

ФГОС

11



И. Г. Семакин  
Е. К. Хеннер  
Л. В. Шестакова

ИНФОРМАТИКА

1

УГЛУБЛЕННЫЙ УРОВЕНЬ



ИЗДАТЕЛЬСТВО

БИНОМ

**ФГОС**

**И. Г. Семакин, Е. К. Хеннер,  
Л. В. Шестакова**

# **ИНФОРМАТИКА**

**УГЛУБЛЕННЫЙ УРОВЕНЬ**

**Учебник для 11 класса**

**в 2-х частях**

**Часть 1**

Рекомендовано  
Министерством образования и науки  
Российской Федерации  
к использованию в образовательном процессе  
в имеющих государственную аккредитацию  
и реализующих образовательные программы  
общего образования образовательных учреждениях



Москва  
БИНОМ. Лаборатория знаний  
2014

## ОТ АВТОРОВ

*Уважаемые старшеклассники!*

Этот учебник предназначен для изучения курса «Информатика» в 11 классе на углубленном уровне. Вы уже не новички в информатике. В 7–9 классах вы изучали курс информатики для основной школы. В результате вы получили необходимые базовые знания и умения в этом предмете. В курсе для 7–9 классов вы познакомились с элементами всех основных разделов современной информатики: теоретической информатики, информационных и коммуникационных технологий, социальной информатики.

Освоив курс для основной школы, вы прошли через самый первый уровень погружения в информатику. Углубленный курс для старших классов, к изучению которого вы приступили в 10 классе и продолжаете изучать в 11 классе, — это второй уровень погружения. Тем из вас, кто после окончания школы поступит в вузы на профильные по отношению к информатике специальности, предстоит третий уровень погружения в этот предмет. Это уже будет профессиональный уровень, в результате освоения которого вы станете специалистами в какой-то определенной области информатики и информационно-коммуникационных технологий (ИКТ).

Авторы курса желают вам успеха в освоении непростого, но очень интересного и актуального учебного предмета!

## Навигационные значки

Обратите внимание на символы навигационной полосы, имеющейся в учебниках. Они означают следующее:



- важное утверждение или определение;



- вопросы и задания;



- материал для подготовки к итоговой аттестации;



- дополнительный материал;



- практические работы на компьютере;



- интернет-ресурсы;



- проектные или исследовательские задания;



- практикум<sup>1)</sup>.

<sup>1)</sup> *Семакин И. Г., Хеннер Е. К., Шеина Т. Ю., Шестакова Л. В.* Информатика. Углубленный уровень: практикум для 10–11 классов. М.: БИНОМ. Лаборатория знаний, 2013.



# Глава 1

## ИНФОРМАЦИОННЫЕ СИСТЕМЫ

### 1.1. Основы системного подхода

#### 1.1.1. Что такое система

Понятие системы, так же как и понятие информации, относится к числу фундаментальных научных понятий. Так же как и для информации, для системы нет единственного общепринятого определения. В то же время это понятие часто используется нами в бытовой речи, употребляется в научной терминологии. Вот ряд примеров употребления понятия «система»: система образования, транспортная система, система связи, Солнечная система, нервная система, Периодическая система химических элементов, система числения, операционная система, информационная система.

Обобщая все приведенные выше примеры, дадим следующее предварительное определение.

*Система — это совокупность элементов (материальных или идеальных), обладающая определенной целостностью.*

*Состав системы — это совокупность входящих в нее частей (элементов).* Рассматривая компьютер как систему, можно выделить следующие составляющие его части: процессор, память, устройства ввода, устройства вывода. Но, в свою очередь, процессор тоже является системой, в состав которой входят: арифметико-логическое устройство (АЛУ), устройство управления, регистры, кэш-память. Процессор входит в состав компьютера, поэтому, подчеркивая его собственную системность, процессор следует назвать *подсистемой* компьютера.

---

**Подсистема** — это система, входящая в состав другой, более крупной системы.

---

В свою очередь, АЛУ процессора тоже является системой. В его состав входят сумматоры, полусумматоры и другие элементы. Следовательно, АЛУ — это подсистема процессора. Таким путем

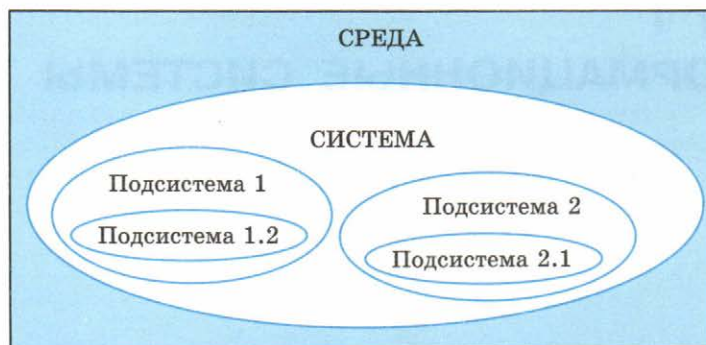


Рис. 1.1. Система — иерархия подсистем

можно продолжать углубляться дальше. Отсюда следует вывод: *всякая система представляет собой иерархию составляющих ее подсистем* (рис. 1.1).

Вопрос о том, что считать системой (подсистемой), а что — простым (неделимым) элементом, субъективен и зависит от решаемой задачи. Например, описывая школу как систему, реализующую функцию обучения и воспитания учащихся, мы будем рассматривать людей (учеников, учителей) в качестве простых элементов. В то же время медицина рассматривает человека как сложную анатомическую систему.

Внешняя система по отношению к данной является *средой ее существования*. Средой существования Земли является Солнечная система; средой существования Солнечной системы — Галактика и т. д. Всякая система *относительно обособлена* от среды своего существования. Это значит, что, с одной стороны, ее можно выделить из среды (рассмотреть отдельно), но в то же время она постоянно связана со своей средой.

Системы бывают естественные и искусственные. Естественные системы — это природные системы. Примеры: системы звезд и планет, растительность и животный мир Земли, молекулы и атомы. Искусственные системы создаются людьми: это заводы, дороги, образование, культура, здравоохранение, компьютеры, самолеты и пр. Некоторые системы объединяют в себе части естественного и искусственного происхождения. Например: гидроэлектростанция, городской парк.

Всякая система обладает *свойством целостности*, поскольку она существует в совокупности своих частей и выполняет свою отдельную функцию в среде своего существования.



**Системный эффект.** Система не является случайным набором частей. Ее состав подчиняется назначению, которое система имеет в природе или в обществе. Искусственные системы человек создает с определенной целью. В связи с этим существует следующее определение системы: *система — это средство достижения цели*. Вот примеры: транспортная система предназначена для перевозки людей и грузов, система здравоохранения — для лечения и укрепления здоровья людей, компьютер — для работы с информацией.

В науке о системах — системологии — сформулирован закон, который называется *принципом эмерджентности* или *законом системного эффекта*. Звучит он так: *целое больше суммы своих частей*. Говоря другими словами: свойства системы не сводятся к совокупности свойств ее частей и не выводятся из них. Слово «эмерджентность» происходит от английского *emergence* — внезапное появление. Например, сложная система организма животного или человека создает системный эффект, который называется жизнью. Выход из строя какой-либо подсистемы организма (кровообращения, пищеварения и др.) приводит к утрате жизни.

**Связи (отношения) в системе.** Части системы всегда связаны между собой, находятся в определенных отношениях. Виды этих связей могут быть самыми разными. В естественных и технических системах они носят материальный характер. Например, планеты Солнечной системы связаны силами гравитации; детали автомобиля — болтами, сваркой, шестеренками; части энергетической системы — линиями электропередач.

Отношения между частями социальных систем бывают различными. Это могут быть отношения подчинения (начальник — подчиненный, министерство — предприятие), отношения вхождения (университет — факультет — кафедра — преподаватель), отношения родственных связей членов семьи. Решающее значение для функционирования таких систем играют *информационные связи* внутри системы, а также с внешней средой. Такие связи реализуются через прямое общение, переписку, технические средства связи, средства массовой информации. Человек является частью многих систем: семьи, класса, производственного коллектива, команды, государства и др. Во всех этих системах он находится в состоянии информационного взаимодействия с другими людьми.

Большое значение информационные связи имеют для деятельности производственных коллективов. Если распоряжение руководителя не доходит до подчиненных или искажается в процессе передачи, то может быть нарушен производственный процесс с самыми серьезными последствиями, вплоть до катастрофы. Во

время боевых действий в армии от работы информационной связи зависят жизни людей. Армия, лишенная связи, не может выполнять своего назначения — эффективно вести военные действия.

Из приведенных примеров следует, что *системный эффект обеспечивается не только наличием нужного состава частей системы, но и существованием необходимых связей между ними.*

**Структурой системы** называется совокупность связей, существующих между частями (элементами) системы. Наглядным примером отображения структуры системы являются схемы электрических цепей. Элементы электрического устройства соединяются между собой двумя способами: последовательно и параллельно. От способа соединения зависит свойство всей цепи. Например, если три проводника, имеющие сопротивления  $R_1$ ,  $R_2$ ,  $R_3$ , соединить последовательно, то общее сопротивление цепи будет равно  $R_1 + R_2 + R_3$ , а если параллельно —  $(R_1 \cdot R_2 \cdot R_3) / (R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3)$ . Первое сопротивление больше второго. Поэтому, например, при пропускании электрического тока в первой цепи будет выделяться больше тепла, чем во второй.

В науке существует много примеров, когда для понимания свойств каких-то систем требовалось понять их структуру. Например, открытие немецким химиком Кекуле структуры молекулы бензола — бензольного кольца — помогло понять химические свойства этого органического вещества. Свойства атома стали лучше понятны физикам после того, как Эрнест Резерфорд открыл планетарную структуру атома, а Нильс Бор сформулировал свои знаменитые постулаты.

Для любой социальной системы, объединенной информационными связями, также характерна определенная структура. Эффективность функционирования системы существенно зависит от ее структуры. Структурная организация любой социальной системы определяется законами, уставами, правилами, инструкциями. Структура государства описана в конституции, структура армии — в уставе.

Обобщая все сказанное о системах, сформулируем следующее определение.

**Система** — целостная, взаимосвязанная совокупность частей, существующая в некоторой среде и обладающая определенным назначением, подчиненная некоторой цели. Система обладает внутренней структурой, относительной обособленностью от окружающей среды, наличием связей со средой.



**Системным подходом** называется научный метод изучения действительности, при котором любой объект исследования рассматривается как система, при этом учитываются его существенные связи с внешней средой.

### Система основных понятий

Что такое система				
<b>Система</b> — целостная, взаимосвязанная совокупность частей, существующая в некоторой среде и обладающая определенным назначением, подчиненная некоторой цели				
Свойства системы				
Целесообразность	Целостность	Структурированность	Иерархическая совокупность подсистем	Связь со средой
<b>Системный эффект:</b> всякая система приобретает новые свойства, не присущие ее составным частям. Целое больше суммы своих частей				
<b>Системный подход</b> — основа научной методологии: рассмотрение всякого объекта изучения в качестве системы, а также учет его существенных связей с внешней средой				

### Вопросы и задания

1. Что такое система? Приведите примеры.
2. Что такое структура системы? Приведите примеры.
3. Приведите примеры систем, имеющих одинаковый состав (одинаковые элементы), но разную структуру.
4. В чем суть системного эффекта? Приведите примеры.
5. Что такое подсистема?
6. Выделите подсистемы в следующих объектах, рассматриваемых в качестве систем:
  - костюм;
  - автомобиль;
  - компьютер;
  - городская телефонная сеть;
  - школа;
  - армия;
  - государство.
7. Удаление каких элементов из систем, названных в задании 6, приведет к потере системного эффекта, т. е. к невозможности выполнения основного назначения систем? Попробуйте выделить существенные и несущественные с позиции системного эффекта элементы этих систем.

## 1.1.2. Модели систем

### Системный анализ

Исследование некоторой реальной системы состоит из двух этапов: анализа и синтеза.

**Анализ системы** — это выделение ее частей с целью прояснения состава системы. В предыдущем разделе мы говорили, что каждая часть системы — это подсистема, и у этой подсистемы есть свои части. Однако невозможно раскладывать систему бесконечно. На чем-то придется остановиться. Какие-то части принять за простые, далее неделимые элементы. Вопрос о том, на чем следует остановить «дробление» системы, зависит от цели исследования. Целью исследования системы является получение ее **модели** — приближенного представления об устройстве и функционировании системы. Полученная модель будет использоваться для прогнозирования поведения системы в некоторых условиях, для управления системой, для диагностики сбоев в функционировании системы и пр.

Однако невозможно понять механизм функционирования системы, выяснив только ее состав. Необходимо знать структуру связей между частями системы. Только в совокупности состава и структуры можно понять состояние и поведение системы. Поэтому анализ системы — это первый этап ее исследования. Второй этап называется **синтезом**. Слово «синтез» означает «соединение». Синтез — это мысленное или реальное соединение частей в единое целое. В результате синтеза создается целостное представление о системе, объясняется механизм системного эффекта.

---

**Системным анализом** называется исследование реальных объектов и явлений с точки зрения системного подхода, состоящее из этапов анализа и синтеза.

---

! Всякое описание системы носит *модельный характер*, т. е. отражает ограниченное число ее свойств. Главный вопрос при построении модели системы: какие ее характеристики являются существенными с точки зрения целей использования будущей модели.

### Модель «черного ящика»

В простейшем случае бывает достаточно иметь представление о взаимодействии системы с внешней средой, не вдаваясь в подробности ее внутреннего устройства. Например, при использовании сложной бытовой техники вам совсем не обязательно знать ее устройство. Достаточно знать, как ею пользоваться, т. е. какие



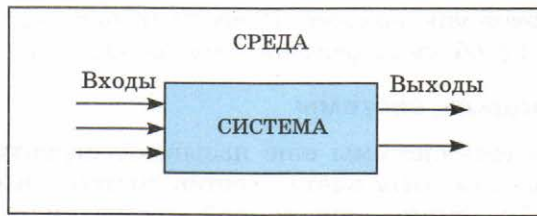


Рис. 1.2. Модель «черного ящика»

управляющие действия можно с ней производить (что на входе) и какие результаты вы будете получать (что на выходе). Все эти сведения содержатся в инструкции для пользователя. Такое описание системы называется моделью «черного ящика» (рис. 1.2).

Вход системы — это воздействие на систему со стороны внешней среды, а выход — это воздействие, оказываемое системой на окружающую среду. В такой модели внутреннее устройство системы скрыто, поэтому ее и называют «черным ящиком».

Например, с точки зрения человека, не связанного с системой высшего образования, университет есть «черный ящик», на входе которого — выпускники школ, а на выходе — дипломированные специалисты.

### Модель состава

Как отмечалось выше, результатом анализа системы является определение ее состава. Если описание системы ограничить перечислением ее частей, то мы получим модель состава. Например, модель состава системы «Университет» представлена на рис. 1.3.

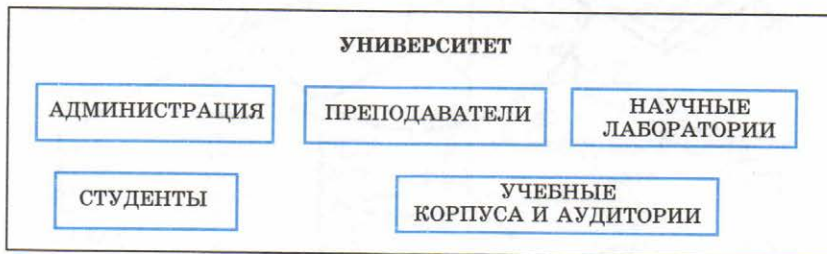


Рис. 1.3. Модель состава университета

Каждая из отмеченных на рисунке составляющих системы «Университет» является подсистемой со своим составом, поэтому для этих подсистем также можно построить свои модели состава. Разумеется, такой модели недостаточно для того, чтобы понять,

как функционирует университет. И все-таки она дает более подробное представление об университете, чем модель «черного ящика».

### Структурная модель системы

Структурную модель системы еще называют **структурной схемой**. На структурной схеме отражается состав системы и ее внутренние связи. Для отображения структурной схемы системы используются **графы**.

Граф состоит из **вершин**, обозначающих элементы системы, и **ребер** — линий, обозначающих связи (отношения) между элементами системы. Знакомая многим схема метрополитена Москвы (рис. 1.4) является примером графа. Вершинами здесь являются станции метро, а ребрами — перегоны для поездов между стан-

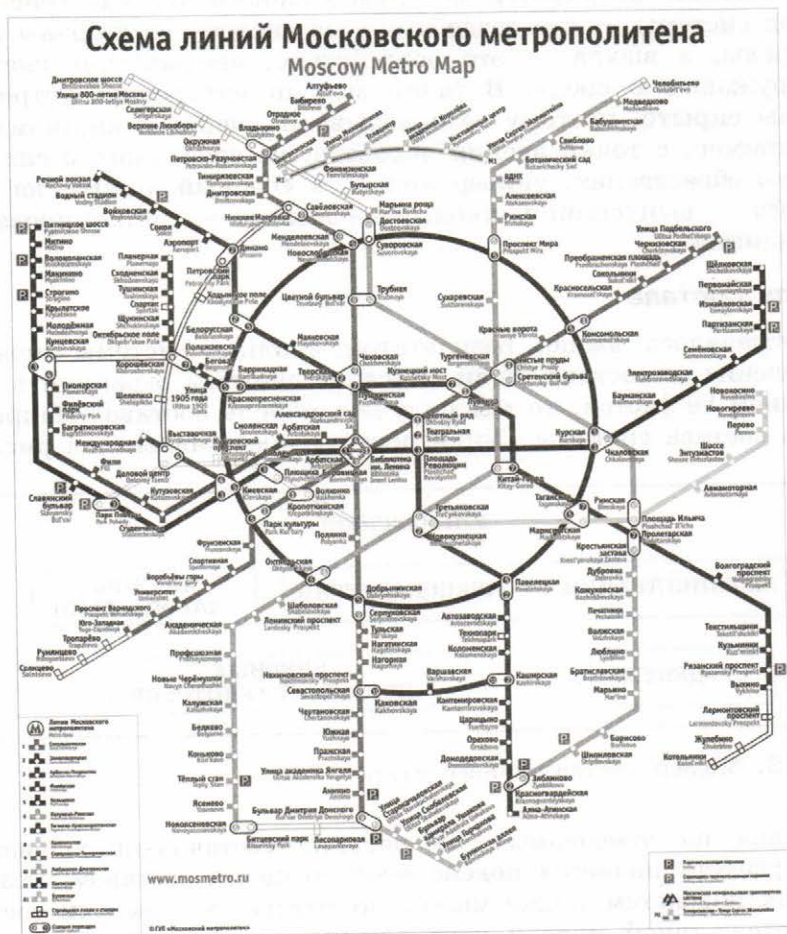


Рис. 1.4. Схема Московского метрополитена



циями и (в некоторых случаях) переходы для пассажиров. Такая схема позволяет пассажиру метро определить маршрут своего перемещения между любыми станциями. Схема метро отражает его радиально-кольцевую структуру.

Еще один пример графа показан на рис. 1.5. Это структурная модель молекулы углеводорода. Вершинами являются атомы водорода и углерода, ребра отображают валентные связи.

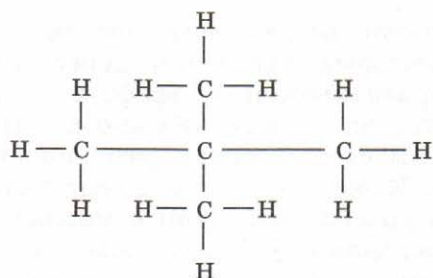


Рис. 1.5. Граф структуры молекулы углеводорода

Связь между двумя станциями метро, соединенными линией движения, является двунаправленной, поскольку поезда могут двигаться в обе стороны. Валентная связь между атомами молекулы также не имеет выделенного направления. Такие графы называются **неориентированными**. Если же связь между двумя элементами системы действует только в одну сторону, то на графе она отображается направленной стрелкой. Такой граф называется **ориентированным**. Направленные линии связи на графе называются **дугами**.

На рисунке 1.6 приведен пример ориентированного графа из области медицины. Известно, что у людей кровь может различаться по группе. Существуют четыре группы крови. При переливании крови от одного человека к другому не все группы совместимы. Граф на рис. 1.6 показывает возможные варианты переливания крови. Группы крови изображаются вершинами гра-

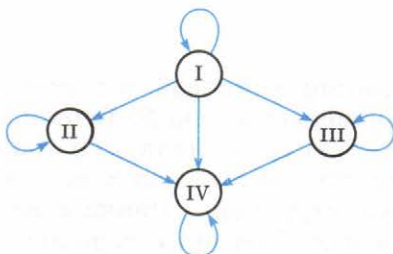


Рис. 1.6. Ориентированный граф системы переливания крови

фа с соответствующими номерами, а стрелки указывают на возможность переливания крови одной группы человеку с другой группой. Например, из этого графа видно, что кровь I группы можно переливать любому человеку, а человек с I группой крови воспринимает только кровь своей группы. Видно также, что человеку с IV группой крови можно переливать любую кровь, но его собственную кровь можно переливать только людям с той же группой.

На практике часто встречаются системы с **иерархической структурой**, граф которых называется **деревом** (рис. 1.7).

Дерево — это ориентированный граф, хотя при его изображении не всегда рисуются стрелки. Обычно вершины дерева располагаются по уровням сверху вниз. Дуги направлены от верхних вершин к нижним. Каждая вершина может быть связана с одной вершиной верхнего уровня (исходной) и множеством вершин нижнего уровня (порожденными). Такая связь называется «один-ко-многим». Единственная вершина самого верхнего уровня называется **корнем** дерева. Вершины самого нижнего уровня, у которых нет порожденных вершин, называются **листьями** дерева. Дерево является **связным графом**. Это значит, что между любыми двумя вершинами имеется хотя бы один **путь**, связывающий их между собой. Поскольку в дереве отсутствуют **петли** — замкнутые траектории связей, путь перемещения по дереву между любыми двумя вершинами всегда является единственным.

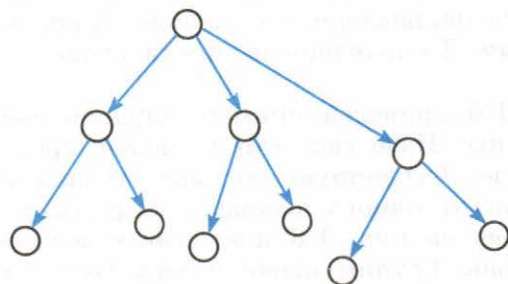


Рис. 1.7. Дерево

Структура организации файловой системы во внешней памяти компьютера является иерархической. Вершинами графа, отображающего файловую структуру, являются папки и файлы. Дуги отражают отношения вхождения одних вершин в другие. Дерево имеет многоуровневую структуру. Папка самого верхнего уровня называется **корнем** дерева. Конечные вершины такого дерева (**листья**) — это файлы и пустые папки.



**Система основных понятий**

Модели систем	
Модель «черного ящика»:	представляет систему на уровне описаний ее входов и выходов
Модель состава:	перечень составляющих систему частей (подсистем, элементов)
Структурная модель:	отражает состав и внутренние связи системы
Граф:	графическое отображение структурной модели; состоит из вершин и линий (ребер, дуг)
Дерево:	ориентированный граф системы с иерархической структурой; связь — «один-ко-многим»; не содержит петель

**Вопросы и задания**

1. Какие существуют типы моделей систем? Чем они различаются?
2. Что такое граф? Из чего он состоит?
3. Какой граф называется неориентированным? Приведите примеры.
4. Какой граф называется ориентированным? Приведите примеры.
5. Нарисуйте в виде графа систему, состоящую из четырех одноклассников, между которыми существуют следующие связи (взаимоотношения): *дружат* — Саша и Маша, Саша и Даша, Маша и Гриша и Саша. Анализируя полученный граф, ответьте на вопрос: с кем Саша может поделиться секретом, не рискуя, что он станет известен кому-то другому?
6. Нарисуйте два варианта графа системы «Компьютер», содержащего следующие вершины: процессор, оперативная память, долговременная память, клавиатура, монитор, принтер. В первом варианте линия связи обозначает отношение «передает информацию», во втором — отношение «управляет».

Практикум. Раздел «Системология»

**1.1.3. Что такое информационная система**

В наше время никого не удивит тем, что для получения справочной информации люди используют компьютеры. Вам, возможно, приходилось посещать торговые центры, где продавцы с помощью компьютера узнают о наличии нужного товара. В некоторых центрах доступ к справочным компьютерам (их называют терминалами) предоставляется посетителям. В железнодорожных

или авиационных кассах кассиры используют компьютер для выяснения вопроса о наличии нужного вам билета. В гостиницах компьютер помогает узнать о наличии свободных номеров, а также забронировать вам номер на нужную дату. Все перечисленные примеры относятся к одной и той же области использования компьютерных технологий, которая называется *информационными системами*.

**Информационная система (ИС)** — совокупность содержащейся в базах данных информации, а также информационных технологий и технических средств, обеспечивающих обработку информации. Всякая ИС имеет определенную сферу применения.

Широкое распространение информационных систем начинается с ЭВМ третьего поколения. Именно тогда на компьютерах стали использоваться в качестве устройств внешней памяти накопители на магнитных дисках. Магнитные диски являются устройствами прямого доступа в отличие от магнитных лент, которые применялись и на машинах первых двух поколений. Благодаря прямому доступу данные на дисках обрабатываются быстрее, чем на лентах. Другой важной особенностью машин третьего поколения стала возможность многопользовательского режима работы на ЭВМ, когда к одной машине одновременно получают доступ множество пользователей через персональные терминалы — устройства для ввода и вывода информации (клавиатура и монитор). Поддержку многопользовательского режима работы обеспечивали операционные системы.

Мощный толчок распространению информационных систем придало развитие сетевых технологий. В рамках одного предприятия, учреждения работают *ИС на базе корпоративной сети*. При этом вся информация может быть сосредоточена на одном узле, а также возможен вариант, когда разные части общедоступных данных хранятся на разных узлах сети.

Наиболее крупные информационные системы работают *на базе глобальных компьютерных сетей*. Примером является «Полет-Сирена» — информационная система воздушного транспорта, главная страница которой показана на рисунке 1.8. Терминалом доступа к этой системе может служить любой компьютер, подключенный к Интернету. Однако существует множество «глобальных» ИС не общего, а ограниченного доступа и масштаба — это *корпоративные системы*. Они могут объединять локальные сети предприятий одного ведомства и способствовать их общему эффективному управлению в рамках региона, министерства и пр.



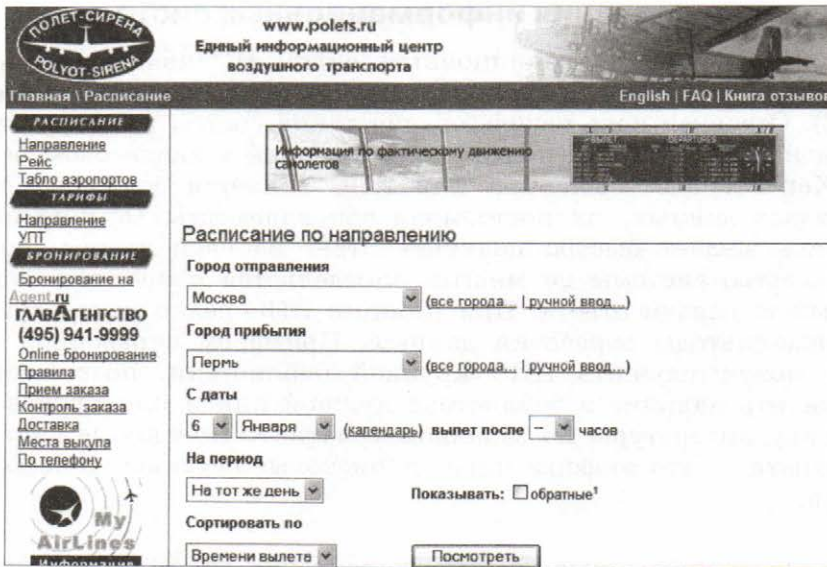


Рис. 1.8. Информационная система «Полет-Сирена»

Основой всякой информационной системы является *база данных*. О базах данных на уровне первоначального представления рассказывалось в курсе информатики 8 класса. Повторим приведенное там определение базы данных.

**База данных** — определенным образом организованная совокупность данных, относящихся к некоторой предметной области, предназначенная для хранения во внешней памяти компьютера и постоянного применения.

База данных (БД) — это всего лишь сохраненная информация. А информационная система должна обеспечивать использование данных из этого хранилища заинтересованными людьми — пользователями. Обслуживание запросов пользователя к БД на поиск данных, их представление в удобном виде, обработку и анализ выполняют программы, которые называются **приложениями баз данных** (клиентскими приложениями).

Пользователь информационной системы не обязан быть специалистом в области вычислительной техники, поэтому клиентские приложения ИС должны обладать простым, наглядным, интуитивно понятным **интерфейсом**, позволяющим пользователю реализовывать все возможности ИС и предотвращающим недопустимые действия с его стороны.

## Области применения информационных систем

Наиболее старым и традиционным видом ИС являются *информационно-справочные*, или **информационно-поисковые системы (ИПС)**. Основная цель использования таких систем — оперативное получение ответов на запросы пользователей в диалоговом режиме. Характерным свойством для ИПС является большой объем хранимых данных, их постоянная обновляемость. Обычно пользователь желает быстро получить ответ на свой запрос, поэтому качество системы во многом определяется скоростью поиска данных и выдачи ответа. При работе с ИПС редко используются сложные методы обработки данных. Примером справочной системы может служить ИПС крупной библиотеки, позволяющая определить наличие в библиотеке нужной книги или произвести подборку литературы по заданной тематике. Поисковые системы Интернета — это информационно-поисковые системы сетевых ресурсов.

**Автоматизированные системы управления (АСУ)** — это многофункциональная информационная система, используемая в управлении предприятием. Руководству предприятием постоянно приходится принимать управленческие решения. Правильность этих решений зависит от полноты и оперативности получения руководителем нужной информации: о финансовых и материальных ресурсах, кадровом составе, транспортных средствах и многом другом. Вся эта информация на большинстве современных предприятий хранится в базе данных и предоставляется по запросам сотрудников автоматизированной информационной системой. При этом компьютер может выполнять достаточно сложную обработку данных на основании заложенных в него математических моделей. Это могут быть технологические или экономические расчеты, т. е. компьютер берет на себя определенные инженерные функции. Крупные АСУ обеспечивают управление предприятиями, энергосистемами и даже целыми отраслями производства.

Еще одним направлением применения информационных систем являются **компьютерные системы обучения**. Простейший вариант такой системы — обучающая программа на ПК, с которой пользователь работает в индивидуальном режиме. В базу данных обучающей системы заложена учебная информация. Существует множество обучающих программ практически по всем школьным предметам и ряду курсов профессионального обучения. Более сложными являются обучающие системы, использующие возможности компьютерных сетей. В локальной сети можно



организовывать обучение с элементами взаимодействия учащихся, используя соревновательную форму или форму деловой игры.

Наиболее сложными и масштабными обучающими системами являются **системы дистанционного обучения**, работающие в глобальных сетях. Дистанционное образование называют образованием XXI века. Уже существуют дистанционные отделения при многих ведущих вузах страны, формируется международная система дистанционного образования. Такие системы открывают доступ к качественному образованию для всех людей, независимо от их места жительства, возраста, возможных физических ограничений. Высокоскоростные системы связи в сочетании с технологией мультимедиа позволяют организовывать обучение в режиме реального времени (on-line), проводить дистанционные лекции, семинары, конференции, принимать зачеты и экзамены.

Широкое распространение в последнее время получили **геоинформационные системы (ГИС)**. ГИС базируются на картах территорий. Большое количество нужной людям информации оказывается привязанным к географическим картам. Это сведения о расположении в городе различных организаций, магазинов, культурных учреждений, больниц и пр. Информация, необходимая для работы геологов, строителей, работников транспорта, для туристов связана с картами. Например, существует электронный *государственный земельный кадастр Российской Федерации* — информационная система, содержащая реестр сведений о земле, находящейся в хозяйственном обороте (сельскохозяйственном, промышленном и т. д.). Очевидно, что без привязки к карте такой кадастр создать было невозможно. Другой, знакомый многим из вас пример ГИС — информационная система *ГИСметео*, позволяющая получать справки о погоде по всему миру (рис. 1.9).

**Экспертные системы** — системы, основанные на моделях знаний в предметных областях. Информационной основой экспертной системы является база знаний. **База знаний** — это формализованная система знаний высококвалифицированного специалиста в определенной предметной области. Экспертная система используется для консультаций пользователя, помощи в принятии сложных решений, решения плохо формализуемых задач. Примерами проблем, которые решаются с помощью экспертных систем, являются: установление диагноза больного; определение причин неисправности сложной техники (например, самолетов); рекомендации по ликвидации неисправности; определение вероятных последствий принятого управляющего решения и т. д. Экспертные системы часто включают в состав АСУ в качестве подсистем.

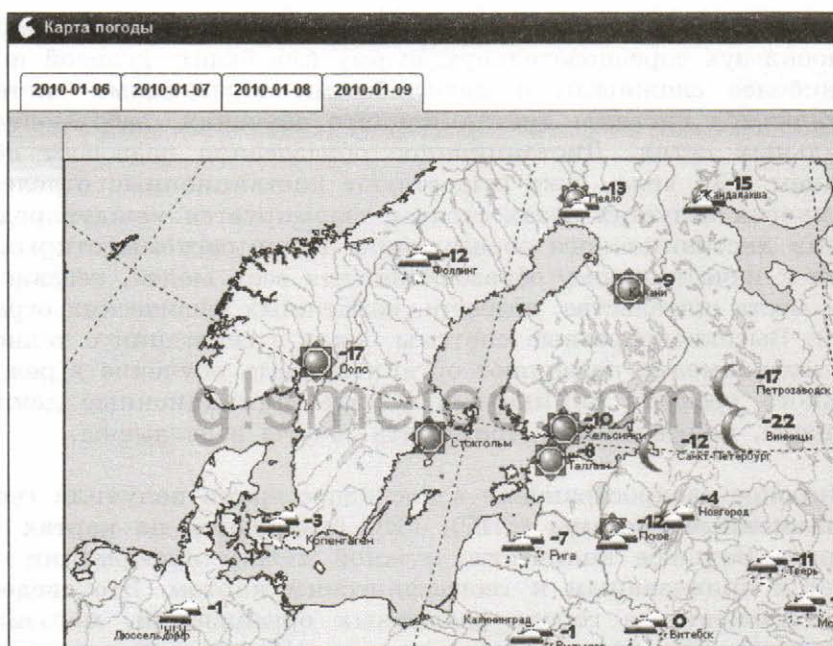


Рис. 1.9. Информационная система ГИСметеo. Карта погоды



### О профессиях: профессии, связанные с созданием информационных систем

В реестре специальностей высшего профессионального образования имеется специальность «**специалист по информационным системам**». Это одна из наиболее востребованных сегодня инженерных профессий. В круг служебных обязанностей специалиста по ИС входит:

- разработка концепции информационной системы;
- проектирование архитектуры ИС;
- проектирование модели данных, на основе которой будет создаваться БД;
- построение логической схемы базы данных;
- реализация базы данных в определенной инструментальной среде;
- разработка клиентских приложений;
- разработка интерфейса пользователя ИС.

Есть еще одна очень важная функция специалиста по ИС: администрирование БД. **Администратор базы данных** отвечает за содержание БД, ее целостность, безопасность, поэтому какие-либо изменения в содержании или структуре базы данных производятся либо самим администратором,



либо под его контролем. Обычно базой данных пользуется большое количество людей. От ее работоспособности часто зависит функционирование всей организации: банка, фирмы, министерства и др. Отсюда понятно, что ответственность работы администратора БД очень велика.

В перечне профессиональных стандартов в области информационных технологий отдельно присутствуют профессии «системный аналитик» и «системный архитектор». Основная функция **системного аналитика** — взаимодействие с заказчиком на разработку ИС с целью осуществления квалифицированного системного анализа предметной области (производства, исследовательской или образовательной деятельности и пр.), получения адекватной инфологической или математической модели. Задача **системного архитектора** — разработка архитектуры будущей системы, т. е. состава и структуры ее аппаратно-программного комплекса, требований к эксплуатационным свойствам, мониторинг удовлетворения этим требованиям созданной системы.

### Система основных понятий



Информационные системы				
<p><b>Информационная система (ИС)</b> — совокупность содержащейся в базах данных информации, а также информационных технологий и технических средств, обеспечивающих обработку информации. Всякая ИС имеет определенную сферу применения</p>				
Техническая база		Состав ИС		
На одном компьютере	На базе компьютерной сети (локальной или глобальной)	База данных	Клиентские приложения	Пользовательский интерфейс
Области применения ИС				
Информационно-справочная (информационно-поисковая) система (ИПС)	АСУ (принятие управленческих решений)	Геоинформационные системы (ГИС)	Обучение, дистанционное образование	Экспертные системы
Профессии, связанные с ИС				
Специалист по информационным системам	Администратор баз данных	Системный аналитик	Системный архитектор	

## Вопросы и задания

1. Какие основные признаки современной информационной системы можно выделить?
2. К каким типам ИС относятся следующие системы:
  - система прогноза погоды для различных регионов страны;
  - система автоматического тестирования для подготовки к сдаче ЕГЭ;
  - система диспетчерской службы крупного аэропорта;
  - система диагностики в кардиологической клинике?
3. Придумайте возможные области использования информационных систем в деятельности школы. К какому типу ИС относится каждая из придуманных вами систем?

### 1.1.4. Инфологическая модель предметной области

Дальнейший разговор об информационных системах мы будем вести в процессе построения небольшой информационно-справочной системы. Всякая информационная система базируется на информационной модели конкретной **предметной области** — области деятельности людей, где эта система будет использоваться. В качестве предметной области выберем учебный процесс в школе, поскольку эта область вам хорошо знакома. Представим себя в роли **системных аналитиков** и займемся построением структурной модели для будущей информационной системы.

Учебный процесс — многокомпонентная и достаточно сложная система. Чтобы описать ее в полном объеме, нам потребовалось бы много времени и места в учебнике. Упростим себе задачу. Для этого сформулируем ограниченный круг целей, на достижение которых будет направлена разработка информационной системы. Зададимся вопросом: кто будет пользователем ИС и какие возможности система должна предоставлять пользователям?

Назовем нашу информационную систему «Классный журнал». Только это будет не бумажный, а электронный журнал. Его пользователями будут: учителя, преподающие в классе, ученики этого класса и их родители. Учителю система должна предоставлять возможность выставлять оценки ученикам. Кроме того, в ИС должны храниться адреса и домашние телефоны учеников. Родители должны иметь возможность получать информацию об успеваемости своих детей по каждому предмету, по датам получения оценок, а также узнавать фамилии, имена и отчества учителей-предметников. Информацию о своих оценках могут получать и сами ученики. Такая система может быть подсистемой



ИПС школы, содержащей информацию об успеваемости всех ее учащихся во всех классах.

Проведем системный анализ. Предметной областью является учебный процесс в отдельном классе. Первый этап — **анализ** предметной области — предполагает разделение исследуемой системы на части. Таких частей (сущностей) будет три:

- 1) ученики одного класса;
- 2) изучаемые предметы;
- 3) оценки, полученные учениками по предметам.

С позиции поставленной цели этого достаточно. Каждый объект, входящий в выделенные части, обладает определенными свойствами, которые называются **атрибутами**. Какие именно атрибуты следует включить в модель, зависит от того, какую информацию мы хотим в дальнейшем из нее извлекать. Исходя из описанных выше целей использования ИС (учителями, родителями, учениками), определим следующие наборы атрибутов для наших объектов.

**Ученик:** фамилия; имя; пол; адрес; телефон.

**Изучаемый предмет:** название предмета; фамилия, имя и отчество учителя.

**Оценка:** фамилия и имя ученика; предмет; дата получения оценки; выставленный балл.

Данное выше описание является моделью состава системы. Второй этап системного анализа — **синтез**. Надо установить и описать связи между частями системы. Очевидно наличие следующих связей: ученики получают оценки, каждая оценка относится к определенному предмету. Эти связи отображены на структурной схеме (рис. 1.10).

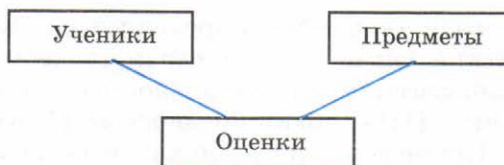


Рис. 1.10. Структурная схема системы

Связь между учениками и предметами тоже существует, но она в этой модели реализуется через оценки. Такой граф называют информационно-логической моделью, или короче — **инфо-логической моделью** предметной области. Для отображения инфологических моделей чаще всего используются диаграммы типа

«сущность–связь» (ER-диаграммы). Для того чтобы придать нашей схеме на рис. 1.10 вид ER-диаграммы, к ней надо добавить ромбы с указанием имени связи и овалы-выноски с указанием атрибутов объектов. Инфологическая модель учебного процесса представлена на рис. 1.11.



Рис. 1.11. Инфологическая модель (ER-диаграмма)

Имя, данное связи (в ромбе), определяет ее смысл. Есть еще одна характеристика связи — тип связи (в теории баз данных она называется показателем кардинальности). Типы связей бывают: «один-к-одному» (1:1), «один-ко-многим» (1:M) и «многие-ко-многим» (M:M). Например, связь между вершинами УЧЕНИКИ и ОЦЕНКИ — «один-ко-многим», потому что один ученик может получить множество оценок, но одна конкретная оценка, записанная в журнале, относится к одному ученику. Такой же тип имеет связь между предметами и оценками: по одному предмету в журнале выставлено множество оценок, но одна оценка относится к одному предмету.

В ER-диаграмме в выносках, связанных с вершинами, указываются атрибуты элементов подсистемы, которую эта вершина обозначает.



