из всех выделенных полей.

В нижней части окна *Конструктора* настраиваются свойства выделенного поля, например:

- максимальный размер для текста;
- количество знаков в дробной части для десятичного числа;
- значение по умолчанию (которое автоматически вписывается в поле при создании новой записи).

Если в таблице в качестве первичного ключа используется поле-счётчик (его значение увеличивается на единицу при добавлении новой записи), для него нужно установить свойство *Автозначение* равным *Да*. Такое поле будет заполняться автоматически.

Можно потребовать, чтобы значение какого-то поля обязательно было задано для каждой добавляемой записи (вспомните обязательные поля при регистрации на веб-сайтах). Для этого нужно установить значение *Да* свойства *Обязательное*.

Формат вывода значения на экран (например, число знаков в дробной части, выравнивание, выделение отрицательных значений красным цветом и т. п.) задаётся с помощью кнопки  в нижней части окна *Конструктора*.

При закрытии окна *Конструктора* будет предложено сохранить таблицу и ввести её название, затем новая таблица появится в списке таблиц.

Исследуйте кнопки  панели инструментов и выясните, какая из них какую операцию выполняет. Запишите в тетради функцию каждой из них и соответствующую команду контекстного меню, которое появляется при щелчке правой кнопкой мыши на названии таблицы.

## Выводы

- Сортировка — это выдача пользователю записей в определённом порядке. При сортировке порядок записей в файле не изменяется.

- Фильтр — это условие для отбора записей. В фильтрах можно использовать логические операции AND (И) и OR (ИЛИ). После применения фильтра все записи, не удовлетворяющие условию, скрываются, но не удаляются из базы.
- Создать новую таблицу можно в режиме дизайна (в режиме *Конструктора*) или с помощью мастера.
- В *Конструкторе* определяются поля таблицы и их свойства.

Нарисуйте в тетради интеллект-карту этого параграфа.

### Вопросы и задания

1. Как вы думаете, какие достоинства и недостатки имеет идея хранения всех объектов БД в одном файле?
2. Почему СУБД (в отличие от табличных процессоров) не разрешает вставлять новую запись в середину таблицы?
3. Что такое многоуровневая сортировка?
4. Можно ли хранить в базе данных несколько разных фильтров для одной таблицы?
5. Что такое мастер?
6. Какие способы создания таблиц вы знаете? Чем они различаются?
7. Зачем каждому полю таблицы присваивается некоторый тип данных?
8. Данные каких типов можно хранить в базах данных?
9. Как изменить ключ таблицы?
10. Выполните по указанию учителя задания в рабочей тетради.

### Практическая работа

Выполните практическую работу № 31 «Работа с базой данных».

# § 33
# Запросы

***Ключевые слова:***

- запрос
- конструктор запросов
- критерий отбора
- запрос с параметрами
- вычисляемое поле

## Что такое запрос?

Для пользователя любой информационной системы в первую очередь важно, чтобы он смог выбрать из базы данных ту информацию, которая ему в данный момент нужна. Для этого используются *запросы*.

**Запрос** — это обращение к СУБД для отбора записей или выполнения других операций с данными.

Как вы уже знаете, в большинстве современных СУБД для управления данными (т. е. для составления запросов) используется язык *SQL*. Изучение этого языка выходит за рамки школьного курса, поэтому мы будем использовать в основном визуальные (наглядные) средства составления запросов. В то же время при желании вы сможете «подсмотреть», как выглядит составленный вами запрос на языке *SQL*.

Используя дополнительные источники, выясните, какие типы запросов можно использовать для управления данными.

## Конструктор запросов

Продолжим работу с базой данных *Футбол*, которую мы недавно создали. Перейдём в окне базы данных на страницу *Запросы* и выберем в области *Задачи* вариант *Создать запрос в режиме дизайна*. Появится окно *Конструктора*, программа предложит добавить в рабочую область таблицу, из которой будут выбираться данные.

Названия полей, которые нужно включить в запрос, можно перетаскивать мышью в пустые столбцы бланка в нижней части окна *Конструктора* (рис. 6.14)

**Рис. 6.14**

В нашем случае в запрос добавлены поля *Команда*, *Победы* и *Зарплата*.

Чтобы увидеть результаты запроса, нужно щёлкнуть на кнопке  *Выполнить запрос* на панели инструментов. Результат выполнения запроса – это таблица, которая не сохраняется в файле, а строится в момент выполнения запроса. Однако она напрямую связана с данными — если вы измените что-то в результатах запроса, эти изменения будут внесены в базу данных. Такие изменения разрешены только тогда, когда в запросе есть первичный ключ таблицы.

---

Запрос, который мы построили, программа переводит на язык *SQL* и отправляет СУБД для выполнения. Чтобы увидеть *SQL*-запрос, нужно щёлкнуть на кнопке  (повторный щелчок возвращает обратно в режим *Конструктора*). В данном случае мы увидим такой запрос:

```
SELECT "Команда»"", "Победы", "Зарплата" FROM "Футбол"
```

Здесь выбираются три поля из таблицы *Футбол*. В режиме *SQL* запрос можно редактировать вручную, если вы знаете этот язык.

Используя дополнительные источники, найдите перевод английских слов *select* и *from*.

---

Перейдём обратно в *Конструктор* и сделаем так, чтобы список команд был отсортирован по убыванию числа побед. Для этого в столбце *Победы* нужно для параметра *Сортировка* выбрать из выпадающего списка значение *по убыванию*. Теперь можно заново выполнить запрос и посмотреть на результат.

Используя всплывающие подсказки, выясните, как можно выполнить запрос с помощью клавиатуры.

Если закрыть окно *Конструктора*, программа предложит сохранить запрос и ввести его имя. Оно не может совпадать с именем уже существующей таблицы или запроса.

После этого запрос появляется в списке запросов в основном окне базы данных. Двойной щелчок на имени запроса открывает окно с результатами. Нажав правую кнопку мыши на заголовке столбца таблицы в этом окне, можно настроить формат столбца (тип данных, выравнивание).

Исследуйте кнопки  на панели инструментов и выясните, какая из них какую операцию выполняет. Запишите в тетрадь функцию каждой из них и соответствующую команду контекстного меню, которое появляется при щелчке правой кнопкой мыши на названии запроса.

## Критерии отбора

Теперь отберём только те команды, которые одержали более 10 побед. Для этого в столбце *Победы* зададим критерий отбора *>10* (строка *Критерий*) и проверим запрос.

Добавим второе условие отбора в той же строке *Критерий*: для поля *Зарплата* установим критерий *>15000*. Теперь СУБД отберёт только те записи, для которых одновременно выполняются оба критерия, т. е. условия в одной строке объединяются с помощью логической операции AND (И). В строке с заголовком *или* можно записать условие, которое будет объединяться с первым с помощью логической операции ИЛИ.

Для текстовых данных можно указывать не только точное значение, но и шаблон (вспомните, как строятся маски имён файлов). Например, если в критерий отбора для поля *Команда* ввести

```
LIKE 'К*'
```

будут отобраны только те команды, названия которых начинаются с буквы «К». Здесь слово LIKE обозначает «такой, как...», «похожий на...», а звёздочка – любое количество любых символов. Кроме звёздочки можно использовать знак «?», обозначающий один любой символ.

Проверьте, что произойдёт, если у какого-то поля выключить флажок в строке *Видимый*. В каких случаях это может быть нужно?

## Запросы с параметрами

Создадим запрос, отбирающий команды, в которых зарплата футболистов больше заданной.

Как сделать, чтобы в результатах запроса остались только команды с зарплатой не менее 15 000 рублей?

Если мы захотим изменить сумму, нужно будет изменить запрос. Чаще всего пользователь не имеет права изменять запросы (это делает только администратор базы данных), поэтому возникает проблема: как дать пользователю возможность менять зна-

чение в запросе, не меняя сам запрос? Чтобы решить эту задачу, применяют **запросы с параметрами.**

**Параметры** — это данные, которые пользователь вводит при выполнении запроса.

В *Конструкторе* параметр задаётся с помощью двоеточия, за которым следует имя запроса, например: *>=:Минимальная_зарплата*. Здесь параметр называется *Минимальная_зарплата*. Для соединения двух слов используется знак подчёркивания, потому что в *OpenOffice Base* имя параметра не может включать пробелы[1].

Когда выполняется запрос, на экране появляется окно, в котором пользователь должен ввести значения всех параметров (рис. 6.15).

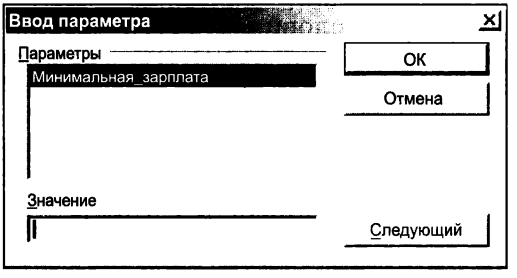



**Рис. 6.15**

## Вычисляемые поля

В базе данных не нужно хранить значения, которые можно вычислить по другим известным данным. Пусть, например, требуется определить количество очков, которые набрала каждая команда в чемпионате, учитывая, что за победу начисляется 3 очка, а за ничью — одно очко. Количество побед и ничьих есть в базе данных, поэтому количество очков хранить не нужно, его можно вычислить тогда, когда оно понадобится.

**Вычисляемое поле** — это значение, которое не хранится в базе данных и вычисляется при выполнении запроса.

В бланке запроса перетащим поле *Зарплата* за заголовок вправо, освободив 3 столбца. В первые два пустых столбца добавим поля *Ничьи* и *Поражения*, а в третий вместо имени поля введем нужную нам формулу:

Ничьи+3*Победы

[1] В программе *Microsoft Access* этого ограничения нет, там имя параметра заключается в квадратные скобки.