## Система основных понятий

Инфологическая модель	
<b>Предметная область</b> — область деятельности людей, для которой создается модель	
<b>Инфологическая модель</b> — результат системного анализа предметной области	
<b>ER-диаграмма</b> (диаграмма типа «сущность–связь»): отражение инфологической модели в виде графа	
сущности	связи
Части (подсистемы) системы — вершины графа. Имеют имена и атрибуты	Отражают отношения между вершинами. Имеют имена и типы: 1:1, 1:M, M:M

### Вопросы и задания

1. Что такое инфологическая модель предметной области?
2. Из каких элементов составляется инфологическая модель?
3. Какие типы связей используются в инфологической модели?
4. В модели учебного процесса для всей школы какой тип связи будет установлен между подсистемами ПРЕДМЕТЫ и УЧИТЕЛЯ?
5. В модели учебного процесса для всей школы какой тип связи будет установлен между подсистемами КЛАССЫ и КЛАССНЫЕ РУКОВОДИТЕЛИ?
6. Какие изменения следует внести в модель на рис. 1.11, если предметной областью для нее становится успеваемость учеников во всей школе? Какие новые подсистемы войдут в модель, какие связи будут установлены?

Практикум. Раздел «Системология»

## 1.2. Реляционные базы данных

### 1.2.1. Реляционные базы данных и СУБД

На основе инфологической модели предметной области создается модель данных, которая затем будет реализована в компьютерной базе данных. Определение базы данных было дано в параграфе 1.1.3. С основными понятиями баз данных вы знакомы из курса информатики 8 класса. Повторим некоторые понятия, важные для разработки нашего проекта.

**Классификация баз данных.** По содержанию хранимой информации базы данных можно разделить на фактографические и документальные. **Фактографические БД** содержат данные, представленные в краткой форме, в строго фиксированных форматах. Такие БД являются аналогами бумажных картотек, например библиотечного каталога или каталога видеотеки. Для **документальных БД** аналогом являются архивы документов, например, архив судебных дел, архив исторических документов и пр. Проектируемая нами база данных со сведениями об успеваемости учеников будет фактографической.

Базы данных классифицируются также по структуре модели данных. Известны три разновидности структур данных: иерархическая, сетевая и табличная. Соответственно по признаку структуры базы данных делятся на **иерархические, сетевые и реляционные** (табличные). Наиболее распространенным типом являются реляционные БД. В теории баз данных доказано, что любую структуру данных можно свести к табличной форме. Базу данных по успеваемости учеников класса будем делать реляционной.

**Организация данных в реляционной БД.** Основной информационной единицей реляционной БД является таблица. Следовательно, реляционные БД используют табличную (реляционную) модель данных. База данных может состоять из одной таблицы — однотабличная БД, или из множества взаимосвязанных таблиц — многотабличная БД.

Структурными составляющими таблицы являются **записи и поля**.

	поле 1	поле 2	поле 3	...
запись 1				
запись 2				
запись 3				
...				

Каждая запись содержит информацию об отдельном экземпляре объектов, составляющих систему (предметную область): одной книге в библиотеке, одном сотруднике предприятия и т. п. А каждое поле — это определенная характеристика (свойство, атрибут) объекта: название книги, автор книги; фамилия сотрудника, год рождения и т. п. Поля таблицы должны иметь имена, отличные друг от друга. *В таблице не должно быть совпадающих записей.*



Для каждой таблицы реляционной БД должен быть определен **первичный ключ** — поле или совокупность полей, однозначно определяющих запись. Иначе говоря, значение первичного ключа не должно повторяться в разных записях. Например, в библиотечной базе данных таким ключом может быть выбран инвентарный номер книги, который не может совпадать у разных книг.

Для строчного представления структуры таблицы применяется следующая форма:

**ИМЯ\_ТАБЛИЦЫ** (ИМЯ\_ПОЛЯ\_1, ИМЯ\_ПОЛЯ\_2, ..., ИМЯ\_ПОЛЯ\_N)

Подчеркиваются поля, составляющие первичный ключ.

Название «реляционная база данных» связано с английским словом *relation*, что в переводе означает «отношение». Отношение в БД — это взаимосвязь полей — элементов данных. Наглядным способом отображения отношения является таблица. Поэтому **ИМЯ\_ТАБЛИЦЫ** — это имя отношения. Примеры отношений:

**БИБЛИОТЕКА** (ИНВ\_НОМЕР, АВТОР, НАЗВАНИЕ, ГОД\_ИЗД, ИЗДАТЕЛЬСТВО).

**БОЛЬНИЦА** (НОМЕР\_ПАЛАТЫ, НОМЕР\_МЕСТА, ПАЦИЕНТ, ДАТА\_ПОСТУПЛЕНИЯ, ДИАГНОЗ, ПЕРВИЧНЫЙ)

Каждое поле таблицы имеет определенный **тип**. С типом связаны два свойства поля: множество значений, которые оно может принимать, и множество операций, которые над ним можно выполнять. Существуют четыре основных типа для полей БД: **символьный**, **числовой**, **логический** и **дата**. Для полей таблиц «Библиотека» и «Больница» могут быть установлены следующие типы:

символьный тип: АВТОР, НАЗВАНИЕ, ИЗДАТЕЛЬСТВО, ПАЦИЕНТ, ДИАГНОЗ;

числовой тип: ИНВ\_НОМЕР, ГОД\_ИЗД, НОМЕР\_ПАЛАТЫ, НОМЕР\_МЕСТА

дата: ДАТА\_ПОСТУПЛЕНИЯ;

логический: ПЕРВИЧНЫЙ.

В последнем случае поле **ПЕРВИЧНЫЙ** обозначает следующий факт: больной поступил в больницу с данным диагнозом впервые или повторно. Те записи, где значение этого поля равно **TRUE** (**ИСТИНА**), относятся к первичным больным, значение **FALSE** (**ЛОЖЬ**) отмечает повторного больного. Таким образом, поле логического типа может принимать только два значения.

В таблице **БОЛЬНИЦА** используется **составной ключ** — он состоит из двух полей: НОМЕР\_ПАЛАТЫ и НОМЕР\_МЕСТА. Только их сочетание не повторяется в разных записях (ведь фамилии пациентов могут совпадать).



**СУБД.** Программное обеспечение, используемое для создания, обновления, администрирования баз данных, а также выполнения запросов к БД, называется **системой управления базами данных (СУБД)**. В зависимости от структуры создаваемых баз различаются иерархические, сетевые и реляционные СУБД. Наибольшее распространение на персональных компьютерах получили реляционные СУБД. Основные действия, которые пользователь может выполнять с помощью СУБД:

- создание структуры БД;
- заполнение БД информацией;
- изменение (редактирование) структуры и содержания БД;
- поиск информации в БД;
- сортировка данных;
- защита БД;
- проверка целостности БД.

### Система основных понятий

Реляционные БД и СУБД	
Классификация баз данных	
<b>Фактографические</b>	Содержат данные, представленные в краткой форме, в строго фиксированных форматах
<b>Документальные</b>	Содержат документы в различных форматах
<b>Иерархические</b>	Имеют иерархическую организацию данных (граф — дерево)
<b>Сетевые</b>	Имеют сетевую организацию данных (граф — сеть)
<b>Реляционные</b>	Имеют табличную организацию данных
Основные понятия реляционных БД	
<b>Таблица (отношение)</b>	Основная информационная единица хранения данных. Содержит множество неповторяющихся строк (записей), отражающих свойства (атрибуты) различных объектов
<b>Запись</b>	Отдельная строка в таблице; описывает отдельный объект
<b>Поле</b>	Столбец таблицы, свойство (характеристика, атрибут) объекта
<b>Первичный ключ</b>	Поле или совокупность полей, однозначно идентифицирующая каждую запись в таблице
<b>Тип поля</b>	Определяет допустимые значения и действия над полем (числовой, символьный, логический, дата, время)
<b>Система управления базами данных (СУБД)</b> — программное обеспечение для работы с базами данных	



## Вопросы и задания

1. Что такое база данных?
2. Как классифицируются базы данных по их содержанию?
3. Как классифицируются базы данных по способу организации данных?
4. Перечислите и определите основные понятия реляционных БД: таблица, запись, поле, первичный ключ, тип поля.
5. Что такое СУБД?



### 1.2.2. Проектирование реляционной модели данных

**Многотабличная модель данных.** Перейдем к следующему этапу разработки информационной системы «Классный журнал». Теперь от инфологической модели системы нам нужно перейти к реляционной модели данных. Для этого требуется:

- описать отношения, определяющие структуры таблиц базы данных;
- определить основные ключи;
- реализовать связи между таблицами.

Если в качестве имен отношений (таблиц) принять имена сущностей ER-диаграммы на рис. 1.11, а в качестве полей — атрибуты сущностей, то получим:

УЧЕНИКИ (ФАМИЛИЯ, ИМЯ, ПОЛ, АДРЕС, ТЕЛЕФОН)

ПРЕДМЕТЫ (НАЗВАНИЕ\_ПРЕДМЕТА, УЧИТЕЛЬ)

ОЦЕНКИ (УЧЕНИК, ПРЕДМЕТ, ДАТА, ОЦЕНКА)

Определим первичные ключи таблиц. В таблице УЧЕНИКИ придется выбрать составной ключ ФАМИЛИЯ + ИМЯ, поскольку только их сочетание не повторяется в каждой записи. Но представьте себе, что в классе есть два полных тезки, например два Королёвых Андрея. Как их различать? Вводить поле ОТЧЕСТВО? Но, во-первых, в школьных журналах это делать не принято, во-вторых, нет гарантии, что и отчество не совпадет. Решение этой проблемы — ввести специальные, уникальные идентификаторы учеников (коды учеников). Удобнее всего сделать их целыми числами. Такой номер присваивается ученику один раз — при занесении записи в базу — и больше никогда не изменяется.

В таблице ПРЕДМЕТЫ выбор ключа не вызывает проблем. Очевидно, что это название предмета.

В таблице ОЦЕНКИ атрибут УЧЕНИК должен идентифицировать конкретного ученика. В качестве такого идентификатора в таблице УЧЕНИКИ теперь есть поле НОМЕР\_УЧЕНИКА. Его

и будем использовать вместо атрибута УЧЕНИК. А вместо атрибута ПРЕДМЕТ будем использовать поле НАЗВАНИЕ\_ПРЕДМЕТА. В таблице ОЦЕНКИ основной ключ будет составным: НОМЕР\_УЧЕНИКА + НАЗВАНИЕ\_ПРЕДМЕТА + ДАТА. Делается допущение, что по одному предмету в один день в журнал ученику может быть выставлено не более одной оценки.

В итоге получены следующие три отношения, составляющие модель данных:

УЧЕНИКИ (НОМЕР\_УЧЕНИКА, ФАМИЛИЯ, ИМЯ, ПОЛ, АДРЕС, ТЕЛЕФОН)

ПРЕДМЕТЫ (НАЗВАНИЕ\_ПРЕДМЕТА, УЧИТЕЛЬ)

ОЦЕНКИ (НОМЕР\_УЧЕНИКА, НАЗВАНИЕ\_ПРЕДМЕТА, ДАТА, ОЦЕНКА)

Теперь о связях между таблицами. Они уже фактически определены нами при выборе ключевых полей. В реляционных базах действуют следующие правила, определяющие типы связей между таблицами.

1. Если две таблицы имеют одинаковые первичные ключи, то между ними действует связь «один-к-одному».
2. Если первичной ключ первой таблицы является частью составного ключа второй таблицы или неключевым полем второй таблицы, то между первой и второй таблицами действует связь «один-ко-многим».

В реляционных базах данных связь «многие-ко-многим» непосредственно между двумя таблицами не реализуется. Для ее реализации приходится вводить дополнительную связывающую таблицу. Для нашего проекта такой проблемы нет.

На этом этап проектирования базы данных завершен, получена реляционная модель данных. Ее еще называют **схемой данных**. Следующий этап — создание базы данных на компьютере средствами СУБД.

**О нормализации данных.** В теории реляционных баз данных существует понятие **нормальной формы** организации данных. Полученная нами модель данных обладает свойствами **третьей нормальной формы**. Отношение (таблица) соответствует третьей нормальной форме, если:

- все поля таблицы — атомарные, т. е. далее не делимые;
- каждое неключевое поле полностью функционально зависит от первичного ключа;
- между полями таблицы нет транзитивных зависимостей.



Чтобы объяснить смысл этих условий, покажем на примере нашей задачи, каким образом можно было осуществлять нормализацию данных. Пусть первоначально мы решили сделать базу данных однотабличной, т. е. собрать все поля в одной таблице. Назовем ее **УСПЕВАЕМОСТЬ**:

**УСПЕВАЕМОСТЬ** (НОМЕР УЧЕНИКА, НАЗВАНИЕ ПРЕДМЕТА,  
ДАТА, ОЦЕНКА, УЧИТЕЛЬ, ФАМИЛИЯ, ИМЯ, ПОЛ,  
АДРЕС, ТЕЛЕФОН)

Каждая запись такой таблицы относится к одной оценке, полученной конкретным учеником по данному предмету в конкретный день учебного года. В базе данных будет столько записей, сколько оценок стоит в журнале. Эта модель удовлетворяет требованиям **первой нормальной формы**, согласно которой все поля должны быть атомарными. Атомарное поле далее не делится. Например, объединение в одно поле ФИО фамилии, имени и отчества человека нарушает принцип атомарности. Понятие атомарности относительно. Например, если в приложениях не потребуется отдельной обработки названия улицы, номера дома и квартиры, то адрес можно не разбивать на составляющие и считать его атомарным.

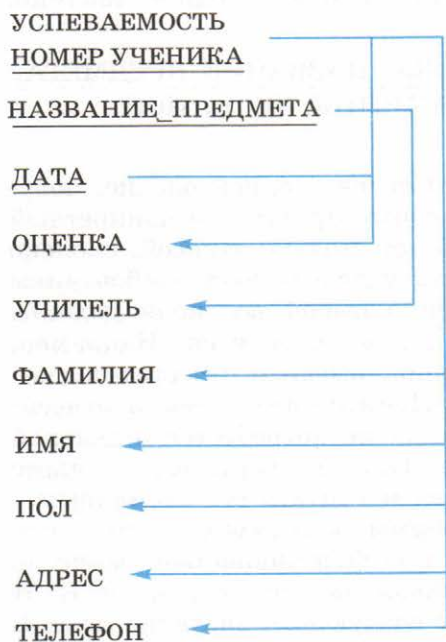
Требование **второй нормальной формы** заключается в том, чтобы каждое неключевое поле полностью функционально зависело от первичного ключа. Функциональная зависимость поля *B* от поля *A* (или группы полей) означает следующее: значение поля *B* однозначно определяется значением поля *A* (или группы полей).

На рисунке 1.12 стрелками показаны все функциональные зависимости полей в таблице **УСПЕВАЕМОСТЬ**. Полностью от составного первичного ключа зависит только поле **ОЦЕНКА**. Поле **УЧИТЕЛЬ** функционально зависит только от поля **НАЗВАНИЕ ПРЕДМЕТА**. Поля **ФАМИЛИЯ**, **ИМЯ**, **ПОЛ**, **АДРЕС**, **ТЕЛЕФОН** функционально зависят от поля **НОМЕР УЧЕНИКА**. Разбив таблицу **УСПЕВАЕМОСТЬ** на три таблицы: **УЧЕНИКИ**, **ПРЕДМЕТЫ**, **ОЦЕНКИ**, получим в каждой из них полную функциональную зависимость от основных ключей, что показано на рис. 1.13.

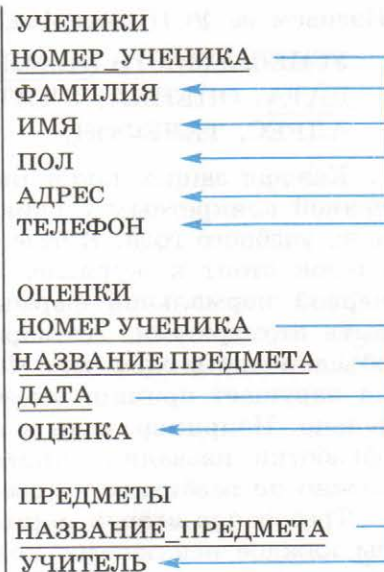
**Транзитивной** называется зависимость между двумя полями *A* и *B* через третье поле *C*:  $A \rightarrow C \rightarrow B$ . Таких зависимостей ни в одной из трех построенных таблиц нет. Значит, полученные таблицы удовлетворяют требованиям **третьей нормальной формы**.

*Основным недостатком ненормализованной модели данных является избыточность информации в БД.* Если бы все данные хранились в таблице **УСПЕВАЕМОСТЬ**, то при занесении разных оценок для одного и того же ученика повторялись бы поля **ФАМИЛИЯ**, **ИМЯ**, **ПОЛ**, **АДРЕС**, **ТЕЛЕФОН**. Во всех записях, относящихся к одному предмету, повторялась бы фами-

ля учителя, преподающего этот предмет. В нормализованной базе эти данные будут храниться всего по одному разу, каждое в своей таблице.



**Рис. 1.12.** Функциональные зависимости в таблице УСПЕВАЕМОСТЬ



**Рис. 1.13.** Функциональные зависимости в трех нормализованных таблицах

## Система основных понятий

Реляционная модель данных		
<b>Этапы построения реляционной модели</b>		
1) описание структуры таблиц	2) определение первичных ключей таблиц	3) реализация связей между таблицами
<b>Типы связей в реляционной модели</b>		
1:1 — две таблицы имеют одинаковые первичные ключи	1:M — первичный ключ первой таблицы является частью составного ключа второй таблицы или неключевым полем второй таблицы	



Нормализация данных		
<b>Первая нормальная форма:</b> все поля таблицы — атомарные	<b>Вторая нормальная форма:</b> каждое неключевое поле полностью функционально зависит от основного ключа	<b>Третья нормальная форма:</b> между полями таблицы нет транзитивных зависимостей
Модель данных в третьей нормальной форме исключает избыточность		

## Вопросы и задания

1. Что такое первичный ключ таблицы?
2. Каким образом между таблицами устанавливаются связи типа 1:1 и 1:M?
3. При проектировании БД были определены следующие отношения:  
 МАГАЗИН (НОМЕР\_МАГ, ТИП, АДРЕС, ДИРЕКТОР, ТЕЛЕФОН)  
 ОТДЕЛ (НАЗВАНИЕ\_ОТД, ЗАВЕДУЮЩИЙ, ТЕЛЕФОН)  
 ПРОДАВЕЦ (ТАБЕЛЬНЫЙ\_НОМ, ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО, КАТЕГОРИЯ)  
 Являются ли эти отношения связанными? Добавьте все, что необходимо для их связи; нарисуйте граф соответствующей инфологической модели.
4. С какой целью производится нормализация модели данных?
5. В чем заключаются требования первой, второй и третьей нормальных форм?
6. Нормализуйте следующие отношения путем декомпозиции (приведите к третьей нормальной форме):  
 ПОЛИКЛИНИКА (ФАМИЛИЯ\_ПАЦИЕНТА, ДАТА\_ПОСЕЩЕНИЯ, ДАТА\_РОЖДЕНИЯ, УЧАСТОК, ВРАЧ, ДИАГНОЗ)  
 РЕЙС\_САМОЛЕТА (НОМЕР\_РЕЙСА, ДАТА\_ВЫЛЕТА, ВРЕМЯ\_ВЫЛЕТА, ПУНКТ\_НАЗНАЧЕНИЯ, ТИП\_САМОЛЕТА, ЧИСЛО\_МЕСТ, ВРЕМЯ\_ПОЛЕТА, КОМАНДИР\_КОРАБЛЯ, КЛАСС\_ПИЛОТА)

### 1.2.3. Создание базы данных

От теоретического проектирования структуры базы данных нам предстоит перейти к ее практическому созданию на компьютере. База данных создается средствами СУБД. Создание происходит в три этапа.

1. Открытие файла, в котором будет храниться БД.
2. Построение структуры таблиц и установка связей.
3. Ввод данных в таблицы.

Обычно открытие файла происходит в среде СУБД через последовательность команд вида **Файл** → **Создать БД** → **Новая БД**.

На втором этапе средствами СУБД создаются таблицы, в каждой из которых определяются имена полей, их типы и форматы. Ниже представлены структуры таблиц УЧЕНИКИ, ПРЕДМЕТЫ и ОЦЕНКИ (табл. 1.1–1.3).

Таблица 1.1

**УЧЕНИКИ**

Имя поля	Тип поля	Длина (формат)
<u>НОМЕР_УЧ</u>	числовой	целый
ФАМИЛИЯ	текстовый	30
ИМЯ	текстовый	15
ПОЛ	текстовый	1
АДРЕС	текстовый	40
ТЕЛЕФОН	текстовый	15

Таблица 1.2

**ПРЕДМЕТЫ**

Имя поля	Тип поля	Длина (формат)
<u>НАЗВ_ПРЕДМЕТА</u>	текстовый	30
УЧИТЕЛЬ	текстовый	40

Таблица 1.3

**ОЦЕНКИ**

Имя поля	Тип поля	Длина (формат)
<u>НОМЕР_УЧ</u>	числовой	целый
<u>НАЗВ_ПРЕДМЕТА</u>	текстовый	30
<u>ДАТА</u>	дата	
ОЦЕНКА	числовой	целый

Затем средствами СУБД устанавливаются связи. Между таблицами УЧЕНИКИ и ОЦЕНКИ связь устанавливается через общее поле НОМЕР\_УЧ. А связь между таблицами ПРЕДМЕТЫ и ОЦЕНКИ устанавливается через поле НАЗВ\_ПРЕДМЕТА. И та, и другая связь будет иметь тип «один-ко-многим». На рисунке 1.14 показана окончательно построенная модель данных в форме схемы<sup>1)</sup>. Здесь в обозначении связи «один-ко-многим» вместо буквы М использован символ  $\infty$  — «бесконечность».

1) Эта и все последующие иллюстрации получены в процессе работы с СУБД Microsoft Access.



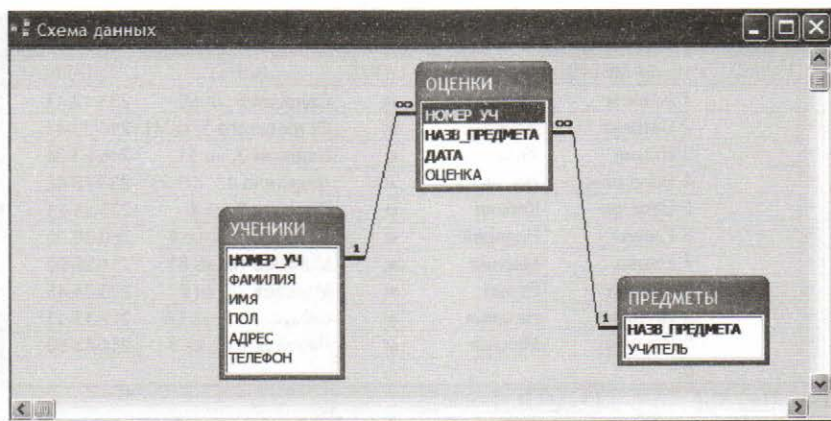


Рис. 1.14. Схема данных

После этого таблицы можно заполнять данными. Современные СУБД предоставляют пользователю удобные средства ввода. Данные можно вводить непосредственно в строки таблиц, отражаемых на экране, или через диалоговые окна — **формы** (рис. 1.15). В процессе ввода данных СУБД осуществляет автоматический контроль соответствия вводимых данных объявленным типам и форматам полей. На рисунке 1.15 показана форма для ввода данных в таблицу **УЧЕНИКИ**.

Рис. 1.15. Форма для ввода и просмотра данных в таблице

После того как таблица будет заполнена, через эту же форму можно просматривать записи таблицы или вносить в них изменения. На рисунке 1.16 показана таблица **УЧЕНИКИ**, заполненная для 10 человек.

Аналогичным способом заполняется таблица **ПРЕДМЕТЫ**, которая показана на рис. 1.17.

	НОМЕР УЧ	ФАМИЛИЯ	ИМЯ	ПОЛ	АДРЕС	ТЕЛЕФОН
▶ +	1	Антонов	Кирилл	м	Садовая 5, кв. 56	233-12-11
+ +	2	Веткина	Ирина	ж	Островского 3, кв.41	214-33-44
+ +	3	Вяткин	Иван	м	Садовая 3, кв. 14	245-23-34
+ +	4	Волочкова	Настасья	ж	Ладыгина 43, КВ.23	233-76-88
+ +	5	Волегов	Кирилл	м	Садовая 3, кв. 4	233-21-21
+ +	6	Гилев	Валерий	м	Лебедева 43, кв. 4	250-30-30
+ +	7	Ежова	Марина	ж	Малкова 76, кв.81	214-98-90
+ +	8	Зиминая	Елена	ж	Малкова 76, кв. 2	233-34-45
+ +	9	Игошина	Наталья	ж	Сибирская 8, кв. 65	212-11-11
+ +	10	Ильиных	Михаил	м	Лебедева 14, кв. 3	244-43-50
* +	0					

Запись: 1 из 10

Рис. 1.16. Таблица УЧЕНИКИ после заполнения

	НАЗВ ПРЕДМЕТА	УЧИТЕЛЬ
+ +	Алгебра	Русланова Анна Васильевна
+ +	Геометрия	Русланова Анна Васильевна
+ +	Информатика	Кирихин Сергей Ильич
+ +	Литература	Попова Нина Семеновна
+ +	Биология	Карлова Светлана Дмитриевна
+ +	Химия	Русских Павел Павлович
▶ +	Английский	Грузберг Клара Григорьевна
+ +	История	Онянов Юрий Петрович
* +		

Запись: 7 из 8

Рис. 1.17. Таблица ПРЕДМЕТЫ после заполнения

Самый большой объем данных будет иметь таблица ОЦЕНКИ. Она будет пополняться учителями каждый день. И только для одного класса в течение учебного года в нее будут занесены сотни, а то и тысячи записей. На рисунке 1.18 показан фрагмент таблицы ОЦЕНКИ, содержащий всего 20 записей.

База данных создана! Теперь можно приступать к разработке клиентских приложений.

*Замечание.* В параграфе 1.2.2 было сказано, что избыточность в БД является ее недостатком. Однако в некоторых ситуациях избыточность допустима. Например, в таблице на рис. 1.17 повторяются фамилии, имена и отчества учителей в их связи с названиями предметов. Положительным качеством такой таблицы является ее наглядность.



НОМЕР_УЧ	НАЗВ_ПРЕДМЕТА	ДАТА	ОЦЕНКА
1	Алгебра	04.09.2009	4
1	Информатика	17.09.2009	5
1	История	04.09.2009	5
1	История	21.09.2009	3
2	Алгебра	23.09.2009	3
2	Информатика	11.09.2009	2
2	История	07.09.2009	5
2	История	15.09.2009	5
2	История	30.09.2009	5
3	Алгебра	11.09.2009	5
3	Алгебра	23.09.2009	4
3	Информатика	14.09.2009	4
3	История	21.09.2009	5
4	Алгебра	23.09.2009	5
4	Информатика	17.09.2009	5
4	История	15.09.2009	5
5	Алгебра	15.09.2009	2
5	Алгебра	23.09.2009	4
5	Информатика	10.09.2009	3
5	История	30.09.2009	4

Запись: 14 из 21

Рис. 1.18. Фрагмент таблицы ОЦЕНКИ после заполнения

## Система основных понятий

Создание базы данных			
Создание БД осуществляется средствами СУБД			
Создание структуры БД		Ввод данных	
1. Создание таблиц:	2. Установка связей (создание схемы):	Ввод в строки таблицы	Ввод через форму
описание полей, типов, форматов, ключей	объединение через общие поля: 1-1 или 1-∞	Автоматический контроль соответствия данных типам и форматам полей	

## Вопросы и задания

1. Что нужно иметь для того, чтобы начать процесс создания базы данных?
2. Какую информацию нужно указать СУБД для создания таблиц БД?
3. Какую информацию нужно указать БД для создания схемы БД?

4. Каким способом можно вводить данные в таблицы?
5. Каким образом СУБД помогает пользователю производить безошибочный ввод данных?



Практикум. Раздел «Базы данных»

### 1.2.4. Простые запросы к базе данных

После того как база данных создана, ее можно использовать в качестве информационного справочника. В этом и состоит основное назначение всякой информационно-поисковой системы.

Действия, выполняемые над информацией, хранящейся в базе данных, называются **манипулированием данными**. К ним относятся: выборка данных по некоторым условиям, сортировка данных, обновление, удаление устаревших и добавление новых данных. Выполнение этих действий производится с помощью запросов.

**Запрос** — это описание действий СУБД с целью выполнения определенного вида манипулирования данными.

Есть универсальный язык, на котором формулируются запросы во многих СУБД. Он называется **SQL (Structured Query Language)** — структурированный язык запросов. Здесь существует выбор, с которым часто приходится сталкиваться в информатике: обучаться ли составлению запросов на языке SQL или воспользоваться каким-то более высокоуровневым вспомогательным средством. В большинстве современных СУБД такие средства имеются.

В учебных целях мы будем использовать строчное описание команд запросов на придуманном (учебном) языке. Он близок к SQL, однако имеет не такой строгий синтаксис и, кроме того, использует русские служебные слова. С учебным языком запросов вы знакомились в курсе информатики 9 класса. Учебный язык запросов соотносится с SQL примерно так же, как учебный Алгоритмический язык соотносится с языками программирования.

Команда **запроса на выборку** данных из БД на учебном языке запросов имеет следующий формат:

**.выбрать** <список выводимых полей> **где** <условие выбора>  
**сортировать** <ключ сортировки> **по** <порядок сортировки>

Не все составляющие этой команды являются обязательными. Могут отсутствовать условия выбора и параметры сортировки. Кроме того, ключей сортировки может быть несколько. Тогда они записываются в порядке приоритетов: первый, второй и т. д.

Опишем серию запросов на учебном языке, которую позже в практикуме реализуем средствами СУБД. В 8 классе вы учились



составлять запросы к однотабличной БД. Теперь рассмотрим примеры запросов, для выполнения которых потребуется извлекать данные из нескольких таблиц. В таком случае имена полей идентифицируются комбинированными именами следующего формата:

**ИМЯ ТАБЛИЦЫ.ИМЯ ПОЛЯ**

**Запрос 1.** Получить список учеников класса, содержащий номер в журнале, фамилию и имя каждого ученика.

**.выбрать УЧЕНИКИ.НОМЕР\_УЧ, УЧЕНИКИ.ФАМИЛИЯ, УЧЕНИКИ.ИМЯ сортировать УЧЕНИКИ.НОМЕР\_УЧ по возрастанию**

В этом запросе все данные берутся из одной таблицы УЧЕНИКИ. Условие выбора отсутствует, поскольку нужно выбрать все записи таблицы. Сортировка осуществляется по возрастанию номеров. Результат запроса представлен в табл. 1.4.

Таблица 1.4

**Список учеников класса**

	НОМЕР_УЧ	ФАМИЛИЯ	ИМЯ
▶	1	Антонов	Кирилл
	2	Веткина	Ирина
	3	Вяткин	Иван
	4	Волочкова	Настасья
	5	Волегов	Кирилл
	6	Гилев	Валерий
	7	Ежова	Марина
	8	Зими́на	Елена
	9	Игошина	Наталья
	10	Ильиных	Михаил

**Запрос 2.** Вывести список всех оценок, полученных Волеговым Кириллом по всем предметам. Указать даты получения оценок.

**.выбрать ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА, ОЦЕНКИ.ДАТА, ОЦЕНКИ.ОЦЕНКА где ОЦЕНКИ.НОМЕР\_УЧ=5 сортировать ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА по возрастанию, ОЦЕНКИ.ОЦЕНКА по убыванию**

В этом запросе, как и в предыдущем, все данные извлекаются из одной таблицы ОЦЕНКИ. Результат запроса представлен в табл. 1.5. Обратите внимание на то, что оценки по алгебре занесены в последовательные строки и упорядочены по убыванию.

Таблица 1.5

**Оценки ученика Волегова Кирилла**

НАЗВ_ПРЕДМЕТА	ДАТА	ОЦЕНКА
Алгебра	23.09.2009	4
Алгебра	15.09.2009	2
Информатика	10.09.2009	3
История	30.09.2009	4

**Запрос 3.** Получить список всех оценок, полученных по алгебре всеми учениками класса. Отсортировать по фамилиям учеников и по датам получения оценок.

**.выбрать** УЧЕНИКИ.ФАМИЛИЯ, ОЦЕНКИ.ДАТА, ОЦЕНКИ.ОЦЕНКА где ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА="Алгебра" **сортировать** УЧЕНИКИ.ФАМИЛИЯ по возрастанию, ОЦЕНКИ.ДАТА по возрастанию

В этом запросе используются данные из двух связанных таблиц УЧЕНИКИ и ОЦЕНКИ. Результат запроса представлен в табл. 1.6.

Таблица 1.6

**Оценки по алгебре**

ФАМИЛИЯ	ДАТА	ОЦЕНКА
Антонов	04.09.2009	4
Веткина	23.09.2009	3
Волегов	15.09.2009	2
Волегов	23.09.2009	4
Волочкова	23.09.2009	5
Вяткин	11.09.2009	5
Вяткин	23.09.2009	4

**Запрос 4.** Вывести список всех двоек, полученных учениками класса, с указанием фамилии ученика, даты получения двойки, предмета и учителя. Отсортировать по фамилиям учеников.

**.выбрать** УЧЕНИКИ.ФАМИЛИЯ, ОЦЕНКИ.ДАТА, ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА, ПРЕДМЕТЫ.УЧИТЕЛЬ где ОЦЕНКИ.ОЦЕНКА=2 **сортировать** УЧЕНИКИ.ФАМИЛИЯ по возрастанию

Для реализации этого запроса были использованы все три связанные таблицы. Результат представлен в табл. 1.7.



Таблица 1.7

Список двоечников

ФАМИЛИЯ	ДАТА	НАЗВ_ПРЕДМЕТА	УЧИТЕЛЬ
Веткина	11.09.2009	Информатика	Кирюхин Сергей Ильич
Волегов	15.09.2009	Алгебра	Русланова Анна Васильевна

## Система основных понятий

Запросы к базе данных		
Запрос — описание действий СУБД с целью выполнения определенного вида манипулирования данными		
Средства формирования запросов		
SQL (Structured Query Language) — структурированный язык запросов	Конструктор запросов (Microsoft Access)	
Структура запроса на выборку		
Список полей	Условие выбора записей	Ключи и порядок сортировки
Имена полей (простые или составные), выводимых по запросу	Логическое выражение, которому удовлетворяют выбираемые записи	Один ключ или последовательность ранжированных ключей. Порядок: по возрастанию, по убыванию

## Вопросы и задания

1. Что входит в понятие манипулирования данными в БД?
2. Какова цель запроса на выборку?
3. Запишите на учебном языке запросов команды выборки для получения следующей информации:
  - получить таблицу с фамилиями, адресами и номерами телефонов всех мальчиков класса;
  - вывести сведения об оценках, полученных учениками 15 сентября 2009 года; указать фамилию ученика, предмет и оценку; отсортировать по убыванию значения оценки.

Практикум. Раздел «Базы данных»

### 1.2.5. Сложные запросы к базе данных

Запросы, которые будут строиться в этом параграфе, мы назвали сложными. Сложность — относительное понятие. По сравнению с запросами из предыдущего параграфа здесь появятся следующие новые элементы:

- использование в условиях выбора логических операций (конъюнкция, дизъюнкция, отрицание);
- применение группировки записей;
- использование статистических функций в группах записей;
- создание вычисляемых полей.

Нумерация запросов, начатая в предыдущем параграфе, здесь будет продолжена.

#### Логические операции в условии выбора

**Запрос 5.** Вывести список всех учеников класса, получивших пятерки по алгебре, и всех учеников, получивших пятерки по истории, указав фамилию, имя, предмет и дату получения оценки.

**.выбрать** УЧЕНИКИ.ФАМИЛИЯ, УЧЕНИКИ.ИМЯ,  
ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА, ОЦЕНКИ.ДАТА

где (ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА="Алгебра" **или**  
ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА="История") **и**  
ОЦЕНКИ.ОЦЕНКА=5

**сортировать** УЧЕНИКИ.ФАМИЛИЯ по возрастанию  
ОЦЕНКИ.ДАТА по убыванию

Результаты этого запроса представлены в табл. 1.8. Условие выборки в запросе является сложным логическим выражением, содержащим операцию дизъюнкции (**или**) и операцию конъюнкции (**и**). Круглые скобки в логическом выражении необходимы, поскольку операция дизъюнкции младше конъюнкции, но выполняться она должна в первую очередь.

Таблица 1.8

Пятерки по истории и алгебре

ФАМИЛИЯ	ИМЯ	НАЗВ_ПРЕДМЕТА	ДАТА
Антонов	Кирилл	История	04.09.2009
Веткина	Ирина	История	30.09.2009
Веткина	Ирина	История	15.09.2009
Веткина	Ирина	История	07.09.2009
Волочкова	Настасья	Алгебра	23.09.2009
Волочкова	Настасья	История	15.09.2009
Вяткин	Иван	История	21.09.2009
Вяткин	Иван	Алгебра	11.09.2009



## Группировка и статистическая обработка

В следующем запросе будет использоваться группировка выбираемых по запросу записей. Сгруппировать, т. е. собрать вместе, можно несколько записей, у которых совпадают значения одного или нескольких выбираемых в запросе полей. Для выделенной группы записей можно выполнить обработку значений некоторого числового поля. Функции, выполняющие такую обработку, называются **статистическими функциями**. Например, можно просуммировать значения числового поля для всей группы (применить функцию Sum), вычислить среднее значение (функция Avg), подсчитать количество элементов записей в группе (функция Count), найти максимум или минимум (функции Max, Min) и пр. Полученное значение можно вывести в итоговой таблице.

**Запрос 6.** Вычислить среднюю оценку для каждого ученика по каждому предмету. Вывести фамилию ученика, название предмета, среднюю оценку, фамилию, имя, отчество учителя по этому предмету.

Такой запрос может выполнить классный руководитель в конце учебной четверти, для того чтобы спрогнозировать ее итоги.

**выбрать** УЧЕНИКИ.ФАМИЛИЯ, ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА, Avg(ОЦЕНКИ.ОЦЕНКА), ПРЕДМЕТЫ.УЧИТЕЛЬ **группировать** УЧЕНИКИ.ФАМИЛИЯ, ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА, ПРЕДМЕТЫ.УЧИТЕЛЬ **сортировать** УЧЕНИКИ.ФАМИЛИЯ по возрастанию, ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА по возрастанию

Таблица 1.9

Средние оценки

ФАМИЛИЯ	НАЗВ_ПРЕДМЕТА	Avg_ОЦЕНКА	УЧИТЕЛЬ
Антонов	Алгебра	4	Русланова Анна Васильевна
Антонов	Информатика	5	Кирюхин Сергей Ильич
Антонов	История	4	Онянов Юрий Петрович
Веткина	Алгебра	3	Русланова Анна Васильевна
Веткина	Информатика	2	Кирюхин Сергей Ильич
Веткина	История	5	Онянов Юрий Петрович
Волегов	Алгебра	3	Русланова Анна Васильевна
Волегов	Информатика	3	Кирюхин Сергей Ильич
Волегов	История	4	Онянов Юрий Петрович
Волочкова	Алгебра	5	Русланова Анна Васильевна
Волочкова	Информатика	5	Кирюхин Сергей Ильич
Волочкова	История	5	Онянов Юрий Петрович
Вяткин	Алгебра	4,5	Русланова Анна Васильевна
Вяткин	Информатика	4	Кирюхин Сергей Ильич
▶ Вяткин	История	5	Онянов Юрий Петрович

Из этого примера видно, что в запросе на выборку можно указывать поля группировки после служебного слова **группировать**. В группы объединяются выбираемые по запросу записи, у которых совпадают поля фамилии учеников, названия предметов и учителя. В каждой такой группе вычисляется среднее арифметическое значение оценок. В итоговой таблице (табл. 1.9) средние значения выводятся в графу с названием Avg-ОЦЕНКА.

**Запрос 7.** Получить список учеников, имеющих среднюю оценку по алгебре выше 4. Вывести фамилию ученика и среднюю оценку по алгебре. Отсортировать список по фамилиям.

Команда для этого запроса будет построена путем модификации команды из предыдущего запроса: добавления условий выбора.

**.выбрать** УЧЕНИКИ.ФАМИЛИЯ, ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА, Avg(ОЦЕНКИ.ОЦЕНКА) **группировать** УЧЕНИКИ.ФАМИЛИЯ, ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА где ОЦЕНКИ.НАЗВ\_ПРЕДМЕТА = "Алгебра" и Avg(ОЦЕНКИ.ОЦЕНКА) >4 **сортировать** УЧЕНИКИ.ФАМИЛИЯ по возрастанию

В результате выполнения этого запроса получится таблица, состоящая из двух строк (табл. 1.10).

Таблица 1.10

**Лучшие по алгебре**

	ФАМИЛИЯ	НАЗВ_ПРЕДМЕТА	Avg_ОЦЕНКА
	Волочкова	Алгебра	5
▶	Вяткин	Алгебра	4,5

**Расширение базы данных**

Добавим в нашу базу данных еще одну полезную информацию: сведения о четвертных оценках по предметам. Назовем новую таблицу ИТОГИ и опишем ее структуру (табл. 1.11).

Таблица 1.11

**ИТОГИ**

Имя поля	Тип поля	Длина (формат)
<u>НОМЕР_УЧ</u>	Числовой	Целое
<u>НАЗВ_ПРЕДМЕТА</u>	Текстовый	30
1_ЧЕТВ	Числовой	Целое
2_ЧЕТВ	Числовой	Целое
3_ЧЕТВ	Числовой	Целое
4_ЧЕТВ	Числовой	Целое



Составной ключ этой таблицы:

НОМЕР\_УЧ + НАЗВ\_ПРЕДМЕТА.

Добавим эту таблицу в схему БД и установим связи. Связь между таблицами УЧЕНИКИ и ИТОГИ будет иметь тип «один-ко-многим». Такой же тип связи установится между таблицами ПРЕДМЕТЫ и ИТОГИ. Схема данных теперь выглядит, как показано на рис. 1.19.

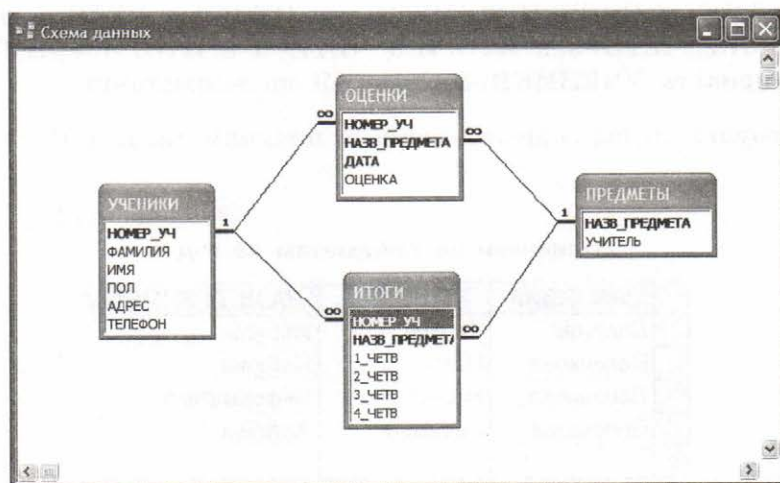


Рис. 1.19. Расширенная схема данных

Теперь заполним таблицу ИТОГИ данными. Ограничимся тремя предметами, как и в таблице ОЦЕНКИ. Введенные данные представлены в таблице на рис. 1.20.

НОМЕР_УЧ	НАЗВ_ПРЕДМ	1_ЧЕТВ	2_ЧЕТВ	3_ЧЕТВ	4_ЧЕТВ
1	Алгебра	4	3	4	4
1	Информатика	4	5	4	4
1	История	3	4	4	3
2	Алгебра	3	3	4	3
2	Информатика	3	3	3	3
2	История	5	4	5	4
3	Алгебра	3	4	3	3
3	Информатика	4	3	4	3
3	История	4	5	4	4
4	Алгебра	5	5	5	5
4	Информатика	5	5	5	5
4	История	5	5	5	5
5	Алгебра	5	4	5	5
5	Информатика	4	4	4	4
5	История	5	5	5	5

Рис. 1.20. Содержание таблицы ИТОГИ

Используя расширенную базу данных, построим к ней несколько запросов.

**Запрос 8.** Получить список отличников по отдельным предметам. Указать фамилию, имя ученика и предмет, по которому он имеет все четвертные пятерки.

**.выбрать** УЧЕНИКИ.ФАМИЛИЯ, УЧЕНИКИ.ИМЯ, ПРЕДМЕТЫ.НАЗВ\_ПРЕДМЕТА где ИТОГИ.1\_ЧЕТВ=5 и ИТОГИ.2\_ЧЕТВ=5 и ИТОГИ.3\_ЧЕТВ=5 и ИТОГИ.4\_ЧЕТВ=5  
**сортировать** УЧЕНИКИ.ФАМИЛИЯ по возрастанию

В результате выполнения запроса получим табл. 1.12.

Таблица 1.12

Отличники по предметам за год

	ФАМИЛИЯ	ИМЯ	НАЗВ_ПРЕДМЕТА
	Волегов	Кирилл	История
	Волочкова	Настасья	История
	Волочкова	Настасья	Информатика
	Волочкова	Настасья	Алгебра
▶			

### Вычисляемые поля

**Запрос 9.** Вычислить суммы оценок по информатике у всех учеников и вывести список в порядке убывания сумм.

В запросе в списке выводимых полей можно определить новое поле, для которого будет задано имя и формула вычисления его значений. Назовем такое поле СУММА.

**.выбрать** УЧЕНИКИ.ФАМИЛИЯ, СУММА: [ИТОГИ].[1\_ЧЕТВ]+[ИТОГИ].[2\_ЧЕТВ]+[ИТОГИ].[3\_ЧЕТВ]+[ИТОГИ].[4\_ЧЕТВ] где ПРЕДМЕТЫ.НАЗВ\_ПРЕДМЕТА="Информатика" **сортировать** СУММА по убыванию

Обратите внимание на двоеточие после имени вычисляемого поля и квадратные скобки в выражении для вычисления суммы. Имя поля, заключенное в скобки, обозначает значение этого поля. В результате получим табл. 1.15.



Таблица 1.15

**Сумма четвертных оценок по информатике**

	ФАМИЛИЯ	СУММА
	Волочкова	20
	Антонов	17
	Волегов	16
	Вяткин	14
	Веткина	12
▶		

**Запрос 10.** Вычислить суммы всех оценок, полученных всеми учениками по каждому предмету в отдельности в каждой четверти. Вывести название предмета и суммы оценок для каждой четверти. Отсортировать по названиям предметов.

Для этого запроса нужны данные только из таблицы ИТОГИ. Нужно сгруппировать все записи по предметам и внутри каждой группы просуммировать оценки за каждую четверть.

**.выбрать** ИТОГИ.НАЗВ\_ПРЕДМЕТА, Sum(ИТОГИ.1\_ЧЕТВ),  
Sum(ИТОГИ.2\_ЧЕТВ), Sum(ИТОГИ.3\_ЧЕТВ), Sum(ИТОГИ.4\_ЧЕТВ)  
**группировать** ИТОГИ.НАЗВ\_ПРЕДМЕТА **сортировать**  
ИТОГИ.НАЗВ\_ПРЕДМЕТА по возрастанию

Из таблицы 1.16 видно, что во всех четвертях самая высокая сумма оценок — по истории. А второе и третье места в рейтинге успеваемости поочередно делят алгебра и информатика.

Таблица 1.16

**Сумма оценок класса по четвертям**

	НАЗВ_ПРЕДМЕТА	Sum-1_ЧЕТВ	Sum-2_ЧЕТВ	Sum-3_ЧЕТВ	Sum-4_ЧЕТВ
	Алгебра	20	19	21	20
	Информатика	20	20	20	19
▶	История	22	23	23	21

**Запрос 11.** Получить годовой рейтинг успеваемости по предметам. Для этого просуммировать по каждому предмету суммы оценок за четыре четверти. Вывести названия предметов и итоговые суммы оценок.

.выбрать ИТОГИ.НАЗВ\_ПРЕДМЕТА, Sum(ИТОГИ.1\_ЧЕТВ), Sum(ИТОГИ.2\_ЧЕТВ), Sum(ИТОГИ.3\_ЧЕТВ), Sum(ИТОГИ.4\_ЧЕТВ), СУММА: [Sum-1\_ЧЕТВ]+[ Sum-2\_ЧЕТВ]+[Sum-3\_ЧЕТВ]+ [Sum-4\_ЧЕТВ] группировать ИТОГИ.НАЗВ\_ПРЕДМЕТА

В результате запроса получим табл. 1.17.

Таблица 1.17

Годовой рейтинг предметов

НАЗВ_ПРЕДМЕТА	Sum-1_ЧЕТВ	Sum-2_ЧЕТВ	Sum-3_ЧЕТВ	Sum-4_ЧЕТВ	СУММА
Алгебра	20	19	21	20	80
Информатика	20	20	20	19	79
История	22	23	23	21	89

Отсюда видно, что и по итогам года самая высокая успеваемость по истории, на втором месте алгебра, на третьем — информатика.

## Система основных понятий

### Запросы к базе данных (продолжение)

Сложное условие запроса	Условие, содержащее логические операции
Поля группировки	Определяют поля, по совпадению значений которых выбираемые записи объединяются в группы
Статистическая обработка	Применение в запросах статистических функций к значениям сгруппированных полей (сумма, среднее значение, максимальное, минимальное значения и др.)
Вычисляемое поле	Поле, формируемое в запросе; задается имя и формула для вычисления по значениям других полей

## Вопросы и задания

1. Какие логические операции можно использовать в сложном условии запроса на выборку?
2. Что такое поля группировки?
3. Какие виды вычислительной обработки данных можно выполнять в следующих запросах на выборку?
  - Получить список учеников, у которых нет троек по информатике.
  - Вывести фамилии учителей, по предметам которых средний балл за четыре четверти у всех учеников больше, чем 4.



4. Спроектируйте схему базы данных, в которой будут содержаться сведения об успеваемости учеников во всех классах данной школы.
5. Придумайте несколько сложных запросов к базе данных, спроектированной в результате выполнения задания 4.
6. Спроектируйте схему базы данных, в которой будут содержаться сведения об успеваемости учеников во всех школах города.

Практикум. Раздел «Базы данных»

ЭОР к главе 1 на сайте ФЦИОР (<http://fcior.edu.ru>)

#### 1.1.

- Назначение и виды информационных моделей
- Построение информационных моделей ИС

#### 1.2.

- Ввод данных в БД
- Высказывание. Простые и сложные высказывания. Основные логические операции
- Запросы на выборку данных
- Понятие СУБД. Классификация СУБД
- Проектирование баз данных
- Проектирование объектов данных
- Проектирование отчетов
- Проектирование экранных форм
- Создание отчетов в БД
- Этапы разработки ИС



## Глава 2

# МЕТОДЫ ПРОГРАММИРОВАНИЯ

Вы уже не новички в программировании. В 10 классе каждый из вас составил не менее десятка программ на Паскале. Что же нового о программировании вы узнаете, изучив данную главу? Почему она называется «Методы программирования»?

Цель изучения этой главы состоит в том, чтобы приблизить ваши знания и умения в области программирования к профессиональному уровню. Для всякого профессионала важно иметь системное представление о предмете своей деятельности. Под фразой «методы программирования» понимается совокупность способов, средств, технологий создания программ для компьютера. В данной главе системно излагаются некоторые из этих методов.

Как принято в нашем учебнике, мы немного заглянем в историю программирования, чтобы понять логику его развития, а потом вы будете осваивать основы современных методов программирования.

## 2.1. Эволюция программирования

**Программирование** для компьютера — процесс создания программ управления работой компьютера с целью решения определенных информационных задач.

### ***Машинно-ориентированное программирование***

С изобретением программно управляемых вычислительных машин появилась новая профессия — программист. Мы уже рассказывали, что первым в истории программистом была Ада Лавлейс, работавшая вместе с Чарльзом Бэббиджем, она разрабатывала программы управления его Аналитической машиной. Но массовой профессия программиста стала только с изобретением электронных вычислительных машин — ЭВМ.



На ламповых ЭВМ первого поколения программисты составляли свои программы, используя непосредственно команды процессора. При этом программисту приходилось распределять ячейки памяти под данные и под команды программы. Нужно было знать систему команд процессора и коды всех команд. Исходные данные и команды представлялись в форме двоичного кода, т. е. непосредственно в том виде, в котором они хранятся в памяти ЭВМ. Для сокращения записи программ на специальных бланках обычно использовали двоично-восьмеричный или двоично-шестнадцатеричный код. Вот пример команды программы для одного из компьютеров первого поколения.

	Адрес команды	Код операции	1-й адрес	2-й адрес	3-й адрес
Шестнадцатеричный код	28	02	C0	C4	D8
Двоичный код	0010 1000	00000 0010	1100 0000	1100 0100	1101 1000

Такая команда называется трехадресной. Код  $02_{16}$  обозначает команду сложения. 1-й и 2-й адреса — это адреса ячеек ОЗУ, в которых хранятся слагаемые, 3-й адрес — адрес ячейки, куда заносится сумма. Сама команда хранится в ячейке ОЗУ с адресом  $28_{16}$ .

Программирование в машинных кодах представляло собой сложный процесс. По этой причине производительность работы программистов была довольно низкой. В 1950-х годах возникает направление, которое получило название «автоматизация программирования». Основная его цель: создание средств, облегчающих и ускоряющих процесс создания программы для ЭВМ. Появляются первые языки программирования.

Первыми языками программирования были машинно-ориентированные автокоды. Позднее за языками такого уровня закрепилось название ассемблеры. Первоначально ассемблером называли программу-переводчик с языка ассемблера в машинные команды. Позднее и сам язык ассемблера стали называть именем «ассемблер». Программирование на ассемблере снимает с программиста заботу о распределении памяти под данные и команды программы. Программист также не должен помнить внутренние коды всех команд процессора. Вот пример той же команды сложения на ассемблере (автокоде):

```
ADD a, b, c
```

Слово ADD обозначает команду «сложить»,  $a$  и  $b$  — имена переменных-слагаемых,  $c$  — переменная, куда помещается результат.

Язык ассемблер называется машинно-ориентированным по той причине, что для каждой команды процессора существует свой аналог команды на ассемблере. Поскольку разные типы ЭВМ имели разные системы команд процессора, ассемблеры у них тоже различались. Современные ассемблеры точно так же ориентированы на определенные типы процессоров. Позже появились так называемые *макроассемблеры*, в языке которых существуют макрокоманды, соответствующие сериям команд (подпрограммам) на языке процессора.

Составление программы на ассемблере проще, чем на языке команд процессора. Работу по распределению памяти под данные и команды, перевод команд ассемблера в машинные команды берет на себя специальная системная программа — **транслятор**.

Из машинной ориентированности программ на ассемблере следует, что такие программы нельзя переносить для исполнения на другие типы ЭВМ с другой системой команд процессора. Эта проблема создавала серьезные ограничения для прикладных программистов. Кроме того, само программирование на ассемблере является достаточно сложным для массового освоения, что ограничивало использование ЭВМ в прикладных областях.

### **Языки программирования высокого уровня**

Следующим этапом развития программирования стало создание **языков программирования высокого уровня (ЯПВУ)**. Примеры ЯПВУ: Паскаль, Бейсик, Фортран. Для каждого языка существует машинно-независимый стандарт. Возможность программирования на данном ЯПВУ зависит от наличия на вашем компьютере транслятора с этого языка. Трансляторы для каждого типа компьютеров составляют системные программисты.

Текст программы на ЯПВУ по своей форме ближе к естественным языкам (чаще всего — к английскому), к языку математики. Та же команда сложения двух величин на ЯПВУ похожа на привычную форму математического равенства:

$c := a + b$  (на Паскале);

$c = a + b$  (на Фортране, Бейсике, Си).

Гораздо проще освоить программирование на языке высокого уровня, чем на ассемблере. Поэтому с появлением ЯПВУ значительно возросло число прикладных программистов, расширилось применение ЭВМ во многих областях.



С середины XX века и до нашего времени были созданы сотни языков программирования высокого уровня. Но распространенными и популярными из них стали не все. Одним из долгожителей в семействе ЯПВУ является язык Фортран (FORTRAN). FORTRAN — сокращение словосочетания *formula translator* — транслятор формул. Первая версия Фортрана была создана в 1954 году. Во времена ЭВМ второго и третьего поколений была популярна версия Фортран-IV. Фортран создавался как специализированный язык для математических расчетов, используемых в науке и технике. И в наше время этот язык, в стандарте Фортран-90 (и в последующих его модификациях Фортран-95, Фортран-2003), остается основным языком программирования для расчетов в области физико-технических проблем.

К числу первых ЯПВУ, созданных в 1950-х годах, относятся Кобол (создан в США) и Алгол (в Европе). Алгол, как и Фортран, был ориентирован на научно-технические расчеты математического характера. Кобол — язык для программирования экономических задач. В Коболе, по сравнению с двумя другими названными языками, слабее развиты математические средства, но зато хорошо представлены средства обработки текстов, организации вывода данных в форме требуемого документа. Для первых ЯПВУ предметная ориентация языков была характерной чертой.

Большое количество языков программирования появилось в 1960–1970-х годах. В 1965 году в Дартмутском университете был разработан язык Бейсик. По замыслу авторов это простой язык, легко изучаемый, предназначенный для программирования несложных расчетных задач. Наибольшее распространение Бейсик получил с появлением микроЭВМ и персональных компьютеров.

Значительным событием в истории языков программирования стало создание в 1969 году языка Паскаль. Его автор — швейцарский профессор Никлаус Вирт разрабатывал Паскаль как учебный язык структурного программирования.

Наибольший успех в распространении языка Паскаль обеспечили персональные компьютеры. Фирма Borland International, Inc (США) разработала систему программирования Turbo Pascal (Турбо Паскаль) для ПК. Турбо Паскаль — это не только язык и транслятор с него, но еще и **интегрированная среда программирования**, дающая пользователю возможность удобно работать на Паскале: вводить и редактировать текст программы, искать синтаксические ошибки, пользоваться библиотеками подпрограмм и модулей, работать с файлами и пр. Турбо Паскаль вышел за рамки учебного предназначения и стал языком профессионального программирования с универсальными возможностями. Паскаль



стал источником многих основных современных языков программирования, например таких, как Ада, Модула-2 и др.

Модула-2 — это еще один язык, предложенный Виргом, являющийся развитием языка Паскаль и содержащий средства для создания больших программ.

Язык программирования Си (английское название — С) появился практически одновременно с Паскалем. Он создавался как инструментальный язык для разработки операционных систем, трансляторов, баз данных и других системных и прикладных программ. Хотя Си и является языком высокого уровня, в нем заложены возможности непосредственного обращения к некоторым машинным командам, к определенным участкам памяти компьютера, что ранее было возможно только в ассемблере. С появлением Си многие системные программисты перешли с ассемблера на Си. Дальнейшее развитие Си привело к созданию языка объектно-ориентированного программирования Си++.

### **Парадигмы программирования**

Слово «парадигма» применительно к программированию означает определенный общепринятый подход к организации вычислений на компьютере. Парадигма определяет систему базовых понятий, на основе которой происходит программирование.

Языки программирования, о которых рассказывалось выше, основаны на **процедурной парадигме программирования**. Алгоритм, реализованный на процедурном языке, основывается на представлениях о фон-неймановской архитектуре компьютера. Базовым понятием является понятие величины, хранимой в памяти компьютера, базовой операцией — операция присваивания. Решение задачи на компьютере происходит путем изменения состояния памяти: в начале в память помещаются исходные данные, в конце получаются результаты. Программа — это описание процедуры перехода памяти из начального состояния в конечное в процессе выполнения последовательности команд. Мышление программиста, работающего в рамках процедурной парадигмы, направлено на сведение решения задачи к последовательности команд, которые умеет выполнять процессор компьютера.

На другом подходе к программированию основана **функциональная парадигма программирования**. Наиболее известным языком, реализующим функциональную парадигму, является ЛИСП.

В теории программирования существует предположительное утверждение (которое не имеет строгого доказательства) о том, что любой алгоритм может быть описан с помощью некоторого набора частично-рекурсивных функций. Известно, что достаточно широ-



кий класс задач, решаемых средствами процедурного программирования, поддается решению и в функциональной парадигме. Язык ЛИСП позволяет моделировать на компьютере достаточно сложные процессы, в частности, интеллектуальную деятельность людей. Помимо ЛИСПа, существуют и другие языки функционального программирования: Haskell, Scheme и др.

**Логическая парадигма программирования** реализована в языке Пролог. Пролог был разработан во Франции в 1972 году. Его, как и ЛИСП, относят к языкам искусственного интеллекта. Пролог основан на аппарате математической логики, отсюда название парадигмы. На Прологе строится база знаний в определенной предметной области, состоящая из совокупности фактов и правил. Для решения некоторой задачи формулируется запрос к базе знаний, который называется целью. Реализация цели (ответ на запрос) происходит через заложенный в интерпретатор Пролога алгоритм, который называется механизмом вывода. Используя Пролог, можно решать логические задачи, аналогичные тем, которые были рассмотрены в § 1.6.4 учебника для 10 класса. Следует отметить, что с помощью Пролога можно решать лишь ограниченный круг задач искусственного интеллекта. К языкам логического программирования также относятся Planner, Mercury, Fril и др.

**Объектно-ориентированная парадигма** программирования основана на концепции объектов и классов. Всякий объект характеризуется набором свойств и действий, которые с ним могут быть выполнены и которые он может выполнять. Объект относится к определенному классу. Объектно-ориентированное программирование (ООП) сводится к выстраиванию иерархий классов, описаний объектов и их взаимодействий, программной реализации различных действий над объектами. Первым языком ООП был Симула, разработанный в 1967 году. Позже элементы ООП стали внедряться в процедурные языки, в том числе в Турбо Паскаль. Наиболее популярными средствами объектно-ориентированного программирования являются языки Си++, Java. В нашем учебнике подробнее об ООП рассказывается в разделе 2.4.

### **Методологии и технологии программирования**

На первых ЭВМ с «тесной» памятью и небольшим быстродействием основным показателем качества программы была ее экономичность по занимаемой памяти и времени счета. Чем программа получалась короче, тем класс программиста считался выше.

С увеличением объема памяти и быстродействия ЭВМ, с совершенствованием языков программирования и трансляторов с этих языков проблема экономичности программы становится менее острой. Всё более важными качественными характеристиками программ становятся их простота, наглядность, надежность. С появлением машин третьего поколения эти качества стали основными.

Уже в 1960-х годах программирование стало достаточно массовой профессиональной деятельностью. Возникают компании (фирмы) по разработке программ. Актуальной становится задача разработки общепринятой методологии программирования, повышающей производительность работы программистов и, что самое главное, качество программных продуктов. Основным качественным показателем программы — ее работоспособность, отсутствие ошибок.

**Методология программирования** — это совокупность определенных способов написания, отладки и сопровождения программ. Первая наиболее известная и распространенная методология программирования получила название **«структурное программирование»**.

Появление структурного программирования связано с именами Эдсгера Дейкстры и Чарльза Хоара. Начиная с 1960-х годов стали появляться языки структурного программирования. Первым из них был Алгол-60, разработанный Дейкстрой, затем был создан Паскаль. Другие, первоначально «неструктурные» языки стали также приобретать «структурные свойства» (Турбо Бейсик, Фортран-77 и пр.). Структурное программирование до настоящего времени остается важнейшей методологией программирования. Соблюдение его принципов позволяет программисту составлять ясные, безошибочные, надежные программы.

**Технология** — это воспроизводимый способ производства какого-то продукта, гарантирующий номинальное качество и оптимальные затраты. Технология предполагает наличие определенных средств, инструментов производства. В 1990-х годах с развитием объектно-ориентированной парадигмы программирования, а также средств графического интерфейса на персональных компьютерах возникает новая **технология программирования — визуальное программирование**. Визуальная технология программирования позволяет программисту легко и быстро строить наглядный графический интерфейс для своих программ на основе стандартного набора шаблонов, графически отображаемых на экране объектов. О технологии визуального программирования в системе Delphi рассказывается в разделе 2.4 нашего учебника.



**О профессиях: профессии, связанные с программированием**

В перечне современных профессий имеется профессия «**программист**». Велико разнообразие профессиональной деятельности программистов. Всё это разнообразие принято делить на две группы: системное программирование и прикладное программирование. Исторически профессия системного программиста берет свое начало с разработки трансляторов с языков программирования. Возникает разделение программистов на системных и прикладных. **Прикладные программисты** используют компьютер для решения прикладных задач из различных предметных областей: математики, физики, экономики и др. А работа **системных программистов** состоит в том, чтобы облегчить, упростить работу прикладным программистам, повысить эффективность использования компьютера для решения прикладных задач. Системные программисты разрабатывают не только трансляторы, но и операционные системы, утилиты, программное обеспечение для функционирования компьютерных сетей, информационных систем и пр.

**Система основных понятий**

<b>Что такое программирование</b>			
<b>Программирование</b> для компьютера — процесс создания программ управления работой компьютера (создание программного обеспечения)			
<b>Машинно-ориентированное программирование</b>			
<i>Программирование в машинных командах:</i> требует от программиста распределения памяти под данные и команды программы, знания кодов операций и форматов команд		<i>Программирование на автокодах (ассемблерах):</i> символическое обозначение переменных, мнемоническое обозначение кодов операций. Нуждается в трансляции	
<b>Машинно-независимое программирование</b> — на языках программирования высокого уровня (ЯПВУ)			
<b>Парадигмы и языки программирования</b>			
Процедурное	Функциональное	Логическое	Объектно-ориентированное
Фортран, Алгол, Паскаль, Бейсик, Си и др.	ЛИСП, Haskell, Scheme и др.	Пролог, Planner, Mercury, Fril и др.	Симула, Си++, Java и др.
<b>Методологии и технологии программирования</b>			
Структурная методология процедурного программирования		Визуальная технология объектно-ориентированного программирования	

## Вопросы и задания

1. Что такое программирование?
2. В какой форме составлялись программы для первых ЭВМ?
3. Почему языки автокоды (ассемблеры) называются машинно-ориентированными языками программирования?
4. Назовите основные процедурные языки программирования в хронологической последовательности их создания.
5. Что такое парадигма программирования?
6. Назовите основные парадигмы программирования и их отличия друг от друга.
7. Что такое структурное программирование?
8. Что такое визуальное программирование?

## 2.2. Структурное программирование

### 2.2.1. Паскаль — язык структурного программирования

#### *История Паскаля*

Язык программирования Паскаль был создан швейцарским профессором Никлаусом Виртом в 1969 году как язык для обучения студентов структурной методике программирования. Язык получил свое название в честь Блеза Паскаля — изобретателя первого вычислительного механического устройства. Позднее фирма Borland International, Inc. (США) разработала систему программирования Турбо Паскаль для персональных компьютеров, которая вышла за рамки учебного применения и стала использоваться для научных и производственных целей. В Турбо Паскаль были внесены некоторые дополнения к базовому стандарту Паскаля, описанного Виртом. Со временем язык развивался. Начиная с версии 5.5, в Турбо Паскаль вводятся средства поддержки объектно-ориентированного программирования. В дальнейшем это привело к созданию Object Pascal — языка с возможностями объектно-ориентированного программирования (ООП). В начале 1990-х годов объединение элементов ООП в Паскале с визуальной технологией программирования привело к созданию системы программирования Delphi.

У вас уже есть опыт программирования на Паскале, который вы получили, изучая информатику в 9 и 10 классах. Обучение там проводилось на примерах программ. Правила языка излага-



лись по мере их использования в рассматриваемых программах. Целью данной главы является систематизация и расширение ваших знаний о языке Паскаль. Как это было и раньше, язык описывается в рамках расширенного стандарта Паскаля, соответствующего диалекту Турбо Паскаль.

В начале данной главы учебника мы будем рассматривать методику структурного программирования на Паскале. Позднее вы познакомитесь с основами ООП и визуального программирования на Delphi.

### Структура процедурных языков программирования высокого уровня

Во всяком языке программирования определены способы организации данных и способы организаций действий над данными. Кроме того, существует понятие «элементы языка», включающее множество символов (алфавит), служебных слов и других изобразительных средств языка программирования. Несмотря на разнообразие процедурных языков, их изучение происходит приблизительно по одной схеме. Это связано с общностью структуры различных процедурных языков программирования высокого уровня, которая схематически отражена на рис. 2.1.



Рис. 2.1. Структура процедурного ЯПВУ

Всякий язык программирования образуют *три его основные составляющие: алфавит, синтаксис и семантика*. **Алфавит** — это множество символов, допустимых в записи текстов программ.

**Синтаксис** — это правописание языковых конструкций (имен, констант, выражений, операторов и пр.). **Семантика** — это смысловое содержание языковой конструкции.

Соблюдение правил в языке программирования должно быть строгим. В разговорном языке оно не такое строгое. Человеческая речь содержит значительное количество избыточной информации. Не расслышав какое-то слово, можно понять смысл фразы в целом. Слышающий или читающий человек может додумать, дополнить, исправить ошибки в воспринимаемом тексте. Компьютер же — автомат, воспринимающий все буквально. В текстах программ нет избыточности, компьютер сам не исправит даже очевидной (с точки зрения человека) ошибки. Он может лишь указать на место, которое «не понял», и вывести замечание о предполагаемом характере ошибки. Исправить же ошибку должен программист.

Вы уже знакомы со многими элементами Паскаля, составляли программы на Паскале. Поэтому целью данного раздела нашего курса является упорядочение имеющихся у вас знаний языка и их дальнейшее развитие, необходимое для того, чтобы научиться решать задачи новых типов.

### Структура программы на Паскале

По определению стандартного Паскаля программа состоит из **заголовка программы** и **тела программы (блока)**, за которым следует **точка** — признак конца программы. В свою очередь блок содержит **разделы описаний** (меток, констант, типов, переменных, подпрограмм) и **раздел операторов**.

```

Program <имя программы>;
Label <раздел меток>;
Const <раздел констант>;
Type <раздел типов>;
Var <раздел переменных>;
Procedure (Function) <раздел подпрограмм>;
begin
    <раздел операторов>
end.
  
```

Раздел операторов имеется в любой программе и является основным. Предшествующие разделы носят характер описаний и не все обязательно присутствуют в каждой программе.

В Турбо Паскале, в отличие от базового стандарта Паскаля, возможно:

- отсутствие заголовка программы;
- разделы **Const**, **Type**, **Var**, **Label** могут следовать друг за другом в любом порядке и повторяться в разделе описаний сколько угодно раз.



### 2.2.2. Элементы языка и типы данных

**Алфавит.** Алфавит языка состоит из множества символов, включающих в себя *буквы, цифры, специальные символы.*

*Латинские буквы:* от *A* до *Z* (заглавные) и от *a* до *z* (строчные).

*Цифры:* 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

*Специальные символы:* + — \* / = < > [ ] . , ( ) : ; { } ^ @ \$ #.

Следующие комбинации специальных символов являются единичными символами (их нельзя разделять пробелами):

:= знак присваивания;

<= меньше или равно;

>= больше или равно;

(\* \*) ограничители комментариев (наряду с { });

<> не равно;

(. .) эквивалент [ ].

*Пробельные символы* — символ пробела (код ASCII 32) и все управляющие символы кода ASCII (от 0 до 31).

**Служебные слова.** К спецсимволам относятся и служебные слова, смысл которых определен однозначно. Служебные слова не могут быть использованы для других целей. С точки зрения языка, они являются единичными элементами алфавита. Вот некоторые служебные слова: **Program, Var, array, if, do, while** и др.

**Идентификаторы.** Идентификатором называется символическое имя определенного программного объекта. Такими объектами являются: имена констант, переменных, типов данных, процедур и функций, программ. *Идентификатор — это любая последовательность букв и цифр, начинающаяся с буквы.* К буквам приравнивается также знак подчеркивания. Длина идентификатора может быть произвольной, но значащими являются только первые 63 символа.

**Комментарии.** Следующие конструкции представляют собой комментарии и поэтому пропускаются компилятором:

{любой текст, не содержащий символ “фигурная скобка” }

(\* любой текст, не содержащий символы “звездочка”, “круглая скобка” \*)

//последующий текст до конца строки

Буквы русского алфавита употребляются только в комментариях, символьных и текстовых константах.

### Концепция типов данных в Паскале

Концепция типов данных является одной из центральных в любом языке программирования. С **типом величины** связаны три ее свойства: форма внутреннего представления, множество принимаемых значений и множество допустимых операций.

Паскаль характеризуется большим разнообразием типов данных (рис. 2.2).



Рис. 2.2. Система типов данных Паскаля

Тип данных называется *порядковым*, если он состоит из счетного количества значений, которые можно пронумеровать. Отсюда следует, что на этом множестве значений существуют понятия «следующий» и «предыдущий».

В стандартном Паскале Вирта отсутствовал строковый тип. Он был внесен в Турбо Паскаль. Кроме того, в Турбо Паскале целые и вещественные — это группы типов. В старших версиях Турбо Паскаля появился процедурный тип и тип «объект».

Каждый тип имеет свой идентификатор. В таблице 2.1 представлена информация о простых типах данных, определенных в Турбо Паскале и последующих диалектах языка. Для вещественных типов в скобках указано количество сохраняемых значащих цифр мантиссы в десятичном представлении числа.



Таблица 2.1

## Типы данных

Имя типа	Длина в байтах	Диапазон (множество) значений	Кол-во десятичных цифр в мантиссе
<b>Целочисленные типы</b>			
integer	2	-32768..32767	
byte	1	0..255	
word	2	0..65535	
shortint	1	-128..127	
longint	4	-2147483648..2147483647	
<b>Вещественные типы</b>			
real	6	$2,9 \cdot 10^{-39} - 1,7 \cdot 10^{38}$	(11-12)
single	4	$1,5 \cdot 10^{-45} - 3,4 \cdot 10^{38}$	(7-8)
double	8	$5 \cdot 10^{-324} - 1,7 \cdot 10^{308}$	(15-16)
extended	10	$3,4 \cdot 10^{-4932} - 1,1 \cdot 10^{4932}$	(19-20)
<b>Логический тип</b>			
boolean	1	true, false	
<b>Символьный тип</b>			
char	1	Все символы кода ASCII	

**Типы пользователя.** Одна из принципиальных возможностей Паскаля состоит в том, что пользователю разрешается определять свои типы данных. Типы пользователя всегда базируются на стандартных типах данных Паскаля. Для описания типов пользователя в Паскале существует раздел типов, начинающийся со служебного слова **Type**. К простым типам пользователя относятся перечислимый и интервальный типы данных.

**Перечислимый тип** задается непосредственно перечислением (списком) всех значений, которые может принимать переменная данного типа.

**Type** <имя типа> = (<список значений>)

Определенное таким образом имя типа затем используется для описания переменных. Например:

```
Type Gaz = (C, O, N, F);
      Metal = (Fe, Co, Na, Cu, Zn);
Var G1, G2, G3: Gaz;
      Met1, Met2: Metall;
      Day: (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
```

Здесь *Gaz* и *Metal* — имена перечислимых типов, которые ставятся в соответствие переменным *G1*, *G2*, *G3* и *Met1*, *Met2*. Переменной *Day* назначается перечислимый тип, которому не присвоено имени.

Значения, входящие в перечислимый тип, являются *константами*. Действия над ними подчиняются правилам, применимым к константам. Каждое значение в перечислимом типе занимает в памяти 2 байта, поэтому число значений этого типа не должно превышать 65 535.

Перечислимый тип — упорядоченное множество. Его элементы пронумерованы, начиная с 0 в порядке следования в описании.

В программе, в которой присутствует данное выше описание переменной *Day*, возможен такой фрагмент:

```
if Day = Sun then Write('Ура! Сегодня выходной!');
```

**Ограниченный тип** задается как упорядоченное ограниченное подмножество некоторого порядкового типа:

```
<константа 1>..<константа 2>
```

Порядковый номер первой константы не должен превышать номера второй константы в соответствующем базовом типе.

При выполнении программы автоматически контролируется принадлежность значений переменной ограниченного типа установленному диапазону. При выходе из диапазона исполнение программы прерывается.

### Пример

```
Type Numbers = 1..31;
      Alf = 'A'..'Z';
Var Data: Numbers;
      Bukva: Alf;
```

**Структурные типы.** Особенностью Паскаля является то, что в нем структуры данных рассматриваются как типы — структурные типы данных. Одна величина *простого типа* имеет одно значение: целое число, вещественное число, символ и пр. Одна величина *структурного типа* — это совокупность множества значений элементов структуры данного типа. Примеры: числовой массив, символьная строка и пр.



Автор Паскаля Вирт придавал большое значение разнообразию типов данных в языке программирования. В своей книге «Алгоритмы и структуры данных» он подчеркивает зависимость алгоритма решения задачи от способа организации данных в программе. Удачно выбранный способ организации данных упрощает алгоритм решения задачи.

### Система основных понятий

Базовые понятия Паскаля		
<p><b>Паскаль</b> — процедурный язык структурного программирования. Этапы развития: стандартный Паскаль (Н. Вирт), Турбо Паскаль, Object Pascal, Delphi</p>		
Состав программы на Паскале		
Элементы языка	Разделы программы	Комментарии
Алфавит, служебные слова, идентификаторы	Заголовок программы Разделы описаний Раздел операторов	Поясняющие тексты, которые пропускаются компилятором
Типы данных		
Стандартные (целочисленные, вещественные, логический, символьный)	Определяемые в программе: перечислимые, ограниченные	
Простые типы	Структурные типы	
Одна величина — одно значение	Одна величина — множество значений (массивы, строки, записи, множества, файлы)	

### Вопросы и задания

1. В какой парадигме программирования реализован язык Паскаль?
2. Входят ли в алфавит Паскаля русские буквы? Для чего их можно использовать?
3. Что такое идентификатор? Каковы правила задания идентификаторов?
4. Чем различаются разные типы данных из группы целочисленных типов?
5. Чем различаются разные типы данных из группы вещественных типов?
6. В чем разница между простыми и структурными типами?
7. Что такое перечислимый и ограниченный типы данных?

### 2.2.3. Операции, функции, выражения

#### Арифметические операции

К числовым типам данных относятся группы вещественных и целочисленных типов. К ним применимы арифметические операции и операции отношений.

Операции над данными бывают унарные (применимые к одному операнду) и бинарные (применимые к двум операндам). **Унарная арифметическая операция** в Паскале одна. Это операция изменения знака. Ее формат:

-<величина>.

**Бинарные арифметические операции** стандартного Паскаля описаны в табл. 2.2. В ней символ I обозначает целые типы, символ R — вещественные типы.

Таблица 2.2

Бинарные операции Паскаля

Знак	Выражение	Типы операндов	Тип результата	Операция
+	A+B	R, R	R	Сложение
		I, I	I	
		I, R; R, I	R	
-	A-B	R, R	R	Вычитание
		I, I	I	
		I, R; R, I	R	
*	A*B	R, R	R	Умножение
		I, I	I	
		I, R; R, I	R	
/	A/B	R, R	R	Вещественное деление
		I, I	R	
		I, R; R, I	R	
div	A <b>div</b> B	I, I	I	Целочисленное деление
mod	A <b>mod</b> B	I, I	I	Остаток от целочисленного деления



### Стандартные функции и процедуры

В Паскале существует большое количество стандартных функций и процедур, к которым программист может обращаться в своих программах. Наиболее часто используются математические функции, например:  $\text{sqrt}(x)$  — корень квадратный,  $\text{abs}(x)$  — абсолютная величина,  $\text{sin}(x)$  и др. Часто используемые стандартные процедуры:  $\text{Read}(\dots)$  — процедура ввода,  $\text{Write}(\dots)$  — процедура вывода данных.

Стандартные функции и процедуры являются внешними подпрограммами по отношению к вызывающей их программе. Они объединены в модули, которые подключаются к основной программе и становятся доступными для использования. Наиболее часто используемые подпрограммы объединены в модуль под названием SYSTEM. Этот модуль подключается к программе автоматически.

Таблица 2.3 содержит описания стандартных математических функций Паскаля.

Таблица 2.3

Стандартные математические функции Паскаля

Обращение	Тип аргумента	Тип результата	Функция
Pi	—	R	Число $\pi = 3,1415926536E+00$
$\text{abs}(x)$	I, R	I, R	Модуль аргумента
$\text{arctan}(x)$	I, R	R	Арктангенс (в радианах)
$\text{cos}(x)$	I, R	R	Косинус (в радианах)
$\text{exp}(x)$	I, R	R	$e^x$ — экспонента
$\text{frac}(x)$	I, R	R	Дробная часть $x$
$\text{int}(x)$	I, R	R	Целая часть $x$
$\text{ln}(x)$	I, R	R	Натуральный логарифм
random	—	R	Псевдослучайное число в полуинтервале $[0, 1)$
$\text{random}(x)$	I	I	Псевдослучайное число в полуинтервале $[0, x)$
$\text{round}(x)$	R	I	Округление до ближайшего целого

Окончание таблицы 2.3

Обращение	Тип аргумента	Тип результата	Функция
sin(x)	I, R	R	Синус (в радианах)
sqr(x)	I, R	I, R	Квадрат $x$
sqrt(x)	I, R	R	Корень квадратный
trunc(x)	R	I	Ближайшее целое, не превышающее $x$ по модулю

Для подключения других модулей необходимо в начале программы (после заголовка) записать строку:

**Uses** <имя модуля>

Для управления символьным выводом на экран используется стандартный модуль CRT. К программе он подключается командой:

**Uses** CRT

В дальнейшем из этого модуля мы будем часто использовать *процедуру очистки экрана* для символьного вывода, обращение к которой производится оператором ClrScr.

### Арифметические выражения

Арифметическое выражение задает порядок выполнения действий над числовыми величинами. Арифметические выражения содержат числовые константы и переменные, арифметические операции, функции, круглые скобки. Одна константа или одна переменная — простейшая форма арифметического выражения.

Например, рассмотрим математическое выражение:

$$\frac{2a + \sqrt{0,5 \sin(x+y)}}{0,2c - \ln(x-y)}$$

На Паскале оно выглядит так:

`(2*A+Sqrt(0.5*sin(X+Y)))/(0.2*C-Ln(X-Y))`

Для того чтобы правильно записывать арифметические выражения, нужно соблюдать следующие правила.

1. Все символы пишутся в строчку на одном уровне. Проставляются все знаки операций (нельзя пропускать знак \*).
2. Не допускаются два следующих подряд знака операций. (Нельзя: A+-B; можно: A+(-B).)



3. Операции с более высоким приоритетом выполняются раньше операций с меньшим приоритетом. Порядок убывания приоритетов:  
 вычисление функции;  
 унарная операция смены знака (-);  
 \*, /, div, mod;  
 +, -.
4. Несколько записанных подряд операций одинакового приоритета выполняются последовательно слева направо.
5. Часть выражения, заключенная в скобки, вычисляется в первую очередь. (Например, в выражении  $(A+B) * (C-D)$  умножение производится после сложения и вычитания.)

Не следует записывать выражения, не имеющие математического смысла, например: деление на нуль, логарифм отрицательного числа и т. п.