В строке *Псевдоним* можно ввести осмысленный заголовок этого столбца — *Очки*, который и будет появляться в таблице с результатами запроса.

С новым столбцом (*вычисляемым полем*) можно делать всё, что и с обычными столбцами, соответствующими реальным полям таблицы, — сортировать, устанавливать условия отбора[1]. Например, можно выбрать сортировку по убыванию, чтобы в начале таблицы оказались команды с самыми высокими результатами.

## Выводы

- Запрос — это обращение к СУБД для отбора записей или выполнения других операций с данными.
- Запросы можно составлять в наглядной (визуальной) форме с помощью *Конструктора*.
- Параметры — это данные, которые пользователь вводит при выполнении запроса.
- Вычисляемое поле — это значение, которое не хранится в базе данных и вычисляется при выполнении запроса.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Объясните, зачем нужны строки *Псевдоним, Сортировка* и *Критерий* в бланке запроса.
2. Как вы думаете, ограничивает ли структура бланка (одна строка *Критерий* и несколько строк *или*) возможность создания сложных запросов?
3. Как изменить существующий запрос? Предложите два способа.
4. Зачем нужны запросы с параметрами?
5. Можно ли обойтись без вычисляемых полей? Предложите разные варианты решения проблемы и обсудите их достоинства и недостатки.
6. Выполните по указанию учителя задания в рабочей тетради.

## Подготовьте сообщение

«Запросы на языке SQL»

## Практическая работа

Выполните практическую работу № 32 «Запросы».

---

[1] В *OpenOffice Base* числа в условиях отбора для вычисляемого поля нужно заключать в апострофы: *>'10'*.

# § 34
# Многотабличные базы данных

***Ключевые слова:***

- многотабличная БД
- связь между таблицами
- внешний ключ

## Почему бы не собрать всё в одной таблице?

До этого момента мы изучали простейшую базу данных, в которой всё данные сведены в одну таблицу, и поэтому искать информацию достаточно просто. Однако у такой модели есть и *недостатки*:

- *дублирование данных*: например, в базе данных школьной библиотеки будет много раз храниться фамилия автора «Пушкин»;
- при изменении каких-то данных (например, адреса фирмы), возможно, придётся изменять *много записей*;
- *нет защиты от ошибок ввода* (опечаток).

Однотабличная база данных — это аналог картотеки, в которой все карточки имеют одинаковую структуру. В то же время обычно в одной базе нужно хранить данные, относящиеся к объектам *разных* типов, которые связаны между собой. Поэтому возникает вопрос: какую модель лучше использовать для описания и хранения этих данных?

Посмотрим, как можно организовать базу данных, в которой хранятся данные об альбомах музыкальных групп. Вот что получается, если свести все данные в одну таблицу (рис. 6.16).

**Альбомы**

| Код альбома | Название | Группа | Год | Число композиций |
|---|---|---|---|---|
| 1 | Хиты про любовь | Браво | 1998 | 13 |
| 2 | Мода | Браво | 2011 | 12 |
| 3 | Вне зоны доступа | Город 312 | 2006 | 16 |
| 4 | Новая музыка | Город 312 | 2010 | 18 |

**Рис. 6.16**

В данном случае в качестве первичного ключа можно выбрать пару свойств *Альбом* + *Группа*, но работать с таким составным ключом неудобно, поэтому мы ввели суррогатный ключ — дополнительное числовое поле *Код альбома*, оно обозначено знаком .

Определите типы данных и ограничения для всех полей таблицы на рис. 6.16.

Сразу видим, что в этой таблице есть дублирование — название группы (символьная строка) повторяется для каждого альбома этой группы. Причина в том, что в таблице на самом деле есть сведения не только об альбомах, но и о группах — объектах совершенно другого класса. Поэтому для хранения всей информации о группах нужно сделать отдельную таблицу (рис. 6.17).

**Группы**

| Код группы | Название | Год создания |
|---|---|---|
| 1 | Браво | 1983 |
| 2 | Город 312 | 2001 |

**Рис. 6.17**

Здесь первичным ключом может быть название группы (символьная строка), но для ускорения работы введён суррогатный ключ — *Код группы* (целое число). В эту таблицу можно добавить другие данные о группе.

Предложите новые поля, которые можно добавить в таблицу Группы.

В таблице *Альбомы* теперь будет храниться не название группы, а её код (рис. 6.18).

**Альбомы**

| Код альбома | Название | Код группы | Год | Число композиций |
|---|---|---|---|---|
| 1 | Хиты про любовь | 1 | 1998 | 13 |
| 2 | Мода | 1 | 2011 | 12 |
| 3 | Вне зоны доступа | 2 | 2006 | 16 |
| 4 | Новая музыка | 2 | 2010 | 18 |

**Рис. 6.18**

Таким образом, база данных состоит из двух таблиц, каждая из которых хранит сведения об объектах одного класса (первая — о группах, вторая — об альбомах). Такая структура понятна человеку и облегчает работу с многотабличными базами.

Две таблицы связаны, их связь можно показать на схеме (рис. 6.19).

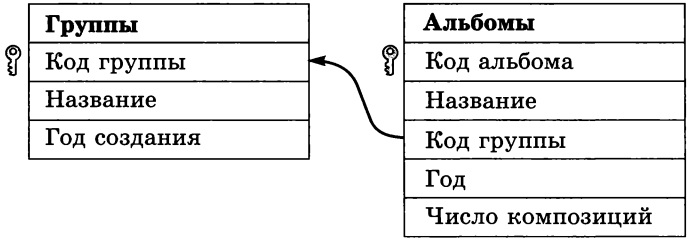



**Рис. 6.19**

На этом рисунке перечислены поля каждой из таблиц и показана их связь: код группы в таблице *Альбомы* должен совпадать с кодом одной из групп в таблице *Группы*.

Каждый альбом связан только с одной группой, поэтому связь между таблицами *Группы* и *Альбомы* — это связь типа «один ко многим» (она кратко обозначается как 1:n или 1-∞). Это значит, что одна запись в таблице *Группы* может быть связана со многими записями в таблице *Альбомы*, но не наоборот.

---

**Внешний ключ** — это неключевое поле таблицы, связанное с первичным ключом другой таблицы.

---

Таким образом, *Код группы* — это внешний ключ в таблице *Альбомы*.

Используя дополнительные источники, выясните, как называется внешний ключ таблицы в англоязычной литературе.

Итак, мы получили две связанные таблицы вместо одной. При этом:

- устранено дублирование данных (повторно хранятся только числовые коды);
- все изменения нужно выполнять только в одном месте;
- есть некоторая защита от ошибок при вводе данных — можно сделать так, что при заполнении таблицы *Альбомы* название группы выбирается из уже готового списка групп.

Однако кое-что *ухудшилось* из-за того, что данные разбросаны по разным таблицам:

- базами данных, в которых более 40—50 таблиц, сложно управлять из-за того, что разработчику трудно воспринимать информацию в таком «раздробленном» виде;
- при поиске приходится «собирать» нужные данные из нескольких таблиц.

Рассматриваемые базы данных основаны на **реляционной модели данных**. Это название связано с тем, что для работы с ними применяется специальный математический аппарат — реляционная алгебра,

разработанная в 70-х годах XX века. Более подробно с реляционными базами данных вы познакомитесь в старшей школе.

Используя дополнительные источники, выясните, от какого слова произошло слово «реляционный» и что оно означает. Какой учёный разработал реляционную алгебру?

## Создание многотабличной базы данных

Построим многотабличную базу данных (рис. 6.20). Создадим в новой базе данных (назовем её *Музыка*) две таблицы (пока без связей). Далее мы будем использовать общепринятые обозначения вида *Группы*.[*Название*] — это значит: «поле *Название* таблицы *Группы*».

Не забудьте, что связи устанавливаются только между однотипными полями, т. е. поля *Группы*.[*Код группы*] и *Альбомы*.[*Код группы*] должны быть одного типа (например, целые числа — тип INTEGER).

Чтобы установить связи между таблицами, выберем пункт главного меню[1] *Сервис* → *Связи*. С помощью специального окна добавим в рабочую область (она пока пустая) обе таблицы (рис. 6.20).


**Рис. 6.20**

Теперь можно «схватить» мышью название какого-то поля и «бросить» его на название поля другой таблицы, с которым нужно установить связь. Установим таким образом связь между полями *Код группы* обеих таблиц (рис. 6.21).

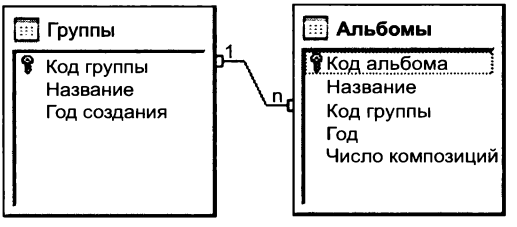



**Рис. 6.21**

Обратите внимание, что программа автоматически установила связь типа 1:n, причём знак 1 установлен у ключевого поля таблицы *Группы*, а знак *n* — у внешнего ключа таблицы *Альбомы*.

[1] В программе *Microsoft Access* нужно щёлкнуть на кнопке *Схема данных* на вкладке *Работа с базами данных*.

Чтобы изменить или удалить связи, нужно зайти в меню *Сервис* → *Связи*. Для удаления связи надо выделить её щелчком мышью и нажать клавишу *Delete*.

## Запросы

Мы построим два запроса к базе данных *Музыка* — простой и итоговый. Начнём с **простого запроса**, в котором будет собрана информация по всем альбомам (рис. 6.22).

| Название | Группа | Год | Число композиций |
|---|---|---|---|
| Хиты про любовь | Браво | 1998 | 13 |
| Мода | Браво | 2011 | 12 |
| Вне зоны доступа | Город 312 | 2006 | 16 |
| Новая музыка | Город 312 | 2010 | 18 |

**Рис. 6.22**

Посмотрим, откуда нужно взять данные. Название группы хранится в таблице *Группы*; а остальные данные — в таблице *Альбомы*. В *режиме дизайна* добавим в рабочую область обе таблицы (рис. 6.23).