**Пример.** Цифрами сверху указан порядок выполнения операций:

1 7 4 5 3      6 2 12 11 10 8      9  
 $(1+y) * (2*x + \text{sqrt}(y) - (x+y)) / (y+1 / (\text{sqrt}(x) - 4))$

Данное арифметическое выражение (на Паскале) соответствует следующему математическому выражению:

$$(1+y) \frac{2x + \sqrt{y} - (x+y)}{y + \frac{1}{x^2 - 4}}$$

В Паскале нет операции или стандартной функции возведения числа в произвольную степень. Для вычисления  $x^y$  рекомендуется поступать следующим образом:

- а) если  $y$  — целое положительное значение, то его степень вычисляется через умножение; например,  $x^3 \rightarrow x \cdot x \cdot x$ ; большие степени следует вычислять умножением в цикле;
- б) если  $y$  — целое отрицательное число, то степень вычисляется так:  $x^y = \left(\frac{1}{x}\right)^{|y|}$ , а при  $y = 0$   $x^0 = 1$ ;
- в) если  $y$  — вещественное, не равное нулю значение, то используется следующая математическая формула:  $x^y = e^{y \ln x}$ . На Паскале получим арифметическое выражение:

$\text{Exp}(y * \text{Ln}(x))$

Очевидно, что в этом случае не допускается нулевое или отрицательное значение  $x$ . Для целого  $y$  такого ограничения нет.

**Пример**

$$\sqrt[3]{a+1} = (a+1)^{\frac{1}{3}}$$

На Паскале это выражение выглядит так:

```
Exp(1/3 * Ln(a + 1))
```

Выражение имеет целочисленный тип, если в результате его вычисления получается величина целочисленного типа. Выражение имеет вещественный тип, если результатом его вычисления является вещественная величина.

**Логические выражения**

С логическими величинами и выражениями вам уже приходилось иметь дело при работе с базами данных, с электронными таблицами, в разделе «Логические основы обработки информации» учебника для 10 класса. Кратко сформулируем основные правила записи логических выражений на Паскале.

**Логическое выражение** есть логическая формула, записанная на языке программирования. Логическое выражение состоит из логических операндов, связанных логическими операциями и круглыми скобками. Результатом вычисления логического выражения является булевская величина (*false* или *true*). Логическими операндами могут быть логические константы, переменные, функции, операции отношения. Один отдельный логический операнд является простейшей формой логического выражения.

**Логические константы:** *true*, *false*.

**Логические переменные:** описываются с типом *boolean*.

**Операции отношения:** осуществляют сравнение двух операндов и определяют, истинно или ложно соответствующее отношение между ними.

**Знаки операций отношения:** = (равно), <> (не равно), > (больше), < (меньше), >= (больше или равно), <= (меньше или равно).

**Логические операции:** **not** — отрицание; **and** — логическое умножение (конъюнкция); **or** — логическое сложение (дизъюнкция), **xor** — «исключающее ИЛИ». Таблица истинности для этих операций выглядит так:

A	B	not A	A and B	A or B	A xor B
false	false	true	false	false	false
false	true	true	false	true	true
true	false	false	false	true	true
true	true	false	true	true	false



Логические операции располагаются в следующем порядке по убыванию старшинства (приоритета): 1) **not**, 2) **and**, 3) **or**, **xor**. Операции отношения имеют самый низкий приоритет. Поэтому если операндами логической операции являются отношения, то их следует заключать в круглые скобки. Например, математическому неравенству  $1 \leq x \leq 50$  соответствует следующее логическое выражение:

$(1 \leq x) \text{ and } (x \leq 50)$

**Логическая функция**  $\text{odd}(x)$  принимает значение *true*, если значение целого аргумента  $x$  является нечетным, иначе — *false*.

### Система основных понятий

Операции, функции, выражения
<b>Арифметические операции</b> — применяются к числовым величинам; нет операции возведения в степень
<b>Тип результата операции</b> зависит от типа операндов (см. табл. 2.2)
<b>Возведение в степень</b> ( $x^y$ ): при целом $y$ реализуется через умножение, при вещественном $y$ реализуется так: $\text{Exp}(y * \text{Ln}(x))$
<b>Стандартные функции и процедуры</b> описаны в модулях, подключаемых к программе. Модуль <code>SYSTEM</code> подключается по умолчанию
<b>Арифметическое выражение:</b> языковая конструкция, определяющая порядок вычисления числовой величины в соответствии с математическим выражением. Типом выражения называется тип результата его вычисления
<b>Логическое выражение:</b> логическая формула, записанная на языке программирования. Логическое выражение состоит из логических операндов, связанных логическими операциями и круглыми скобками

### Вопросы и задания

1. Для следующих математических выражений запишите соответствующие арифметические выражения на Паскале.

а)  $a + bx + cyz$ ;    б)  $[(ax - b)x + c]x - d$ ;    в)  $\frac{a+b}{c} + \frac{c}{ab}$ ;

г)  $\frac{x+y}{a_1} \cdot \frac{a_2}{x-y}$ ;    д)  $10^4\alpha + 3\frac{1}{5}\beta$ ;    е)  $\left(1 + \frac{x}{2!} + \frac{y}{3!}\right) / \left(1 + \frac{2}{3+xy}\right)$ .

2. Запишите математические выражения, соответствующие следующим выражениям на Паскале.

- а)  $(p+q)/(r+s) - p*q/(r*s)$   
 б)  $1E3 + beta/(x-gamma*delta)$   
 в)  $a/b*(c+d) - (a-b)/b/c + 1E-8$

3. Для следующих математических выражений запишите соответствующие арифметические выражения на Паскале.

- а)  $(1+x)^2$ ; б)  $\sqrt{1+x^2}$ ; в)  $\cos^2 x^2$ ; г)  $\log_2 \frac{x}{5}$ ; д)  $\arcsin x$ ; е)  $\frac{e^x + e^x}{2}$ ;  
 ж)  $x^{\sqrt{2}}$ ; з)  $\sqrt[3]{1+x}$ ; и)  $\sqrt{x^8 + 8^x}$ ; к)  $\frac{xyz - 3,3|x + \sqrt[4]{y}|}{10^7 + \ln 4!}$ ; л)  $\frac{\beta + \sin^2 \pi^4}{\cos 2 + |\operatorname{ctg} y|}$ .

4. Вычислите значения выражений.

- а) `trunc(6.9)`      е) `round(6.2)`  
 б) `trunc(6.2)`      ж) `20 mod 6`  
 в) `20 div 6`      з) `2 mod 5`  
 г) `2 div 5`      и) `3*7 div 2 mod 7/3 - trunc(sin(1))`  
 д) `round(6.9)`

5. Определите тип выражения.

- а) `1+0.0`    в) `sqr(4)`      д) `sin(0)`  
 б) `20/4`    г) `sqrt(16)`    е) `trunc(-3.14)`

6. Вычислите значения логических выражений.

- а)  $K \bmod 7 = K \operatorname{div} 5 - 1$  при  $K=15$ ;  
 б) `odd(trunc(10*P))` при  $P = 0,182$ ;  
 в) `not odd(n)` при  $n = 0$ ;  
 г) `t and (P mod 3 = 0)` при  $t = \text{true}$ ,  $P = 10101$ ;  
 д)  $(x*y <> 0) \text{ and } (y > x)$  при  $x = 2$ ,  $y = 1$ ;  
 е) `a or not b` при  $a = \text{False}$ ,  $b = \text{True}$ .

## 2.2.4. Оператор присваивания, ввод и вывод данных

**Присваивание** — это действие, в результате которого переменная величина получает определенное значение. В программе на Паскале существуют три способа присваивания значения переменной:

- 1) оператор присваивания;
- 2) оператор ввода;
- 3) передача значения через параметры подпрограммы.

**Оператор присваивания** имеет следующий формат:

`<переменная> := <выражение>`

Например:

- 1) `x := 2*a + sqrt(b);`
- 2) `b := (x > y) and (k <> 0);`



Сначала вычисляется выражение, затем полученное значение присваивается переменной. В первом примере приведен **арифметический оператор присваивания**. Здесь  $x$  — переменная вещественного типа. Во втором примере — **логический оператор присваивания**. Здесь  $b$  — переменная типа `boolean`.

Типы переменной и выражения должны совпадать. Из этого правила есть одно исключение: переменной вещественного типа можно присваивать значение целочисленного выражения. В таком случае значение целого числа преобразуется к формату с плавающей точкой и присвоится вещественной переменной.

### Ввод и вывод данных

Под вводом понимается передача данных с внешнего устройства компьютера в оперативную память. При выводе данные передаются из оперативной памяти на внешнее устройство (рис. 2.3).

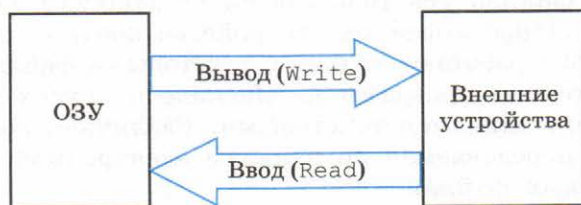


Рис. 2.3. Структура процедурного ЯПВУ

Операцию ввода называют чтением и выполняют с помощью оператора `Read`. Вывод называют записью, и для его выполнения используется оператор `Write`.

К внешним устройствам относятся устройства ввода и вывода (клавиатура, монитор, принтер и др.) и устройства внешней памяти (магнитные и оптические диски, флеш-память и др.). Данные на внешних устройствах организованы в виде файлов.

Для внешних запоминающих устройств (ВЗУ) **файл** — это *поименованная область памяти* этого устройства. В файлы на ВЗУ можно записывать данные по команде `Write` и можно читать данные из файлов по команде `Read`. На одном устройстве ВЗУ может храниться множество файлов одновременно. Правила именования файлов на ВЗУ определяются операционной системой. Имена для файлов, создаваемых пользователем, задает сам пользователь.

Устройства ввода с клавиатуры и вывода на экран монитора являются однофайловыми. Считается, что с клавиатурой связан один системный файл с именем `INPUT`, поэтому ввод с клавиатуры равнозначен чтению из файла `INPUT`. С монитором связан

системный файл, который называется OUTPUT. Вывод на экран — это запись данных в файл OUTPUT.

По форме хранения данных файлы бывают *типизированными*, *нетипизированными* и *текстовыми*. В типизированных и нетипизированных файлах данные различных типов хранятся в том же формате, что и в оперативной памяти. Следовательно, при чтении и записи в такие файлы данные копируются без изменения объема и формы представления. В текстовых файлах данные хранятся в символьном формате, поэтому при вводе (чтении) чисел из текстового файла происходит преобразование их представления из символьной формы к форме их внутреннего представления (с фиксированной или плавающей точкой). А при выводе (записи) чисел в текстовый файл они преобразуются из внутренней формы в символьную.

Далее мы будем использовать только текстовые файлы.

**Текстовые файлы.** Текстовый файл — наиболее часто употребляемая разновидность файлов. Устройства ввода с клавиатуры и вывода на экран работают только с текстовыми файлами. Файлы, содержащие тексты программ на Паскале и других языках программирования, являются текстовыми. Различная документация, информация, передаваемая по каналам электронной связи, записана в текстовых файлах.

Содержимое текстового файла представляет собой символьную последовательность, разделенную на строки. Каждая строка заканчивается специальным признаком EOLN (end of line — конец строки). Весь файл заканчивается признаком EOF (end of file — конец файла). Схематически это выглядит так:

$S_1$	$S_2$	...	$S_{k1}$	EOLN	$S_1$	$S_2$	...	$S_{k2}$	EOLN	...	EOF
-------	-------	-----	----------	------	-------	-------	-----	----------	------	-----	-----

Здесь  $S_i$  обозначает  $i$ -й символ в строке. Каждый символ представлен во внутреннем коде (ASCII) и занимает 1 байт. Признак EOLN состоит из двух однобайтовых управляющих кодов: CR (код ASCII 13) — возврат к началу строки и LF (код ASCII 10) — перевод строки. При выводе содержимого текстового файла на экран или на печать признак EOLN обеспечивает визуальное разделение строк: переход к продолжению вывода с начала новой строки.

Текстовый файл можно создать или преобразовать с помощью текстового редактора. Файл можно просмотреть на экране монитора или распечатать на принтере.

**Ввод с клавиатуры** производится путем обращения к стандартной процедуре Read в следующем формате:

Read(<список ввода>)



Чтение происходит из системного файла INPUT, всегда доступного для любой программы. Элементами списка ввода могут быть переменные символьного типа, числовых типов и строковые переменные.

### Пример

```
Read(a, b, c, d)
```

При выполнении этого оператора происходит прерывание исполнения программы, после чего пользователь должен набрать на клавиатуре значения переменных *a*, *b*, *c*, *d*, отделяя их друг от друга пробелами или нажатием клавиши Enter. При этом вводимые значения высвечиваются на экране. Значения следует вводить в строгом соответствии с синтаксисом Паскаля.

### Пример

```
Var T: Real; J: integer; K: char;
```

```
begin
```

```
  Read(T, J, K);
```

Набираем на клавиатуре:

```
253.98 100 G [Enter]
```

Если в программе имеются несколько подряд идущих операторов Read, то каждый следующий оператор продолжает считывать данные из той же строки, что и предыдущий (если строка не закончилась). Поэтому данные для них можно вводить последовательно (на экране они отображаются в одной строке) и лишь в конце ввода нажать клавишу Enter.

### Пример

```
Var A, B: Integer;
```

```
  C, D: Real;
```

```
begin
```

```
  Read(A, B); Read(C, D);
```

Набираем на клавиатуре и видим на экране:

```
18758 34 2.62E-02 1.54E+01 [Enter].
```

Другой вариант оператора ввода с клавиатуры имеет вид:

```
ReadLn(<список ввода>)
```

Здесь слово ReadLn означает *read line* — читать строку. Нажатие клавиши Enter в процессе ввода вырабатывает признак EOLN — конец строки, и данные при выполнении следующего оператора ввода будут отображаться на экране с начала новой строки. Если в предыдущем примере заменить операторы Read на ReadLn:

```
ReadLn(A, B); ReadLn(C, D);
```

то ввод значений будет происходить из двух строк, отображенных на экране:

```
18758 34 [Enter]
2.62E-02 1.54E+01 [Enter]
```

Если в строке ввода содержится значений больше, чем количество переменных в списке ввода, то лишние значения игнорируются. Обратная ситуация приводит к ошибке ввода из-за нехватки данных.

### Ввод из файла на диске

Исходные данные могут быть заранее подготовлены с помощью текстового редактора и сохранены в файле на диске под определенным именем. Ввод исходных данных из файла производится автоматически, и при этом не происходит задержки выполнения программы, которая есть при клавиатурном вводе.

Для организации ввода данных из текстового файла следует:

- 1) объявить в программе переменную с типом `text` (она называется **файловой переменной**);
- 2) связать файловую переменную с файлом на ВЗУ, содержащим исходные данные, с помощью оператора `Assign`;
- 3) открыть файл для чтения с помощью процедуры `Reset`;
- 4) осуществлять чтение из файла с помощью операторов `Read` или `Readln`;
- 5) закрыть файл с помощью оператора `Close`.

**Пример.** В текстовом файле с именем `abc.txt` хранятся пять чисел, разделенных на две строки:

```
2.5 3.1 4.0
0.7 1.5
```

В следующей программе организован ввод этих данных в вещественные переменные `a`, `b`, `c`, `d`, `e`:

```
Var a, b, c, d, e: real;
    FD: text; {Описание файловой переменной}
begin
  Assign(FD, 'abc.txt'); {переменная FD связывается с файлом
                        abc.txt}
  Reset(FD); {файл открывается для чтения с его начала}
  Readln(FD, a, b, c); {чтение первой строки файла}
  Readln(FD, d, e); {чтение второй строки файла}
  Close(FD); {разрывается связь переменной FD с файлом}
```

Здесь `FD` — файловая переменная. `Assign`, `Reset`, `Readln`, `Close` — операторы обращения к стандартным процедурам, имеющим следующие форматы:



```
Assign(<файловая переменная>, <имя файла>);  
Reset(<файловая переменная>);  
Readln(<файловая переменная>, <список ввода>);  
Close(<файловая переменная>);
```

Если файл хранится не в текущем каталоге, то в операторе `Assign` кроме имени файла надо указывать полный путь к нему. Имя файла можно задавать в строковой константе или переменной.

**Вывод на экран** производится по оператору обращения к стандартной процедуре:

```
Write(<список вывода>)
```

Здесь элементами списка вывода могут быть выражения различных типов (в частности, константы и переменные).

### Пример

```
Write('Сумма ', A, '+', B, '=', A+B)
```

Если, например,  $A = 5$  и  $B = 7$ , то на экране получим:

```
Сумма 5+7=12
```

При выводе на экран нескольких значений в строку они не отделяются друг от друга пробелами. Программист сам должен позаботиться о таком разделении. В приведенном примере предусмотрен пробел после слова «Сумма».

Второй вариант процедуры вывода на экран:

```
Writeln(<список вывода>)
```

`Writeln` означает *write line* — писать строку. Действие оператора `Writeln` отличается от действия оператора `Write` тем, что после вывода последнего в списке значения происходит перевод курсора к началу следующей строки. Оператор `Writeln`, записанный без параметров, вызывает перевод строки.

В списке вывода могут присутствовать указатели форматов вывода (форматы). Формат определяет представление выводимого значения на экране. Формат отделяется от соответствующего ему элемента двоеточием. Если указатель формата отсутствует, то компьютер выводит значение по определенному правилу, предусмотренному «по умолчанию».

### **Вывод в текстовый файл**

Запись результатов выполнения программы в текстовый файл позволяет их сохранить для того, чтобы в дальнейшем можно было их просмотреть с помощью текстового редактора, распечатать на принтере, а также использовать в качестве исходных данных для другой программы.

Для организации вывода данных в текстовый файл следует:

- 1) объявить в программе файловую переменную с типом `text` ;
- 2) связать файловую переменную с файлом на ВЗУ с помощью оператора `Assign`;
- 3) открыть файл для записи с помощью процедуры `Rewrite`;
- 4) осуществлять запись в файл с помощью операторов `Write` или `Writeln`;
- 5) закрыть файл с помощью оператора `Close`.

### Пример

Требуется записать в текстовый файл таблицу умножения на 2.

```
var A: integer;
    TM: text; {Описание файловой переменной}
begin
  Assign(TM, 'E:\TabMul.txt'); {Связывание переменной TM}
                               {с файлом}
  Rewrite(TM); {Открытие файла для записи}
  {Циклический вывод в файл таблицы умножения на 2}
  for A:=2 to 9 do Writeln(TM, 2, '*', A, '=', 2*A);
  Close(TM) {Закрытие файла}
end.
```

Процедуры открытия файла для записи и запись в файл имеют следующий формат:

```
Rewrite(<файловая переменная>);
Write(<файловая переменная>, <список вывода>);
Writeln(<файловая переменная>, <список вывода>);
```

Если файла с именем, указанным в операторе `Assign`, на диске не было, то программа его создаст. Если такой файл уже был, то его прежнее содержание будет утеряно и запишутся новые данные. В конце выполнения оператора `Writeln` выставляется признак `EOLN`. Оператор `Write` этого не делает. Закрытие файла приводит к выставлению признака `EOF`.

В результате выполнения программы в корневом каталоге диска `E:` появится файл с именем `TabMul.txt`. Открыв его в текстовом редакторе, увидим:

```
2*2=4
2*3=6
2*4=8
2*5=10
2*6=12
2*7=14
2*8=16
2*9=18
```



## Система основных понятий



## Присваивание, ввод, вывод

**Оператор присваивания:** `<переменная>:=<выражение>`

Типы переменной и выражения должны совпадать. Исключение: вещественной переменной можно присваивать значение целого выражения

**Ввод** — передача данных с внешнего устройства в ОЗУ*Ввод с клавиатуры:*`Read(<список ввода>)` или  
`Readln(<список ввода>)`*Ввод из файла:*`Read(<ФП>, <список ввода>)` или  
`Readln(<ФП>, <список ввода>)`.  
ФП — файловая переменная**Вывод** — передача данных из ОЗУ на внешнее устройство*Вывод на экран:*`Write(<список вывода>)` или  
`Writeln(<список вывода>)`*Вывод в файл:*`Write(<ФП>, <список вывода>)` или  
`Writeln(<ФП>, <список вывода>)`.  
ФП — файловая переменная**Операторы (стандартные процедуры) работы с файлами**`Assign` — назначение связи между файловой переменной и файлом на внешнем устройстве;`Reset` — открытие файла для чтения;`Rewrite` — открытие файла для записи;`Close` — закрытие файла (разрыв связи с файловой переменной)

## Вопросы и задания



1. Назовите последовательность действий при выполнении оператора присваивания.
2. Сформулируйте правило соответствия типов для оператора присваивания. Какое существует исключение из этого правила?
3. Если  $y$  — вещественная переменная, а  $n$  — целая, то какие из следующих операторов присваивания правильные, а какие — нет?
 

а) $y:=n+1;$	б) $n:=y-1;$
в) $n:=4.0;$	г) $y:=\text{trunc}(y);$
д) $y:=n \text{ div } 2;$	е) $y:=y \text{ div } 2;$
ж) $n:=n/2;$	з) $n:=\text{sqr}(\text{sqrt}(n));$
4. Напишите последовательность операторов присваивания, в результате выполнения которой целые переменные  $x$  и  $y$  обменяются значениями. При этом нельзя использовать дополнительные переменные. Найдя такой алгоритм, определите, в чем его недостаток по сравнению с методом обмена через третью переменную. Можно ли его применять для вещественных чисел?
5. Напишите оператор присваивания, в результате выполнения которого целой переменной  $h$  присвоится значение цифры, стоящей в раз-

ряде сотен в записи положительного целого числа  $k$  (например, если  $k = 28796$ , то  $h = 7$ ).

6. Напишите оператор присваивания, в результате выполнения которого целой переменной  $S$  присвоится значение суммы цифр трехзначного целого числа  $k$ .
7. Напишите программу, по которой из текстового файла с именем kvur.txt будут прочитаны три числа:  $a$ ,  $b$ ,  $c$  — коэффициенты квадратного уравнения, затем будут вычислены корни этого уравнения и выведены на экран и в текстовый файл korni.txt.

Практикум. Раздел «Программирование»

## 2.2.5. Структуры алгоритмов и программ

### Базовые алгоритмические структуры

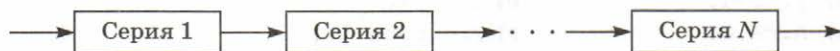
В основе структурного программирования лежит теорема, доказанная Эдсгером Дейкстрой в 1969 году. Суть ее в том, что *алгоритм для решения любой логической задачи можно составить только из структур СЛЕДОВАНИЕ, ВЕТВЛЕНИЕ, ЦИКЛ*. Их называют **базовыми алгоритмическими структурами**.

В параграфе 1.7.1 учебника для 10 класса уже рассказывалось о базовых структурах. По сути дела, мы и раньше во всех рассматриваемых примерах программ придерживались принципов структурного программирования. Еще раз покажем, как изображаются базовые структуры в схемах алгоритмов и как они программируются на Паскале.



Эдсгер В. Дейкстра  
(1930–2002)

**Следование** — это линейная последовательность действий:

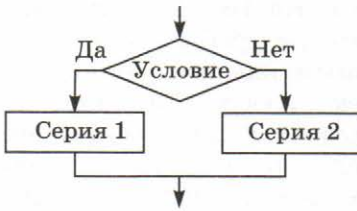


В программе на Паскале серия — это либо один отдельный оператор, либо **составной оператор**: последовательность операторов, заключенная в операторные скобки. Операторными скобками называются служебные слова **begin** и **end**.

**Ветвление** — алгоритмическая альтернатива. Управление передается одному из двух блоков в зависимости от истинности или ложности условия. Затем происходит выход на общее продолже-



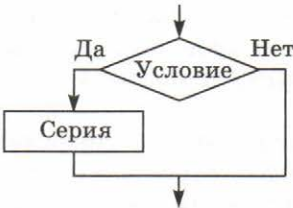
ние. Вот как изображается ветвление на блок-схеме и программируется на Паскале с помощью условного оператора:



```

if <логическое выражение>
then <серия 1>
else <серия 2>;
  
```

Неполная форма ветвления имеет место, когда на ветви «нет» пусто:

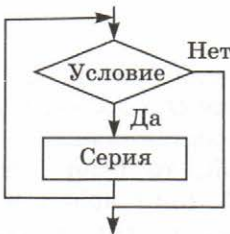


```

if <логическое выражение>
then <серия>;
  
```

В конструкциях на Паскале операторы на ветвях могут быть как простыми, так и составными.

Цикл — повторение некоторой группы действий по условию. Различаются два типа цикла. Первый — цикл с предусловием (цикл-пока):

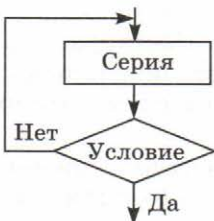


```

while <логическое выражение> do
  <серия>;
  
```

Пока условие истинно, выполняется серия, образующая тело цикла.

Второй тип циклической структуры — цикл с постусловием (цикл-до):



```

repeat
  <серия>
until <логическое выражение>
  
```

Здесь тело цикла предшествует условию цикла. Тело цикла повторяет свое выполнение, если условие ложно. Повторение закончится, когда условие станет истинным. На Паскале для тела цикла с постусловием операторных скобок не требуется.

Теоретически необходимым и достаточным является лишь первый тип цикла — цикл с предусловием. Любой циклический алгоритм можно построить с его помощью. Это более общий вариант цикла, чем цикл-до. В самом деле, тело цикла-до хотя бы один раз обязательно выполнится, так как проверка условия происходит после завершения его выполнения. А для цикла-пока возможен такой вариант, когда тело цикла не выполнится ни разу, поэтому в любом языке программирования можно было бы ограничиться только циклом-пока. Однако в ряде случаев применение цикла-до оказывается более удобным, и поэтому он используется.

Иногда в литературе структурное программирование называют «программированием без `goto`» — *оператора безусловного перехода*. Действительно, при таком подходе нет места безусловному переходу. Неоправданное использование в программах оператора `goto` лишает ее структурности, а значит, всех связанных с этим положительных свойств: «прозрачности» и надежности алгоритма. Хотя во всех процедурных языках программирования этот оператор присутствует, однако, с точки зрения структурного подхода, его употребления следует избегать.

### **Комбинации базовых структур**

Сложный алгоритм состоит из соединенных между собой базовых структур. Соединяться эти структуры могут двумя способами: *последовательным* и *вложенным*. Если блок, составляющий тело цикла, сам является циклической структурой, то имеют место вложенные циклы. В свою очередь внутренний цикл может иметь внутри себя еще один цикл и т. д. В связи с этим вводится представление о *глубине вложенности циклов*. Точно так же и ветвления могут быть вложенными друг в друга.

Структурный подход требует соблюдения стандарта в изображении блок-схем алгоритмов. Чертить их нужно так, как это делалось во всех приведенных примерах. Каждая базовая структура должна иметь один вход и один выход. Нестандартно изображенная блок-схема плохо читается, теряется наглядность алгоритма. Несколько примеров структурных блок-схем алгоритмов приведено на рис. 2.4.

Такие блок-схемы легко читаются. Их структура хорошо воспринимается зрительно. Структуре каждого алгоритма можно дать название.



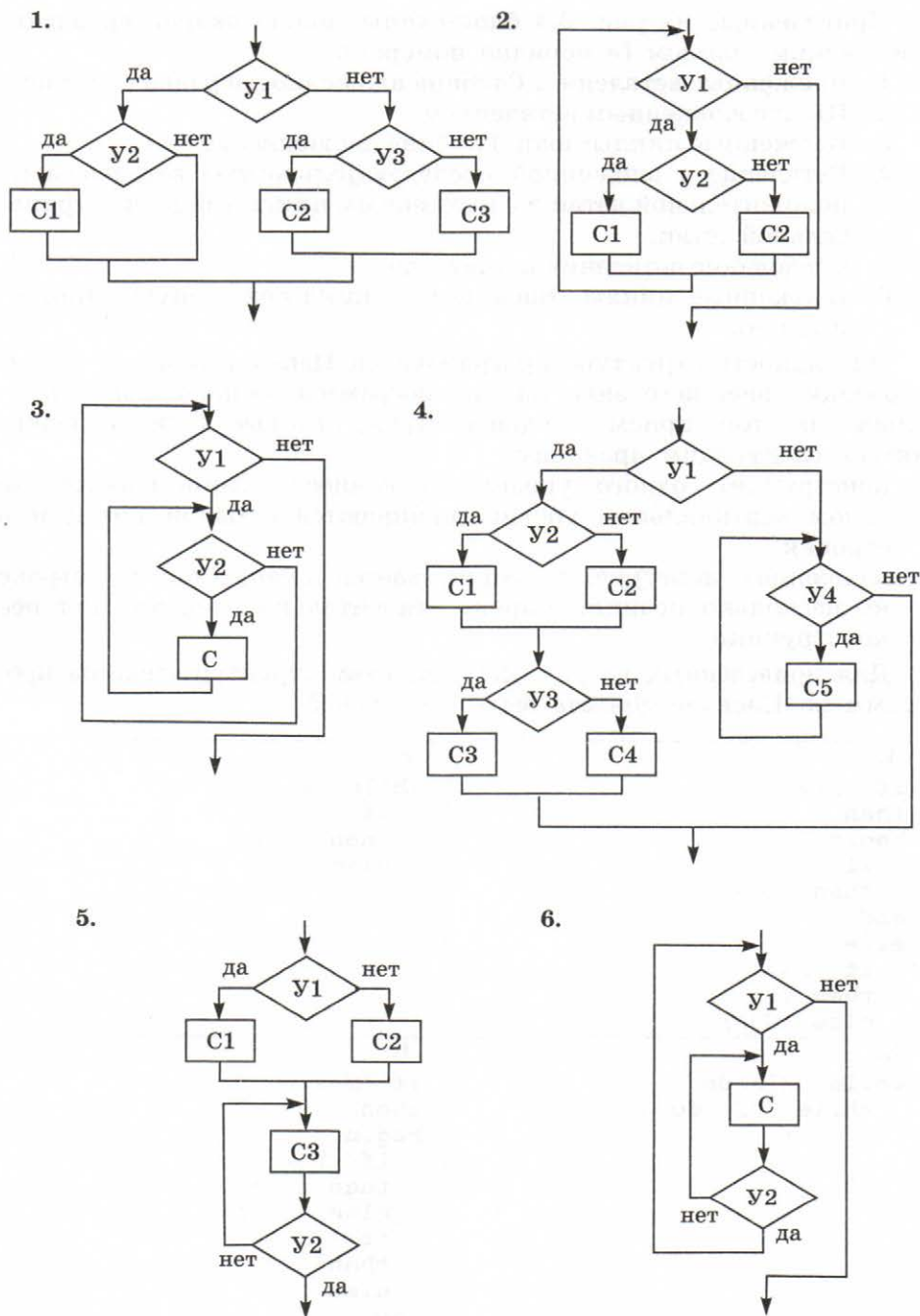


Рис. 2.4. Структурные схемы алгоритмов

Приведенные на рис. 2.4 блок-схемы можно охарактеризовать следующим образом (в порядке номеров):

1. Вложенные ветвления. Глубина вложенности равна единице.
2. Цикл с вложенным ветвлением.
3. Вложенные циклы-пока. Глубина вложенности — 1.
4. Ветвление с вложенной последовательностью ветвлений на положительной ветви и с вложенным циклом-пока на отрицательной ветви.
5. Следование ветвления и цикла-до.
6. Вложенные циклы. Внешний — цикл-пока, внутренний — цикл-до.

Наглядность структуре программы на Паскале придает структуризация внешнего вида текста программы. Основной используемый для этого прием — сдвиги строк, которые должны подчиняться следующим правилам:

- конструкции одного уровня вложенности записываются на одном вертикальном уровне (начинаются с одной позиции в строке);
- вложенная конструкция записывается смещенной по строке на несколько позиций вправо относительно внешней для нее конструкции.

Для приведенных на рис. 2.4 блок-схем структура текстов программ на Паскале должна быть следующей.

<pre> 1. if &lt;V1&gt; then begin   if &lt;V2&gt;   then &lt;C1&gt; end else   if &lt;V3&gt;   then &lt;C2&gt;   else &lt;C3&gt;; </pre>	<pre> 2. while &lt;V1&gt; do   if &lt;V2&gt;   then &lt;C1&gt;   else &lt;C2&gt;; </pre>
<pre> 3. while &lt;V1&gt; do   while &lt;V2&gt; do     &lt;C&gt;; </pre>	<pre> 4. if &lt;V1&gt; then begin   if &lt;V2&gt;   then &lt;C1&gt;   else &lt;C2&gt;;   if &lt;V3&gt;   then &lt;C3&gt;   else &lt;C4&gt; end else while &lt;V4&gt; do &lt;C5&gt;; </pre>



<p>5.</p> <pre> if &lt;V1&gt; then &lt;C1&gt; else &lt;C2&gt;; repeat   &lt;C3&gt; until &lt;V2&gt;; </pre>	<p>6.</p> <pre> while &lt;V1&gt; do   repeat     &lt;C1&gt;   until &lt;V2&gt;; </pre>
---	--

Структурное программирование — это не только форма описания алгоритма и программы, но еще и *способ мышления программиста*. Размышляя над алгоритмом, нужно стремиться составлять его из стандартных структур. Если использовать строительную аналогию, то структурная методика построения алгоритма подобна сборке здания из стандартных секций в отличие от складывания по кирпичику.

## Система основных понятий

Структуры алгоритмов и программ		
Базовые алгоритмические структуры		
Следование	Ветвление	Цикл
Линейная последовательность действий	Выбор одной из двух серий действий с выходом на общее продолжение	Повторение серии действий по условию. Цикл с предусловием, цикл с постусловием
<b>Комбинации базовых структур:</b> последовательность, вложенность		
<b>Структурный алгоритм (программа):</b> построенный из базовых алгоритмических структур, не содержащий команды безусловного перехода ( <code>goto</code> )		
<b>Структурирование текста программы:</b> использование сдвигов строк для вложенных конструкций		

## Вопросы и задания

1. Перечислите основные базовые алгоритмические структуры и средства их программирования в Паскале.
2. Какой алгоритм называется структурным?
3. Нарисуйте блок-схемы и напишите на Паскале два варианта программы решения задачи: выбрать из двух числовых величин большее зна-

чение. Первый вариант — с полным ветвлением, второй вариант — с неполным ветвлением.

4. Нарисуйте блок-схемы и напишите на Паскале два варианта программы решения задачи: выбрать из трех числовых величин наименьшее значение. Первый вариант — с вложенными ветвлениями, второй вариант — с последовательными ветвлениями.
5. Для данного натурального числа  $N$  требуется вычислить сумму:  $1 + 1/2 + 1/3 + \dots + 1/N$ . Постройте алгоритм и напишите два варианта программы на Паскале: с циклом-до и с циклом-пока.
6. Какую структуру будет иметь алгоритм решения следующей задачи? Дано целое положительное число  $N$ . Если  $N$  — четное, то вычислить  $N! = 1 \cdot 2 \cdot \dots \cdot N$ . Если  $N$  — нечетное, то вычислить сумму:  $1 + 2 + \dots + N$ . Составьте программу на Паскале.
7. В следующем фрагменте программы  $k$  и  $S$  — переменные целого типа:

```

Readln(k);
S:=0;
while k>0 do
begin
  S:=S+k mod 10;
  k:=k div 10
end;
Writeln(S);

```

Какая решается задача? Запрограммируйте решение этой же задачи с использованием цикла с постусловием.

### 2.2.6. Программирование ветвлений

Загляните в параграф 1.7.4 учебника для 10 класса. Там изображен структурный алгоритм решения квадратного уравнения и написана программа на Паскале, в которой соблюдены все вышеперечисленные правила структуризации текста программы. Алгоритм имеет структуру вложенных ветвлений. Рассмотрим другие задачи, которые решаются с помощью ветвящегося алгоритма.

#### Пример 1

Требуется перевести пятибалльную оценку в ее наименование: 5 — «отлично», 4 — «хорошо», 3 — «удовлетворительно», 2 — «неудовлетворительно».

Блок-схема алгоритма приведена на рис. 2.5.



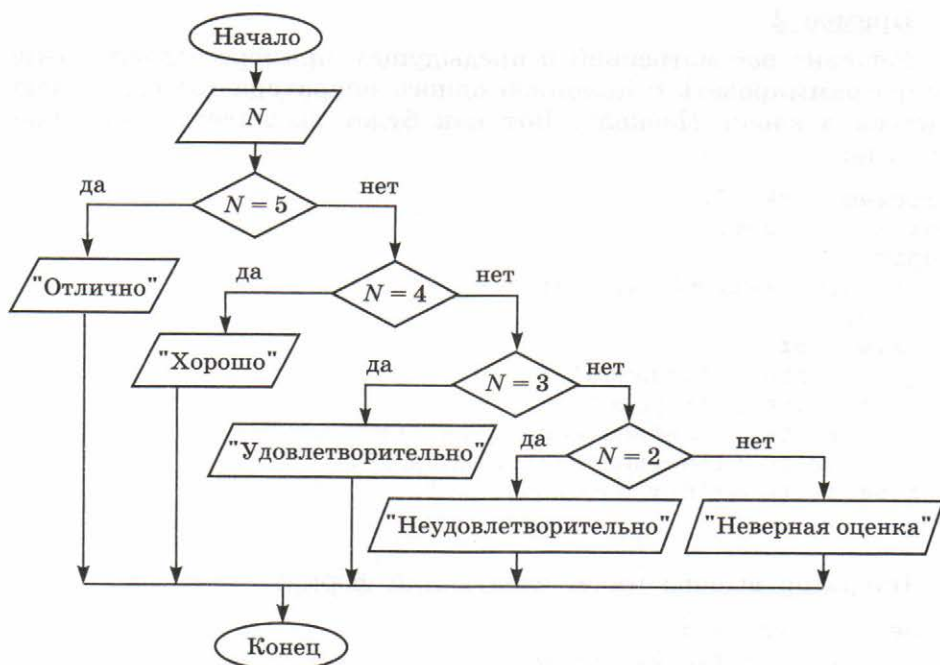


Рис. 2.5. Алгоритм перевода числовой оценки в словесную

Этот алгоритм имеет структуру вложенных ветвлений и может быть запрограммирован с использованием условного оператора **if** следующим образом:

```

Program Marks_1;
Var N: integer;
Begin
  WriteLn('Введите оценку:');
  ReadLn(N);
  if (N=5)
  then WriteLn('Отлично')
  else
    if (N=4)
    then WriteLn('Хорошо')
    else
      if (N=3)
      then WriteLn('Удовлетворительно')
      else
        if (N=2)
        then WriteLn('Неудовлетворительно')
        else WriteLn('Неверная оценка')
  end.
  
```

### Пример 2

Решение рассмотренной в предыдущем примере задачи можно запрограммировать с помощью одного **оператора выбора**, имеющегося в языке Паскаль. Вот как будет выглядеть такая программа:

```

Program Marks_2;
Var N: integer;
Begin
  WriteLn('Введите оценку:');
  ReadLn(N);
  case N of
    5: WriteLn('Отлично');
    4: WriteLn('Хорошо');
    3: WriteLn('Удовлетворительно');
    2: WriteLn('Неудовлетворительно');
  else WriteLn('Нет такой оценки')
  end;

```

**Оператор выбора** имеет следующий формат:

```

case <селектор> of
  <список констант>: <оператор>;
  ...
  <список констант>: <оператор>;
else <оператор>
end

```

Здесь <селектор> — это выражение любого *порядкового типа*; <константа> — постоянная величина того же типа, что и селектор; <список констант> — последовательность констант, разделенная запятыми; <оператор> — любой простой или составной оператор.

Выполнение оператора выбора происходит так: вычисляется выражение — селектор; затем в списках констант ищется такое значение, которое совпадает с полученным значением селектора; далее исполняется оператор, помеченный данной константой. Если такой константы не найдено, то происходит переход к выполнению оператора, следующего после слова **else**.

### Пример 3

В этом примере демонстрируется использование списка констант в операторе выбора. Программа сообщает, сдал студент экзамен или не сдал. Если оценка одна из следующих: 3, 4, 5, то экзамен сдан; если оценка 2, то не сдан.



```
case N of
  3, 4, 5: WriteLn('Экзамен сдан');
  2: WriteLn('Экзамен не сдан');
else WriteLn('Нет такой оценки');
```

Так же как условный оператор, оператор выбора может использоваться в неполной форме, т. е. без ветви **Else**.

Если применить условный оператор, то эта программа запишется так:

```
if (N=3) or (N=4) or (N=5)
then WriteLn('Экзамен сдан')
else
  if N=2
  then WriteLn('Экзамен не сдан')
  else WriteLn('Нет такой оценки');
```

В условии ветвления использовано сложное логическое выражение, содержащее операции логического сложения **or** (или).

#### Пример 4

Составить программу определения принадлежности заданной точки с координатами  $(x, y)$  на плоскости области, заштрихованной на рис. 2.6, включая ее границы<sup>1)</sup>.

Запишем систему неравенств, которой удовлетворяют точки, принадлежащие данной области:

$$\begin{cases} y \leq 1; \\ y \geq \sin x; \\ x \geq 0; \\ x \leq \pi/2. \end{cases}$$

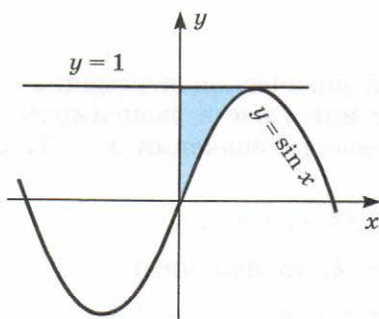


Рис. 2.6. Область на плоскости

<sup>1)</sup> Задача взята из демоверсии ЕГЭ по информатике.

Рассмотрим два варианта применения ветвления для решения этой задачи.

```

Program Variant_1;
Var x, y: real;
begin
  Write('Введите x, y:');
  Readln(x, y);
  if y<=1 then
    if y>=sin(x) then
      if x>=0 then
        if x<=Pi/2 then
          Write('Да')
        else Write('Нет')
      else Write('Нет')
    else Write('Нет')
  else Write('Нет')
end.