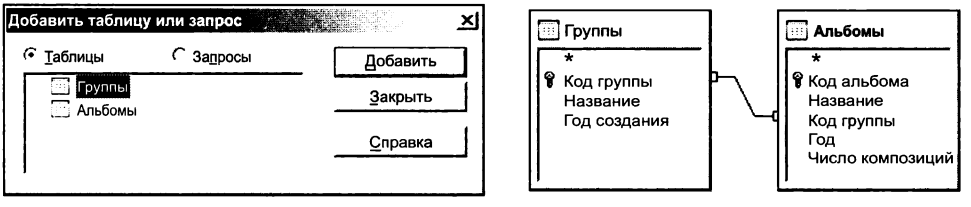



**Рис. 6.23**

Теперь перетащим в столбцы бланка все нужные поля и выполним запрос (рис. 6.24).

| | Название | Название | Год | Число композиций |
|---|---|---|---|---|
| ▷ | Хиты про любовь | Браво | 1998 | 13 |
| | Мода | Браво | 2011 | 12 |
| | Вне зоны доступа | Город 312 | 2006 | 16 |
| | Новая музыка | Город 312 | 2010 | 18 |

**Рис. 6.24**

Посмотрите, как выглядит этот запрос на языке *SQL*. Попытайтесь разобраться в нём. Попробуйте его изменить.

Возникла одна проблема: в запросе два поля *Название*, одно из них обозначает название группы, а второе — название альбома. Для того чтобы исправить ситуацию, зайдём в *Конструктор* и добавим правильные псевдонимы (заголовки) для этих столбцов (рис. 6.25).

| Поле | Название | Название | Год | Число композиций |
|---|---|---|---|---|
| Псевдоним | **Альбом** | **Группа** | | |
| Таблица | Альбомы | Группы | Альбомы | Альбомы |

**Рис. 6.25**

Теперь окно *Конструктора* можно закрыть, при сохранении дайте запросу название *ЗапросАльбомы*. Обратите внимание, что нельзя называть запрос именем, которое совпадает с именем таблицы (например, *Альбомы*) или другого запроса.

Теперь построим **итоговый запрос**: для каждой группы подсчитаем количество альбомов и число композиций во всех альбомах (рис. 6.26).

| Группа | Альбомов | Общее число композиций |
|---|---|---|
| Браво | 2 | 25 |
| Город 312 | 2 | 34 |

**Рис. 6.26**

Создадим новый запрос с помощью *Конструктора*. Все нужные нам данные можно взять из двух таблиц, но они уже собраны вместе в запросе *ЗапросАльбомы*. Поэтому можно выбрать этот запрос как единственный источник данных. Выберем из него поля *Группа*, *Альбомы* и *Общее число композиций* (рис. 6.27).

| Поле | Группа | Альбом | Общее число композиций |
|---|---|---|---|
| Псевдоним | | | |
| Таблица | ЗапросАльбомы | ЗапросАльбомы | ЗапросАльбомы |
| Сортировка | по возрастанию | | |
| Функция | **Group** | **Количество** | **Сумма** |

**Рис. 6.27**

Для того чтобы создать итоговый запрос, мы используем строку *Функция*. В столбце *Группа* выбрана операция *Group*, это означает, что итоговые данные вычисляются для каждой группы. В столбцах *Альбом* и *Общее число композиций* выбраны функции *Количество* и *Сумма*, это значит, что для каждой группы мы подсчитаем количество альбомов и общее количество композиций во всех альбомах.

## Выводы

- Чтобы избежать дублирования данных, информацию в базах данных представляют в виде набора связанных таблиц.
- Каждая таблица в многотабличной БД хранит свойства объектов одного класса.
- Чаще всего используют связи между таблицами типа 1:$n$ — одной записи в первой таблице может соответствовать множество записей во второй таблице (но не наоборот).
- Внешний ключ — это неключевое поле таблицы, связанное с первичным ключом другой таблицы.
- В запросах можно объединять данные из нескольких связанных таблиц.
- Итоговый запрос — это запрос, в котором вычисляются общие характеристики по группам записей: количество, сумма, среднее арифметическое и др.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Почему собирать все данные в одной таблице во многих случаях невыгодно?
2. По какому принципу данные разбиваются на несколько таблиц?
3. Чем различаются понятия «ключ таблицы» и «внешний ключ таблицы»?
4. Как вы думаете, что означают связи 1:1 и $n$:$n$ между таблицами?
5. Как строится запрос с выбором данных из нескольких таблиц?
6. Чем отличается итоговый запрос от простого?
7. Как вы думаете, почему нельзя назвать запрос так же, как и таблицу?
8. Выполните по указанию учителя задания в рабочей тетради.

## Практические работы

Выполните практические работы:
№ 33 «Многотабличная база данных»;
№ 34 «Запросы в многотабличной базе данных».

## ЭОР к главе 6 из Единой коллекции цифровых образовательных ресурсов (school-collection.edu.ru)

www

Создание таблиц в режиме конструктора таблиц в СУБД Access
Ввод и просмотр данных в режиме таблицы в СУБД Access
Основные объекты базы данных в СУБД Access
Создание запроса на выборку в режиме конструктора запросов в СУБД Access
Создание запросов на добавление, удаление, обновление в СУБД Access
Установка связей между таблицами в СУБД Access
Интерактивный задачник, раздел «Логические выражения в запросах»
Интерактивный задачник, раздел «Реляционные структуры данных»

# Глава 7
# ИНФОРМАТИКА И ОБЩЕСТВО

## § 35
## История и перспективы развития компьютеров

***Ключевые слова:***

- Аналитическая машина
- ЭВМ
- поколения компьютеров
- персональные компьютеры
- суперкомпьютеры

### Вычисления до компьютеров

С незапамятных времён люди стремились облегчить себе вычисления. Первым счётным «прибором» были пальцы рук. Позднее для расчётов стали использоваться различные приспособления, такие как **абак** (в простейшем варианте — это дощечка, на которой раскладывались камешки), его русский вариант — **счёты**, и многие другие (рис. 7.1).

Абак (V век до н. э.) Счёты

**Рис. 7.1**

Первая механическая вычислительная машина была изготовлена известным французским учёным **Блезом Паскалем** в 1645 г. В дальнейшем было создано множество подобных машин (их часто называли **арифмометрами**), но все они могли работать лишь при непосредственном участии человека.

Используя дополнительные источники, найдите ответы на вопросы.

— Какие арифметические действия могла выполнять машина Паскаля?

-- Сколько лет было Б. Паскалю, когда он изобрёл счётную машину?

Идея об автоматических вычислениях по программе впервые была разработана английским учёным **Чарльзом Бэббиджем.** Он спроектировал и описал **Аналитическую машину,** набор устройств и принципы действия которой фактически повторились в компьютерах XX века. Бэббидж посвятил всю свою жизнь работе над машиной, но построить её из механических деталей не удалось: уровень техники XIX века не позволял изготовить столь сложный и точный механизм. Работающую программируемую вычислительную машину смогли создать только с появлением электронных схем в середине XX века.

**Ч. Бэббидж (1791–1871)**

Разработкой принципов программирования Аналитической машины Бэббиджа занималась **Ада Лавлейс** (дочь английского поэта Д. Г. Байрона). Её идеи оказали большое влияние на развитие программирования. Например, ей принадлежат термины «цикл» и «рабочая ячейка».

**Ада Лавлейс (1815–1852)**

Используя дополнительные источники, найдите ответы на вопросы.
— Какой язык программирования был назван в честь Ады Лавлейс?
— Когда и где был разработан этот язык? Где он использовался?

## Поколения компьютеров

Прогресс вычислительной техники во многом определялся развитием её **элементной базы,** т. е. тех базовых деталей (элементов), из которых собирались все устройства электронных машин. В истории вычислительной техники было четыре типа таких базовых элементов (рис. 7.2), поэтому говорят о четырёх поколениях компьютеров.


**Рис. 7.2**

Компьютеры **первого поколения** (примерно 1945–1955 годы) работали на электронных лампах. Первой **электронно-вычислительной машиной (ЭВМ**[1]**)**, которая могла выполнять вычисления по заложенной в неё программе, принято считать **ЭНИАК** (сокращение от английского словосочетания *Electronic Numeric Integrator and Computer*) — рис. 7.3 (фото с сайта **www.fi.edu**). Эта ЭВМ была построена в 1944 году в США.

**Рис. 7.3**

www Используя дополнительные источники, выясните, какие ЭВМ первого поколения были построены в СССР.

**Второе поколение** (примерно 1955—1965 годы) появилось тогда, когда на смену лампам в электронных схемах пришли **транзисторы** — приборы для управления электрическими сигналами на основе кристаллов кремния. На транзисторах удалось собрать логические элементы для выполнения вычислений и элементы памяти.

www Используя дополнительные источники, найдите ответы на вопросы.
— Кто и когда изобрёл транзистор?
— Когда и где была построена первая ЭВМ на транзисторах? Как она называлась?

---

[1] Термин «ЭВМ» использовался в литературе приблизительно до 80-х годов XX века. Сейчас все разновидности вычислительной техники называют компьютерами, но старые модели по традиции именуются ЭВМ.

ЭВМ на транзисторах были значительно меньше и имели существенно более высокое быстродействие; они потребляли гораздо меньше энергии, были надёжнее и не требовали таких громоздких систем отвода тепла, как ламповые машины. Многие машины второго поколения уже помещались в обычной комнате среднего размера.