```

```

Program Variant_2;
Var x, y: real;
begin
  Write('Введите x, y:');
  Readln(x, y);
  if (y<=1) and (y>=sin(x)) and
    (x>0) and (x<=Pi/2)
  then Write('Да')
  else Write('Нет')
end.

```

Первый вариант имеет структуру вложенных ветвлений с простыми условиями. Во втором варианте — одно ветвление с условием, являющимся сложным логическим выражением. Рассмотрим еще один вариант программы решения этой задачи — с линейной структурой.

```

Program Variant_3;
Var x, y: real; Flag: Boolean;
begin
  Write('Введите x, y:'); Readln(x, y);
  Flag:=(y<=1) and (y>=sin(x)) and (x>0) and (x<=Pi/2);
  Write('Точка принадлежит области?', Flag:6)
end.

```

Здесь используется логическая переменная *Flag*, которая принимает значение *true* или *false* в зависимости от положения точки. Например, если ввести значения  $x = 1$ ,  $y = 1$ , то на экран выведется:

Точка принадлежит области? True.

А если  $x = 1$ ,  $y = 2$ , то получим:

Точка принадлежит области? False.

Формат вывода :6 логической величины позволяет отделить логическое значение от предыдущего текста пробелами.



## Система основных понятий



Программирование ветвлений	
Условный оператор	Оператор выбора
<p><b>if</b> (&lt;логическое выражение&gt;  <b>then</b> &lt;оператор 1&gt;  <b>else</b> &lt;оператор 2&gt;  <b>else</b> &lt;логическое выражение&gt;          истинно, то выполняется &lt;оператор 1&gt;, иначе выполняется &lt;оператор 2&gt;. &lt;Оператор 1&gt; и &lt;оператор 2&gt; — простые или составные операторы. В неполном ветвлении отсутствует ветвь <b>else</b></p>	<p><b>case</b> &lt;селектор&gt; <b>of</b>          &lt;список констант 1 &gt;:          &lt;оператор 1&gt;;          ...          &lt;список констант N&gt;:          &lt;оператор N&gt;  <b>else</b> &lt;оператор&gt;  <b>end</b>          &lt;селектор&gt; — выражение порядкового типа; константы имеют тот же тип. Выполняется только одна из ветвей выбора, которая содержит константу, совпадающую со значением селектора. Ветвь <b>else</b> может отсутствовать</p>

## Вопросы и задания

1. Какие операторы используются для программирования ветвящихся алгоритмов?
2. Является ли оператор выбора необходимым для программирования ветвящихся алгоритмов?
3. В каких случаях удобно использование оператора выбора?
4. Составьте на Паскале программу полного решения биквадратного уравнения.
5. Используя оператор выбора, составьте программу, которая по введенному номеру месяца в году будет выводить соответствующее время года (зима, весна, лето, осень).

Практикум. Раздел «Программирование»



## 2.2.7. Программирование циклов

Рассмотрим приемы программирования циклов на Паскале. Различают циклы с заданным числом повторений и итеративные циклы.

**Задача.** Известно, что сумма следующего бесконечного числового ряда:

$$1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{i!} + \dots$$

в пределе стремится к значению константы  $e = 2,71828182\dots$ . Функция  $e^x$  называется экспонентой, а логарифм по основанию  $e$  называется натуральным логарифмом и обозначается  $\ln x$ .

Требуется составить программу, вычисляющую эту константу по сумме числового ряда.

Если слагаемые в этом выражении обозначить следующим образом:

$$a_0 = 1, a_1 = \frac{1}{1!}, a_2 = \frac{1}{2!}, a_3 = \frac{1}{3!}, \dots,$$

то обобщенная формула для  $i$ -го элемента будет такой:

$$a_i = \frac{1}{i!}.$$

Нетрудно увидеть, что между элементами данной последовательности имеется зависимость:

$$a_0 = 1, a_1 = \frac{a_0}{1}, a_2 = \frac{a_1}{2}, a_3 = \frac{a_2}{3} \text{ и т. д.}$$

Такая зависимость называется **рекуррентной зависимостью**, а соответствующая числовая последовательность — **рекуррентной последовательностью**. Данная рекуррентная последовательность может быть описана следующим образом:

$$a_i = \begin{cases} 1, & \text{при } i = 0; \\ \frac{a_{i-1}}{i}, & \text{при } i > 0. \end{cases}$$

### Циклы с заданным числом повторений

Составим программу, по которой будет вычислена сумма заданного количества слагаемых. Постановка задачи такая: дано целое положительное значение  $n$ . Требуется вычислить сумму:

$$1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{n!}.$$

Приведем два варианта программы решения этой задачи. В первом варианте используется цикл с предусловием, во втором — цикл с постусловием.



```
Program Summa_1;
Var E, a: real;
    N, i: integer;
begin
  Write('N='); Readln(N);
  E:=0; i:=0; a:=1;
  while i<=N do
  begin
    E:=E+a;
    i:=i+1;
    a:=a/i
  end;
  Writeln('E=', E)
end.
```

```
Program Summa_2;
Var E, a: real;
    N, i: integer;
begin
  Write('N='); Readln(N);
  E:=0; i:=0; a:=1;
  repeat
    E:=E+a;
    i:=i+1;
    a:=a/i;
  until i>N;
  Writeln('E=', E)
end.
```

Обратите внимание на то, как цикл с предусловием преобразуется в цикл с постусловием: условие цикла помещается после тела цикла и заменяется на противоположное:

$\text{Not}(i \leq N) = i > N$ .

И тот, и другой цикл повторит свое выполнение  $N$  раз. Переменная  $i$  выполняет роль не только знаменателя в дроби  $1/i!$ , но и является счетчиком числа повторений цикла. Такие переменные называются **параметрами цикла**.

Выполнение этих программ на компьютере для значения  $N = 7$  приводит к следующему результату:  $E = 2,7182539$ .

Для программирования циклов с заданным числом повторений при постоянном шаге изменения параметра цикла в Паскале существует **цикл с параметром**. Вот как выглядит программа решения той же задачи с использованием цикла с параметром:

```
Program Summa_3;
Var E, a: real; N, i: integer;
begin
  Write('N='); Readln(N);
  E:=1; a:=1;
  for i:=1 to N do
  begin
    a:=a/i;
    E:=E+a
  end;
  Writeln('E=', E)
end.
```

В программе используется оператор цикла **for**, для которого существуют два варианта:

- 1) **for** <параметр цикла>:=<выражение 1> **to** <выражение 2>  
**do** <оператор>
- 2) **for** <параметр цикла>:=<выражение 1> **downto** <выражение 2>  
**do** <оператор>

Здесь <параметр цикла> — имя простой переменной порядкового типа. Выполнение оператора **for** в первом варианте (**to**) происходит по следующей схеме.

1. Вычисляются значения <выражения 1> и <выражения 2>. Это делается только один раз при входе в цикл.
2. Параметру цикла присваивается значение <выражения 1>.
3. Значение параметра цикла сравнивается со значением <выражения 2>. Если параметр цикла меньше или равен этому значению, то выполняется тело цикла, в противном случае выполнение цикла заканчивается.
4. Значение параметра цикла изменяется на следующее значение в его типе (для целых чисел — увеличивается на единицу); происходит возврат к пункту 3.

Оператор цикла **for** объединяет в себе действия, которые при использовании цикла **while** выполняют несколько операторов: присваивание параметру начального значения, сравнение с конечным значением, замена значения на следующее.

Во втором варианте оператора **for** слово **downto** буквально можно перевести как «вниз до». В таком случае параметр цикла изменяется по убыванию, т. е. при каждом повторении цикла параметр изменяет свое значение на предыдущее (равносильно `i:=pred(i)`).

Работая с оператором **for**, учитывайте следующие правила:

- параметр цикла не может иметь вещественный тип;
- в теле цикла нельзя изменять переменную — параметр цикла;
- при выходе из цикла значение переменной-параметра является неопределенным.

В следующем примере в качестве параметра цикла **For** используется символьная переменная. Пусть требуется получить на экране десятичные коды букв латинского алфавита. Как известно, латинские буквы в таблице кодировки упорядочены по алфавиту. Вот фрагмент такой программы:

```
for C := 'a' to 'z' do
  Write (C, ' - ', Ord(C));
```

Здесь переменная `C` имеет тип `char`.

А теперь подумайте, как вывести кодировку латинского алфавита в обратном порядке (от 'z' до 'a').



### Итерационные циклы

Итерационными называются циклы, точное число повторений которых заранее неизвестно. Оно определится только в результате выполнения цикла. В итерационном цикле при каждом его повторении происходит последовательное приближение к вычисляемой величине и проверка условия достижения искомого результата. Выход из итерационного цикла осуществляется в случае выполнения заданного условия.

Снова рассмотрим задачу вычисления суммы того же числового ряда. Но теперь условие будет таким: в сумму нужно включить только слагаемые, значение которых больше некоторой малой величины  $\varepsilon$ . При этом полученная сумма будет отличаться от предельного значения (константы  $e$ ) на величину, не большую  $\varepsilon$ .

С увеличением значения  $i$  величина  $1/i!$  уменьшается, следовательно, в сумму надо включать все слагаемые, предшествующие первому значению, меньшему  $\varepsilon$ . Приведем две программы решения этой задачи: с циклом с предусловием и циклом с постусловием.

```
Program Summa_4;
Var E, a, eps: real; i:integer;
begin
  Write('Eps='); Readln(eps);
  E:=0; i:=0; a:=1;
  while a>eps do
  begin
    E:=E+a;
    i:=i+1;
    a:=a/i
  end;
  Writeln('E=', E,
          'Слагаемых:', i)
end.
```

```
Program Summa_5;
Var E, a, eps: real; i:integer;
begin
  Write('Eps='); Readln(eps);
  E:=0; i:=0; a:=1;
  repeat
    E:=E+a;
    i:=i+1;
    a:=a/i
  until a<=eps;
  Writeln('E=', E,
          'Слагаемых:', i)
end.
```

Решить эту задачу, используя цикл с параметром, нельзя. Итерационные циклы программируются путем использования либо цикла-пока, либо цикла-до.

В качестве результата выводится значение суммы и число вошедших в нее слагаемых. Выполнение этих программ для значения  $\varepsilon = 10^{-8}$  дает в результате:

E=2,71828182 Слагаемых: 12

Таким образом, за 12 повторений цикла значение константы  $e$  получено с точностью до 8 знаков после запятой.

## Система основных понятий

Программирование циклов		
Разновидности циклических алгоритмов		
Циклы с заданным числом повторений	Итерационные циклы	
Имеется управляющий параметр, изменяющийся с постоянным шагом в определенном диапазоне значений. Реализуется всеми типами операторов цикла	Число повторений цикла заранее неизвестно. Реализуется операторами цикла-пока и цикла-до	
Операторы цикла		
Цикл-пока	Цикл с параметром	Цикл-до
<p><b>while</b> &lt;логическое выражение&gt;  <b>do</b> &lt;оператор&gt;;            &lt;оператор&gt; — тело цикла. Цикл повторяет выполнение, пока истинно &lt;логическое выражение&gt;</p>	<p>Параметр — переменная порядкового типа</p> <p>1) <b>for</b> &lt;параметр цикла&gt; := &lt;выражение 1&gt; <b>to</b> &lt;выражение 2&gt; <b>do</b> &lt;оператор&gt;            — по возрастанию параметра</p> <p>2) <b>for</b> &lt;параметр цикла&gt; := &lt;выражение 1&gt; <b>downto</b> &lt;выражение 2&gt; <b>do</b> &lt;оператор&gt;            — по убыванию параметра</p>	<p><b>repeat</b>            &lt;оператор&gt;  <b>until</b>            &lt;логическое выражение&gt;</p> <p>Повторяется выполнение тела цикла до того, как &lt;логическое выражение&gt; станет истинным</p>

## Вопросы и задания

1. Чем отличается итерационный цикл от цикла с заданным числом повторений?
2. Почему для программирования итерационных циклов не используется оператор цикла с параметром?
3. Значение функции  $e^x$  (экспонента от  $x$ ) равно сумме сходящегося бесконечного ряда:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} + \dots$$



Получите рекуррентную формулу для слагаемых. Используя операторы цикла **while**, **repeat** и **for**, составьте три варианта программы вычисления суммы с заданным числом слагаемых.

4. Запрограммируйте итерационный цикл вычисления функции  $e^x$  с заданной точностью  $\varepsilon$ . Составьте два варианта программы: с циклами **while** и **repeat**. Сопоставьте полученный результат со значением стандартной функции  $\exp(x)$ .
5. Вычислите сумму квадратов всех целых чисел, попадающих в интервал  $(\ln x, e^x)$ , для заданного  $x > 1$ .
6. Вычислите количество точек с целочисленными координатами, попадающих в круг радиуса  $R$  ( $R > 0$ ) с центром в начале координат.
7. Получите таблицу значений функций  $\sin x$  и  $\cos x$  на отрезке  $[0, 1]$  с шагом  $0,1$  в следующем виде:

x	sin x	cos x
0.0000	0.0000	1.0000
0.1000	0.0998	0.9950
...	...	...
1.0000	0.8415	0.5403

8. Получите в возрастающем порядке все трехзначные числа, в десятичной записи которых нет одинаковых цифр.
9. Дано целое  $n > 2$ . Получите все простые числа из отрезка  $[2, n]$ .

Практикум. Раздел «Программирование»

### 2.2.8. Вспомогательные алгоритмы и подпрограммы

Еще одним важнейшим методологическим приемом структурного программирования является *декомпозиция решаемой задачи на подзадачи* — более простые, с точки зрения программирования, части исходной задачи. Алгоритмы решения таких подзадач называются **вспомогательными алгоритмами**.

В языках программирования вспомогательные алгоритмы называются **подпрограммами**. В Паскале различаются две разновидности подпрограмм: **процедуры** и **функции**. Рассмотрим этот вопрос на примере следующей задачи: даны два натуральных числа  $a$  и  $b$ . Требуется определить наибольший общий делитель трех величин:  $a + b$ ,  $a^2 + b^2$ ,  $a \cdot b$ . Запишем это так:  $\text{НОД}(a + b, a^2 + b^2, a \cdot b)$ .

Идея решения состоит в следующем математическом факте: если  $x, y, z$  — три натуральных числа, то  $\text{НОД}(x, y, z) = \text{НОД}(\text{НОД}(x, y), z)$ . Иначе говоря, нужно найти НОД двух ве-

личин, а затем НОД полученного значения и третьего числа (попробуйте это доказать).

Очевидно, что вспомогательным алгоритмом для решения поставленной задачи является алгоритм получения наибольшего общего делителя двух чисел. Эта задача решается с помощью алгоритма Евклида, который подробно обсуждался в 9 классе. Напомним, что идея алгоритма Евклида основана на следующей формуле:

$$\text{НОД}(M, N) = \begin{cases} M, & \text{при } M = N; \\ \text{НОД}(M - N, N), & \text{при } M > N; \\ \text{НОД}(N, N - M), & \text{при } N > M. \end{cases}$$

Приведем алгоритм решения поставленной задачи на учебном Алгоритмическом языке (АЯ). Алгоритм состоит из процедуры «Евклид» и основного алгоритма «Задача», в котором присутствуют два обращения к процедуре:

Процедура Евклид (цел M, N, K);

нач

пока M <> N

нц

если M > N

то M := M - N

иначе N := N - M

кв

кц

    K := M

кон

алг задача;

цел a, b, c;

нач

    ввод(a, b);

    Евклид(a+b, a\*a+b\*b, c);

    Евклид(c, a\*b, c);

    вывод(c)

кон

Здесь M, N и K являются формальными параметрами процедуры. M и N — параметры-аргументы, K — параметр-результат.

**Процедуры в Паскале.** Основное отличие процедур в Паскале от процедур в Алгоритмическом языке состоит в том, что процедуры в Паскале описываются в разделе описания подпрограмм, а



в АЯ процедура является внешней по отношению к вызывающей программе. Теперь посмотрим, как решение поставленной задачи программируется на Паскале.

```
Program NOD1;
Var A, B, C: integer;
Procedure Evklid(M, N: integer; var K: integer);
begin
  while M<>N do
    if M>N
    then M:=M-N
    else N:=N-M;
  K:=M
end;
begin
  Write('a = '); ReadLn(A);
  Write('b = '); ReadLn(B);
  Evklid(A+B, A*A + B*B, C);
  Evklid(C, A*B, C);
  WriteLn('НОД = ', C)
end.
```

В данном примере обмен аргументами и результатами между основной программой и процедурой производится через параметры. Описание процедуры на Паскале имеет следующий формат:

```
Procedure <имя процедуры> [(список формальных параметров)];
<блок>
```

Квадратные скобки указывают на то, что список формальных параметров может отсутствовать; т. е. возможна процедура без параметров.

Параметры могут быть параметрами-переменными и параметрами-значениями. Параметры-переменные записываются следующим образом:

```
Var <список переменных>: <тип>
```

Параметры-значения указываются так:

```
<список переменных>: <тип>
```

Чаще всего аргументы представляются как параметры-значения (хотя могут быть и параметрами-переменными). А для передачи результатов используются параметры-переменные. Процедура в качестве результата может передавать в вызывающую программу множество значений (в частном случае — одно), а может и ни одного. Теперь рассмотрим правила обращения к процедуре. Обращение к процедуре производится в форме оператора процедуры: <имя процедуры>[(список фактических параметров)]

Если описана процедура с формальными параметрами, то обращение к ней производится оператором процедуры с фактическими параметрами. Правила соответствия между формальными и фактическими параметрами: соответствие *по количеству*, соответствие *по последовательности* и соответствие *по типам*.

Взаимодействие формальных и фактических параметров через параметры-переменные называется **передачей по ссылке**: при обращении к процедуре ей передается ссылка на переменную, заданную в качестве фактического параметра. Эта ссылка и используется процедурой для доступа к этой переменной.

Другой вариант взаимодействия формальных и фактических параметров называется **передачей по значению**: вычисляется значение фактического параметра (выражения), и это значение присваивается соответствующему формальному параметру.

В рассмотренном нами примере формальные параметры  $M$  и  $N$  являются параметрами-значениями. Это аргументы процедуры. При обращении к ней первый раз им соответствуют значения выражений  $A+B$  и  $A \cdot A+B \cdot B$ ; второй раз —  $C$  и  $A \cdot B$ . Параметр  $K$  является параметром-переменной. В ней получается результат работы процедуры. В обоих обращениях к процедуре соответствующим фактическим параметром является переменная  $C$ . Через эту переменную основная программа получает результат.

Теперь рассмотрим другой вариант программы, решающей ту же задачу. В ней используется процедура **без параметров**.

✓ **Program** NOD2;  
**Var** A, B, K, M, N: Integer;  
**Procedure** Evklid;  
**begin**  
  **while** M<>N **do**  
    **if** M>N  
      **then** M:=M-N  
      **else** N:=N-M;  
  K:=M  
**end**;  
**begin**  
  Write('a= '); ReadLn(A);  
  Write('b= '); ReadLn(B);  
  M:=A+B; N:=A\*A+B\*B);  
  Evklid;  
  M:=K; N:=A\*B;  
  Evklid;  
  WriteLn('НОД равен ', K)  
**end**.



Чтобы разобраться в этом примере, требуется объяснить новое для нас понятие: область действия описания.

**Областью действия описания** любого программного объекта (переменной, типа, константы и т. д.) является тот блок, на который это описание распространяется. Если данный блок вложен в другой (подпрограмма), то присутствующие во вложенном блоке описания являются **локальными**. Они действуют только в пределах внутреннего блока. Описания же, расположенные во внешнем блоке, называются **глобальными** по отношению к внутреннему блоку. Если глобально описанный объект используется во внутреннем блоке, то на него распространяется внешнее (глобальное) описание.

В программе NOD1 переменные  $M$ ,  $N$ ,  $K$  являются локальными внутри процедуры; переменные  $A$ ,  $B$ ,  $C$  — глобальные. Однако внутри процедуры переменные  $A$ ,  $B$ ,  $C$  не используются. Связь между внешним блоком и процедурой осуществляется через параметры.

В программе NOD2 все переменные являются глобальными. В процедуре *Evklid* нет ни одной локальной переменной (нет и параметров). Переменные  $M$  и  $N$ , используемые в процедуре, получают свои значения через оператор присваивания в основном блоке программы и изменяют значения в подпрограмме. Результат получается в глобальной переменной  $K$ , значение которой выводится на экран. Здесь **обмен** значениями между основной программой и процедурой производится **через глобальные переменные**.

Использование механизма передачи через параметры делает процедуру более универсальной, независимой от основной программы. Однако в некоторых случаях оказывается удобнее использовать передачу через глобальные переменные. Чаще такое бывает с процедурами, работающими с большими объемами данных. В этой ситуации глобальное взаимодействие экономит память компьютера.

**Функции.** Теперь выясним, что такое подпрограмма-функция. Обычно функция используется в том случае, когда результатом работы подпрограммы должна быть скалярная (простая) величина. Тип результата называется типом функции. В Турбо Паскале допускаются функции строкового типа. Формат описания функции следующий:

```
Function <имя функции> [( <список формальных параметров> )]:  
    <тип функции>; <блок>
```

Как и у процедуры, у функции в списке формальных параметров могут присутствовать параметры-переменные и параметры-значения. Все это — аргументы функции. Параметры могут вообще отсутствовать, если аргументы передаются глобально.

Программа решения рассмотренной выше задачи с использованием функции будет выглядеть следующим образом:

```

Program NOD3;
Var A, B, Rez: integer;
Function Evklid(M, N: integer): integer;
begin
    while M<>N do
        if M>N
        then M:=M-N
        else N:=N-M;
    Evklid:=M
end;
begin
    Write('a = '); ReadLn(A);
    Write('b = '); ReadLn(B);
    Rez:= Evklid(Evklid(A+B, A*A+B*B), A*B);
    WriteLn('NOD равен ', Rez)
end.

```

Из примера видно, что тело функции отличается от тела процедуры только тем, что в функции *результат присваивается идентификатору функции*: Evklid:=M.

Обращение к функции является операндом в выражении. Оно записывается в следующей форме:

<имя функции> (<список фактических параметров>)

Правила соответствия между формальными и фактическими параметрами все те же. Сравнивая приведенные выше программы, можно сделать вывод, что программа NOD3 имеет определенные преимущества перед другими. Функция позволяет получить результат путем выполнения одного оператора присваивания. Здесь также иллюстрируется возможность того, что фактическим аргументом при обращении к функции может быть эта же функция.

По правилам стандарта Паскаля, возврат в вызывающую программу из подпрограммы происходит, когда выполнение подпрограммы доходит до ее конца (последний **end**). Однако в Турбо Паскале есть средство, позволяющее выйти из подпрограммы в любом ее месте. Это оператор-процедура **Exit**. Например, функцию определения большего из двух данных вещественных чисел можно описать так:

```

Function Max(X, Y: real): real;
begin
    Max:=X;
    if X>Y then Exit else Max:=Y
End;

```



**Модифицированный алгоритм Евклида.** Подпрограмму алгоритма Евклида можно составить иначе, если воспользоваться операцией `mod` — получением остатка от деления, имеющейся в Паскале. Идея алгоритма исходит из справедливости следующих равенств:

$$\text{НОД}(M, N) = \begin{cases} M, & \text{при } N = 0; \\ \text{НОД}(N, M \bmod N), & \text{при } N \neq 0. \end{cases}$$

В таком случае функцию `Evklid` можно переписать так:

```
Function Evklid(M, N: integer): integer;
Var R: integer;
begin
  while N <> 0 do
    begin R:=M mod N; M:=N; N:=R end;
  Evklid:=M
end;
```

## Система основных понятий

Подпрограммы	
Процедуры	Функции
Результат — <i>любое число величин</i> Описание: <b>Procedure</b> <имя процедуры> [(список формальных параметров)]; <блок>	Результат — <i>одна величина</i> Описание: <b>Function</b> <имя функции> [(список формальных параметров)]: <тип функции>; <блок>
Обращение — <i>оператор процедуры</i> : <имя процедуры> [(список фактических параметров)]	Обращение — <i>операнд выражения</i> : <имя функции> (<список фактических параметров>)
Параметры подпрограмм	
Параметры-переменные	Параметры-значения
Описание: <b>Var</b> <список переменных>: <тип>	Описание: <список переменных>: <тип>
Фактические параметры: <i>переменные</i>	Фактические параметры: <i>выражения</i>

## Вопросы и задания

1. Для чего используются подпрограммы?
2. В чем различие между процедурами и функциями?
3. Какие существуют способы передачи данных между подпрограммой и вызывающей ее программой?
4. Составьте программу вычисления площади кольца по значениям внутреннего и внешнего радиусов, используя подпрограмму вычисления площади круга (два варианта: с процедурой и с функцией).
5. Составьте программу сложения двух простых дробей. Результат должен быть несократимой дробью. Используйте подпрограмму вычисления НОД по алгоритму Евклида. Простая дробь задается двумя целыми числами: числителем и знаменателем.
6. По координатам вершин треугольника вычислите его периметр, используя подпрограмму вычисления длины отрезка между двумя точками.
7. Даны три целых числа. Определите, у которого из них больше сумма цифр. Подсчет суммы цифр организуйте через подпрограмму.

Практикум. Раздел «Программирование»

### 2.2.9. Массивы

**Массивом** в Паскале называют переменную величину регулярного типа.

**Регулярный тип** — это структурный тип данных, представляющий собой совокупность пронумерованных однотипных величин.

**Описание массивов.** Переменная регулярного типа описывается в разделе описания переменных в следующей форме:

```
Var <идентификатор>: array [<тип индекса>] of <тип компонентов>
```

В данном случае квадратные скобки — это обязательные символы, которые называются индексными скобками. Чаще всего в качестве типа индекса употребляется ограниченный тип. Например, массив вещественных чисел, хранящий 12 значений среднемесячных температур в течение года, опишется так:

```
Var T: array [1..12] of real;
```

Описание массива определяет, во-первых, размещение массива в памяти, во-вторых, правила его дальнейшего употребления в программе.

Элемент массива идентифицируется в виде переменной с индексами:

```
<идентификатор массива> [<индексы элемента>]
```



Для одномерного массива записывается один индекс — одно значение. Для многомерных массивов индексы — множество значений. В качестве индекса может употребляться любое выражение соответствующего типа. Например, для элементов массива температур возможны обозначения:  $T[5]$ ,  $T[k]$ ,  $T[i+j]$ ,  $T[m \text{ div } 2]$ .

Последовательные элементы массива располагаются в последовательных ячейках памяти ( $T[1]$ ,  $T[2]$  и т. д.), причем значения индекса не должны выходить за диапазон  $1..12$ .

Тип индекса может быть любым скалярным порядковым типом, кроме `integer`. Например, в программе могут присутствовать следующие описания:

```
Var Cod: array[char] of 1..100;  
    L: array[boolean] of char;
```

В такой программе допустимы следующие обозначения элементов массивов:

```
Cod['x']; L[true]; Cod[chr(65)]; L[a>0].
```

В некоторых случаях бывает удобно в качестве индекса использовать перечислимый тип. Например, данные о количестве учеников в четырех десятых классах одной школы могут храниться в следующем массиве:

```
Type Index = (A, B, C, D);  
Var Class_10: array[Index] of byte;
```

И если, например, элемент `Class_10[A]` равен 35, то это означает, что в 10А классе 35 человек. Такое индексирование улучшает наглядность программы.

Часто структурному типу присваивается имя в разделе типов, которое затем используется в разделе описания переменных.

```
Type Mas1 = array [1..100] of integer;  
    Mas2 = array [-10..10] of char;  
Var Num: Mas1; Sim: Mas2;
```

До сих пор речь шла об одномерных массивах, в которых типы элементов скалярные.

**Многомерный массив** в Паскале трактуется как одномерный массив, тип элементов которого также является массивом (массив массивов).

В качестве примера рассмотрим таблицу с информацией о среднемесячных температурах за 10 лет, например с 2001 по 2010 год. Очевидно, что для этого удобна прямоугольная (двумерная) таблица, в которой столбцы соответствуют годам, а строки — месяцам.

Год	Месяц											
	1	2	3	4	5	6	7	8	9	10	11	12
2001	-23	-17	-8,4	6,5	14	18,6	25	19	12,3	5,6	-4,5	-19
2002	-16	-8,5	-3,2	7,1	8,4	13,8	28,5	21	6,5	2	-13	-20
2003	-9,8	-14	-9,2	4,6	15,6	21	17,8	20	11,2	8,1	-16	-21
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2010	-25	-9,7	-3,8	8,5	13,9	17,8	23,5	17,5	10	5,7	-14	-20

Для обработки такой таблицы в программе следует описать массив:

```
Var Tabl: array[2001..2010] of array[1..12] of real;
```

Вот примеры обозначения некоторых элементов этого массива: Tabl[2001][1]; Tabl[2005][10]; Tabl[2010][12].

Однако чаще употребляется другая, эквивалентная форма обозначения элементов двумерного массива:

```
Tabl[2001,1]; Tabl[2005,10]; Tabl[2010,12].
```

Переменная Tabl[2001] обозначает всю первую строку таблицы, т. е. весь массив температур за 2001 год. Другим эквивалентным вариантом приведенному выше описанию является следующее:

```
Type Month = array [1..12] of real;
```

```
Year = array [2001..2010] of Month;
```

```
Var Tabl: Year;
```

Наиболее краткий вариант описания данного массива такой:

```
Var Tabl: array[2001..2010, 1..12] of real;
```

Продолжая по аналогии, можно определить трехмерный массив как одномерный массив, у которого элементами являются двумерные массивы. Вот пример описания трехмерного массива:

```
Var A: array[1..10, 1..20, 1..30] of integer;
```

Это массив, состоящий из  $10 \cdot 20 \cdot 30 = 6000$  целых чисел и занимающий в памяти  $6000 \cdot 2 = 12\,000$  байтов. В Паскале нет ограничения сверху на размер массива. Однако в каждой конкретной реализации Паскаля ограничивается объем памяти, выделяемый под массивы. В Турбо Паскале это ограничение равно 64 килобайтам.



По аналогии с математикой одномерные числовые массивы часто называют векторами, а двумерные — матрицами.

В Паскале не допускается употребление динамических массивов, т. е. таких, размер которых определяется в процессе выполнения. Изменение размеров массива происходит через изменение в тексте программы и повторную компиляцию. Для упрощения таких изменений удобно определять индексные параметры в разделе констант:

```
Const Imax = 10; Jmax = 20;  
Var Mas: array[1..Imax, 1..Jmax] of integer;
```

Теперь для изменения размеров массива *Mas* и всех операторов программы, связанных с этими размерами, достаточно отредактировать только одну строку в программе — раздел констант.

**Действия над массивом как единым целым.** Такие действия допустимы лишь в случае присваивания значений одного массива другому. При этом оба массива должны иметь одинаковые типы (тип индексов и тип элементов).

### Пример 1

```
Var P, Q: array[1..5, 1..10] of real;
```

При выполнении операции присваивания

```
P:=Q
```

все элементы массива *P* станут равными соответствующим элементам массива *Q*.

### Пример 2

Как уже отмечалось, в многомерных массивах переменная с индексом может обозначать целый массив. Тогда, если массив *Tabl* описан так:

```
Type Mas = array [1..12] of real;  
Var Tabl: array[2001..2010] of mas;
```

и в нем требуется данные за 2009 год сделать такими же, как за 2001 год (девятой строке присвоить значение первой строки), то это можно сделать одним присваиванием:

```
Tabl[2009]:= Tabl[2001].
```

А если нужно поменять местами значения этих строк, то это делается через третью переменную того же типа:

```
P:=Tabl[2009]; Tabl[2009]:=Tabl[2001]; Tabl[2001]:=P;
```

где переменная  $P$  описана так:

```
Var P: Mas;
```

**Ввод и вывод массивов** производится покомпонентно. Вот примеры ввода с клавиатуры значений одномерного и двумерного массивов:

```
for I:=1 to 12 do
  ReadLn(T[I]);
for I:=1 to Imax do
  For J:=1 to Jmax do
    ReadLn(Mas[I, J]);
```

Здесь каждое следующее значение будет вводиться с новой строки. Для построчного ввода используется оператор `Read`.

Аналогично в цикле по индексной переменной организуется вывод значений массива на экран. Например:

```
For I:=1 To 12 Do Write(T[I]:8:4);
```

Напомним, что модификатор формата `8:4` означает вывод числа в формате с фиксированной точкой в 8 позиций, из которых в 4 последних позициях размещается дробная часть.

Следующий фрагмент программы организует построчный вывод матрицы на экран:

```
for I:=1 to Imax do
begin
  for J:=1 to Jmax do
    Write(Mas[I,J]:6);
  WriteLn
end;
```

После вывода очередной строки матрицы оператор `WriteLn` без параметров переведет курсор в начало новой строки. Следует заметить, что в последнем примере матрица на экране будет получена в естественной форме прямоугольной таблицы, если  $Jmax$  не превышает 12 (подумайте почему).

Для массивов большого размера удобно производить **ввод значений из текстового файла**, заранее подготовив такой файл с исходными данными. Пусть в текстовом файле с именем `matr.txt` с помощью текстового редактора записана следующая числовая матрица размером  $4 \times 4$ :

```
5  7 10  3
3  2  1 23
7 12  6 10
9  2  6 14
```



В следующей программе производится ввод этой матрицы в двумерный массив *M*:

```

Var M: array[1..4, 1..4] of integer;
    i, j: byte;
    F1: text; {Файловая переменная}
begin
  Assign(F1, 'matr.txt'); {Связывание F1 с файлом matr.txt}
  Reset(F1);             {Открытие файла для чтения}
  for i:=1 to 4 do
    begin
      for j:=1 to 4 do
        Read(F1, M[i,j]); {Последовательное чтение}
                          {из одной строки}
        Readln(F1)       {Переход к следующей строке}
      end;
    Close(F1);           {Закрытие файла}
  ...

```

## Система основных понятий

Массивы	
Массив — переменная величина регулярного типа	
Регулярный тип — структурный тип данных, представляющих собой совокупность пронумерованных однотипных величин	
Описание массива	Идентификация элементов массива
<b>Var</b> <идентификатор>: <b>array</b> [<тип индекса>] <b>of</b> <тип компонентов> <тип индекса> — любой порядко- вый тип, кроме <i>integer</i> ; тип ком- понентов — любой простой или структурный тип	<идентификатор массива> [<индексы элемента>] Для одномерного массива — один индекс (одно значение), для много- мерного массива индексы — мно- жество значений
Действия над массивом как единым целым	
Присваивание однотипных массивов	Отношения «равно», «не равно» для однотипных массивов
Ввод/вывод массивов производится покомпонентно с клавиатуры или из файла	

## Вопросы и задания

1. Что такое регулярный тип данных? Что такое массив?
2. Какие типы допустимы для индексов массива?

3. Как в Паскале трактуется многомерный массив?
4. Какие действия можно выполнять над массивом как единым целым?
5. Дан вектор  $\{z_i\}$ ,  $i = 1, \dots, 50$ . Составьте программу ввода значений и вычисления длины этого вектора по следующей формуле:

$$L = \sqrt{z_1^2 + z_2^2 + \dots + z_{50}^2}.$$

6. Даны значения массива  $\{a_i\}$ ,  $i = 0, \dots, 10$ , и переменной  $x$ . Составьте программу вычисления алгебраического многочлена 10-й степени по формуле Горнера:  
 $a_{10}x^{10} + a_9x^9 + \dots + a_1x + a_0 = (((...(a_{10}x + a_9)x + a_8)x + \dots + a_1)x + a_0$ .
7. Значения вектора  $\{x_i\}$ ,  $i = 1, \dots, 20$  вводятся из текстового файла. Требуется подсчитать количество его элементов, значения которых лежат на отрезке  $[0, 1]$ .
8. Введите из текстового файла целочисленную матрицу размером  $6 \times 8$ . Переверните матрицу, поменяв 1-ю строку с 6-й, 2-ю — с 5-й, 3-ю — с 4-й, и запишите полученную матрицу в другой файл.
9. Введите с клавиатуры одномерный числовой массив из 9 элементов. Сверните его в матрицу размером  $3 \times 3$ , разместив первую тройку элементов в 1-й строке матрицы, 2-ю тройку — во второй строке, 3-ю тройку — в третьей строке.
10. Введите с клавиатуры построчно в двумерный массив числовую матрицу размером  $4 \times 4$ . Разверните ее по столбцам в одномерный массив. Запишите массив в текстовый файл.

## 2.2.10. Типовые задачи обработки массивов

### Заполнение массива

Значения массива могут задаваться вводом с клавиатуры, чтением из файла или вычислением в программе. В некоторых задачах статистического характера требуется заполнять массивы случайными числами.

#### Пример 1

Заполнить массив равномерно распределенными целыми случайными числами в диапазоне от 0 до 100.

```
Var X: array[1..20] of integer; i: integer;
begin
  Randomize;
  for i:=1 to 20 do X[i]:=Random(100);
  For i:=1 to 20 do Write(X[i]:4)
end.
```

Стандартная функция `Random(x)` возвращает псевдослучайное целое число на отрезке  $[0, x - 1]$ , где  $x$  — целое число. Такие функции называют датчиками случайных чисел. Стандартная



процедура `Randomize` случайным образом устанавливает начальное состояние датчика. Благодаря этому при повторном запуске программы последовательности чисел, вырабатываемые датчиком, повторяться не будут.

Если требуется изменить диапазон случайных чисел, то это всегда можно сделать путем сдвига. Например, если нужно получить числа на отрезке  $[-50, 50]$ , то в программе переписывается оператор присваивания:

```
X[i]:=Random(100)-50;
```

Для получения вещественных случайных чисел используется функция `Random` без аргументов. Она возвращает случайные дробные значения в полуинтервале  $[0, 1)$ . С помощью сдвига и множителя эти значения можно привести к любому диапазону. Например, следующее выражение будет вычислять случайное вещественное число в полуинтервале  $[-5, 5)$ :  $10 * \text{Random} - 5$ .

### Пример 2

Заполнить верхнетреугольную матрицу указанного вида и вывести ее на экран.

Матрица:

```
1 2 3 4
0 2 3 4
0 0 3 4
0 0 0 4
```

Программа решения задачи:

```
Var M: array[1..4, 1..4] of integer;
    i, j: integer;
begin
  for i:=1 to 4 do
    for j:=1 to 4 do
      if j<i then M[i,j]:=0 else M[i,j]:=j;
    for i:=1 to 4 do
      begin
        for j:=1 to 4 do
          Write (M[i,j]:3);
        Writeln
      end;
    end.
end.
```

### Поиск в массиве

Алгоритмы поиска рассматривались в параграфе 1.7.6 учебника для 10 класса. Напомним, что если значения массива не упорядочены по возрастанию или убыванию, то поиск искомой величи-



ны приходится выполнять методом последовательного просмотра всего массива. В упорядоченном массиве поиск можно ускорить, применив алгоритм бинарного поиска.

Еще одной часто решаемой задачей является поиск в массиве наибольшего или наименьшего значения.

### Пример 3

Оформить в виде процедуры подпрограмму поиска максимального элемента в одномерном массиве. Заполнить одномерный массив случайными числами (как в примере 1). С помощью процедуры найти в нем максимальное значение и индекс первого вхождения этого значения в массив.

```
Const n = 20;
Type: mas = array[1..n] of integer;
Var X: array[1..n] of integer; i, Xmax, imax: integer;

{Начало описания процедуры}
Procedure MaxArray(Var A: mas; Var MaxA, k: integer);
Var j: integer;
begin
  MaxA:=A[1]; k:=1;
  for j:=2 to n do
    if A[j]>MaxA then
      begin MaxA:=A[j]; k:=j end
  end; {Конец описания процедуры}

begin
  Randomize;
  for i:=1 to n do X[i]:=Random(100); {Заполнение массива}
  MaxArray(X, Xmax, imax);           {Обращение к процедуре}
  Write('Xmax=', Xmax, 'imax=', imax) {Вывод результатов}
end.
```

Процедура `MaxArray` имеет три параметра: *A* — исходный массив, *MaxA* — переменная для найденного максимального значения, *k* — переменная для индекса максимального значения. При обращении к процедуре им соответствуют фактические параметры: *X*, *Xmax*, *imax*. Размер массива определяется глобальной константой *n*, значение которой используется как в основной программе, так и в процедуре.

### Сортировка массива

В параграфе 1.7.7 учебника для 10 класса рассматривались два алгоритма сортировки: сортировка методом выбора максимального элемента и сортировка методом пузырька. Используем алгоритм метода пузырька для построчной сортировки матрицы.



**Пример 4**

Заполнить квадратную матрицу случайными целыми числами. Упорядочить каждую строку матрицы по возрастанию значений элементов.

```
Const n = 4;
Type: vector = array[1..n] of integer;
      matrix = array[1..n] of vector;
Var Matr: matrix; k, l: integer;

{Начало процедуры сортировки}
procedure SortVector(Var A: vector);
var i, j, X: integer; Flag: boolean;
begin
  Flag:=true; i:=1;
  while (i<=n-1) and Flag do
  begin
    Flag:=false;
    for j:=1 to n-i do
      if A[j]>A[j+1] then
        begin X:=A[j]; A[j]:=A[j+1]; A[j+1]:=X; Flag:=true end;
        i:=i+1
      end
    end; {Конец процедуры сортировки}

{Начало процедуры вывода матрицы}
Procedure PrintMatr(Var M: matrix);
Var i, j: integer;
begin
  for i:=1 to n do
  begin
    for j:=1 to n do Write (M[i][j]:3);
    Writeln
  end
end; {Конец процедуры вывода матрицы}

{Основная программа}
begin
  Randomize;
  {Заполнение матрицы}
  for k:=1 to n do
    for l:=1 to n do
      Matr[k][l]:=Random(10);

  {Вывод исходной матрицы}
  Writeln('Исходная матрица:'); PrintMatr(Matrx);
  {Построчная сортировка}
  for k:=1 to n do SortVector(Matrx[k]);
```



```
{Вывод отсортированной матрицы}
Writeln('Отсортированная матрица:'); PrintMatr(Matrx);
end.
```

В этой программе имеются две процедуры: `SortVector` — сортировка одномерного массива и `PrintMatr` — вывод матрицы. *Matrx* — сортируемая квадратная матрица размера  $n \times n$ . Каждая строка матрицы — это одномерный массив (вектор). Сортировка строк производится поочередным обращением к процедуре `SortVector` с указанием в переменной *k* номера сортируемой строки. После заполнения матрицы случайными числами она выводится на экран. После сортировки ее вывод повторяется. Чтобы не программировать дважды вывод матрицы, создана процедура `PrintMatr`. В основной программе к ней дважды происходит обращение. Исполнение этой программы на компьютере дало следующие результаты:

Исходная матрица:

```
4 1 3 9
4 2 9 0
8 4 3 7
3 6 2 9
```

Отсортированная матрица:

```
1 3 4 9
0 2 4 9
3 4 7 8
2 3 6 9
```

## Система основных понятий

### Задачи обработки массива

Типовые задачи:

- заполнение массива путем ввода, вычисления, случайными числами;
- поиск в массиве: заданного значения, максимального или минимального значения;
- сортировка массива

### Датчик случайных равномерно распределенных чисел

Целые числа: случайное число на отрезке  $[0, x - 1]$ :  
Функция `Random(x)`, где *x* — целое число

Вещественные числа: случайное число в полуинтервале  $(0, 1]$ :  
Функция `Random` (без аргумента)



## Вопросы и задания

1. Какими способами можно заполнить массив значениями?
2. Что такое датчик случайных чисел?
3. Как можно получать целые случайные числа на отрезке  $[-50, -1]$ ?
4. Как можно получать вещественные случайные числа в полуинтервале  $[2,5, 10)$ ?
5. Даны два вектора  $\{x_i\}$ ,  $\{y_i\}$ ,  $i = 1, \dots, 10$ , упорядоченные по возрастанию. Соедините их в один вектор  $\{z_i\}$ ,  $i = 1, \dots, 20$  так, чтобы сохранилась упорядоченность.
6. Дан массив, состоящий из 100 целых чисел:
  - а) выведите все числа, которые встречаются в этом массиве по несколько раз;
  - б) выведите все числа, которые встречаются в массиве только по одному разу.
7. В целочисленной матрице размером  $10 \times 10$  найдите максимальное значение и индексы всех элементов, равных ему.
8. Матрицу размером  $5 \times 10$  заполните случайными двоичными цифрами (0 и 1). Определите номер строки с наибольшим количеством нулей.
9. В двоичной матрице размером  $10 \times 10$  найдите совпадающие строки.

Практикум. Раздел «Программирование»

### 2.2.11. Метод последовательной детализации

В соответствии с методологией структурного программирования метод последовательной детализации является основным подходом при проектировании сложных алгоритмов. Суть метода состоит в следующем:

- 1) анализируется исходная задача, в ней выделяются подзадачи, строится иерархия таких подзадач (рис. 2.7);

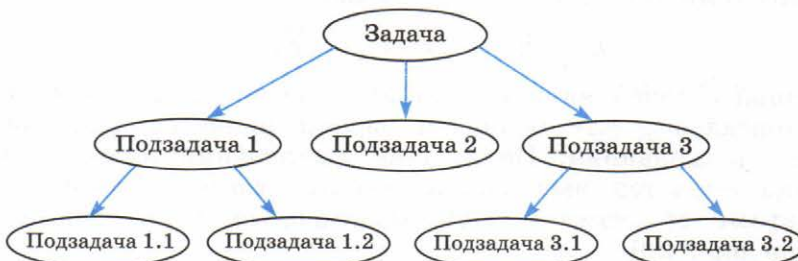


Рис. 2.7. Иерархия подзадач

- 2) составляются алгоритмы (программы), начиная с основного алгоритма (основной программы), далее — вспомогательные алгоритмы (подпрограммы) с последовательным углублением уровня.

Проиллюстрируем применение метода последовательной детализации на несложном примере.

### Пример 1

Вычислить площадь выпуклого  $N$ -угольника, заданного координатами своих вершин (рис. 2.8).

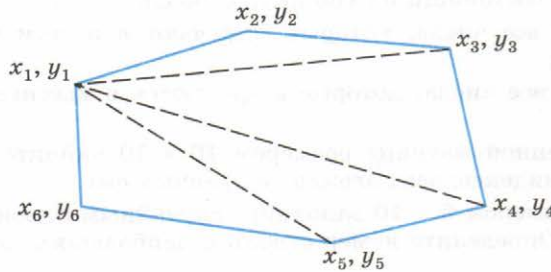


Рис. 2.8. Выпуклый многоугольник

Метод решения задачи заключается в следующем: выпуклый  $N$ -угольник разбивается диагональными линиями, выходящими из одной вершины, на  $N - 2$  треугольника. Площадь многоугольника вычисляется как сумма площадей треугольников. Площади треугольников вычисляются по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где  $p$  — полупериметр треугольника,  $a$ ,  $b$ ,  $c$  — длины сторон треугольника. Длины сторон вычисляются по формуле, следующей из теоремы Пифагора. Например, длина отрезка между точками с координатами  $(x_1, y_1)$ ,  $(x_2, y_2)$  равна:

$$L_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Основной задачей является вычисление площади многоугольника. Подзадачей для основной задачи является вычисление площади треугольника. Подзадачей вычисления площади треугольника является вычисление длины отрезка. Соотношение между этими задачами и соответствующими программами показано на рис. 2.9.

**Организация данных:** исходные данные — координаты вершин  $N$ -угольника будут храниться в двух массивах:  $X[1..N]$ ,  $Y[1..N]$ .





Рис. 2.9. Связи: структура задачи — структура программы

На первом шаге детализации составляется основная программа без подробного программирования используемых в ней подпрограмм первого уровня. Однако должны быть записаны интерфейсы подпрограмм первого уровня. **Интерфейс** здесь — это заголовок подпрограммы: имя и список формальных параметров. Интерфейсы необходимы для того, чтобы в основной программе можно было организовать обращения к подпрограммам первого уровня детализации.

В программе об  $N$ -угольнике подпрограмму `Treugolnik` сделаем процедурой.

```

Program N_ugolnik;
Const N=6;
Var X, Y: array[1..N] of real; S, SNugol: real; i: integer;
Procedure Treugolnik(Var x1, y1, x2, y2, x3, y3, R: real);
{блок процедуры не записывается}
begin
  {Ввод координат вершин многоугольника}
  for i:=1 to N do
    begin
      Write('X[' , i , ']='); Readln(X[i]);
      Write('Y[' , i , ']='); Readln(Y[i])
    end;
  SNugol:=0; {Переменная для вычисления площади фигуры}
  {Суммирование площадей треугольников}
  for i:=2 to N-1 do
    begin
      Treugolnik(X[1], Y[1], X[i], Y[i], X[i+1], Y[i+1], S);
      SNugol:=SNugol+S
    end;
  Writeln('Площадь фигуры =', Snugol)
end.
  
```

На втором шаге детализации запрограммируем процедуру `Treugolnik`. В разделе подпрограмм этой процедуры запишем лишь интерфейс подпрограммы `Line`, которую сделаем функцией.

```

Procedure Treugolnik(Var x1, y1, x2, y2, x3, y3, R: real);
Var L1, L2, L3, p: real;
Function Line(Var Xa, Ya, Xb, Yb: real): real;
{Блок функции не записывается}
begin
  L1:= Line(x1,y1,x2,y2); {Длина 1-й стороны}
  L2:= Line(x2,y2,x3,y3); {Длина 2-й стороны}
  L3:= Line(x1,y1,x3,y3); {Длина 3-й стороны}
  p:=(L1+L2+L3)/2;      {Полупериметр}
  R:=sqrt(p*(p-L1)*(p-L2)*(p-L3)) {Площадь треугольника}
end;

```

На третьем шаге детализации запрограммируем функцию `Line`. Формальные координаты концов отрезка заданы параметрами:  $(Xa, Ya)$  — первая точка,  $(Xb, Yb)$  — вторая точка.

```

Function Line(Var Xa, Ya, Xb, Yb: real): real;
begin
  Line:=sqrt(sqr(Xa-Xb)+sqr(Ya-Yb))
end;

```

Из составленных фрагментов собираем окончательный вариант программы.

```

Program N_ugolnik;
Const N=6;
Var X, Y: array[1..N] of real; S, SNugol: real; i: integer;
Procedure Treugolnik(Var x1, y1, x2, y2, x3, y3, R: real);
Var L1, L2, L3, p: real;
Function Line(Var Xa, Ya, Xb, Yb: real): real;
begin
  Line:=sqrt(sqr(Xa-Xb)+sqr(Ya-Yb))
end;
begin
  L1:= Line(x1,y1,x2,y2);
  L2:= Line(x2,y2,x3,y3);
  L3:= Line(x1,y1,x3,y3);
  P:=(L1+L2+L3)/2;
  R:=sqrt(p*(p-L1)*(p-L2)*(p-L3))
end;

```



```
begin
  for i:=1 to N do
    begin
      Write('X[' , i , ']='); Readln(X[i]);
      Write('Y[' , i , ']='); Readln(Y[i])
    end;
  SNugol:=0;
  for i:=2 to N-1 do
    begin
      Treugolnik(X[1], Y[1], X[i], Y[i], X[i+1], Y[i+1], S);
      SNugol:=SNugol+S
    end;
  Writeln('Площадь фигуры =', SNugol)
end.
```

В этой программе значение  $N$  может быть любым, начиная с 3, т. е.  $N \geq 3$ . Константа  $N$  описана глобально в основной программе, поэтому область этого описания распространяется на все подпрограммы. Все остальные величины в подпрограммах определены локально.

Показанный в рассмотренном примере способ построения программы называют еще **программированием «сверху вниз»**: начиная с основной программы, последовательно переходя к подпрограммам все более глубокого уровня детализации.

Работу полученной программы можно проверить на простом примере. Пусть  $N = 4$ . Вычислим площадь квадрата с длинами сторон, равными 2 и следующими координатами вершин:

```
X[1]=0, X[2]=0, X[3]=2, X[4]=2
Y[1]=0, Y[2]=2, Y[3]=2, Y[4]=0
```

После ввода этих значений в результате получим:

```
Площадь фигуры =4
```

Применение метода последовательной детализации позволяет разделить работу над большим программным проектом между несколькими программистами. Один человек — руководитель группы проектирует многоуровневую структуру алгоритма и составляет основную программу, а написание подпрограмм поручается другим членам группы. Для согласования работы программ договариваются лишь о интерфейсах: именах и параметрах подпрограмм. А внутреннее устройство подпрограммы — дело программиста, ее составляющего. При составлении больших проектов подпрограммы объединяются в модули.



## Система основных понятий

### Метод последовательной детализации

**Метод последовательной детализации** — один из основных методов структурного программирования, заключающийся в разработке сложных алгоритмов путем построения иерархии подзадач

**Программирование «сверху вниз»:** первой составляется основная программа, затем подпрограмма 1-го уровня детализации, затем — 2-го уровня и т. д. Подпрограммы последнего уровня содержат только простые команды, не вызывающие других подпрограмм

**Интерфейс подпрограммы:** это ее заголовок, позволяющий организовать обращение к подпрограмме (имя, тип, формальные параметры)



## Вопросы и задания

1. В чем заключается основная идея метода последовательной детализации?
2. Каким образом применение метода последовательной детализации позволяет организовать работу коллектива программистов над большим проектом?
3. Что такое интерфейс подпрограммы?



Практикум. Раздел «Программирование»

## 2.2.12. Символьный тип данных

Величина типа «символ» может принимать значения любых символов компьютерного алфавита. Символьная величина занимает 1 байт памяти, в котором хранится код этого символа, соответствующий используемой кодовой таблице. Заметим, что в Delphi наряду с однобайтовой кодировкой символов используется и двухбайтовая.

Символьная константа записывается между апострофами. Например: 'R', '+', '9', ']'.

Символьный тип называется `char`. Пример описания символьных переменных:

```
var c1, c2: char;
```

Символьный тип относится к порядковым типам данных. Из этого следует:

- символы — упорядоченное множество;
- у каждого символа в этом множестве есть свой порядковый номер;



- между символами работает соотношение «следующий — предыдущий».

Порядковый номер символа — это его десятичный код, который лежит в диапазоне от 0 до 255. Например, в кодовой таблице ASCII десятичный код латинской буквы «А» равен 65, а цифры «5» — 53. О стандартах кодирования символов подробно рассказывалось в параграфе 1.4.2 учебника для 10 класса.

#### Функция Ord (x)

Ord(x) — функция от аргумента порядкового типа, которая возвращает порядковый номер значения x в этом типе данных. Если x — символьная величина, то результатом функции будет десятичный код x в кодовой таблице. Например:

Ord('A')=65, Ord('5')=53

#### Функция Chr (x)

Chr(x) — функция от целочисленного аргумента, результатом которой является символ с кодом, равным x. Например:

Chr(65)='A', Chr(53)='5'.

Поскольку коды символов лежат в диапазоне от 0 до 255, желательно тип x определять либо как byte, либо как ограниченный тип 0..255. В параграфе 1.4.2 учебника для 10 класса содержится программа получения таблицы кодировки символов с номерами от 20 до 255. В этой программе последовательность кодов и соответствующих им символов получается в следующем цикле:

```
for kod:=20 to 255 do Write(Chr(kod):3, kod:4);
```

Напомним, что коды, меньшие 20, являются управляющими и на экране не отражаются.

Функция Chr является обратной к функции Ord. Отсюда следует:  $Chr(Ord(x))=x$ . Например:  $Chr(Ord('A'))='A'$ .

#### Принцип последовательного кодирования алфавитов

*В любой кодовой таблице выполняется принцип последовательного кодирования латинского (английского) алфавита и алфавита десятичной системы счисления. Это важное обстоятельство, которое часто учитывается в программах обработки символьной информации.*

При выполнении операций отношений применительно к символьным величинам учитываются коды этих величин. Чем больше значение кода, тем символ считается больше. Истинными являются следующие отношения: 'A' < 'B', 'Z' > 'Y', 'a' > 'A'. Значение символьной переменной C является заглавной латинской буквой, если истинно логическое выражение:

$(C >= 'A')$  and  $(C <= 'Z')$

Значение символьной переменной *C* является цифрой, если истинно логическое выражение:

```
(C>='0') and (C<='9').
```

В латинском алфавите 26 букв. Поэтому разница между кодами букв «Z» и «A», а также «z» и «a» равна 25.

### Задача 1

С помощью датчика случайных чисел заполнить массив *Sim*[0..10] строчными английскими буквами. Затем массив отсортировать в алфавитном порядке.

```
Uses CRT;
```

```
Var Sim: array[0..10] of char;
```

```
    C: char; i, k: integer;
```

```
begin
```

```
  ClrScr;
```

```
  Randomize; {Заполнение массива случайными буквами}
```

```
  Writeln('Исходный массив:');
```

```
  for i:= 0 to 10 do
```

```
  begin
```

```
    Sim[i]:=Chr(Random(26)+Ord('a'));
```

```
    Write(Sim[i])
```

```
  end;
```

```
  Writeln;
```

```
  {Сортировка методом пузырька}
```

```
  for i:=0 to 9 do
```

```
    for k:=0 to 9-i do
```

```
      if Sim[k]>Sim[k+1] then
```

```
      begin
```

```
        C:=Sim[k]; Sim[k]:=Sim[k+1]; Sim[k+1]:=C
```

```
      end;
```

```
  Writeln('Отсортированный массив:');
```

```
  for i:=0 to 10 do Write(Sim[i]);
```

```
end.
```

При тестировании программы было получено:

Исходный массив:

```
gnkbeqgmsin
```

Отсортированный массив:

```
beggikmnnqs
```

### Задача 2

На вход программе подаются строчные английские буквы. Ввод этих символов заканчивается точкой (другие символы, отличные от «.» и букв «a»..«z», во входных данных отсутствуют). Написать программу на Паскале, которая будет выводить буквы,



встречающиеся во входной последовательности, в порядке уменьшения частоты их встречаемости. Каждая буква должна быть выведена один раз. Точка при этом не учитывается<sup>1)</sup>.

*Идея алгоритма.* Формируется массив символов английского алфавита от «а» до «z». Дадим массиву имя *Alf*, а элементы пронумеруем от 0 до 25. В другом массиве *Schet*[0..25] будем вести счетчики повторений каждой буквы в последовательности вводимых символов. В *Schet*[0] — счетчик «а», в *Schet*[1] — счетчик «б» и т. д. В начале счетчики обнуляются.

В цикле посимвольно вводятся данные и подсчитывается количество каждой буквы в своем счетчике. Цикл заканчивается, когда будет введена точка. Затем сортируется массив счетчиков по убыванию значений. Применяется алгоритм сортировки методом пузырька. Одновременно с перестановками в массиве *Schet* производятся аналогичные перестановки в массиве *Alf*. В конце по порядку выводятся оба этих массива, при этом пропускаются счетчики, равные нулю, и соответствующие им буквы.

```
Program Bukvy;
Uses CRT;
Var Alf: array[0..25] of char;
    Schet: array[0..25] of byte;
    X, i, k: byte;
    C: char; Flag: boolean;
begin
  ClrScr;
  {Заполнение массива Alf буквами алфавита}
  {и массива Schet нулями}
  for i:=0 to 25 do
  begin Alf[i]:=Chr(Ord('a')+i); Schet[i]:=0 end;
  Readln(C); {Ввод первого символа}
  {Циклический ввод символов до точки}
  while C<>'.' do
  begin
    k:=Ord(C)-Ord('a'); {Номер символа C}
    Schet[k]:=Schet[k]+1; {+1 в счетчик данного символа}
    Readln(C); {Ввод очередного символа}
  end;
  {Сортировка массива Schet методом пузырька}
  Flag:=true; i:=0;
  while (i<=24) and Flag do
  begin
    Flag:=false;
```

<sup>1)</sup> Задача взята из демоверсии ЕГЭ по информатике.

```

for k:=0 to 24-i do
  if Schet[k]<Schet[k+1] then
    begin X:=Schet[k]; Schet[k]:=Schet[k+1]; Schet[k+1]:=X;
          C:=Alf[k]; Alf[k]:=Alf[k+1];
          Alf[k+1]:=C;
          Flag:=true
    end;
  i:=i+1
end;
for i:=0 to 25 do
  if Schet[i]>0 then Writeln(Alf[i]:2, '-', Schet[i]:3)
end.

```

При тестировании программы была введена последовательность символов:  
asadsdghka.

В результате получено:

```

a-3
d-2
s-2
g-1
h-1
k-1

```

## Система основных понятий

### Символьный тип данных

**Величины символьного типа (char):** константы и переменные, принимающие значения символов компьютерного алфавита

1 символ занимает 1 байт памяти (в 8-битовых кодировках)

**Ord(x)** — функция от аргумента порядкового типа, которая возвращает порядковый номер значения  $x$  в этом типе данных. Если  $x$  — символьная величина, то функция возвращает код символа

**Chr(x)** — функция от целочисленного аргумента, результатом которой является символ с кодом, равным  $x$

В любой кодовой таблице выполняется **принцип последовательного кодирования** латинского (английского) алфавита и алфавита десятичной системы счисления

## Вопросы и задания

1. Как в программе на Паскале обозначаются символьные константы и переменные?
2. С помощью какой стандартной функции определяется код символа?



3. С помощью какой стандартной функции можно определить символ по его коду?
4. Что такое принцип последовательного кодирования алфавитов? Приведите примеры алгоритмов, где он может быть использован.
5. Определите результаты вычисления выражений (типы и значения):
  - 1) `Chr(Ord('B'))`;
  - 2) `Ord('A')-Ord('Z')`;
  - 3) `Ord('A')-Ord('a')= Ord('Z')-Ord('z')`;
  - 4) `Ord('9')-Ord('0')`;
  - 5) `Chr(Ord('a')+Ord('R')-Ord('r'))`.

### 2.2.13. Строки символов

Рассмотрим еще один структурный тип данных — строковый тип. **Строковый тип данных** был введен в Турбо Паскале. Он позволяет программировать обработку слов, предложений, текстов.

**Строка** — это последовательность символов. Каждый символ занимает 1 байт памяти (код ASCII). Количество символов в строке называется ее длиной. Длина строки может находиться в диапазоне от 0 до 255. Строковые величины могут быть константами и переменными.

**Строковая константа** записывается как последовательность символов, заключенная в апострофы. Например:

```
' Язык программирования ПАСКАЛЬ '  
' IBM PC - computer '  
'33-45-12'
```

**Строковая переменная** описывается в разделе описания переменных следующим образом:

```
Var <идентификатор>: string[<максимальная длина строки>]
```

Например:

```
Var Name: string[20]
```

Параметр длины может и не указываться в описании. В таком случае подразумевается, что он равен максимальной величине — 255. Например:

```
Var slovo: string
```

Строковая переменная занимает в памяти на 1 байт больше, чем указанная в описании длина. Дело в том, что один (нулевой) байт содержит значение *текущей длины строки*. Если строковой переменной не присвоено никакого значения, то ее текущая длина равна нулю. По мере заполнения строки символами ее текущая длина возрастает, но она не должна превышать максимальной по описанию величины.



Символы внутри строки индексируются (нумеруются), начиная с единицы. Каждый отдельный символ идентифицируется именем строки с индексом, заключенным в квадратные скобки. Например:

```
Name[5], Name[i], slovo[k+1].
```

Значение индекса может быть задано положительной константой, переменной, выражением целочисленного типа. Оно не должно выходить за границы описания.

Тип `string` и стандартный тип `char` совместимы: строки и символы могут употребляться в одних и тех же выражениях.

Строковые выражения строятся из строковых констант, переменных, функций и знаков операций. Над строковыми данными допустимы операции сцепления и операции отношения.

**Операция сцепления (+)** применяется для соединения нескольких строк в одну результирующую строку. Сцеплять можно как строковые константы, так и переменные.

Например:

```
'ЭВМ'+ ' IBM'+ ' PC'
```

В результате получится строка:

```
'ЭВМ IBM PC'
```

Длина результирующей строки не должна превышать 255.

**Операции отношения** : `=`, `<`, `>`, `<=`, `>=`, `<>` производят сравнение двух строк, в результате чего получается логическая величина (*true* или *false*). Операция отношения имеет более низкий приоритет, чем операция сцепления. Сравнение строк производится слева направо до первого несовпадающего символа, и та строка считается больше, в которой первый несовпадающий символ имеет больший номер в таблице символьной кодировки.

Если строки имеют различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная. Строки равны, если они полностью совпадают по длине и содержат одни и те же символы.

### **Пример**

Выражение:	Результат:
'cosm1' < 'cosm2'	True
'pascal' > 'PASCAL'	True
'Ключ_' > 'Ключ'	True
'MS DOS' = 'MS DOS'	True



**Функции и процедуры**

Функция  $\text{Copy}(S, \text{Poz}, N)$  выделяет из строки  $S$  подстроку длиной  $N$  символов, начиная с позиции  $\text{Poz}$ .  $N$  и  $\text{Poz}$  — целочисленные выражения.

**Пример**

Значение $S$ :	Оператор:	Результат:
'ABCDEFGH'	$\text{Copy}(S, 2, 3)$	'BCD'
'ABCDEFGH'	$\text{Copy}(S, 4, 4)$	'DEFG'

Функция  $\text{Concat}(S_1, S_2, \dots, S_N)$  выполняет сцепление (конкатенацию) строк  $S_1, \dots, S_N$  в одну строку.

**Пример**

Выражение:	Результат:
$\text{Concat}('AA', 'XX', 'Y')$	'AAXXY'

Функция  $\text{Length}(S)$  определяет текущую длину строки  $S$ . Результат — значение целочисленного типа.

**Пример**

Значение $S$ :	Оператор:	Результат:
'test-5'	$\text{Length}(S)$	6
'(A+B)*C'	$\text{Length}(S)$	7

Функция  $\text{Pos}(S_1, S_2)$  обнаруживает первое появление в строке  $S_2$  подстроки  $S_1$ . Результат — целое число, равное номеру позиции, где находится первый символ подстроки  $S_1$ . Если в  $S_2$  не обнаружена подстрока  $S_1$ , то результат равен 0.

**Пример**

Значение $S_2$ :	Оператор:	Результат:
'abcdef'	$\text{Pos}('cd', S_2)$	3
'abcdcdef'	$\text{Pos}('cd', S_2)$	3
'abcdef'	$\text{Pos}('k', S_2)$	0

Процедура  $\text{Delete}(S, \text{Poz}, N)$  — удаление  $N$  символов из строки  $S$ , начиная с позиции  $\text{Poz}$ .

**Пример**

Исходное значение:	Оператор:	Конечное значение:
'abcdefg'	$\text{Delete}(S, 3, 2)$	'abefg'
'abcdefg'	$\text{Delete}(S, 2, 6)$	'a'

В результате выполнения процедуры уменьшается текущая длина строки в переменной  $S$ .

Процедура  $\text{Insert}(S1, S2, \text{Poz})$  — вставка строки  $S1$  в строку  $S2$ , начиная с позиции  $\text{Poz}$ .

### Пример

Начальное $S2$ :	Оператор:	Конечное $S2$ :
'ЭВМ PC'	$\text{Insert}('IBM-', S2, 5)$	'ЭВМ IBM-PC'
'Рис.2'	$\text{Insert}('N', S2, 6)$	'Рис.N2'

## Примеры программ обработки строк

### Пример 1

Составить программу, формирующую символьную строку, состоящую из  $N$  звездочек ( $N$  — целое число,  $1 \leq N \leq 255$ ).

```

Program Stars;
Var A: string;
    N, I: byte;
begin
  Write('Введите число звездочек');
  ReadLn(N);
  A:='';
  for I:=1 to N do
    A:=A+'*';
  WriteLn(A)
end.

```

Здесь строковой переменной  $A$  вначале присваивается значение пустой строки, обозначаемой двумя апострофами (''). Затем к ней присоединяются звездочки.

### Пример 2

В символьной строке подсчитать количество цифр, предшествующих первому символу «!».

```

Program C;
Var S: string;
    K, i: byte;
begin
  WriteLn('Введите строку');
  ReadLn(S);
  K:=0; i:=1;
  while (i<=Length(S)) and (S[i]<>'!') do
  begin
    if (S[i]>='0') and (S[i]<='9')

```



```

    then K:=K+1;
    i:=i+1
end;
WriteLn ('Количество цифр до символа "!" равно ', K)
end.

```

В этой программе переменная  $K$  играет роль счетчика цифр, а переменная  $i$  — роль параметра цикла. Цикл закончит выполнение при первом же выходе на символ '!' или если в строке такого символа нет, то при выходе на конец строки. Символ  $S[I]$  является цифрой, если истинно отношение:  $'0' < S[I] < '9'$ .

### Пример 3

В этом примере запрограммируем одну из функций текстового редактора: выравнивание строки по ширине. Например, дан текст, состоящий из четырех строк:

```

Буря мглою небо кроет,
Вихри снежные крутя.
То как зверь она завоет,
То заплачет, как дитя.

```

Строки выровнены по левому краю. Пусть длина строки должна быть равна  $L$ . Нужно путем равномерной вставки пробелов между словами растянуть строки таким образом, чтобы последний символ каждой строки оказался в позиции номер  $L$ . Например, если  $L = 30$ , то текст должен приобрести такой вид:

```

Буря   мглою   небо   кроет,
Вихри   снежные   крутя.
То   как   зверь   она   завоет,
То   заплачет,   как   дитя.

```

Будем строить программу методом последовательной детализации. Сначала составим основную программу, которая будет вводить исходный текст в массив *Tekst*, состоящий из  $N$  строк, последовательно выравнивать строки путем обращения к процедуре *RovStr* и выводить преобразованный текст на экран.

```

Program Editor;
Const N=4; L=30;
Var Tekst: array[1..N] of string;
    i: byte;
procedure RovStr(Var Str: string);
{Блок процедуры RovStr}
begin
    {Ввод текста}
    Writeln('Введите текст');
    for i:=1 to N do Readln(Tekst[i]);

```



```

{Выравнивание строк}
for i:=1 to N do RovStr(Tekst[i]);
{Вывод после выравнивания}
for i:=1 to N do Writeln(Tekst[i])
end.

```

Здесь константа  $N$  указывает на число строк в тексте, константа  $L$  — на длину строки. Процедура имеет один параметр — строку  $Str$ . Она представляет одновременно и входные, и выходные данные. На входе — невыровненная строка, на выходе — эта же строка после выравнивания.

На втором шаге детализации запрограммируем процедуру  $RovStr$ . Идея алгоритма состоит в следующем: в промежутки между словами вставляются по очереди дополнительные пробелы до тех пор, пока длина строки не станет равной  $L$ .

```

Procedure RovStr(Var Str: string);
Var j: byte;
Begin
  j:=1; {Номер первого символа в строке}
  {Цикл, пока длина строки не станет равной L}
  while length(Str)<L do
  begin
    j:=j mod length(Str);
    {Обнаружение конца слова}
    {и вставка дополнительного пробела}
    if (Str[j]<>' ') and (Str[j+1]=' ') then
    begin Insert(' ',Str,j+1); j:=j+1 end;
    j:=j+1 {Номер следующего символа}
  end
end;

```

Постарайтесь самостоятельно понять назначение следующего оператора в этой программе:

```
j:=j mod length(Str);
```

Объединив основную программу с процедурой, получаем окончательный текст программы:

```

Program Editor;
Const N=4; L=30;
Var Tekst: array[1..N] of string;
    i: byte;
Procedure RovStr(Var Str: string);
Var j: byte;
begin
  j:=1;
  while length(Str)<L do

```





```

begin
  j:=j mod length(Str);
  if (Str[j]=' ') and (Str[j+1]=' ') then
    begin Insert(' ',Str,j+1); j:=j+1 end;
  j:=j+1 {Номер следующего символа}
end
end;
begin
  Writeln('Введите текст');
  for i:=1 to N do Readln(Tekst[i]);
  for i:=1 to N do RovStr(Tekst[i]);
  for i:=1 to N do Writeln(Tekst[i])
end.

```

### Система основных понятий

Строки символов
<b>Строка</b> — последовательность символов
<b>Описание строковой переменной:</b> <b>Var</b> <идентификатор>: string[<длина строки>] Максимальная длина строки — 255
<b>Обозначение символа в строке:</b> <идентификатор строки>[<индекс>]
<b>Операции над строками:</b> сцепление (+), отношение (=, <, >, <=, >=, <>)
<b>Стандартные функции:</b> Copy(S, Poz, N) — выделение подстроки; Concat(S1, S2, ..., SN) — сцепление (конкатенация) строк; Length(S) — определение текущей длины строки; Pos(S1, S2) — определение первого вхождения подстроки в строку
<b>Стандартные процедуры:</b> Delete(S, Poz, N) — удаление подстроки; Insert(S1, S2, Poz) — вставка подстроки

### Вопросы и задания

1. Как в программе обозначается строковая константа; как определяется строковая переменная?
2. Какой может быть максимальная длина строки?
3. Составьте программу получения из слова «дисковод» слова «воск», используя операцию сцепления и функцию Copy.
4. Составьте программу получения слова «правило» из слова «операция», используя процедуры Delete, Insert.

5. В данном слове замените первый и последний символы на '\*'.
6. В заданном слове произведите обмен первого и последнего символов.
7. К заданному слову присоедините столько символов '!', сколько в нем имеется букв (например, из строки 'УРА' надо получить 'УРА!!!').
8. В заданной строке вставьте пробел после каждого символа.
9. Переверните введенную строку (например, из 'ДИСК' должно получиться 'КСИД').
10. В заданной строке удалите все пробелы.
11. Строка представляет собой запись целого числа. Составьте программу ее перевода в соответствующую величину целого типа.

Практикум. Раздел «Программирование»

### 2.2.14. Комбинированный тип данных

Все структурные типы данных, с которыми вы уже познакомились (массивы, строки), представляют собой совокупности однотипных величин. **Комбинированный тип данных** — это структурный тип, состоящий из фиксированного числа компонентов (полей) разных типов.

Комбинированный тип объявляется в программе в разделе типов:

```
Type <имя> = record
    <имя поля 1>: <тип>;
    ...
    <имя поля N>: <тип>
end
```

Поля могут иметь любые типы, в том числе и комбинированный тип.

Например, данные о результатах экзаменов, полученных учеником по трем предметам, могут быть представлены одной величиной комбинированного типа:

```
Type results = record
    Family: string[15]; {Фамилия ученика}
    Rus: 2..5;          {Оценка по русскому языку}
    Alg: 2..5;          {Оценка по алгебре}
    Phiz: 2..5          {Оценка по физике}
end;
```

После этого в разделе переменных следует описание:

```
Var exam: results;
```

Величина комбинированного типа называется **записью**. Элементы записи идентифицируются составными именами следующей структуры:

```
<переменная комбинированного типа>.<имя поля>
```



Например: exam.family, exam.rus

В программе может использоваться массив, элементами которого являются записи.

### Пример 1

На экзаменационном листе содержатся сведения о результатах экзаменов, сданных 30 учениками класса. Ввести эти данные в компьютер и получить список всех отличников.

В программе используется описание комбинированного типа *result1*, приведенное выше. Исходные данные организуются в массив следующей структуры.

```
Var list: array[1..30] of results;
```

После ввода в этот массив исходных данных следует фрагмент программы:

```
for i:=1 to 30 do
  if (list[i].rus=5) and (list[i].alg=5) and (list[i].phiz=5)
  then Writeln(list[i].family);
```

Программа отбирает записи, в которых все поля с оценками равны 5, и выводит соответствующие поля фамилий.

А теперь обсудим проблему: как наиболее удобным способом организовать ввод данных в этой программе? Вводить с клавиатуры неудобно из-за большого объема данных. При каждом повторном запуске программы нужно начинать ввод сначала. А при отладке это наверняка придется делать многократно. Гораздо удобнее подготовить файл с исходными данными с помощью текстового редактора. После этого без проблем можно повторять ввод многократно. Так и поступим. Подготовим текстовый файл следующего вида:

Таблица успеваемости 10А класса

Фамилия	Русский язык	Алгебра	Физика
Антонов			
	4	5	5
Андреева			
	5	3	4
Боброва			
	5	5	5
...			

Таблица содержит данные с фамилиями и оценками 30 учеников класса. Обратите внимание на то, что фамилии записываются в отдельных строках. Необходимость этого связана с реализацией алгоритма (см. далее): при вводе символьной строки прочитыва-

ется полностью очередная строка текстового файла до признака EOLN. При этом фамилии должны содержать не более 15 символов, а первые оценки (по русскому языку) — располагаться не раньше 16-й позиции в своей строке.

Сохраним этот файл в корневом каталоге логического диска E: под именем 10\_a.txt. Составим программу с вводом таблицы успеваемости и выводом списка отличников. Фамилии отличников выведем на экран и сохраним в файле с именем Best.txt.

```

Program Examen;
Type results = record
    Fam: string[15];
    Rus: 2..5;
    Alg: 2..5;
    Phiz: 2..5
end;
Var list: array[1..30] of results; {Массив записей}
    i: integer; F1, F2: text;
begin
    Assign(F1, 'E:\10_a.txt'); {Связывание F1 с файлом 10_a.txt}
    Assign(F2, 'E:\Best.txt'); {Связывание F2 с файлом Best.txt}
    Reset(F1); {Открытие файла F1 для чтения}
    Rewrite(F2); {Открытие файла F2 для записи}
    Readln(F1); Readln(F1); {Пропуск 2-х строк в файле F1}
    {Цикл ввода из файла F1}
    for i:=1 to 30 do
        Readln(F1, list[i].Fam, list[i].Rus, list[i].Alg, list[i].Phiz);
    {Цикл отбора отличников и вывода их фамилий}
    for i:=1 to 30 do
        if (list[i].rus=5) and (list[i].alg=5) and
            (list[i].phiz=5)
        then
            begin
                Writeln(list[i].fam); {Вывод фамилии на экран}
                Writeln(F2, list[i].fam) {Запись фамилии в файл F2 }
            end;
        Close(F1); Close(F2) {Закрытие файлов}
    end.

```

### Пример 2

Решая рассмотренную задачу с оценками, можно обойтись без массива записей. Кроме того, можно не ставить ограничения на число учеников в классе. Их число выяснится в процессе чтения



файла с таблицей успеваемости. Составим программу, которая кроме вывода списка фамилий отличников подсчитает их количество и процент отличников по отношению к полному составу класса.

```
Program Examen_2;  
Type results = record  
    Fam: string[15];  
    Rus: 2..5;  
    Alg: 2..5;  
    Phiz: 2..5  
end;  
Var list: results; {Одна переменная комбинированного типа}  
    I, K: integer;  
    F1, F2: text;  
begin  
    Assign(F1, 'E:\10_a.txt');  
    Assign(F2, 'E:\Best.txt');  
    Reset(F1); Rewrite(F2);  
    Readln(F1); Readln(F1);  
    I:=0; K:=0; {Инициализация счетчиков}  
    {Цикл до конца чтения файла}  
    while not EOF(F1) do  
    begin  
        Readln(F1, list.Fam, list.Rus, list.Alg, list.Phiz);  
        I:=I+1; {Подсчет числа учеников}  
        if (list.rus=5) and (list.alg=5) and (list.phiz=5) then  
        begin  
            Writeln(list.fam);  
            Writeln(F2, list.fam);  
            K:=K+1; {Подсчет числа отличников}  
        end  
    end;  
    Writeln('Из ', I, ' учеников в классе ', K,  
        ' отличников, что составляет ', K/I*100:4:1, '%');  
    Close(F1); Close(F2) {Закрытие файлов}  
end.
```

В этой программе переменная  $I$  используется как счетчик числа учеников, а переменная  $K$  — как счетчик числа отличников.

Стандартная логическая функция EOF (end of file) примет значение *true*, когда процесс чтения из файла дойдет до его конца.

В результате выполнения программы кроме списка отличников на экран выведется строка:

Из 30 учеников в классе 10 отличников, что составляет 33,3%.

**Система основных понятий**

Комбинированный тип данных
<b>Комбинированный тип данных</b> — структурный тип, объединяющий разнотипные компоненты (поля) данных
<b>Тип поля:</b> любой простой или структурированный тип (кроме файлового)
<b>Запись</b> — величина комбинированного типа
<b>Идентификация поля записи</b> — составное имя: <имя записи>.<имя поля>

**Вопросы и задания**

1. Чем комбинированный тип данных отличается от регулярного типа данных (массива)?
2. Что такое запись?
3. Опишите комбинированный тип для записей, содержащих следующие данные учеников: фамилию, имя, год рождения, рост (в сантиметрах), вес (в килограммах).
4. Опишите содержимое текстового файла, из которого будут вводиться данные, соответствующие описанию из предыдущего задания, для нескольких учеников класса (не менее пяти).
5. Напишите программу, по которой будут введены данные из файла, описанного в предыдущем задании, и выполнена следующая обработка:
  - определение среднего роста и среднего веса всех учеников;
  - вывод на экран и в файл `rost.txt` списка (фамилии, имена, возраст) учеников, рост которых выше среднего;
  - вывод на экран и в файл `ves.txt` списка учеников, вес которых ниже среднего.
6. Решите предыдущую задачу, не используя в программе массив записей. Подсказка: оператор `Reset` можно использовать в программе многократно для повторного чтения файла с его начала.

Практикум. Раздел «Программирование»





## 2.3. Рекурсивные методы программирования

### 2.3.1. Рекурсивные подпрограммы

#### Частично рекурсивная функция

В параграфе 2.2.7 было введено понятие **рекуррентной последовательности** на примере следующего числового ряда:

$$a_0 = 1, \quad a_1 = \frac{1}{1!}, \quad a_2 = \frac{1}{2!}, \quad a_3 = \frac{1}{3!}, \dots, a_n = \frac{1}{n!}, \dots$$

Было показано, что вычисление элементов этого ряда можно производить по формуле:

$$a_i = \begin{cases} 1, & \text{при } i = 0; \\ \frac{a_{i-1}}{i}, & \text{при } i > 0. \end{cases}$$

Такая формула называется **одношаговой рекуррентной формулой**. Одношаговость обозначает тот факт, что имеется одно начальное значение и каждое следующее значение ряда вычисляется по одному предыдущему значению.

Будем рассматривать  $a_n$  в качестве значений функции  $F(n)$  от целого аргумента  $n$ . Определение этой функции будет выглядеть так:

$$F(n) = \begin{cases} 1, & \text{при } n = 0; \\ F(n-1)/n, & \text{при } n > 0. \end{cases}$$

Определение функции через саму себя (здесь  $F(n)$  определяется через  $F(n-1)$ ) называется в математике **рекурсивным определением**. Для того чтобы рекурсивно-определенная функция была вычислимой, она должна иметь некоторое известное начальное (частное) значение. В нашем примере это  $F(0) = 1$ . Вычисление на компьютере таких функций можно производить с помощью **рекурсивных алгоритмов**, которые на языках программирования реализуются через **рекурсивные подпрограммы**.

#### Рекурсивные подпрограммы-функции и процедуры

В описаниях подпрограмм-функций на Паскале допускается использование в теле функции вызова этой же самой функции. Такая подпрограмма-функция называется рекурсивной.

**Пример 1**

Вот два варианта подпрограммы-функции вычисления  $F(n) = 1/n!$

Нерекурсивная функция	Рекурсивная функция
<pre>Function FN(n: integer): real; Var i: integer; F: real; begin   F:=1;   for i:=1 to n do F:=F/i;   FN:=F end;</pre>	<pre>Function FN(n: integer): real; begin   if n=0 then FN:=1   else FN:=FN(n-1)/n end;</pre>

Первый, нерекурсивный вариант подпрограммы имеет циклическую структуру. Во втором, рекурсивном варианте используется ветвление. На положительной ветви (**then**) значение функции вычисляется явно ( $FN:=1$ ), на отрицательной ветви (**else**) происходит обращение функции к себе самой с уменьшенным на единицу значением аргумента.