Одна из наиболее быстродействующих ЭВМ второго поколения — **БЭСМ-6** (СССР, 1967 год), построенная под руководством **Сергея Алексеевича Лебедева**. Некоторые ЭВМ этой серии использовались до начала 90-х годов XX века (рис. 7.4).

**С. А. Лебедев (1902–1974)**

**Рис. 7.4.** ЭВМ БЭСМ-6. Фото из Единой коллекции цифровых образовательных ресурсов (school-collection.edu.ru).

Появление **третьего поколения** ЭВМ (примерно 1967–1975 годы) связано с изобретением *интегральной микросхемы*. Так называется кристалл, в котором размещается целая электронная схема: несколько транзисторов, резисторы, конденсаторы и другие радиодетали.

Используя дополнительные источники, выясните, кто и когда разработал первую интегральную микросхему.

Казалось бы, размеры этих ЭВМ снова должны были существенно уменьшиться, но этого не произошло. Дело в том, что ЭВМ третьего поколения были предназначены для коллективной (многопользовательской) работы. Это было время крупных вычислительных центров, предоставлявших услуги огромному числу пользователей. Поэтому главное внимание уделялось не уменьшению

размеров и стоимости машин, а повышению их вычислительной мощности при обработке больших объёмов данных.

В это время начали выпускать семейства вычислительных машин, которые совместимы между собой как аппаратно (все устройства сконструированы по одинаковым стандартам), так и программно (имеют одинаковую систему команд).

Впервые идею общей архитектуры, обеспечивающей выполнение написанных ранее программ на любой новой модели, предложила фирма **IBM**, которая разработала семейства больших ЭВМ **IBM/360** и **IBM/370**.

www Используя дополнительные источники, выясните, какие семейства ЭВМ третьего поколения разрабатывались в нашей стране.

В результате прогресса в электронике существенно увеличилась плотность «упаковки» элементов на кристалле, и в одной микросхеме стали собирать целые узлы, например **микропроцессор**. Такие микросхемы стали называть **БИС** (большие интегральные схемы, от 1000 до 10 000 элементов на кристалле), а позднее — **СБИС** (сверхбольшие ИС, более 10 000 элементов). Именно они стали основой **четвёртого поколения компьютеров**, которое существует примерно с 1975 года до настоящего времени.

www Используя дополнительные источники, выясните, когда и кем был разработал первый микропроцессор. Как он назывался и для чего бы предназначен?

Увеличение плотности схемы позволило повысить быстродействие и надёжность компьютеров. Однако при этом увеличивается теплоотдача от миниатюрных деталей, поэтому требуются специальные меры по отводу тепла (например, установка вентиляторов охлаждения).

Первый процессор **Intel 8080**, предназначенный специально для компьютеров, был выпущен в 1974 году. На его базе был разработан **персональный компьютер Альтаир**, имевший большой коммерческий успех. Он вошел в историю ещё и потому, что в 1975 году студент **Билл Гейтс** со своим другом **Полом Алленом** написали для него транслятор языка программирования *BASIC*. Чуть позднее они создали известную компанию **Microsoft**.

В 1976 году два молодых приятеля Стив Джобс и Стив Возняк в гараже

**Б. Гейтс и П. Аллен**

родителей Джобса собрали ПК **Apple**, положивший начало известному ныне семейству компьютеров. А в 1981 году был продемонстрирован первый компьютер другого семейства — **IBM PC** (*IBM Personal Computer*), потомки которого в нашей стране особенно широко распространены.

Важное направление в компьютерах четвёртого поколения — **параллельная** (одновременная) **обработка данных**. Если задачу удаётся разбить на независимые друг от друга подзадачи, то их не обязательно выполнять друг за другом, они могут для экономии времени работать одновременно. Правда, для этого требуется несколько процессоров, но современный уровень техники это позволяет. Более того, в последние годы построены многоядерные процессоры, т. е. несколько процессоров «упакованы» в одном кристалле.

Мощные многопроцессорные компьютеры, в которых выполняется параллельная обработка данных, называют **суперкомпьютерами**. Это уникальные устройства, поэтому они изготавливаются штучно. В литературе часто упоминаются суперкомпьютеры серии **CRAY**, разработанные под руководством **Сеймура Крэя**.

Используя дополнительные источники, выясните, когда был построен первый суперкомпьютер фирмы *CRAY*. Какое быстродействие он имел? www

Все развитые страны ведут жёсткую конкуренцию в области суперкомпьютеров, поскольку обладание такой техникой позволяет решать стратегически важные вычислительные задачи:

- исследование геофизики Земли, прогнозирование изменений климата на планете;
- создание математических моделей молекул (полимеров, кристаллов и т. п.), синтез новых материалов и лекарств;
- расчёт процессов горения и взрыва, а также моделирование других физических задач (полёт летательных аппаратов, расчёт на прочность кузовов автомобилей);
- моделирование и прогнозирование ситуации в экономике;
- моделирование работы человеческого мозга;
- расчеты процессов нефте- и газодобычи, а также сейсморазведки недр;
- проектирование новых электронных устройств.

В 2009 году в МГУ введён в строй самый мощный **российский суперкомпьютер «Ломоносов»** производительностью около 400 Тфлопс (рис. 7.5[1]). В его состав входят 8892 многоядерных процессора (общее число ядер — 35776). На момент запуска «Ломоносов»

[1] Фотография предоставлена компанией «Т-Платформы» (**t-platforms.ru**).

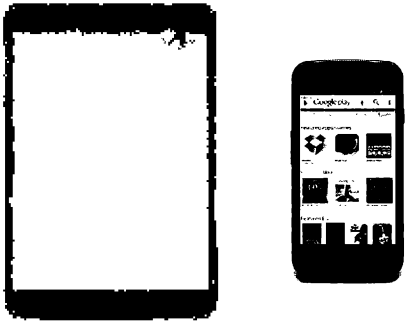
занимал в мировом рейтинге суперкомпьютеров *Top500* двенадцатое место. В 2014 году запущен новый суперкомпьютер **«Ломоносов-2»**, который в несколько раз быстрее своего предшественника.

Используя дополнительные источники, выясните, что такое «флопс» и «Тфлопс».

Используя дополнительные источники, выясните, какой суперкомпьютер считается самым мощным в мире на сегодняшний день. Каково его быстродействие?

**Рис. 7.5**

Настоящая революция в информационных технологиях связана с появлением разнообразных **мобильных компьютеров: ноутбуков, планшетных компьютеров, смартфонов, электронных книг** (рис. 7.6). Многие из них имеют сенсорные экраны и могут выполнять функции мобильного телефона и GPS-навигатора.



Ноутбук

Планшетный компьютер

Смартфон

**Рис. 7.6**

## Рост возможностей компьютеров

С каждым новым поколением вычислительных машин развивались их **аппаратные возможности**. ЭВМ становились всё более мощными и универсальными. Одновременно с этим увеличивалось разнообразие и сложность **внешних устройств**, присоединяемых к ЭВМ.

В самых первых ЭВМ данные вводились с помощью переключателей, а выводились на индикаторные лампочки. Потом появились клавиатуры и устройства ввода с перофлент и перфокарт, где двоичные данные кодировались в виде отверстий на бумажных лентах и карточках (рис. 7.7).

Перфолента Перфокарта

**Рис. 7.7**

Для вывода стали использоваться печатающие устройства; сначала они могли выводить на бумагу только столбики цифр, потом — буквы и другие символы. Для построения графиков и чертежей были разработаны **графопостроители**.

**Мониторы** (сначала текстовые, а затем и графические) значительно облегчили общение человека с компьютером. Были созданы устройства памяти на магнитных лентах и дисках (рис. 7.8).

Магнитная лента Пакет магнитных дисков (29 Мбайт)

**Рис. 7.8**

Современные компьютеры окружены огромным количеством самых разнообразных внешних устройств: это устройства для

хранения данных на оптических дисках, мышь, джойстик, шлемы виртуальной реальности и др. С помощью кабелей и беспроводных соединений к ним можно подключать устройства бытовой электроники (фотоаппараты, музыкальные плееры, сотовые телефоны и др.).

Увеличение разнообразия внешних устройств неизбежно вело к расширению **обрабатываемых типов данных**. Первоначально машины создавались только для обработки чисел (не случайно их и назвали вычислительными!). Когда появились печатающие устройства, стало возможно печатать тексты. Для этого машины стали «осваивать» всё более сложные действия над символами. Затем компьютеры «научились» работать с графической информацией, обрабатывать аудио- и видеоданные.

Каждое новое поколение компьютеров расширяет возможности **программного обеспечения (ПО)**. Использовать современный компьютер без ПО практически невозможно. Стоимость ПО нередко намного превышает стоимость аппаратной части (в первых машинах было наоборот!).

Самые первые программы разрабатывали на машинном языке «нулей и единиц» хорошо подготовленные специалисты. Стандартного программного обеспечения тогда практически не было. Потом появились первые языки программирования, такие как **Фортран** (1957) и **Алгол** (1960). Написать программу на таком языке было значительно проще. Дальнейшее развитие вычислительной техники привело к созданию **операционных систем** (ОС), которые управляли работой компьютеров в многопользовательском режиме и большим количеством внешних устройств (в первую очередь магнитными дисками). Для решения задач в конкретных областях создавались **пакеты прикладных программ**.

Современный пользователь управляет компьютером с помощью меню, кнопок и сенсорных экранов. Появились программы, предназначенные специально для смартфонов и планшетных компьютеров. В человеко-машинном общении наблюдается переход от машинного языка к языкам, естественным для человека.

## Перспективы развития компьютеров

Все современные компьютеры построены на основе идей четвёртого поколения. Для движения вперёд нужны новые идеи.