Пусть в основной программе для вычисления  $1/3!$  имеется следующий оператор присваивания:  $P:=FN(3)$ . При его выполнении произойдет цепочка обращений к рекурсивной функции  $FN$  с последующим возвратом:

Вход в рекурсию  $\rightarrow FN(3) \rightarrow FN(2) \rightarrow FN(1) \rightarrow FN(0):=1 \rightarrow$   
 Результат  $0,1666.. \leftarrow FN(3)=0,5/3 \leftarrow FN(2)=1/2 \leftarrow FN(1)=1/1 \leftarrow$  возврат

При каждом обращении к функции в специальный раздел памяти помещается необходимая информация для реализации обратного процесса вычислений: адрес команды, к которой надо вернуться, ячейки для размещения промежуточных значений функции и т. д. Такой раздел памяти называется **стеком**. После выхода на граничное значение (присваивание единицы) с помощью стека происходит обратный процесс последовательного вычисления промежуточных значений функции, пока не будет получен окончательный результат. Выполнение рекурсивно определенной функции займет на компьютере больше времени, чем функции без рекурсии.

Вычисление частично-рекурсивной функции может быть реализовано также и в форме рекурсивно определенной процедуры. Ниже показан пример программы, в которой описана рекурсивная процедура вычисления  $1/n!$ . С помощью этой процедуры вычисляется значение  $1/5!$ .



```

Var X: real;
Procedure FN(n: integer; var F: real);
begin
  if n=0 then F:=1
  else begin FN(n-1,F); F:=F/n end
end;
begin
  FN(5,X); Write(X)
end.

```

В результате выполнения программы получено число: 0.00833333333333.

### Пример 2

Вспомним задачу вычисления наибольшего общего делителя двух чисел, описанную в параграфе 2.2.8. Задача решается с помощью алгоритма Евклида. Модифицированный алгоритм Евклида основан на следующей системе равенств:

$$\text{НОД}(M, N) = \begin{cases} M, & \text{при } N = 0; \\ \text{НОД}(N, M \bmod N), & \text{при } N \neq 0. \end{cases}$$

Но это частично-рекурсивное определение функции НОД от двух аргументов! Рекурсивная подпрограмма-функция получается путем непосредственной реализации этого определения в форме оператора ветвления. В следующей программе описана рекурсивная функция и обращение к ней.

```

Function NOD(M, N: integer): integer;
begin
  if N=0 then NOD:=M
  else NOD:=NOD(N, M mod N)
end;
begin
  Writeln(NOD(12, 32))
end.

```

В результате выполнения программы получим число 4. В этой рекурсивной функции, как и в примере с факториалом, дано одно начальное значение. А теперь рассмотрим пример с двумя начальными значениями.

### Пример 3

С XIII века в математике известна замечательная числовая последовательность, названная именем своего автора: числа Фибоначчи. У этой последовательности имеется множество приложений, с одним из которых вы познакомились в 10 классе: это

основание фибоначчиевой системы счисления. Первые два значения числового ряда Фибоначчи равны единице. Каждое следующее значение равно сумме двух предыдущих. Если функцию вычисления  $n$ -го элемента ряда обозначить как  $Fib(n)$ , то ее математическое определение запишется так:

$$Fib(n) = \begin{cases} 1, & \text{при } n=1 \text{ и } n=2; \\ Fib(n-1) + Fib(n-2), & \text{при } n > 2. \end{cases}$$

Очевидно, что это частично-рекурсивная функция с двумя начальными (граничными) значениями. На основании данной формулы можно запрограммировать рекурсивную подпрограмму-функцию на Паскале. В следующей программе описана такая функция и с ее помощью вычислены первые 10 чисел Фибоначчи:

```

Program Fibonachi;
Var i: integer;
Function Fib(n: integer): integer;
begin
    if (n=1) or (n=2) then Fib:=1
    else Fib:=Fib(n-1)+Fib(n-2)
end;
begin
    for i:=1 to 10 do Write(Fib(i):4)
end.

```

В результате выполнения программы получим числовую последовательность первых десяти чисел Фибоначчи:

1 1 2 3 5 8 13 21 34 55

При выполнении этой программы компьютер будет строить два стека, поскольку в определении функции присутствуют два рекурсивных обращения к ней самой. Вопреки лаконичной простоте текста программы, ее выполнение — достаточно тяжелый процесс для компьютера с точки зрения времени счета и расхода памяти на построение стека.

#### Пример 4

Снова вернемся к задаче из параграфа 2.2.7 — вычислению суммы числового ряда вида:

$$1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{n!} + \dots$$

В примере 1 мы получили частично-рекурсивную функцию для вычисления слагаемых этой суммы. Оказывается, что и саму сумму можно вычислить как частично-рекурсивную функцию. Если для слагаемых использовать обозначения:  $a_0 = 1$ ,  $a_n = 1/n!$  для



$n = 1, 2, \dots$ , а сумму обозначить как функцию  $S(n)$ , то будет справедливым следующее определение этой функции:

$$S(n) = \begin{cases} 1, & \text{при } n = 0; \\ S(n-1) + a_n, & \text{при } n > 0. \end{cases}$$

Ниже приведены два варианта решения этой задачи. Первый — циклическая программа из параграфа 2.2.7. Во втором варианте используется рекурсивная подпрограмма-функция SUM для вычисления  $S(n)$ . В ней, в свою очередь, использована рекурсивная функция FN для вычисления  $a_n$ , составленная в примере 1.

Нерекурсивная программа	Программа с рекурсивными функциями
<pre> Program Summa_1; Var E, a: real;     N, i: integer; begin   Write('N='); Readln(N);   E:=0; i:=0; a:=1;   while i&lt;=N do   begin     E:=E+a;     i:=i+1;     a:=a/i   end;   Writeln('E=', E) end. </pre>	<pre> Program Summa_4; Var M: integer; Function SUM(K: integer): real; Function FN(n: integer): real; begin   if n=0 then FN:=1   else FN:=FN(n-1)/n end; begin   if K=0 then SUM:=1   else SUM:=SUM(K-1)+FN(K); end; {Основная программа} begin   Write('M='); Readln(M);   Writeln('E=', SUM(M)) end. </pre>

Сопоставьте эти программы и сделайте вывод, какая из них вам кажется проще с точки зрения процесса программирования. С точки зрения оптимизации работы компьютера, предпочтительным является нерекурсивный вариант.

Сделаем выводы из сказанного в этом параграфе. Частично-рекурсивную функцию можно запрограммировать на Паскале, используя рекурсивную подпрограмму-функцию или подпрограмму-процедуру. Такая подпрограмма имеет ветвящуюся структуру алгоритма вместо циклической структуры в нерекурсивной программе. В некоторых случаях рекурсивный подход упрощает программирование. Однако рекурсивно определенная подпрограмма

занимает больше машинного времени при исполнении и требует дополнительного расхода памяти на организацию стека.



### Система основных понятий

#### Рекурсивные подпрограммы

***n*-шаговая рекуррентная последовательность:** числовая последовательность, имеющая *n* заданных начальных элементов и формулу для вычисления последующих элементов через *n* предшествующих

**Рекуррентная формула:** формула для вычисления элементов рекуррентной последовательности

**Рекурсивная подпрограмма (функция, процедура):** подпрограмма, содержащая в своем описании вызов себя самой. Должна иметь выход, не содержащий рекурсивного обращения



### Вопросы и задания



1. Можно ли арифметическую и геометрическую прогрессии назвать рекуррентными последовательностями? Если да, то напишите для них рекуррентные формулы.
2. Напишите нерекурсивный вариант подпрограммы-функции вычисления чисел Фибоначчи. Сопоставьте ее с рекурсивной функцией с точки зрения времени выполнения на компьютере.
3. Напишите два варианта подпрограммы-функции вычисления *n*-го элемента арифметической прогрессии: нерекурсивный и рекурсивный.
4. Опишите рекурсивную подпрограмму-функцию `pow(x, n)` от вещественного *x* ( $x \neq 0$ ) и целого *n*, которая вычисляет величину  $x^n$  согласно формуле:

$$x^n = \begin{cases} 1, & \text{при } n = 0; \\ x^{-n}, & \text{при } n < 0; \\ x \cdot x^{n-1}, & \text{при } n > 0. \end{cases}$$

### 2.3.2. Задача о Ханойской башне

Применение рекурсивных методов для решения вычислительных задач не всегда эффективно. В большинстве случаев для решения той же задачи можно построить оптимальный нерекурсивный ал-



горитм. В то же время существуют задачи не вычислительного содержания, решить которые без использования рекурсии оказывается крайне проблематичным. К числу таких задач относится известная головоломка под названием «Ханойская башня» (рис. 2.10).

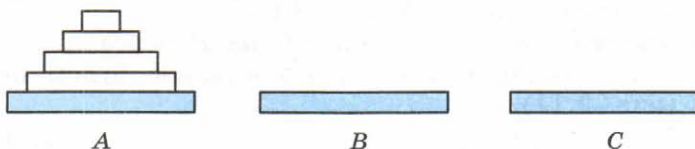


Рис. 2.10. Исходное состояние задачи

На площадке (назовем ее  $A$ ) находится пирамида, составленная из дисков уменьшающегося от основания пирамиды к ее вершине размера. Эту пирамиду в том же виде требуется переместить на площадку  $B$ . При выполнении работы необходимо соблюдать следующие ограничения:

- перекладывать можно только по одному диску, взятому сверху пирамиды;
- класть диск можно только либо на основание площадки, либо на диск большего размера;
- в качестве дополнительной можно использовать площадку  $C$ .

Название «Ханойская башня» связано с легендой, согласно которой в давние времена монахи одного ханойского храма взялись переместить по этим правилам башню, состоящую из 64 дисков. С завершением их работы должен был наступить конец света.

Нетрудно решить эту задачу для двух дисков. Обозначая перемещения диска, например, с площадки  $A$  на площадку  $B$  так:  $A \Rightarrow B$ , напомним алгоритм для этого случая:

$A \Rightarrow C$ ;  $A \Rightarrow B$ ;  $C \Rightarrow B$ .

Всего 3 хода! Для трех дисков алгоритм длиннее:

$A \Rightarrow B$ ;  $A \Rightarrow C$ ;  $B \Rightarrow C$ ;  $A \Rightarrow B$ ;  $C \Rightarrow A$ ;  $C \Rightarrow B$ ;  $A \Rightarrow B$ .

Уже 7 ходов.

Подсчитать количество ходов ( $N$ ) для  $k$  дисков можно по следующей рекуррентной формуле:

$N(1) = 1$ ;  $N(k) = 2 \cdot N(k - 1) + 1$ .

Например,  $N(10) = 1023$ ,  $N(20) = 104857$ . А вот сколько перемещений нужно сделать ханойским монахам:

$N(64) = 18\ 446\ 744\ 073\ 709\ 551\ 615$ .

Попробуйте оценить, сколько лет на это потребуется.

Составим программу, по которой машина рассчитает алгоритм перемещения дисков и выведет его для любого значения  $n$  (ко-

личества дисков). Пусть на площадке  $A$  находится  $n$  дисков. Алгоритм решения задачи будет следующим.

1. Если  $n = 0$ , то ничего не делать.
2. Если  $n > 0$ , то:
  - переместить  $n - 1$  дисков на  $C$  через  $B$ ;
  - переместить диск с  $A$  на  $B$  ( $A \Rightarrow B$ );
  - переместить  $n - 1$  дисков с  $C$  на  $B$  через  $A$ .

При выполнении пункта 2 последовательно будем иметь три состояния (рис. 2.11).

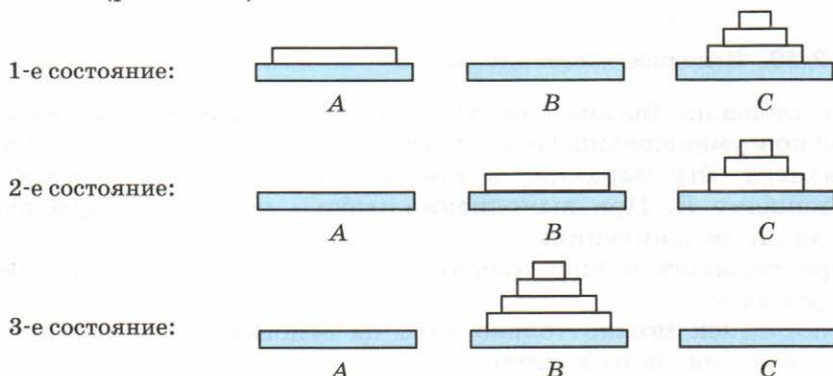


Рис. 2.11. Процесс перемещения башни

Описание алгоритма имеет явно рекурсивный характер. Перемещение  $n$  дисков описывается через перемещение  $n - 1$  дисков. А где же выход из этой последовательности рекурсивных ссылок? Он в пункте 1, как бы ни показалось странным его тривиальное содержание.

А теперь составим программу на Паскале. В ней имеется рекурсивная процедура `Hanoi`, выполнение которой заканчивается только при  $n = 0$ . При обращении к процедуре используются фактические имена площадок, заданные их номерами: 1, 2, 3. Поэтому на выходе цепочка перемещений будет описываться в таком виде:

$1 \Rightarrow 2 \quad 1 \Rightarrow 3 \quad 2 \Rightarrow 3$  и т. д.

```

Program Monahi;
Var N: byte;
Procedure Hanoi(N: byte; A, B, C: char);
begin
  if N>0 then
    begin Hanoi(N-1, A, C, B);
      WriteLn(A, '=>', B);
      Hanoi(N-1, C, B, A)
    end
end;

```



**begin**

```
WriteLn('Укажите число дисков: ');
```

```
ReadLn(N);
```

```
Hanoi(N, '1', '2', '3')
```

**end.**

Удивительная программа, не правда ли? Попробуйте воспроизвести ее на компьютере. Проследите, как изменяется число ходов с ростом  $n$ . Для этой цели можете сами добавить в программу счетчик ходов и в конце вывести его значение или выводить ходы с порядковыми номерами.

### 2.3.3. Алгоритм быстрой сортировки

Об алгоритмах сортировки рассказывалось в параграфе 1.7.7 учебника для 10 класса. Там описаны два алгоритма сортировки: методом поиска максимального элемента и методом пузырька. Сейчас мы рассмотрим оптимальный по времени работы алгоритм, который называется алгоритмом быстрой сортировки. Этот алгоритм был разработан Энтони Хоаром в 1960 году. Покажем, как он реализуется через рекурсивную процедуру.

В алгоритме быстрой сортировки используются три идеи:

- 1) разделение сортируемого массива на две части: левую и правую;
- 2) взаимное упорядочение двух частей (подмассивов) так, чтобы все значения элементов левой части не превосходили значений элементов правой части;
- 3) рекурсия, при которой подмассив упорядочивается точно таким же способом, как и весь массив.

Для первоначального разделения массива на две части нужно выбрать некоторое «барьерное» значение. Это значение должно удовлетворять единственному условию: лежать в диапазоне значений для данного массива (т. е. между минимальной и максимальной величинами). В качестве «барьера» можно выбрать значение любого элемента массива, например первого или последнего, или находящегося в середине.

Далее нужно сделать так, чтобы в левом подмассиве оказались все элементы со значениями, меньшими или равными барьеру, а в правом — большими или равными. Для этого, просматривая массив слева направо, нужно найти позицию первого элемента со значением, большим или равным барьеру, а просматривая справа налево, найти первый элемент со значением, меньшим или равным барьеру. Поменять местами эти значения. Затем продолжить встречное движение до следующей пары элементов, предназначенных для обмена. Так продолжать до тех пор, пока индекс лево-

го просмотра не станет больше индекса правого просмотра. Эти индексы будут разделителями двух взаимно упорядоченных подмассивов. Далее алгоритм рекурсивно применяется к каждому из подмассивов (левому и правому). В конечном счете приходим к совокупности из  $n$  взаимно упорядоченных одноэлементных массивов, которые делить дальше невозможно. Эта совокупность образует один полностью упорядоченный массив. Сортировка завершена!

В следующей программе вещественный массив  $A$  заполняется случайными числами в диапазоне от 0 до 10. Затем этот массив сортируется с помощью процедуры быстрой сортировки  $QSort$ .

```

Program Sortirovka;
Const N = 20;
Var A: array[1..N] of real;
    i: integer;
Procedure Qsort(L, R: integer);
Var i, j: integer; bar, w: real;
begin i:=L; j:=R;
    bar:=A[(L+R) div 2]; {Установка барьера}
    repeat
        {Поиск элемента слева для обмена}
        while A[i]<bar do i:=i+1;
        {Поиск элемента справа для обмена}
        while A[j]>bar do j:=j-1;
        {Обмен элементов и смещение по массиву}
        if i<=j then
            begin
                w:=A[i]; A[i]:=A[j]; A[j]:=w;
                i:=i+1; j:=j-1
            end;
    until i>j;
    {сформированы взаимно упорядоченные подмассивы}
    {сортировка левого подмассива}
    if L<j then Qsort(L,j);
    {Сортировка правого подмассива}
    if i<R then Qsort(I,R);
end; {Qsort}
{Основная программа}
begin
    Randomize;
    for i:=1 to N do A[i]:=Random*10; {Заполнение массива}
                                     {случайными числами}
    Qsort(1,N); {Обращение к процедуре сортировки}
    for i:=1 to N do Write(A[i]:5:1) {Вывод отсортированного}
                                     {массива}
end.

```



Константа  $N$  и массив  $A$  описаны в программе глобально. Процедура  $Qsort(L, R)$  сортирует по возрастанию значений элементы массива  $A$ . Параметр  $L$  процедуры обозначает нижнее значение индекса сортируемого массива (крайний левый индекс), параметр  $R$  — верхнее значение индекса (крайний правый индекс). Барьерное значение  $bar$  вычисляется как значение элемента, имеющего индекс, равный целой части среднего арифметического между  $L$  и  $R$ .

### Система основных понятий

#### Примеры рекурсивного программирования

Задача о Ханойской башне

Быстрая сортировка (Э. Хоар)

### Вопросы и задания

1. Сформулируйте идею рекурсивного алгоритма решения задачи о Ханойской башне.
2. Реализуйте на компьютере программу решения задачи о Ханойской башне.
3. Сформулируйте идею алгоритма быстрой сортировки.
4. Реализуйте на компьютере программу сортировки массива с использованием процедуры быстрой сортировки.

Практикум. Раздел «Программирование»

## 2.4. Объектно-ориентированное программирование

### 2.4.1. Базовые понятия объектно-ориентированного программирования

В этом разделе вы познакомитесь с основами объектно-ориентированной парадигмы программирования (ООП). Как уже рассказывалось в параграфе 2.2.1, язык Паскаль первоначально предназначался только для процедурного программирования с опорой на структурную методику. Затем в поздних версиях языка Турбо Паскаль появились элементы объектно-ориентированного программирования. Этот процесс привел к созданию объектно-ориентиро-



ванной версии Паскаля, которая получила название Object Pascal. Результатом объединения языка программирования Object Pascal с визуальной технологией программирования и библиотекой визуальных компонентов стала система программирования Delphi. Далее будет рассмотрена объектная модель, положенная в основу Delphi.

### Классы, объекты, инкапсуляция

Центральными понятиями объектно-ориентированного программирования являются «класс» и «объект». **Класс** — это тип данных, описываемый программистом; это категория объектов, обладающих одинаковым набором свойств и методов воздействия на объекты. **Объект** — это экземпляр определенного класса с конкретными значениями свойств.

Класс в Object Pascal является структурированным типом данных, элементы которого — поля и методы. Формат описания типа «класс» следующий:

```
Type <имя класса> = class[(<имя класса родителя>)]
    <описания полей>
    <объявление методов и описания свойств>
end;
```

Поля — это переменные величины (простые или структурные), методы — это процедуры и функции, работающие с полями этого класса. Процесс объединения в единую структуру данных (полей) и действий над этими данными (методов) называется **объектно-ориентированной декомпозицией**. Реализация полей и методов объекта скрыта от других объектов, взаимодействующих с ним. Этот принцип ООП называется **инкапсуляцией**. Инкапсуляция исключает возможность изменения значений полей другими способами, кроме методов данного класса. *Инкапсуляция — первый базовый принцип ООП.*

Рассмотрим пример программы вычисления длины отрезка прямой на языке Object Pascal, в которой используется тип данных «класс».

```
Program Geometry;
{----- Описание класса TLine-----}
Type TLine = class
    FX1, FY1, FX2, FY2: real; // Поля — координаты концов линии
    Function Length: real; // Метод — функция вычисления длины
    Procedure Input; // Метод — процедура ввода координат
end;
{ -----Описание объекта типа TLine----- }
var Line: TLine; //
{-----Реализация методов-----}
```



```
Function Line.Length: real;
begin
  Line.Length:=sqrt (sqr (FX1-FX2)+sqr (FY1-FY2))
end;
Procedure Line.Input;
begin
  Write('x1='); Readln(FX1);
  Write('y1='); Readln(FY1);
  Write('x2='); Readln(FX2);
  Write('y2='); Readln(FY2)
end;
{-----Основная программа-----}
begin
  Line:=TLine.Create; //Создание объекта
                        //в динамической памяти
  Line.Input;         //Ввод данных
                        //Вычисление и вывод длины отрезка
  Writeln('Длина отрезка=', Line.Length)
  Line.Destroy       //Удаление объекта из динамической памяти
end.
```

В описании класса методы представлены в виде объявлений заголовков процедур и функций, их реализующих. Подробное описание этих процедур и функций помещается ниже в разделе подпрограмм. Обратите внимание на то, что в описаниях функции `Length` и процедуры `Input` не указываются параметры, поскольку доступными для них величинами являются поля класса.

Конструктор (`Create`) — это стандартная функция, предназначенная для выделения в динамической памяти компьютера места под данные объекта. Деструктор (`Destroy`) — стандартная процедура, освобождающая динамическую память от объекта.

Обращение к полям и методам объекта данного класса производится с помощью составного имени, как это делается в записях:

```
<имя объекта>.<имя поля>, <имя объекта>.<имя метода>
```

### **Наследование и полиморфизм**

Следующая программа также будет иметь геометрическое содержание. В ней объявлены два класса: класс выпуклых четырехугольников (`TFourAngl`) и класс квадратов (`TKvadrat`). Четырехугольник — более общее понятие, чем квадрат. Квадрат является частным случаем четырехугольника. Общее свойство, характерное для любых четырехугольников, — наличие четырех вершин. Поэтому полями класса `TFourAngl` будут координаты четырех вершин. Кроме того, в число полей класса четырехугольников включим длины четырех сторон и двух диагоналей четырехугольника. Сре-

ди методов класса `TFourAngl` объявим ввод координат вершин, вычисление длин отрезков (сторон и диагоналей) и вычисление площади произвольного выпуклого четырехугольника.

Класс `TKvadrat` будет объявлен как потомок класса `TFourAngl`. Имя родительского класса указывается в объявлении класса-потомка в круглых скобках после слова `class`. Класс-потомок наследует у класса-родителя все его элементы, т. е. поля и методы, поэтому в объявлении класса-потомка их повторять не следует. Хотя в классе `TFourAngl` существует метод вычисления площади произвольного выпуклого четырехугольника — функция `Square`, однако в классе `TKvadrat` объявлена функция с тем же именем `Square` с той целью, чтобы для квадрата можно было применять более простой способ вычисления площади (как квадрат длины стороны), что повысит точность вычислений и сократит машинное время.

Рассмотрим полный текст программы.

```

Program Geometry;
Type TKoord = record
  X, Y: real
end; //Тип координат вершин
{----- Объявление базового класса (родителя) -----}
Type TFourAngl = class
  P: array[0..3] of TKoord; //Координаты 4 вершин
  L: array[0..5] of real; //Длины сторон и диагоналей
  Procedure Init; //Метод: ввод координат
  Procedure Line; //Метод: вычисление длин отрезков
  Function Square: real; //Метод: вычисление площади
end;
{----- Объявление порожденного класса (потомка) -----}
Type TKvadrat = class(TFourAngl)
  Function Square: real; //Метод: вычисление площади квадрата
End;
{----Описание объектов Четырехугольник и Квадрат ----}
Var FourAngl: TFourAngl;
    Kvadrat: TKvadrat;
{----- Описание процедур реализации методов -----}
Procedure TFourAngl.Init;
Var i: integer;
begin
  for i:=0 to 3 do
  begin
    Write('Ввод X' , i, ':'); Readln(P[i].X);
    Write('Ввод Y' , i, ':'); Readln(P[i].Y)
  end
end;

```



```

Procedure TFourAngl.Line;
Var i: integer;
begin
  for i:=0 to 3 do
    L[i]:=sqrt(sqr(P[i].X-P[(i+1) mod 4].X)+
              sqr(P[i].Y-P[(i+1) mod 4].Y));
  L[4]:=sqrt(sqr(P[0].X-P[2].X)+ sqr(P[0].Y-P[2].Y));
  L[5]:=sqrt(sqr(P[1].X-P[3].X)+ sqr(P[1].Y-P[3].Y))
end;

Function TFourAngl.Square: real;
Var pp1, pp2: real;
begin
  Line;
  pp1:=(L[0]+L[1]+L[4])/2; pp2:=(L[2]+L[3]+L[4])/2;
  Square:=sqrt(pp1*(pp1-L[0])* (pp1-L[1])* (pp1-L[4]))+
           sqrt(pp2*(pp2-L[2])* (pp2-L[3])* (pp2-L[4]))
end;

Function TKvadrat.Square: real;
begin
  Line;
  Square:=sqr(FourAngl.L[0])
end;

{-----Основная программа-----}
begin
  FourAngl:=TFourAngl.Create; //Создание объекта FourAngl
  Kvadrat:=TKvadrat.Create;  //Создание объекта Kvadrat
  FourAngl.Init;             //Ввод координат вершин
  FourAngl.Line;             //Вычисление длин сторон и диагоналей
  with FourAngl do
    begin
      {----Распознавание квадрата и вычисление площади-----}
      if (L[0]=L[1]) and (L[1]=L[2]) and (L[4]=L[5])
      then Writeln('Это квадрат, площадь = ', Kvadrat.Square)
      else Writeln('Это не квадрат, площадь = ', Square)
    end
  end.

```

Встроенные в текст комментарии позволяют понять назначение отдельных фрагментов программы. В этой программе применен *второй базовый принцип объектно-ориентированного программирования* (помимо инкапсуляции), который называется **наследованием**. Наследование — способ создания новых классов в качестве наследников уже существующих.

Класс-потомок наследует от своего родительского класса все поля и методы. При этом если родительский класс также унаследовал от своих предков некоторые элементы, то они передаются по наследству его потомку. Технология ООП позволяет выстраи-



вать ветвящиеся иерархии наследования, т. е. один потомок может иметь несколько родителей. В нашем примере используется простое наследование: один родитель — один потомок, что соответствует ограничению, действующему в Object Pascal.

**Третий базовый принцип ООП** называется **полиморфизмом** (многообразие). Элементы классов с одинаковым интерфейсом могут иметь разную реализацию. В рассмотренной программе полиморфизм проявляется в том, что функция с одним и тем же именем `Square` определена дважды, по-разному: сначала в родительском классе `TFourAngl`, а затем она *переопределена* в классе-потомке `TKvadrat`. В результате площадь произвольного четырехугольника и площадь квадрата будут вычисляться по разным алгоритмам.

### Система основных понятий

Базовые понятия ООП	
Класс	Структурный тип данных, определяющий категорию объектов, обладающих одинаковым набором свойств и методов воздействия на них
Объект	Экземпляр определенного класса с конкретными значениями свойств (величина типа «класс»)
Инкапсуляция	Первый базовый принцип ООП: объединение в единую структуру (класс) данных — полей и действий над этими данными — методов
Наследование	Второй базовый принцип ООП: класс-потомок получает от своего родительского класса все поля и методы
Полиморфизм	Третий базовый принцип ООП: возможность переопределения метода с одним интерфейсом в разных классах

### Вопросы и задания

1. Что такое класс и что такое объект в Object Pascal?
2. Дайте определения понятий: инкапсуляция, наследование, полиморфизм.
3. Реализуйте на компьютере (в системе программирования на Object Pascal) программу `Geometry`.
4. Добавьте в описание класса `TFourAngl` поля для значений углов четырехугольника и метод для вычисления угла. В основной части программы реализуйте вычисление всех углов.

Практикум. Раздел «Программирование»



## 2.4.2. Система программирования Delphi

**Delphi** — система программирования, предназначенная для создания **объектно-ориентированных** приложений Windows путем использования **визуальной технологии программирования**. Визуальная среда Delphi относится к средам категории RAD (Rapid Application Development — среда быстрой разработки приложений). Договоримся также термином Delphi называть и язык программирования, являющийся современным диалектом Паскаля, который также называют Delphi Pascal.

Основной подход к разработке программного обеспечения в подобных средах заключается в использовании стандартных визуальных компонентов — заранее подготовленных классов, которые программист подключает к программе, помещая их в свой проект.

Процесс разработки программы в Delphi может быть предельно упрощен. В первую очередь это относится к созданию интерфейса, на который обычно уходит порядка 80% времени разработки программы. Программисту необходимо просто поместить нужные компоненты на поверхность окна (в Delphi оно называется **формой**) и настроить их свойства, используя специальный инструмент (*Object Inspector*). С его помощью можно связать **события** компонентов (нажатие кнопки, выбор мышью элемента в списке и т. д.) с процедурами их обработки, которые должен составить сам программист, — и простое приложение готово. При этом программист имеет в своем распоряжении необходимые средства отладки, удобную контекстную справочную систему, средства коллективной работы над проектом и пр.

### **Среда системы программирования Delphi**

Среда системы программирования Delphi показана на рис. 2.12. Она состоит из следующих элементов.

**Строка заголовка** (вверху окна).

**Строка главного меню** и командные кнопки (под строкой заголовка).

**Окно конструктора форм**. Располагается в центре экрана на вкладке *Design* (см. рис. 2.12). Форма используется для конструирования интерфейса проектируемого приложения путем размещения на ней элементов управления (элементов интерфейса).

**Окно программного кода**. Располагается в центре экрана на вкладке *Code*. В начале создания программы в окне программного кода помещается стандартный шаблон (рис. 2.13). Далее в процессе программирования он будет заполняться текстом программы.

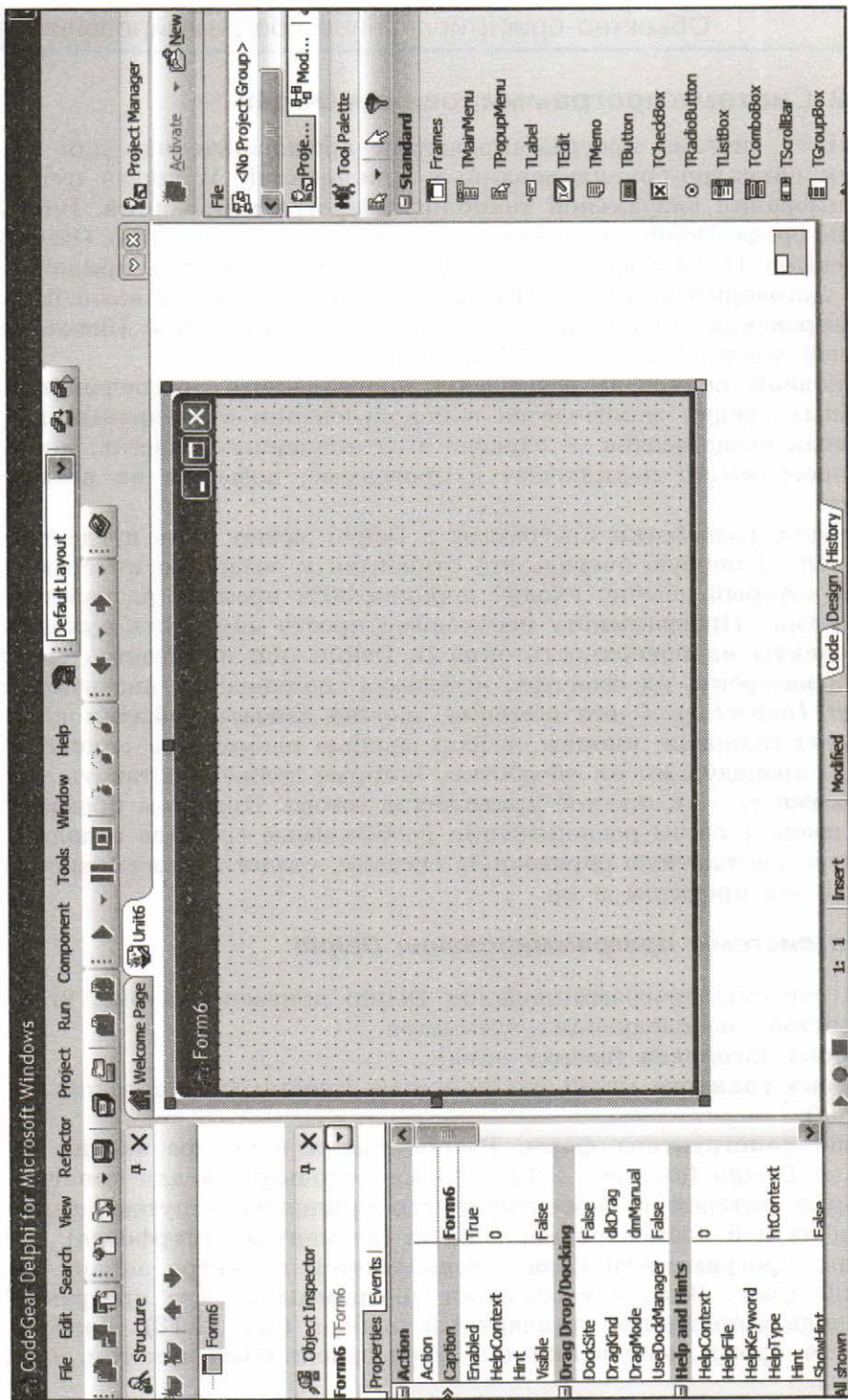


Рис. 2.12. Среда программирования Delphi с окном конструктора форм



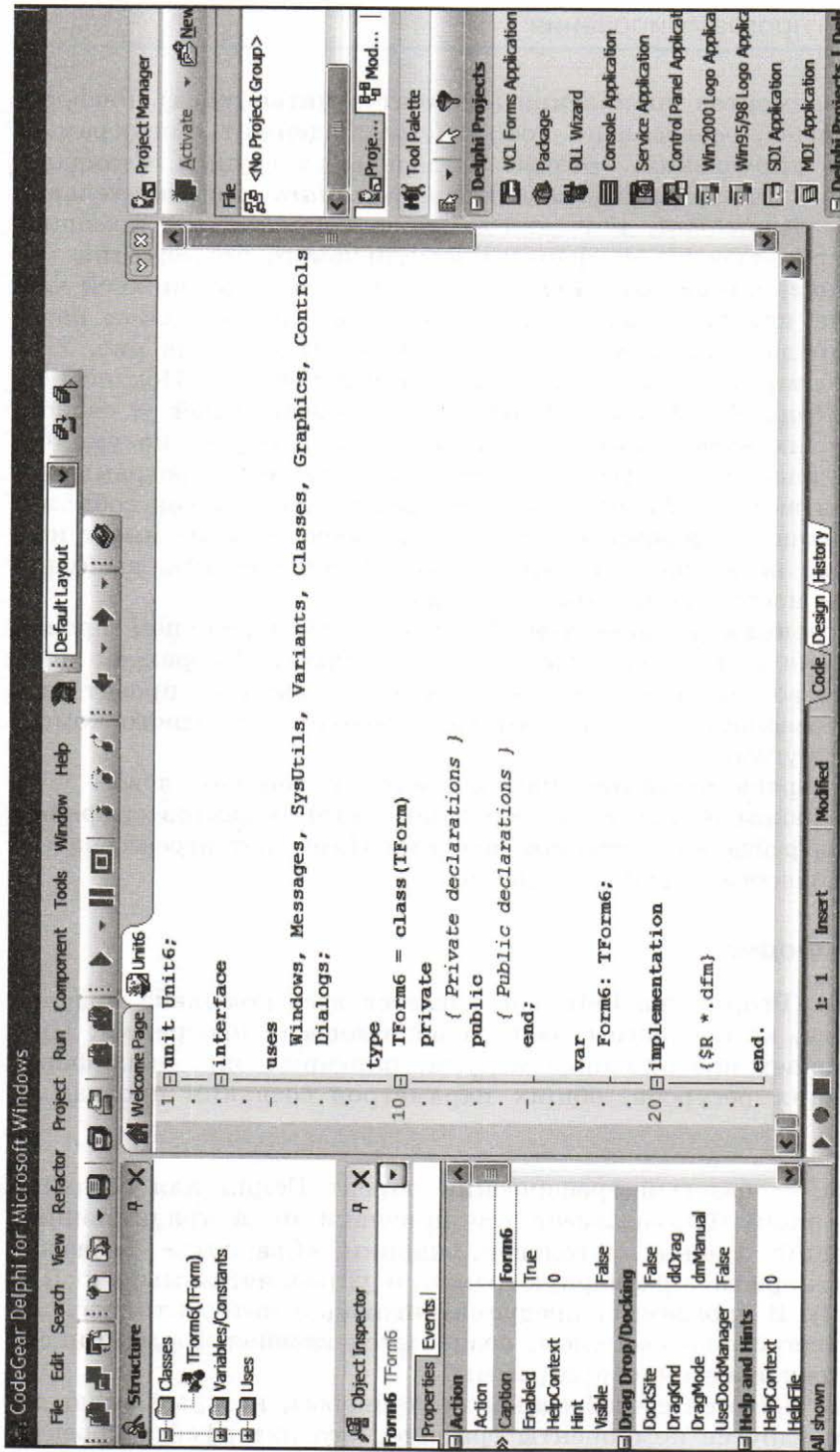


Рис. 2.13. Среда программирования Delphi с окном программного кода

**Окно элементов управления (элементов интерфейса).** Располагается справа, озаглавлено *Tool Palette*. Содержит пиктограммы элементов управления, которые разделены на группы. Стандартная группа пиктограмм озаглавлена *Standard*, дополнительная группа — *Additional*, и т. д. Элементы управления с помощью мыши перемещаются на форму проектируемого приложения.

**Окно инспектора объектов.** Располагается в левой нижней части экрана под заголовком *Object Inspector*. Вверху окна находится выпадающий список с названиями объектов (на рис. 2.12 имеется единственный объект с названием *Form 6*). Под ним на вкладке *Properties* (Свойства) содержится список полей — свойств объекта с их названиями и значениями, принятыми по умолчанию, которые могут быть изменены вручную или программным путем. На вкладке *Events* (События) содержится список событий, относящихся к выбранному объекту. Каждое событие имеет имя и указание на метод — программу обработки события, выполняемую при наступлении этого события.

**Окно менеджера проектов.** Располагается в верхнем правом углу экрана под заголовком *Project Manager*. Отображает файловое дерево компонентов, составляющих текущий проект (см. далее). Позволяет щелчком мышью переходить от одного компонента к другому.

**Окно дерева объектов.** Располагается в верхнем левом углу под заголовком *Structure*. Отображает состав объектов графического интерфейса в создаваемом проекте. Позволяет переключаться между объектами щелчком мышью.

### **Проект. Форма**

**Проектом (Project)** на Delphi называется весь комплекс модулей и ресурсов, из которого создается исполняемая программа. Проект включает программные модули, описания экранных форм, графических ресурсов, общих параметров создания программы и т. д.

**Форма** — базовый графический объект Delphi для создания рабочих окон. Форма имеет все признаки окна традиционных приложений: значок, заголовок, кнопки «Свернуть», «Развернуть», «Закрыть», размерную рамку и управляется мышью (см. рис. 2.12). В программе, предусматривающей интерактивное взаимодействие с пользователем, назначается главная форма, описывающая основное окно программы.

Форма является своеобразным контейнером, который включает в себя все другие компоненты графического интерфейса проекта:



кнопки, метки, окна ввода и пр. **Конструктор форм** позволяет выполнить во время разработки проекта следующие действия:

- добавить компоненты на форму;
- модифицировать форму и ее компоненты;
- связать обработчик события компонента с процедурой или функцией, содержащейся в редакторе кода.


В общих чертах процесс разработки программы на Delphi выглядит следующим образом: из окна элементов управления с помощью мыши выбираются компоненты интерфейса (кнопка, надпись, редактор текста и др.), помещаются на форму, и задаются значения их свойств в области *Properties* (Свойства). Среда Delphi анализирует содержимое формы, создает соответствующий программный модуль (*Unit*), связанный с формой, а программист вносит в него программные коды процедур — **обработчиков событий**.


Проект может содержать несколько форм и, следовательно, несколько программных модулей. Помимо программных модулей форм, в проект включаются файлы с некоторыми другими программными модулями.


### Элементы управления. Свойства

**Элементы управления** — это классы объектов, являющихся компонентами графического интерфейса проекта, обладающие набором свойств, определяющих их внешний вид и состояние, а также реагирующие на события, производимые пользователем или программой.

Примеры элементов управления и их свойств:

 *TLabel* — метка. Служит для отображения текста на экране. В числе свойств: надпись (*Caption*), имя (*Name*), параметры шрифта (*Font*), цвет (*Color*), размер на экране, координаты размещения на экране и др.

 *TEdit* — экранное поле редактирования. Служит для ввода данных пользователем в процессе выполнения программы. Основные свойства: имя (*Name*), текст в поле ввода/редактирования (*Text*), шрифт (*Font*), размеры и др.

 *TButton* — командная кнопка. Позволяет выполнять какие-либо действия при нажатии кнопки во время выполнения программы. Основные свойства: надпись (*Caption*), имя (*Name*), размеры и положение на экране.

### Элементы управления. События

**Событие** — изменение некоторого состояния объекта в результате действия пользователя или программного воздействия на объект. Для каждого класса объектов имеется свой набор событий, отображенный на вкладке *Events* окна *Object Inspector*.

Примерами событий, вызываемых действиями пользователя, являются:

*OnClick* — щелчок указателем мыши на объекте;

*OnDblClick* — двойной щелчок указателем мыши на объекте;

*OnMouseMove* — движение указателя мыши над объектом;

*OnChange* — изменение содержания (например, текста в окне редактирования).

### Методы — процедуры обработки событий

Реакцией Delphi на событие является вызов процедуры — обработчика события. Такую процедуру составляет сам программист. Выбрав событие на вкладке *Events*, следует произвести двойной щелчок на поле, напротив имени события. В окне программного кода появится заготовка процедуры, в которую нужно вписать программный код реакции на это событие.

Программы, создаваемые в системе Delphi, называются *событийно-управляемыми*. Это значит, что выполнение различных задач, решаемых программой, инициируется определенными событиями.

### Система основных понятий

Система программирования Delphi	
Delphi	Система программирования для создания объектно-ориентированных приложений Windows. Используется визуальная технология разработки интерфейса
Форма	Базовый графический объект Delphi, на основе которого создается программный проект (Project)
Элементы управления	Классы объектов графического интерфейса проекта, вкладываемых в форму (метки, кнопки, окна редактирования и пр.). Обладают набором свойств ( <i>Properties</i> ) и событий ( <i>Events</i> )
Методы	Процедуры обработки событий (составляются программистом)
Событийно-управляемое программирование	Выполнение различных задач, решаемых программой, инициируется определенными событиями, которые запускают методы их обработки

### Вопросы и задания

1. Для чего предназначена система Delphi?
2. Назовите основные компоненты среды программирования Delphi.



3. Что такое форма?
4. Что такое элементы управления? Чем они характеризуются?
5. Что такое методы?
6. Почему программирование на Delphi называется событийно-управляемым программированием?

### 2.4.3. Этапы программирования на Delphi

С помощью системы программирования Delphi можно создать несколько различных типов приложений. В том числе можно реализовать и приложение без графического интерфейса, которое называется **консольным приложением**. Такие приложения вам уже приходилось создавать, например, в системе Турбо Паскаль или PascalABC.

#### Создание консольного приложения

Приведем в качестве первого примера для программирования задачу перевода целого недесятичного числа в десятичную систему счисления. В параграфе 1.3.3 учебника для 10 класса приведена такая программа на Паскале. Практически в том же виде ее можно перенести в среду Delphi в режим создания консольного приложения. Делается это следующим образом: после запуска системы Delphi через главное меню отдать команду на создание нового проекта: **File** → **New**. Затем в открывшемся окне выбрать тип приложения *Console Application*. В открывшемся окне редактора программного кода с готовым шаблоном ввести текст программы. Программа будет выглядеть следующим образом:

```
Program Project1;
{$APPTYPE CONSOLE} //Директива компилятору о типе приложения
Uses SysUtils;      //Подключение системной библиотеки
Var N10, Np, k: longint;
    p: 2..9;
begin
  Write('p='); Readln(p); //Ввод основания системы счисления
  Write('N', p, '=');
  Readln(Np);          //Ввод исходного р-ичного числа
  k:=1; N10:=0;
  while (Np<>0) do //Цикл выполняется, пока Np не равно нулю
  Begin
    N10:=N10+(Np mod 10)*k; //Суммирование развернутой формы
    k:=k*p;                //Вычисление базиса: p, p в степени 2,
                          //p в степени 3, ...
    Np:=Np div 10         //Отбрасывание младшей цифры
  end;
```



```

Writeln('N10=', N10); //Вывод десятичного числа
Readln              //Задержка окна результата на экране
end.

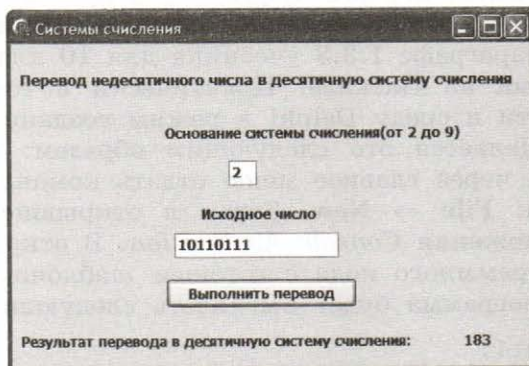
```

Выполнение программы инициируется через главное меню по команде **Run** → **Run**.

### Создание оконного приложения

Теперь решение задачи о переводе чисел реализуем с помощью оконного проекта с графическим интерфейсом. Для этого после команды **File** → **New** выберем тип приложения *VCL Forms Application*.

**1. Проектирование и конструирование интерфейса.** С самого начала работы надо представить себе конечный результат: в какой форме мы хотим увидеть на экране результат выполнения программы. Такое представление для задачи о числах показано на рис. 2.14.



**Рис. 2.14.** Интерфейс задачи о числах

Используя конструктор формы и панель элементов управления, наносим на шаблон формы нужные элементы интерфейса, как это показано на рис. 2.15. Здесь присутствуют пять элементов типа *метка (Label)*, два *окна редактирования (Edit)* и одна *командная кнопка (Button)*. В окна редактирования будут вводиться исходные данные: основание системы счисления и число в этой системе. Результат будет выводиться в поле *Label5*.

Затем для объекта формы изменяется надпись (свойство *Caption*) на «Системы счисления». Для меток с 1-й по 4-ю назначаются требуемые надписи. На метке *Label5* надпись убирается. Для командной кнопки назначается надпись «Выполнить перевод». Окончательный вид макета интерфейса показан на рис. 2.16.



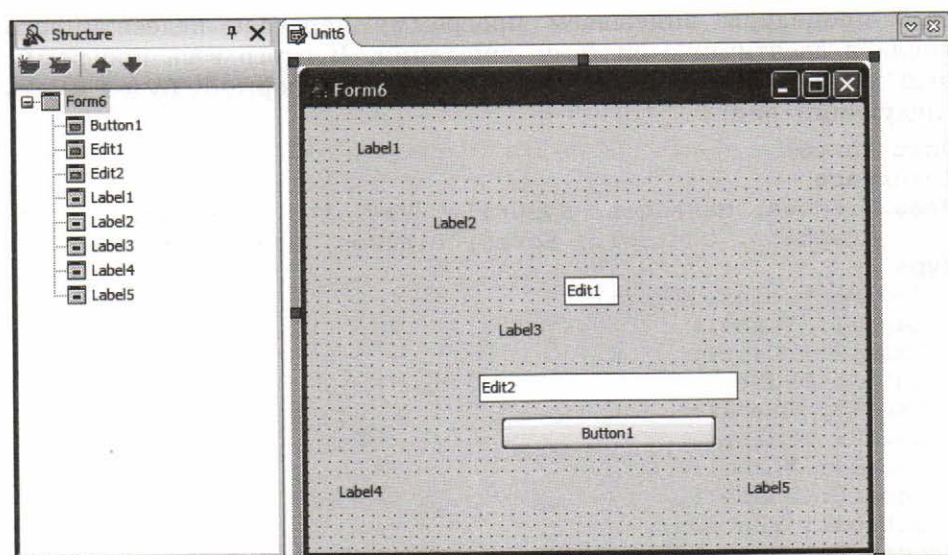


Рис. 2.15. Структура интерфейса

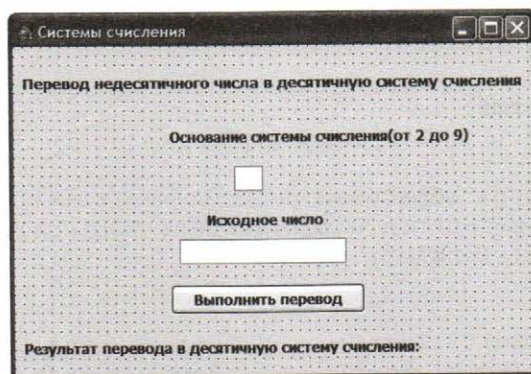


Рис. 2.16. Макет интерфейса программы

**2. Реализация обработки событий.** Выполнение перевода числа в десятичную систему счисления должно происходить по щелчку на кнопке «Выполнить перевод». Иницилируем вызов процедуры обработки события *OnClick* для объекта *Button1*. В окне редактора программного кода появляется шаблон для процедуры с заголовком:

```
Procedure TForm6.Button1Click(Sender: TObject);
```

Здесь в скобках указывается класс *TObject* — базовый класс Delphi, потомками которого являются все другие классы. В тело

этой процедуры вписываем программу перевода десятичного числа в десятичную систему счисления. В итоге весь программный модуль (*Unit*), связанный с созданной формой, будет иметь следующий вид:

```

Unit Unit6;
Interface
Uses Windows, Messages, SysUtils, Variants, Classes,
    Graphics, Controls, Forms, Dialogs, StdCtrls;
Type
  TForm6 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Label3: TLabel;
    Edit2: TEdit;
    Label4: TLabel;
    Label5: TLabel;
    Button1: TButton;
  Procedure Button1Click(Sender: TObject);
  private
    {Private declarations}
  public
    {Public declarations}
  end;
Var Form6: TForm6;
implementation
{$R *.dfm}
{-----Процедура обработки события-----}
Procedure TForm6.Button1Click(Sender: TObject);
Var N10, Np, k: longint;
    p: 2..9;
begin
  p:=StrToInt(Edit1.Text); //Ввод из окна Edit1
  Np:=StrToInt(Edit2.Text); //Ввод из окна Edit2
  k:=1; N10:=0;
  while (Np<>0) do
  begin
    N10:=N10+(Np mod 10)*k;
    k:=k*p;
    Np:=Np div 10
  end;
  Label5.Caption:=IntToStr(N10) //Вывод в поле метки Label5
end //Конец процедуры обработки события
end.

```

Выполнение программы инициируется через главное меню по команде **Run** → **Run**. На экране появляется окно интерфейса



программы. В окне редактирования вводятся исходные данные: основание системы счисления и переводимое число. Затем производится щелчок на командной кнопке, и в поле метки *Label5* получаем результат (см. рис. 2.14).

**Ввод и вывод данных.** Ввод исходных данных производится с помощью поля *Text* элемента интерфейса *Edit*. При вводе с клавиатуры числа в окно редактирования оно воспринимается как символьная строка. Если вводится целое число, то строка затем преобразуется в величину целого типа с помощью функции *StrToInt*(строка). Если нужно ввести вещественное число, то используется функция *StrToFloat*(строка).

Вывод результата выполнен путем присваивания свойству *Caption* метки *Label5* строки со значением итогового числа. Перевод целого числа в строку символов происходит с помощью функции *IntToStr*(целое число). Если требуется вывести вещественное число, то используется функция *FloatToStr*(вещественное число).

Если требуется вводить одно из конечного множества значений, то удобно использовать другие элементы интерфейса: **списки** (*ListBox*), **поля со списком** (*ComboBox*), **счетчики** (*UpDown*), **ползунки** (*TracBar*), **переключатели** (*RadioButton*). На рисунке 2.17 продемонстрировано использование поля со списком в программе «Системы счисления» для ввода основания системы. При нажатии кнопки справа выпадает список значений для выбора одного из них. Значение выбирается щелчком мышью.

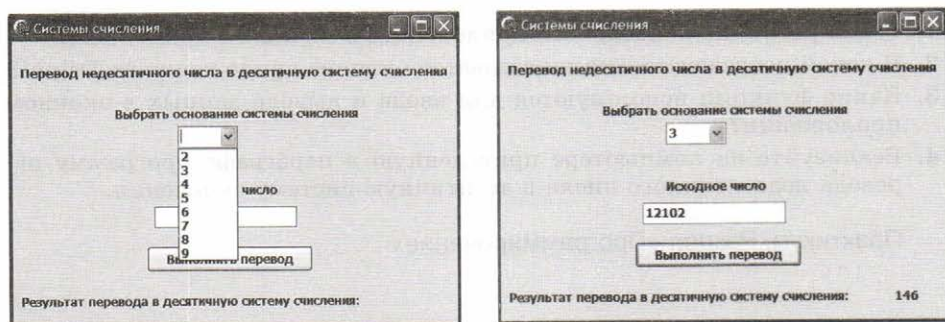


Рис. 2.17. Использование для ввода поля со списком

Значения, включаемые в список, вводятся во время программирования с использованием свойства *ComboBox1.Items*. Элементы поля со списком проиндексированы — получают порядковые номера, начиная с нуля. Каждый элемент идентифицируется индексированным именем элемента: *ComboBox1.Items[индекс]*.

После выбора элемента из списка его индекс присваивается полю `ComboBox1.ItemIndex`. В программе перевода чисел ввод значения переменной  $p$  реализуется следующим оператором:

```
p:=StrToInt(ComboBox1.Items[ComboBox1.ItemIndex]);
```

Поле со списком является способом организации интерфейса для представления одномерного массива.



### Система основных понятий

Этапы программирования на Delphi	
Консольное приложение:	приложение Delphi без графического интерфейса
Оконное приложение:	приложение Delphi с графическим интерфейсом
Проектирование интерфейса:	разработка эскиза графического интерфейса
Конструирование интерфейса:	создание формы и наполнение ее элементами управления
Реализация обработки событий:	программирование процедур методов обработки событий



### Вопросы и задания

1. В чем различие консольного приложения и оконного приложения?
2. В какой последовательности создается оконное приложение на Delphi?
3. Какие функции используются для ввода и вывода данных в оконном приложении?
4. Реализуйте на компьютере приведенную в параграфе программу перевода десятичного числа в десятичную систему счисления.

Практикум. Раздел «Программирование»

#### 2.4.4. Программирование метода статистических испытаний

Метод статистических испытаний (метод Монте-Карло) — численный метод, использующий моделирование случайных величин и получение статистических оценок искомых величин. Одно из применений этого метода — вычисление площадей фигур и объемов



тел. Составим программу вычисления числа  $\pi$  с помощью метода статистических испытаний.

Идея метода состоит в следующем. Около единичной окружности описывается квадрат, длина стороны которого равна 2 (рис. 2.18). С помощью датчика случайных чисел с равномерным законом распределения вероятности производится «стрельба» по квадрату, т. е. случайный выбор точек внутри квадрата. Каждый такой выбор будем называть испытанием. Испытание заключается в том, что вычисляются координаты точки  $(x, y)$  с помощью функции Random в пределах значений от  $-1$  до 1. Затем определяется, лежит ли эта точка внутри круга. Условие выполняется, если  $x^2 + y^2 \leq 1$ . Если точка попадает в круг, то в счетчик попаданий добавляется единица.

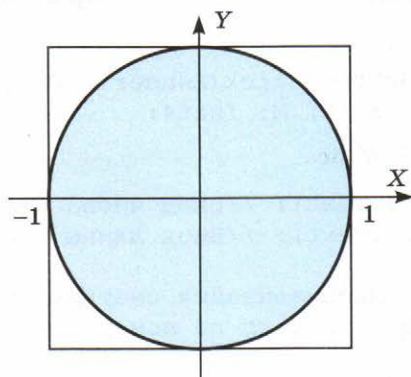


Рис. 2.18. Погружение в квадрат

Пусть  $P$  — общее число испытаний. Из них произошло  $M$  попаданий в круг. Площадь квадрата равна 4. При условии равномерного покрытия испытательными точками площади квадрата, для площади круга справедлива формула:

$$S_{\text{кр}} = 4 \cdot \lim_{P \rightarrow \infty} \frac{M}{P}.$$

Смысл формулы состоит в том, что с увеличением количества испытаний отношение  $M/P$  все больше приближается к отношению площадей круга и квадрата и в пределе при  $P$ , стремящемся к бесконечности, становится ему равно. Поскольку площадь круга радиуса 1 равна  $\pi$ , при достаточно большом значении  $P$  будет выполняться приближенное равенство:

$$\pi \approx 4 \cdot \frac{M}{P}.$$

Чем больше  $P$ , тем это равенство точнее.

Интерфейс программы на Delphi решения этой задачи показан на рис. 2.19. Чтобы можно было проследить за установлением значения числа  $\pi$ , испытания разбиваются на серии. В одной серии производится  $N$  испытаний, а число таких серий равно  $K$ . После завершения каждой серии на экран выводится результат. Окно с формой включает в себя следующие элементы интерфейса:

- *Memo1* — поле для ввода/редактирования многострочных текстов (до 32 Кб);
- *Edit1* и *Edit2* — поля для ввода значений  $K$  и  $N$ ;
- *Label1* и *Label2* — метки к полям ввода;
- *Button1* — командная кнопка для запуска вычислений;
- *ListBox1* — поле для вывода прокручиваемого списка.

Обработка события *Button1Click* производится следующей процедурой:

```

Procedure TForm6.Button1Click(Sender: TObject);
Var i, j: integer; K, N, M: Int64;
    X, Y, Pi: double;
begin
    K:=StrToInt(Edit1.Text); //Ввод числа серий
    N:=StrToInt(Edit2.Text); //Ввод длины серии испытаний
    Randomize;
    M:=0; //Инициализация счетчика попаданий в круг
    for i:=1 to K do //Цикл по номеру серии испытаний
    begin
        for j:=1 to N do //Повторение испытаний
        begin
            X:=2*Random-1; //Вычисление X-координаты точки
            Y:=2*Random-1; //Вычисление Y-координаты точки
            If X*X+Y*Y<=1 Then M:=M+1 //Подсчет числа попаданий
                                //в круг
        end;
        Pi:=4*M/(i*N); //Вычисление "пи" после окончания серии
        //Вывод числа испытаний и "пи"
        ListBox1.Items.Add(IntToStr(i*N) + ' испытаний: Pi='
                            + FloatToStr(Pi))
    end
    end;

```

Вывод в поле *ListBox* производится вызовом метода *ListBox1.Items.Add(строка)*. Каждое обращение к этому методу добавляет строку к списку *ListBox*. Для просмотра всего списка выведенных строк используется линейка прокрутки.



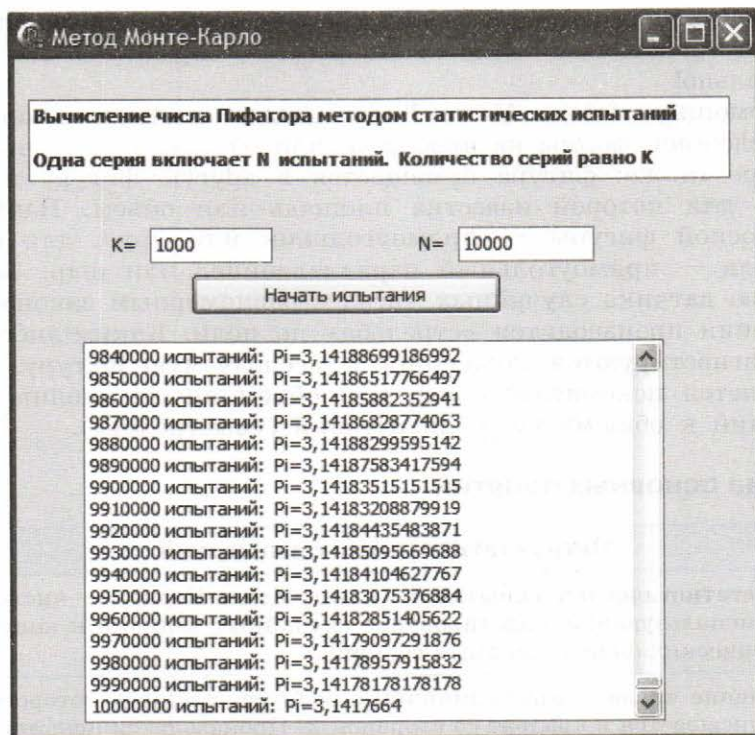


Рис. 2.19. Интерфейс программы «Метод Монте-Карло»

Обсудим результаты, представленные на рис. 2.19. Выполнены расчеты для 1000 серий, каждая из которых состоит из 10000 испытаний. Последний результат соответствует десяти миллионам испытаний. В результате получились только четыре верные цифры в значении числа  $\pi$ : 3,141. Более точное значение:  $\pi = 3,14159265$ .

Возникает вопрос: почему такой дорогой ценой (десять миллионов испытаний!) дали всего 4 цифры числа  $\pi$ ? И отсюда следующий вопрос: а если продолжать увеличивать число испытаний, то можно ли таким же способом получить сколько угодно верных цифр числа  $\pi$ ? Ответ: теоретически — да, практически — нет! В чем же дело?

Причина заключается в погрешности машинных вычислений с вещественными числами. Об этом говорилось в параграфе 2.4.2 учебника для 10 класса. В результате этой погрешности для точек, которые удалены от границы круга на расстояние порядка машинной ошибки, диагностика попадания в круг оказывается не всегда верной. Существует некоторое критическое число испыта-

ний, превышение которого ведет не к повышению, а к понижению точности вычисления числа  $\pi$ . Попробуйте определить его экспериментально!

С помощью метода Монте-Карло можно вычислять площади фигур сложной формы на плоскости или объемы трехмерных тел. Идея все та же: фигура помещается в другую фигуру простой формы, для которой известна площадь или объем. Например, для плоской фигуры — прямоугольник или круг, для объемного тела — прямоугольный параллелепипед или шар. Затем с помощью датчика случайных чисел с равномерным законом распределения производится «стрельба» по цели. Каким-либо образом диагностируются попадания в исследуемую фигуру. Далее вычисляется искомая площадь или объем по отношению числа попаданий к общему числу испытаний («выстрелов»).

### Система основных понятий

#### Метод статистических испытаний

**Метод статистических испытаний** (метод Монте-Карло) — численный метод, использующий моделирование случайных величин и получение статистических оценок искомых величин

**Вычисление числа  $\pi$ :** круг единичного радиуса, площадь которого равна  $\pi$ , вписывается в квадрат со стороной 2. Производятся испытания — случайный выбор точек внутри квадрата с подсчетом числа точек, попавших в круг. Точная формула:

$$\pi = 4 \cdot \lim_{P \rightarrow \infty} \frac{M}{P},$$

где  $P$  — число испытаний,  $M$  — число попаданий в круг

**Статистическая оценка:**  $\pi \approx 4 \cdot \frac{M}{P}$  для больших значений  $P$

**Причины погрешности результата:** конечное значение числа испытаний  $P$  и погрешность машинных вычислений

### Вопросы и задания

1. Что такое метод статистических испытаний (метод Монте-Карло)?
2. В чем состоит идея применения метода Монте-Карло для вычисления площадей и объемов?
3. Реализуйте на компьютере приведенную в параграфе программу вычисления числа  $\pi$ . Проследите за изменением результата с ростом числа испытаний.

Практикум. Раздел «Программирование»



### 2.4.5. Построение графика функции

Составим программу получения в окне вывода графика функции  $y = f(x)$ . Для получения графических изображений в Delphi используется объект *Canvas* — холст. Объект *Canvas* применяется не как самостоятельный компонент, а как свойство формы (*Form.Canvas*). Вывод рисунка на холст происходит с возникновением события *Paint* формы. Это событие появляется каждый раз, когда возникает необходимость вывода окна приложения (при запуске приложения или при обновлении вида окна). Обработка этого события осуществляется процедурой *FormPaint*.

Рисование на холсте происходит путем закрашивания точек на *графической поверхности* окна. Совокупность таких точек (пикселей) образует графическую сетку. Положение каждой точки сетки характеризуется двумя координатами: горизонтальной ( $x$ ) и вертикальной ( $y$ ). Следует учитывать, что координаты отсчитываются от левого верхнего угла. Один шаг графической сетки равен расстоянию между соседними пикселями (рис. 2.20).

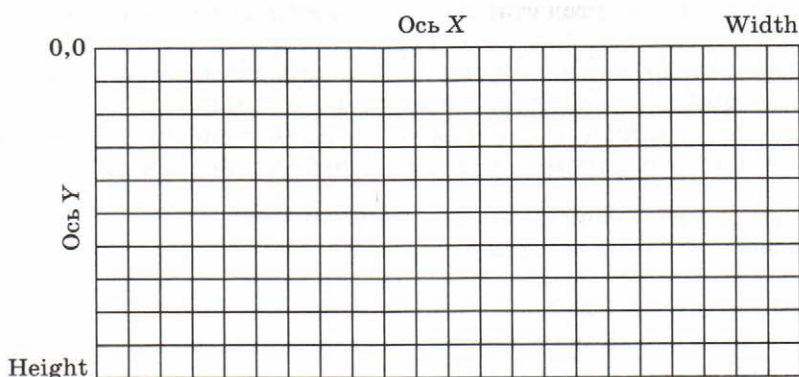


Рис. 2.20. Структура и параметры графической сетки

Максимальное число шагов по оси  $X$  хранит свойство формы *ClientWidth*, а по оси  $Y$  — свойство *ClientHeight*.

Для создания любых рисунков используются основные графические примитивы — точки, линии, окружности, прямоугольники и др. Чтобы нарисовать такие примитивы, используются соответствующие методы объекта *Canvas*.

В приведенной ниже программе рисования графика функции использованы следующие методы для построения изображения на холсте и нанесения надписей:

MoveTo(x, y: integer)	Устанавливает указатель текущей точки для рисования в позицию с указанными значениями координат $x, y$
LineTo(x, y: integer)	Вычерчивает линию из текущей точки в точку с указанными координатами. Вид линии определяет свойство <i>Pen</i> (перо)
TextOut(x, y: integer; s: string)	Выводит на экран строку $s$ от точки с координатами $(x, y)$ . Шрифт определяет свойство <i>Font</i> поверхности ( <i>Canvas</i> ), на которую выводится текст. Цвет закрашки области вывода текста определяет свойство <i>Brush</i> (Кисть) этой же поверхности

Сведения о других методах см. в описаниях Delphi.

График функции строится по точкам с помощью установки значения свойства `Pixels[x,y]:=z`, где  $x, y$  — координаты пикселя,  $z$  — цвет пикселя.

Следующая программа выполняет построение графика функции вида:  $f(x) = 2\sin x \cdot e^{x/5}$  на отрезке  $x \in [-10, 10]$ . Для вычисления значений функции составлена подпрограмма-функция  $f$ . Построение графика выполняет процедура с именем `GrOfFunc`. Текст программы снабжен подробными комментариями. Постарайтесь внимательно его изучить и понять смысл программы!

```
// ФУНКЦИЯ ДЛЯ ГРАФИЧЕСКОГО ПРЕДСТАВЛЕНИЯ
Function f(x: real): real;
begin
  f:=2*sin(x)*exp(x/5)
end;
//ПРОЦЕДУРА ПОСТРОЕНИЯ ГРАФИКА
Procedure GrOfFunc;
Var x1, x2, //границы измерения аргумента функции
    y1, y2, // границы изменения значения функции
    x,      // аргумент функции
    y,      // значение функции в точке x
    dx: real; // приращение аргумента
    l, b: integer; // левый нижний угол области вывода графика
    w, h: integer; // ширина и высота области вывода графика
    mx, my: real; // масштаб по осям X и Y
    x0, y0: integer; // точка — начало координат
    n, sh, s: integer; // число единичных меток на осях
begin
  // Расчет области вывода графика
  l:=20;
```



```
b:=Form6.ClientHeight-20; // у-координата левого нижнего
                        // угла
h:=Form6.ClientHeight-40; // высота
w:=Form6.ClientWidth-40; // ширина
x1:=-10; // нижняя граница диапазона аргумента
x2:=10; // верхняя граница диапазона аргумента
dx:=0.01; // шаг аргумента
// Вычисление максимального и минимального значения
// функции на отрезке [x1,x2]
y1:=f(x1); // переменная для минимума
y2:=f(x2); // переменная для максимума
x:=x1;
repeat
  y:=f(x);
  if y<y1 then y1:=y;
  if y>y2 then y2:=y;
  x:=x+dx;
until (x>=x2);
// Вычисление масштаба
my:=h/abs(y2-y1); // масштаб по оси Y
mx:=w/abs(x2-x1); // масштаб по оси X
x0:=1+abs(Round(x1*mx)); //положение начала
y0:=b-abs(Round(y1*my)); // координат
// РИСОВАНИЕ И РАЗМЕТКА ОСЕЙ КООРДИНАТ
with Form6.Canvas do
begin
  MoveTo(x0,b); LineTo(x0,b-h); //рисуем ось Y
  MoveTo(l,y0); LineTo(l+w,y0); //рисуем ось X
  //Разметка оси OY
  TextOut(x0+5,b-h,FloatToStrF(y2,ffGeneral,6,3)); //нанесем
                        //на ось Y max
  TextOut(x0+5,b,FloatToStrF(y1,ffGeneral,6,3)); //и min
                        //значения функции
  n:=Round(Form6.ClientHeight/40); //количество меток
  if (y2-y1)<100 then //подбираем шаг расстановки меток
                        //в зависимости от диапазона
  begin //изменения значений функции
    sh:=round((y2-y1)/n);
    s:=Round(y1)
  end
  else
  begin
    sh:=(Round((y2-y1)/n) div 10)*10; //выделим только
                        //десятки
    s:=(Round(y1) div 10)*10
  end;
end;
```

```

repeat
  MoveTo(Round(x0-2), Round(y0+s*my)); //рисует риски
                                           //меток
  LineTo(Round(x0+2), Round(y0+S*my)); //и проставляем
                                           //числа
  TextOut(x0-30,Round(y0-5+s*my),
          FloatToStrF(-1*S,ffGeneral,6,3));
  S:=S+sh;
until (s>y2);
//Разметка оси OX
n:=Round(w/40);
if (x2-x1)<100
then //аналогично, для оси OX
begin
  sh:=Round((x2-x1)/n);
  s:=Round(x1)
end
else
begin
  sh:=(Round((x2-x1)/n) div 10)*10;
  s:=(Round(x1) div 10)*10
end;
repeat
  MoveTo(Round(x0+s*mx),y0-2);
  LineTo(Round(x0+S*mx),y0+2);
  TextOut(Round(x0-5+s*mx),y0+15,
          FloatToStrF(S,ffGeneral,6,3));
  S:=S+sh;
until (s>x2);
// ПОСТРОЕНИЕ ГРАФИКА ФУНКЦИИ
x:=x1;
repeat
  y:=f(x);
  //Закрашивание точки графической сетки красным цветом
  Pixels [x0+Round(x*mx),y0-Round(y*my)]:=clRed;
  x:=x+dx;
until (x>=x2);
end;
end;

//Запуск построения графика с открытием формы
Procedure TForm6.FormPaint(Sender: TObject);
begin
  GrOfFunc;
end;

```



```
//Перерисовывание графика с изменением размера формы  
Procedure TForm6.FormResize(Sender: TObject);  
begin  
    // очистить форму  
    Form6.Canvas.FillRect(Rect(0,0,ClientWidth,ClientHeight));  
    // построить график  
    GrOfFunc  
end;
```

График строится в пределах прямоугольника, имеющего в окне вывода левое, правое, верхнее и нижнее поля, равные 20 шагам графической сетки. Шаг изменения математического значения координаты  $x$  равен 0,01. Масштабы по оси  $X$  и оси  $Y$  (переменные  $tx$  и  $ty$ ) равны числу шагов графической сетки, приходящихся на единицу изменения математических значений аргумента  $x$  и функции  $y$ . Принято использовать следующую терминологию: математические значения аргумента  $x$  и функции  $y$  определяют положение графика функции в мировой системе координат. А структура, представленная на рис. 2.20, называется экранной системой коор-

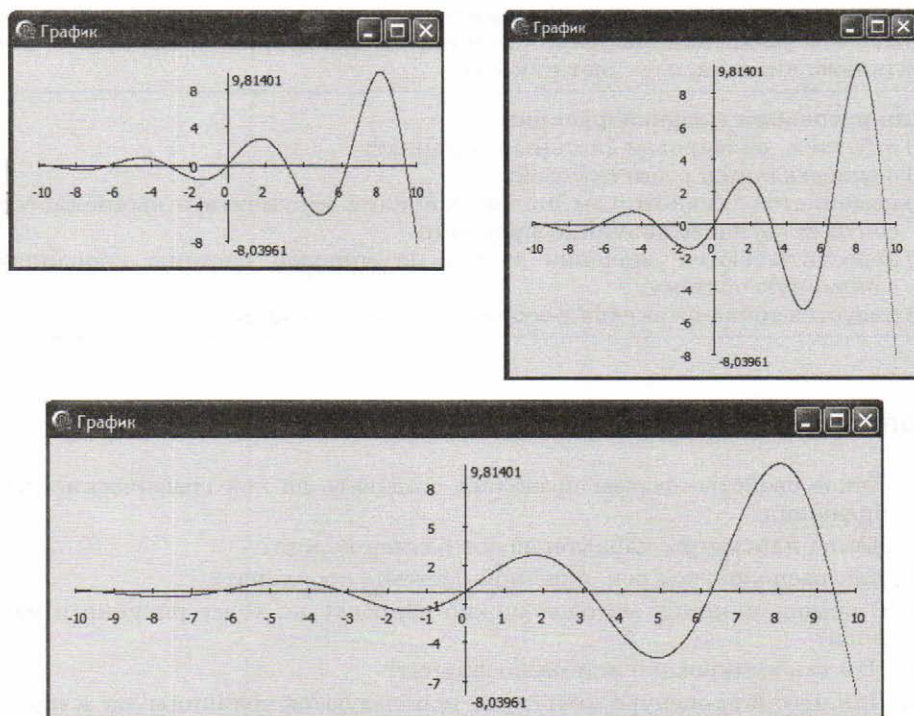


Рис. 2.21. Графики функции для разных размеров окна вывода, полученные с помощью процедуры GrOfFunc

*динат*. Масштабы  $tx$  и  $ty$  используются для пересчета положения точек графика из мировой системы в экранную систему координат.

Если вручную изменять размеры окна вывода, то происходит событие формы, которое называется *Resize*. Это событие обрабатывается процедурой `FormResize`, которая производит очистку формы от старого рисунка и обеспечивает перерисовывание графика для нового размера окна. На рисунке 2.21 показаны три варианта изображения графика для разных размеров холста (окна вывода).



## Система основных понятий

### Построение графика функции

**Canvas** (холст) — свойство формы, реализованное как объект, позволяющее наносить на форму графические изображения

**Свойства холста:** параметры графической сетки `ClientWidth` и `ClientHeight`, определяющие горизонтальный и вертикальный размеры

**Рисование на холсте:** производится с помощью команд изображения графических примитивов или закрашиванием точек графической сетки посредством установки значения свойства `Pixels[x, y] := z`, где  $x, y$  — координаты пикселя,  $z$  — цвет пикселя

**Для рисования графика функции:**

- 1) строятся оси мировой системы координат;
- 2) осуществляется разметка осей;
- 3) изменяется с некоторым шагом значение аргумента и вычисляется соответствующее значение функции;
- 4) пересчитываются значения  $x$  и  $y$  из мировой системы координат в экранную систему;
- 5) рисуется точка на экране в соответствующей позиции



## Вопросы и задания

1. Какое свойство формы позволяет создавать на ней графические изображения?
2. Какие параметры характеризуют размер холста?
3. Как расположены оси экранной системы координат?
4. С помощью каких методов можно наносить на холст рисунки и надписи?
5. Что такое мировая система координат?
6. Для чего в процедуре `GrOfFunc` используются величины  $tx$  и  $ty$ ?
7. При наступлении какого события выполняется программа рисования графика?



8. За счет чего происходит перерисовывание графика с изменением размеров окна вывода (формы)?
9. Что нужно изменить в программе для того, чтобы нарисовать график другой функции в другом диапазоне значений  $x$ ?
10. Какие операторы надо добавить в программу для того, чтобы на концах осей координат нарисовать стрелочки и написать обозначения осей  $X$  и  $Y$ ?
11. Каким образом можно изменить цвет линии графика?
12. Почему при изменении размера окна вывода изменяется разметка осей? Проанализируйте, как это делается в программе.

Практикум. Раздел «Программирование»

ЭОР к главе 2 на сайте ФЦИОР (<http://fcior.edu.ru>)

## 2.2.

- Начальные сведения о программах на языке Pascal
- Простейшие операции языка Pascal.
- Этапы разработки программы, ее структура. Создание шаблона программы на языке Pascal
- Объявление переменных в программе (на примере языка Pascal). Использование. Присваивание. Практическая работа
- Основные типы данных: Integer, Real, Boolean, Character и String. Работа с переменными и константами (на примере языка Pascal) (И)
- Объявление переменных в программе. Перечислимые и интервальные типы (на примере языка Pascal). Практическая работа
- Реализация основных алгоритмических конструкций
- Операторы ветвления if и case (на примере языка Pascal). Практическая работа
- Основные элементы языка программирования (на примере языка Pascal). Циклы. Работа с циклами. Использование циклов в программе. Вложенные циклы.
- Работа с массивами. Одномерные массивы. Алгоритмы работы с массивами. Обработка массива в цикле. Подсчет суммы элементов, максимум и минимум, поиск и сортировка элементов в массиве (на примере языка Pascal). (И)

## 2.4.

- Основные понятия и принципы ООП
- Этапы объектно-ориентированного программирования
- Основные принципы объектно-ориентированного программирования: понятие классов и объектов, их свойств и методов. Практика.



# ОГЛАВЛЕНИЕ

От авторов .....	3
Глава 1. Информационные системы.....	5
<b>1.1. Основы системного подхода .....</b>	<b>5</b>
1.1.1. Что такое система .....	5
1.1.2. Модели систем .....	10
1.1.3. Что такое информационная система .....	15
<i>О профессиях: профессии, связанные с созданием</i> <i>информационных систем .....</i>	<i>20</i>
1.1.4. Инфологическая модель предметной области.....	22
<b>1.2. Реляционные базы данных .....</b>	<b>25</b>
1.2.1. Реляционные базы данных и СУБД .....	25
1.2.2. Проектирование реляционной модели данных .....	29
1.2.3. Создание базы данных .....	33
1.2.4. Простые запросы к базе данных .....	38
1.2.5. Сложные запросы к базе данных .....	42
Глава 2. Методы программирования.....	50
<b>2.1. Эволюция программирования.....</b>	<b>50</b>
<i>О профессиях: профессии, связанные с программированием .....</i>	<i>57</i>
<b>2.2. Структурное программирование.....</b>	<b>58</b>
2.2.1. Паскаль — язык структурного программирования .....	58
2.2.2. Элементы языка и типы данных.....	61
2.2.3. Операции, функции, выражения .....	66
2.2.4. Оператор присваивания, ввод и вывод данных.....	72
2.2.5. Структуры алгоритмов и программ .....	80
2.2.6. Программирование ветвлений.....	86
2.2.7. Программирование циклов .....	91
2.2.8. Вспомогательные алгоритмы и подпрограммы .....	97
2.2.9. Массивы .....	104
2.2.10. Типовые задачи обработки массивов .....	110
2.2.11. Метод последовательной детализации.....	115
2.2.12. Символьный тип данных.....	120
2.2.13. Строки символов .....	125
2.2.14. Комбинированный тип данных .....	132
<b>2.3. Рекурсивные методы программирования .....</b>	<b>137</b>
2.3.1. Рекурсивные подпрограммы .....	137
2.3.2. Задача о Ханойской башне .....	142
2.3.3. Алгоритм быстрой сортировки .....	145
<b>2.4. Объектно-ориентированное программирование .....</b>	<b>147</b>
2.4.1. Базовые понятия объектно-ориентированного программирования.....	147
2.4.2. Система программирования Delphi.....	153
2.4.3. Этапы программирования на Delphi.....	159
2.4.4. Программирование метода статистических испытаний ...	164
2.4.5. Построение графика функции .....	169



УДК 004.9  
ББК 32.97  
С30

**Семакин И. Г.**

**С30 Информатика. Углубленный уровень : учебник для 11 класса : в 2 ч. Ч. 1 / И. Г. Семакин, Е. К. Хеннер, Л. В. Шестакова. — М. : БИНОМ. Лаборатория знаний, 2014. — 176 с. : ил.**

ISBN 978-5-9963-1687-8 (Ч. 1)

ISBN 978-5-9963-1689-2

Учебник предназначен для изучения курса информатики на углубленном уровне в 11 классах общеобразовательных учреждений. Содержание учебника опирается на изученный в 7–9 классах курс информатики для основной школы и разработано в соответствии с Федеральным государственным образовательным стандартом для среднего (полного) образования 2012 г. Рассматриваются теоретические основы информатики, аппаратное и программное обеспечение компьютера, современные информационные и коммуникационные технологии.

Учебник входит в учебно-методический комплект, включающий также учебник для 10 класса, практикум и методическое пособие для учителя.

УДК 004.9  
ББК 32.97

---

*Учебное издание*

**Семакин Игорь Геннадьевич  
Хеннер Евгений Карлович  
Шестакова Лидия Валентиновна**

**ИНФОРМАТИКА.  
УГЛУБЛЕННЫЙ УРОВЕНЬ**

**Учебник для 11 класса**

**В двух частях**

**Часть первая**

*Ведущий редактор О. А. Полежаева*

*Ведущие методисты: И. Л. Сретенская, И. Ю. Хлобыстова*

*Художники: Н. А. Новак, Я. В. Соловцова, Ю. С. Белаш*

*Технический редактор Е. В. Денюкова*

*Корректор Е. Н. Клитина*

*Компьютерная верстка: В. А. Носенко*

Подписано в печать 20.03.14. Формат 70×100/16.

Усл. печ. л. 14,30. Тираж 15 000 экз. Заказ № 35585.

Издательство «БИНОМ. Лаборатория знаний»

125167, Москва, проезд Аэропорта, д. 3

Телефон: (499) 157-5272

e-mail: binom@Lbz.ru

<http://www.Lbz.ru>, <http://e-umk.Lbz.ru>, <http://metodist.Lbz.ru>

При участии ООО Агентство печати «Столица»

[www.apstolica.ru](http://www.apstolica.ru); e-mail: [apstolica@bk.ru](mailto:apstolica@bk.ru)

Отпечатано в соответствии с качеством предоставленных издательством электронных носителей в ОАО «Саратовский полиграфкомбинат».

410004, г. Саратов, ул. Чернышевского, 59. [www.sarpk.ru](http://www.sarpk.ru)

---

ISBN 978-5-9963-1687-8 (Ч. 1)  
ISBN 978-5-9963-1689-2

© БИНОМ. Лаборатория знаний, 2014