Разработчики процессоров постоянно стремятся повысить скорость их работы (быстродействие). К сожалению, в последние годы быстродействие процессоров практически не растёт. Дело в том, что увеличивать его бесконечно не получится из-за ограничений, связанных с действием законов физики.

Для того чтобы обрабатывать данные, части процессора должны обмениваться сигналами. Скорость передачи электрических сигналов огромна, но не бесконечна — она не может быть больше, чем скорость света в вакууме. Поэтому единственный способ снизить время передачи данных — уменьшить размеры элементов процессора. Но при этом появляются другие проблемы.

- Во-первых, чем мельче детали, тем сложнее их изготовить с нужной точностью.
- Во-вторых, при обмене сигналами электронные схемы сильно нагреваются и при перегреве могут перестать работать. Чем мельче элементы, тем труднее охладить процессор.
- В-третьих, если проводники, проводящие электрический ток, расположить слишком близко, между ними может произойти короткое замыкание, и вся схема выйдет из строя.

Таким образом, законы физики не позволяют безгранично увеличивать быстродействие процессора, и каждый новый шаг в этом направлении даётся всё с бóльшим и бóльшим трудом. Сейчас для увеличения быстродействия применяют **многоядерные процессоры** — несколько процессоров в одном кристалле кремния. Однако эта идея тоже не решает всех проблем.

- Прежде всего, далеко не все задачи можно разделить на части так, чтобы несколько процессоров могли выполнять эти части одновременно (использовать параллельные вычисления).
- Если несколько процессоров решают одну задачу, они должны обмениваться данными. Обмен между процессорами происходит медленнее, чем внутри одного процессора. Из-за этих потерь времени вся экономия от параллельных вычислений может быть «съедена», так что никакого увеличения скорости работы не получится.

Учёные ищут неэлектронные средства хранения и обработки данных. В первую очередь попытались использовать в качестве носителя информации свет — так появились **оптические процессоры**. В них можно применять параллельную обработку данных, например одновременно выполнять какую-то операцию со всеми пикселями изображения.

Большие надежды связаны с разработкой **квантовых компьютеров**, в которых применяются идеи квантовой физики, описывающей законы микромира и поведение элементарных частиц. Особые свойства *кубитов* — квантовых битов — позволяют организовать параллельную обработку данных так же, как и в многопроцессорных системах. Поэтому многие задачи, для решения которых сейчас не хватает вычислительных ресурсов (например, взлом систем шифрования), будут достаточно быстро решены, как только квантовый компьютер будет построен.

Ведётся разработка **биологических компьютеров** (*биокомпьютеров*), которые работают как живой организм. Ячейки памяти биокомпьютеров — это молекулы сложных органических соединений, например молекулы ДНК, в которых хранится наследственная информация. Сам процесс вычислений — это химическая реакция, результат — состав и строение получившейся молекулы.

Проводятся также исследования в области **нанотехнологий**, с помощью которых планируют построить транзистор размером с молекулу.

## Выводы

- Выделяют четыре поколения компьютеров в зависимости от элементной базы.
- Компьютеры первого поколения (1945–1955) строились на электронных лампах, второго поколения (1955–1965) — на транзисторах, третьего поколения (1965–1975) — на интегральных микросхемах, четвёртого поколения (после 1975) — на сверхбольших интегральных микросхемах.
- Современные компьютеры вплотную подошли к пределу быстродействия, достижимого для электронных приборов.
- Наиболее перспективные направления развития компьютерной техники — квантовые компьютеры, биокомпьютеры и нанокомпьютеры.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. По какому принципу ЭВМ делятся на поколения?
2. Как вы понимаете термин «персональный компьютер»?
3. Что можно сказать по поводу роли программного обеспечения: уменьшается она или увеличивается по сравнению с предыдущими поколениями?
4. Предположим, что появился процессор с каким-то принципиально новым свойством. Как быстро этим свойством смогут воспользоваться потребители? Какова роль программного обеспечения в этом?
5. Быстродействие вычислительной техники постоянно растёт. Как же тогда объяснить, что пользователи жалуются на «медлительные» компьютеры и все время стараются купить новые, ещё более производительные?

6. Как вы думаете, влияет ли развитие программных средств на развитие аппаратной части?
7. Зачем были созданы языки программирования?
8. Когда появились операционные системы и с чем это связано?
9. Насколько сейчас, по-вашему, актуально умение программировать? Попробуйте найти аргументы «за» и «против» (учитывайте разные цели работы людей на компьютере).
10. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «Чарльз Бэббидж и Аналитическая машина»
б) «Первая программистка»
в) «Персональные компьютеры»
г) «Суперкомпьютеры»
д) «Мобильные устройства»
е) «Большие данные (Big Data)»

### Интересные сайты

**computer-museum.ru** — виртуальный компьютерный музей
**computerhistory.narod.ru** — виртуальный музей истории вычислительной техники в картинках
**lib.ru/MEMUARY/MALINOWSKIJ/0.txt** — Б. Н. Малиновский «История вычислительной техники в лицах»

# § 36
# Информация и управление

***Ключевые слова:***

- система
- кибернетика
- система управления
- разомкнутая система
- замкнутая система
- обратная связь
- автоматическая система
- автоматизированная система
- адаптивная система

## Кибернетика

Человек с древних времён хотел использовать предметы и силы природы в своих целях, т. е. **управлять** ими. В середине XX века ученые поняли, что управление неразрывно связано с информацией и информационными процессами. В 1948 году появилась книга американского математика **Норберта Винера** «Кибернетика, или управление и связь в животном и машине», с которой началось развитие новой науки — *кибернетики*.

**Норберт Винер (1894–1964)**

**Кибернетика** — это наука, изучающая общие закономерности процессов управления и передачи информации в машинах, живых организмах и обществе.

Используя дополнительные источники, выясните, от какого слова произошло слово «кибернетика». Что оно означает?

Основная идея Н. Винера состояла в том, что управление в любых системах — технических, биологических, общественных — определяется одними и теми же законами и тесно связано с обменом информацией. Но прежде, чем говорить про управление, нужно разобраться в том, что такое система.

## Что такое система?

Представьте себе, что человек, у которого поднялась температура, приходит к врачу. Врач может реагировать на его жалобу двумя способами:

- просто дать жаропонижающее или...
- попытаться выяснить причину повышения температуры и лечить именно заболевший орган.

В первом случае врач применил простейшую модель: если повысилась температура, нужно дать таблетку, чтобы её снизить. Во втором варианте врач понимает, что повышение температуры вызвано какой-то болезнью в организме, и лечить нужно именно эту болезнь. Он рассматривает человека как единое целое, как *систему*, т. е. использует *системный подход*.

Используя дополнительные источники, выясните, от какого слова произошло слово «система». Что оно означает?

**Система** — это группа объектов и связей между ними, выделенных из среды и рассматриваемых как одно целое (рис. 7.9).




**Рис. 7.9**

Системой можно назвать организм животного и человека, автомобиль, компьютер, семью, общество, и вообще любой достаточно сложный объект, который можно разбить на части.

С какими системами вы познакомились в курсе информатики 7–9 классов?

Объекты, из которых строится система, называют ее **компонентами**. За счёт связей между частями система обладает особыми свойствами, которых нет ни у одного отдельного компонента.

**Системный эффект** состоит в том, что свойства системы нельзя свести к «сумме» свойств её компонентов.

Например, скопление клеток — это не живой организм; процессор, память и устройства ввода и вывода, не связанные между собой — это не компьютер. Все части самолёта тяжелее воздуха, т. е. каждая из них стремится упасть на землю, однако составленная из них система (самолёт) летает.

Одни и те же компоненты, связанные по-разному, могут представлять собой совершенно разные системы.

Используя дополнительные источники, выясните, в чём сходство и различие между алмазом и графитом.

**Подсистема** — это компонент системы, в котором можно выделить отдельные части и связи между ними.

Как правило, для каждой системы существует **надсистема** — система более высокого уровня. Например, процессор — это подсистема в составе компьютера, а сам компьютер — подсистема в составе компьютерной сети (надсистемы).

Простейшие компоненты системы, которые в данной задаче (!) нет смысла «разбирать» на части, называются её **элементами.** Например, винтик в составе машины можно считать элементом системы. Однако с точки зрения химии металл состоит из атомов, связанных в кристаллическую решетку, поэтому при изучении строения вещества тот же винт можно считать системой. На рис. 7.10 система $S$ состоит из двух подсистем $S_1$ и $S_2$, а также элементов $Г$ и $Д$.

**Рис. 7.10**

**Системный подход** состоит в том, что объект исследования рассматривается как система с учётом всех взаимосвязей между её частями. В середине XX века появилось новое научное направление — системный анализ. Задача **системного анализа** — изучение сложных систем (технических, биологических, экономических, социальных).

## Информация и управление

Во многих системах взаимодействие между подсистемами можно рассматривать как управление. Такие системы (их называют **системами управления**) всегда содержат **управляющий объект** (его также называют **регулятором**) и **управляемый объект** (его также называют просто объектом или объектом управления).

Цель управления задаётся надсистемой, т. е. регулятор является только исполнителем. Например, водитель служебной машины никогда не решает сам, куда везти своего начальника.

Регулятор действует на объект (управляет им) так, чтобы цель была достигнута (рис. 7.11). Это значит, что при управлении передаётся информация.

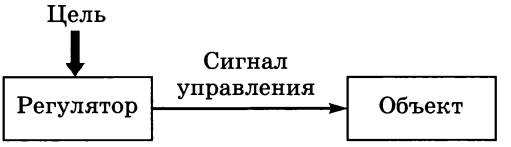



**Рис. 7.11**

На схеме, изображённой на рис. 7.11, регулятор не получает никакой информации о состоянии объекта, т. е. действует «вслепую». Такие простейшие системы управления называют **разомкнутыми**, в них информация идёт только в одну сторону — от регулятора к объекту. Приведём некоторые примеры разомкнутых систем:

- светофор (выдаёт вам световой сигнал, но не получает информации от вас);
- табло на вокзале или в аэропорту;
- начальник, который отдаёт приказания, но не проверяет их выполнение.

Разомкнутые системы используются в двух случаях. Во-первых, когда регулятору всё равно, реагирует ли объект на управление (светофор, табло). Во-вторых, когда регулятор настолько хорошо знает объект (имеет точную модель объекта), что уверен в выполнении своих команд. Например, начальник, уверенный в исполнительности своих подчинённых, может позволить себе (иногда) не контролировать их.

Достоинство разомкнутых систем — в их простоте. Например, во многих случаях частоту вращения электродвигателя регулируют простым реостатом, при этом не нужно ставить сложный и дорогостоящий датчик частоты вращения.

В ответственных случаях разомкнутые системы лучше не применять, потому что они имеют серьёзные *недостатки*:

- для достижения цели регулятор должен иметь точную модель объекта; на практике такая модель чаще всего неизвестна;
- на объект всегда действует внешняя среда, и из-за этого воздействия свойства объекта могут непредсказуемо меняться; регулятор в разомкнутой системе не получает об этом никакой информации.

Даже очень исполнительные работники могут в какой-то момент «схалтурить» из-за личных проблем. Поэтому нельзя

гарантировать, что цель управления будет достигнута с помощью разомкнутой системы.

## Обратная связь

Для достижения цели регулятор должен получать информацию от объекта. Этот канал передачи информации называют *обратной связью*, потому что по нему информация передаётся (с помощью датчиков) в «обратном» направлении — от объекта к регулятору (рис. 7.12).

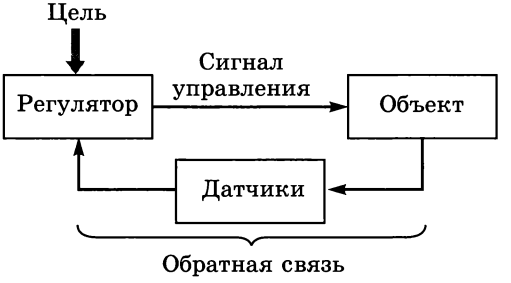



**Рис. 7.12**

**Обратная связь** — это канал передачи информации от объекта к регулятору.

Системы с обратной связью называют *замкнутыми* системами управления, потому что информация передаётся по замкнутому контуру (регулятор — объект — датчики — регулятор).

**Замкнутая система управления** — это система управления с обратной связью.

Задача регулятора — сравнить поставленную цель и реальное состояние объекта, а потом выдать нужный управляющий сигнал. Если цель достигнута, как правило, сигнал управления больше не изменяется.

Например, регулировщик, управляющий движением на перекрёстке, использует обратную связь. Он оценивает (с помощью глаз-датчиков) количество машин, движущихся в разных направлениях, и «открывает» то или другое направление. Человек управляет лучше, чем светофор, потому что учитывает фактическую обстановку, которая может изменяться непредсказуемо.

Чаще всего реальные системы управления (в природе, технике, обществе) — это замкнутые системы. Они могут решать задачу даже тогда, когда модель объекта неточна или свойства объекта изменяются во времени, позволяют учитывать случайные воздействия внешней среды. За это приходится расплачиваться усложнением системы — нужны датчики, которые передают информацию от объекта.

Представьте себе, что в тёмной комнате упала на пол пуговица, и вы не слышали, как она стукнулась. Сказать точно, где лежит пуговица, не сходя с места, невозможно, потому что вы этого не знаете (модель неточна). Вы можете попробовать искать её на ощупь (используя обратную связь — осязание). Если и это не получается, можно включить свет (зрительная обратная связь даёт больше информации), и тогда пуговица точно будет найдена. В этом примере датчиками служат наши органы чувств.

Системы с обратной связью широко используются в технике: это автопилоты на самолётах и судах, регуляторы частоты вращения турбин, роботы, оборудованные датчиками (в том числе устройствами «компьютерного зрения»).

Обратная связь существует и в обществе. Жалобы граждан должны помогать работе органов управления, сигнализируя о том, что не всё в порядке. Не зря все официальные учреждения имеют контактные телефоны и сайты в Интернете, с которых можно отправить сообщение с просьбой или предложениями по их работе.

Обратная связь, при которой регулятор стремится уменьшить разницу между заданной целью и фактическим состоянием объекта, называется *отрицательной*, потому что изменение сигнала управления в этом случае определяется разностью между целью и сигналом обратной связи.

Если нужно быстро перевести объект из одного состояния в другое, иногда применяют *положительную* обратную связь, при которой регулятор стремится увеличить разницу между заданным значением и сигналом обратной связи. Часто в этом случае система переходит в колебательный режим, поэтому положительная обратная связь используется в генераторах колебаний. Другой пример положительной обратной связи — цепные реакции: горение, взрыв, ядерные реакции.

## Какие бывают системы управления?

Системы управления делятся на автоматические и автоматизированные. **Автоматические системы** работают без участия человека (например, автопилот). В **автоматизированных системах** сбор и

предварительную обработку информации выполняет компьютер, а решение по поводу управления принимает человек.

Многие системы умеют «подстраиваться» под изменения внешних условий или изменение свойств объекта управления. Они называются **адаптивными**. В технических системах адаптивные регуляторы могут управлять объектом, свойства которого очень плохо известны или меняются. Они решают две задачи — управляют объектом и одновременно уточняют модель, перестраивая закон управления при изменении этой модели.

## Выводы

- Кибернетика — это наука, изучающая общие закономерности процессов управления и передачи информации в машинах, живых организмах и обществе.
- Система — это группа объектов и связей между ними, выделенных из среды и рассматриваемых как одно целое. Системный эффект состоит в том, что свойства системы нельзя свести к «сумме» свойств её компонентов.
- Подсистема — это компонент системы, в котором можно выделить отдельные части и связи между ними.
- Система управления содержит регулятор (управляющий объект) и объект управления.
- Цель управления задаётся надсистемой.
- Управление (управляющее воздействие) — это сигнал, который поступает от регулятора к объекту.
- Обратная связь — это канал передачи информации от объекта к регулятору.
- Замкнутая система управления — это система с обратной связью.
- Автоматическая система работает без участия человека.
- В автоматизированных системах сбор и обработку информации выполняет компьютер, а решения принимает человек.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Как связана информация с управлением?
2. Что такое система управления? Какие элементы в ней всегда есть?
3. Приведите примеры разомкнутых систем. В чём их достоинства и недостатки?
4. Почему разомкнутые системы не используют в ответственных случаях?

5. Назовите достоинства и недостатки замкнутых систем.
6. Приведите примеры управления с обратной связью из разных областей, из вашей жизни.
7. Чем различаются автоматические и автоматизированные системы?
8. Выполните по указанию учителя задания в рабочей тетради.

### Подготовьте сообщение

а) «Вклад Н. Винера в науку»
б) «Системы управления в природе»
в) «Системы управления в обществе»
г) «Отрицательная и положительная обратная связь»
д) «Что такое адаптивная система?»

# § 37
# Информационное общество

***Ключевые слова***:

- информационное общество
- информационные технологии
- информационная культура
- стандарт

## Что такое информационное общество?

По мере развития человечества постоянно повышалась роль информации в жизни общества и отдельного человека. Самые важные достижения в этой области — это:

- *письменность*;
- *книгопечатание;*
- *средства связи* (телеграф, телефон, радио, телевидение — конец XIX–начало XX века);
- *компьютеры* (вторая половина XX века).

Используя дополнительные источники, выясните, когда и где были изобретены письменность, книгопечатание, средства связи, компьютеры.

Сейчас считается, что мы переходим от индустриального общества, в котором главная роль отводится промышленности, к *информационному* (постиндустриальному).

**Информационное общество** — это такая ступень развития цивилизации, на которой главными продуктами производства становятся информация и знания.

Переход к информационному обществу часто называют **информатизацией**.

Япония, США и некоторые страны Европы (например, Германия) уже приблизились к информационному обществу. Об этом можно судить по следующим признакам:

- внедрение компьютеров и информационных технологий во все сферы жизни;
- развитие *коммуникаций* (средств связи) — компьютерных сетей, сотовой связи и т. п.;
- необходимость компьютерной грамотности для любого человека;
- свобода доступа к информации;
- доступность образования, в том числе дистанционного (через Интернет);
- изменение структуры экономики — всё больше людей занимаются не производством товаров, а получением и обработкой информации;
- изменение уклада жизни людей (например, общение через Интернет вместо личной встречи; интернет-магазины и использование электронных денег и т. п.).

В результате индустриализации машины заменили человека на производстве, а теперь компьютеры начинают самостоятельно (без участия человека) собирать и обрабатывать информацию, заменяя людей в умственной работе, которую до этого мог сделать только человек.

С одной стороны, переход к информационному обществу облегчает жизнь людей, потому что на всех рутинных и тяжёлых работах их заменяют компьютеры и роботы. С другой стороны, существует немало *негативных последствий*:

- усиление влияния средств массовой информации (СМИ), с помощью которых несколько человек могут влиять на большие массы людей; так, широкое освещение террора в СМИ приводит к значительному влиянию террористических организаций на нашу жизнь, поэтому некоторые учёные говорят о том, что общество теряет устойчивость;
- в результате доступности информации нередко разрушается частная жизнь людей и целых организаций (например, в Интернет часто попадают сведения, не предназначенные для всеобщего доступа);

- в огромном потоке информации очень сложно выбрать качественные и достоверные данные (в Интернете, например, очень много ложной и противоречивой информации);
- личное общение людей всё больше заменяется общением в Интернете, реальная жизнь подменяется виртуальной;
- многим людям старшего поколения очень сложно приспособиться к меняющимся условиям, они могут оказаться «за бортом».

## Информационные технологии

Один из признаков информационного общества — широкое внедрение информационных технологий во все сферы жизни.

Любая **технология** — это способ сделать «продукт» из исходных материалов (с гарантированным результатом!).

---

**Информационные технологии** (IT-технологии) — это технологии, связанные с использованием компьютеров для хранения, обработки, передачи и защиты информации.

---

К информационным технологиям относятся:

- подготовка цифровых документов;
- поиск информации;
- телекоммуникации (компьютерные сети, Интернет, электронная почта);
- автоматизированные системы управления (АСУ);
- системы автоматизированного проектирования (САПР);
- геоинформационные системы (на основе карт, снимков со спутника);
- обучение с помощью компьютеров (электронные учебники, компьютерные тренажёры, дистанционное обучение через Интернет).

Остановимся на некоторых информационных технологиях более подробно.

Примером **АСУ** может служить система, предназначенная для автоматизации работы ресторанов. На рабочих местах барменов, официантов, менеджеров и администраторов установлены персональные компьютеры, связанные в единую локальную сеть. При получении заказа информация о нём сразу передаётся в базу данных и распечатывается на принтерах, установленных на кухне.

В промышленности широко используются **автоматизированные системы управления технологическими процессами (АСУ ТП)**. Представим себе организацию, которая отвечает за работу нескольких дизель-генераторов, расположенных в разных местах. Каждый генератор связан с управляющим компьютером, который

собирает информацию о работе установки и передаёт ее (по кабельной линии или через радиомодем) в центральный офис, где за обстановкой следит диспетчер. В случае необходимости диспетчер передаёт обратно команду на изменение режима работы генератора или принимает решение о выезде ремонтной бригады.

Современный инженер для подготовки чертежей не использует традиционные инструменты: кульман, линейку, лекала, перья. Всё делается на компьютере в **системах автоматизированного проектирования**. Все **САПР** используют векторное кодирование изображений. Наиболее известны системы:

- **AutoCAD** — самая популярная система автоматизированного проектирования и черчения;
- **ArchiCAD** — для проектирования зданий, ландшафтов и мебели;
- **OrCAD** — для проектирования электронных схем;
- **КОМПАС** — отечественная САПР, позволяющая оформлять чертежи в соответствии с российскими стандартами.

**Геоинформационные системы** (географические информационные системы, **ГИС**) — это инструменты для работы с цифровыми картами. Они могут строиться на основе карт в векторном формате или спутниковых снимков поверхности Земли. На основной слой накладываются растровые и векторные слои с дополнительной информацией: дороги, города и посёлки, улицы, другие объекты. С помощью ГИС можно измерить расстояние между точками, проложить маршрут по существующим дорогам, и даже посмотреть, как выглядят улицы некоторых крупных городов — это позволяют сделать сервисы **Яндекс.Панорамы** и **Google Street View**.

Информационные технологии открывают большие возможности для **обучения**. Электронные учебники позволяют, в отличие от бумажных, использовать *мультимедийные* технологии: звук, видео, анимацию. Они становятся интерактивными, т. е. учащийся может использовать их как тренажёры, выполняя задания. Некоторые обучающие программы строят модель конкретного ученика и выбирают «маршрут», оптимальный именно для него.

С развитием Интернета стало возможным **дистанционное обучение**. На сервере выкладываются учебные материалы и практические задания. Группе учащихся выделяется преподаватель (*тьютор*), у которого они могут свободно консультироваться, используя электронную почту, чаты и др. Все проблемы можно обсудить с однокурсниками на форуме. С помощью такой системы можно осваивать курс университета, не выходя из дома (нужен только доступ в Интернет).

Используя дополнительные источники, выясните, от какого иностранного слова произошло слово «тьютор».

Для освоения сложной техники используют **компьютерные тренажёры.** Они необходимы для обучения персонала в тех случаях, когда ошибка при работе с реальным объектом может привести к тяжёлым последствиям. Очень важны тренажёры для обучения лётчиков и судоводителей, они позволяют отработать действия в разнообразных аварийных ситуациях. Часто тренажёры применяются в связке с реальной аппаратурой[1] (рис. 7.13).

Тренажёр для обучения лётчиков

Тренажёр для обучения судоводителей

**Рис. 7.13**

Простейшие тренажёры — это компьютерные игры-симуляторы, моделирующие полёт самолета или вождение автомашины. Они строятся на основе математических моделей, которые приближённо описывают реальные объекты.

## Информационная культура

Возрастание роли информации в современном обществе требует от каждого человека (и от общества в целом) определённой культуры обращения с информацией и информационными технологиями, т. е. информационной культуры.

**Информационная культура общества** — это способность общества:

- эффективно использовать информационные ресурсы и средства обмена информацией;
- применять передовые достижения в области информационных технологий.

---

[1] Фотографии тренажёров предоставлены компаниями «Динамика» (**www.dinamika-avia.ru**) и «Транзас» (**www.transas.ru**).

**Информационная культура человека** — это его умение использовать современные технологии для решения своих задач, связанных с поиском и обработкой информации. Современный человек должен уметь:

- сформулировать свою потребность в информации;
- находить нужную информацию, используя различные источники;
- отбирать и анализировать информацию (в том числе конспектировать, составлять рефераты);
- представлять информацию в разных видах;
- обрабатывать информацию и создавать новую информацию;
- использовать информацию для принятия решений.

Это не просто какие-то требования к «идеальному» человеку. Дело в том, что в новом информационном обществе успешность человека и его возможность реализовать все свои способности будут зависеть от его умения грамотно работать с информацией.

## Личное информационное пространство пользователя

Сейчас бóльшая часть данных, необходимых человеку, хранится в цифровом виде, на компьютерах. Такие данные образуют **личное информационное пространство пользователя**. Для того чтобы человеку было легко находить, добавлять и изменять эти данные, нужно содержать их в порядке. Кроме того, надо следить, чтобы данные не были случайно потеряны и к ним не могли получить доступ посторонние.

Вы знаете, что данные в долговременной памяти хранятся в виде **файлов**. Имена файлов должны говорить о содержании этих файлов. Например, *Новый_файл.docx* — это неудачное имя файла, а *Схема_дирижабля.docx* — значительно лучше.

На компьютере обычно хранятся сотни ваших файлов, поэтому их нужно раскладывать в папки — объединять вместе файлы, которые связаны по какому-то признаку. Например, для фотографий можно создать папку *Фото*, в ней — вложенные папки для каждого года (или для каждого места, которое вы посетили).

К личному информационному пространству относится и **программное обеспечение**, которое вы используете — операционная система, прикладные программы и т. д. Операционную систему можно настраивать «на свой вкус», так, чтобы вам было удобно работать. Прежде всего, вы можете выбрать рисунок Рабочего стола, разместить на нём ярлыки только тех программ и доку-

ментов, которые вы часто используете. Программы, которыми вы не пользуетесь, лучше удалить с компьютера, часто после этого его работа ускоряется.

Многие пользователи хранят свои данные в **облачных хранилищах** (*Google Диск*, *Яндекс.Диск* и др.). Конечно, это очень удобно, ведь доступ к ним можно получить с любого компьютера, имеющего доступ к Интернету. Сервис *Google Документы* (*Google Docs*) позволяет нескольким авторам редактировать один документ, сохраняя всю историю изменений. В то же время нужно помнить о безопасности этих данных: они хранятся на серверах в Интернете и неизвестно, кто имеет к ним доступ. Поэтому размещать в облачных хранилищах секретные данные нельзя.

Для общения в сети Интернет вы можете создать свой блог, сайт, форум или страницу в социальной сети. Это тоже элементы вашего личного информационного пространства.

**Блог** — это личный сетевой дневник. Записи **(посты)** автора появляются в хронологическом порядке, поэтому блог можно использовать как доску объявлений (ленту новостей). Материалы блога можно обсуждать прямо в блоге — пользователи комментируют посты, автор отвечает на комментарии.

**Сайтом** обычно называют более профессиональный проект[1)], там меньше личного материала, статьи систематизированы (распределены по разделам). Часто сайт связан с работой автора, компании или с каким-то проектом (например, с разработкой программы).

**Форум** — это публичное обсуждение вопросов по какой-то тематике. На форуме можно обсуждать, например, разведение хомячков или программирование роботов.

На личной странице в **социальной сети** можно разместить любую информацию: сведения о себе, фотографии, звукозаписи, видеоролики и т. п. Причём часто пользователи сами заполняют подробную анкету о себе, размещают личные и семейные фотографии, которые они вряд ли показали бы первому встречному. Строго говоря, если вы хотите выложить в сеть какие-то фотографии, нужно убедиться, что все, кто на них изображён, не возражают против этого.

Когда вы пойдете устраиваться на работу в какую-то компанию, вашу страничку в сети, скорее всего, будут просматривать сотрудники службы безопасности. Бывает и так, что работников даже увольняют за некорректные комментарии или фотографии.

---

1) Часто считают, что блог — это один из видов сайта.

Ещё хуже, если ваши данные попадут в поле зрения злоумышленников. Например, они могут попытаться выманить деньги у вас или у ваших родственников. Поэтому личные данные (адреса, телефоны и т. п.) лучше не размещать в открытом доступе.

Что же ещё угрожает безопасности наших данных? Прежде всего, возможны **сбои аппаратуры**: отключение питания, повреждение носителей данных. При этом все данные на жёстком диске или флэш-накопителе могут потеряться. Поэтому необходимо делать резервное копирование — создавать копии важных данных в нескольких разных местах, например на другом компьютере, на оптических дисках, «флэшках», переносных жёстких дисках, или в облаке.

Пользователь может потерять данные по своей вине, например по ошибке удалить файлы или сохранить испорченную версию файла вместо правильной под тем же именем.

При установке нелицензионных программ на компьютер может проникнуть вредоносная программа, которая позволит кому-то управлять вашим компьютером через Интернет.

Наконец, очень опасны **внешние атаки из Интернета** со стороны программ, которые пытаются узнать ваши пароли к сайтам, данные банковских карточек и т. п. Абсолютной защиты от таких атак нет (кроме как отключить Интернет!), но во многих случаях самых простых мер оказывается достаточно:

- используйте антивирусную программу-монитор, которая постоянно находится в памяти и отслеживает подозрительную активность других программ;
- не открывайте электронные письма от неизвестных пользователей, без темы сообщения или без текста; не запускайте файлы-приложения к ним;
- не отвечайте на письма-спам;
- придумывайте сложные пароли: не менее 6–8 символов, включая прописные и строчные буквы и цифры.

## Электронная цифровая подпись (ЭЦП)

С давних времён для того, чтобы установить авторство бумажного документа, проверяли стоящие на нём подпись и печать. В современном мире всё чаще обмен документами между людьми и организациями происходит в электронном виде. При этом проверить авторство достаточно сложно: изображение подписи и печати на отсканированном изображении может быть подделано! Решить эту проблему удалось с помощью электронной цифровой подписи.

**Электронная цифровая подпись (ЭЦП)** — это набор символов, который получен в результате шифрования сообщения с помощью личного секретного ключа отправителя.

Особенность алгоритма шифрования ЭЦП состоит в том, что сообщение шифруется одним ключом (секретным, который знает только отправитель), а расшифровывается — другим, открытым, который знают все. Такое шифрование называется **асимметричным.**

Специальные удостоверяющие центры, честность которых не вызывает сомнения, выдают **сертификат** на электронную подпись — документ, подтверждающий, что открытый ключ действительно принадлежит организации или частному лицу. Свой секретный ключ владелец сертификата должен хранить в тайне.

Итак, отправитель вместе с исходным сообщением передаёт электронную цифровую подпись — такое же сообщение, зашифрованное с помощью своего секретного ключа. Получатель расшифровывает цифровую подпись с помощью открытого ключа. Если она совпала с незашифрованным сообщением, можно быть уверенным, что его отправил тот человек, который знает секретный ключ. Если сообщение было изменено при передаче, оно не совпадёт с расшифрованной цифровой подписью. Так как сообщение может быть очень длинным, для сокращения объёма передаваемых данных чаще всего шифруется не всё сообщение, а только его контрольная сумма, зависящая от всех битов сообщения.

Электронная подпись в России используется при обмене цифровыми документами между бухгалтериями предприятий, банками, Пенсионным фондом, торговыми организациями, судами. Согласно **Федеральному закону «Об электронной подписи»**, электронный документ, подписанный электронной цифровой подписью, признаётся равнозначным бумажному документу, подписанному собственноручной подписью.

## Информационная этика и право

Понятие «информационная культура» включает в себя этичное поведение при использовании информации. Неэтично подавлять высказывания других, даже если это противоречит вашим убеждениям. Неэтично беспокоить других и угрожать им. При возникновении конфликтов нужно применить все возможные меры, чтобы решить проблему без обращения в суд.

Неэтично распространять высказывания, изображения или мнения других без их согласия. Если вы хотите разместить в общем доступе (например, в социальной сети) информацию, касающуюся частной жизни других людей (фотографии, личную переписку), следует спросить их разрешения.

Многие материалы, размещённые в Интернете, охраняются **законами об авторских правах**. Поэтому эти материалы нельзя взять и использовать «просто так», как своё. Если вы включаете какую-то информацию (текст, картинку) в учебную работу (например, реферат), необходимо обязательно привести ссылку на сайт, где вы её взяли. В некоторых случаях, например при желании использовать чужой рисунок или текст на своем сайте или в печатном издании, нужно получить письменное согласие автора.

Использование чужого материала без ссылки (т. е. фактически присвоение чужих результатов) называется **плагиатом**. Если будет доказано, что этим автору причинён крупный материальный ущерб (более 50 тыс. рублей), начинает действовать Уголовный кодекс (УК РФ, статья 146), предусматривающий денежный штраф, исправительные работы или даже арест на срок до 6 месяцев.

Неправомерный доступ к чужой информации («взлом» сайтов, почтовых ящиков, личных страничек) — это уголовное преступление, которое наказывается лишением свободы на срок до 5 лет (УК РФ, статья 272). То же самое относится и к созданию, использованию и распространению вредоносных компьютерных программ (УК РФ, статья 273, до 7 лет лишения свободы).

С одной стороны, развитие компьютерной техники и коммуникационных технологий предоставляет широкие возможности и (кажущуюся) полную свободу. С другой стороны, в информационном обществе продолжают действовать все нормы права и морали, которые выработало человечество за свою историю.

## Стандарты в сфере информационных технологий

В современном обществе важную роль играют стандарты.

**Стандарт** — это нормативный документ, в котором определены требования к некоторому объекту или процессу.

Давайте представим себе, что никаких стандартов нет. Это значит, например, что все электрические лампочки могут иметь различный диаметр резьбы, и лампочка одной фирмы не подойдёт к светильнику другой. Мониторы для компьютеров разных фирм

будут в этом случае иметь разные разъёмы и соединительные кабели.

Чтобы не было такой неразберихи, вводят стандарты. Например, способ записи чисел в десятичной системе счисления и алфавиты различных языков — это стандарты докомпьютерной эпохи. В области современных информационных технологий существуют стандарты на:

- использование технических терминов (они определяют значение каждого термина);
- разъёмы (для кабелей питания, для видеосигналов и др.);
- величины напряжения и силы тока, которые подаются на различные устройства компьютера;
- протоколы обмена данными (например, для сети Интернет используется семейство протоколов TCP/IP);
- языки программирования;
- оформление документации и многое другое.

Профессиональные стандарты определяют требования к работникам разных профессий (программисту, системному администратору, тестировщику, техническому писателю и т. д.).

Разработкой международных стандартов в области информационных технологий занимаются Международная организация по стандартизации (**ISO**) и Международная электротехническая комиссия (**МЭК**).

Пример успешного применения стандартов — **открытая архитектура** компьютеров, которую предложила фирма IBM в конце XX века. Она опубликовала все стандарты, необходимые для подключения к компьютеру новых устройств, так что любой мог разработать новое устройство или программное обеспечение без покупки лицензий. Это привело к большому успеху серии персональных компьютеров IBM PC.

## Выводы

- Информационное общество — это такая ступень развития цивилизации, на которой главными продуктами производства становятся информация и знания.
- Информационные технологии — это технологии, связанные с использованием компьютеров для хранения, обработки, передачи, защиты, информации.
- Стандарт — это нормативный документ, в котором определены требования к некоторому объекту или процессу.

Нарисуйте в тетради интеллект-карту этого параграфа.

## Вопросы и задания

1. Чем информационное общество отличается от индустриального?
2. Назовите негативные последствия информатизации общества. Как, на ваш взгляд, с ними можно бороться?
3. Какие новые возможности для обучения появляются в информационном обществе? Видите ли вы какие-нибудь негативные стороны в компьютерном обучении?
4. Верно ли, что информационная культура и компьютерная грамотность — это одно и то же? Обоснуйте свой ответ.
5. На каких условиях можно использовать информацию, найденную в сети Интернет?
6. Что такое этичное поведение в Интернете?
7. Какие действия, связанные с информационными технологиями, являются уголовными преступлениями?
8. Объясните, зачем нужны стандарты в области информационных технологий.
9. Выполните по указанию учителя задания в рабочей тетради.

## Подготовьте сообщение

а) «Информатизация общества: плюсы и минусы»
б) «Дистанционное обучение»
в) «Компьютерные тренажёры»
г) «Стандарты в области информационных технологий»

## ЭОР к главе 7 из Единой коллекции цифровых образовательных ресурсов (school-collection.edu.ru)

История средств обработки информации
История средств передачи информации
История средств хранения информации
Поколения ЭВМ: поколение I
Поколения ЭВМ: поколение II
Поколения ЭВМ: поколение III
Поколения ЭВМ: поколение IV
Многопроцессорная архитектура
Многоядерный процессор
Компьютер и управление
Линейные алгоритмы управления

Нелинейные алгоритмы управления
Зарождение и предмет кибернетики
Вычислительный бортовой цифровой комплекс «Аргон-16»
Неустойчивость внутри нас
Информация в человеческом обществе — новостная информация
Информационные преступления и информационная безопасность
Меры обеспечения информационной безопасности
Виртуальные и реальные

# ОГЛАВЛЕНИЕ

*Учебное издание*

**Поляков** Константин Юрьевич

**Еремин** Евгений Александрович

**ИНФОРМАТИКА**

**9 класс**

Ведущий редактор *О. Полежаева*
Ведущие методисты *И. Сретенская, И. Хлобыстова*
Художник *Н. Новак*
Технический редактор *Е. Денюкова*
Корректор *Е. Клитина*
Компьютерная верстка: *Е. Голубова*

Подписано в печать 28.02.17. Формат 70x100/16. Усл. печ. л. 23,4.
Тираж 3000 экз. Заказ № м4311.

ООО «БИНОМ. Лаборатория знаний»
127473, Москва, ул. Краснопролетарская, д. 16, стр. 1,
тел. (495)181-5344, e-mail: binom@Lbz.ru
http://www.Lbz.ru, http://metodist.Lbz.ru

Отпечатано в филиале «Смоленский полиграфический комбинат»
ОАО «Издательство «Высшая школа». 214020, г. Смоленск, ул. Смольянинова, 1
Тел.: +7 (4812) 31-11-96. Факс: +7 (4812) 31-31-70
E-mail: spk@smolpk.ru http://www.smolpk.ru