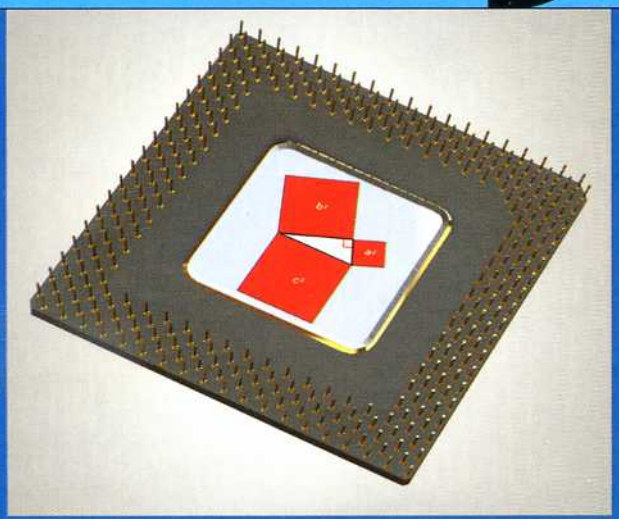
ФГОС

9



И. Г. Семакин
Л. А. Залогова
С. В. Русаков
Л. В. Шестакова

ИНФОРМАТИКА

ИЗДАТЕЛЬСТВО БИНОМ

ФГОС

И. Г. Семакин, Л. А. Залогова,
С. В. Русаков, Л. В. Шестакова

ИНФОРМАТИКА

Учебник для 9 класса

3-е издание

Рекомендовано
Министерством образования и науки Российской Федерации
к использованию при реализации имеющих государственную
аккредитацию образовательных программ начального общего,
основного общего, среднего общего образования

Москва
БИНОМ. Лаборатория знаний
2015

УДК 004.9
ББК 32.97
С30

Семакин И. Г.

С30 Информатика : учебник для 9 класса / И. Г. Семакин, Л. А. Залогова, С. В. Русаков, Л. В. Шестакова. — 3-е изд. — М. : БИНОМ. Лаборатория знаний, 2015. — 200 с. : ил.

ISBN 978-5-9963-1938-1

Учебник предназначен для изучения курса информатики в 9 классе общеобразовательной школы. Учебник содержит теоретический материал курса, вопросы и задания для закрепления знаний, в конце каждой главы в схематическом виде представлена система основных понятий этой главы. Некоторые главы учебника содержат дополнительный раздел, позволяющий изучить данную тему на углубленном уровне.

Учебник входит в учебно-методический комплект по информатике, наряду с учебниками для 7 и 8 классов, задачником-практикумом, методическим пособием для учителя и цифровыми образовательными ресурсами (ЦОР) из Единой коллекции. Соответствует федеральному государственному образовательному стандарту основного общего образования (2010 г.).

УДК 004.9
ББК 32.97

Учебное издание

Семакин Игорь Геннадьевич
Залогова Любовь Алексеевна
Русаков Сергей Владимирович
Шестакова Лидия Валентиновна

ИНФОРМАТИКА

Учебник для 9 класса

Редактор *Е. В. Баклашова*. Ведущий методист *И. Л. Сретенская*
Обложка: *С. Инфантэ*. Художественный редактор *Н. А. Новак*
Иллюстрации: *М. Ю. Ларкин*. Технический редактор *Е. В. Денюкова*
Корректор *Е. Н. Клитина*. Компьютерная верстка: *В. А. Носенко*

Подписано в печать 27.01.15. Формат 70×100/16.
Усл. печ. л. 16,25. Тираж 25 000 экз. Заказ № 36446.

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; e-mail: 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

ISBN 978-5-9963-1938-1

Оглавление

Уважаемые ученики!

В работе с книгой вам помогут навигационные значки:

 — Важное утверждение или определение.

 — Ссылка на Единую коллекцию цифровых образовательных ресурсов (далее — ЕК ЦОР) в двух частях: часть 1 (8 класс); часть 2 (9 класс).

Адрес ЕК ЦОР: http://school-collection.edu.ru

Название ЦОР:

Часть 1:

☑ «Информатика — базовый курс», 8 класс Семакина И., Залоговой Л., Русакова С., Шестаковой Л.

Часть 2:

☑ «Информатика — базовый курс», 9 класс Семакина И., Залоговой Л., Русакова С., Шестаковой Л.

Название ЦОР включает часть, главу, параграф и номер ресурса.

 — Вопросы и задания для использования в подготовке к итоговой аттестации.

 — Вопросы и задания к параграфу.

 — Выполни программу на компьютере.

 — Выполни упражнение или лабораторную работу.

 — Домашний эксперимент или проект.

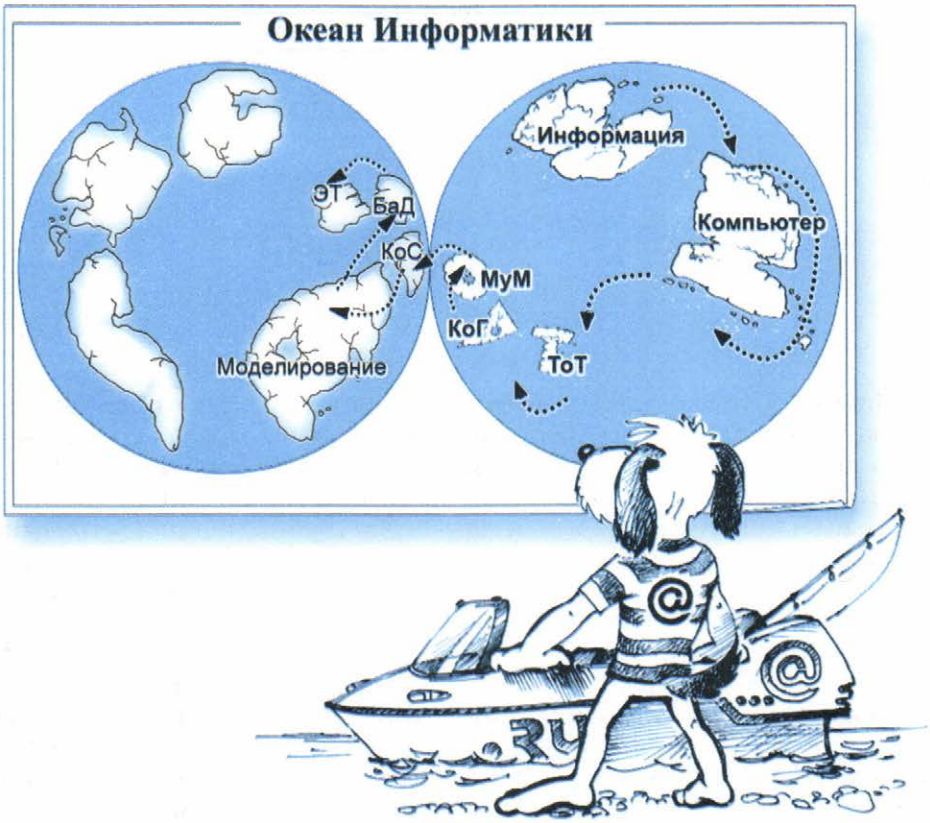
Введение

Начинается финальная часть вашего путешествия по океану Информатики. В начале этой части вы познакомитесь еще с одной обширной областью использования компьютеров — областью управления. С помощью компьютеров управляют самолетами, ракетами, промышленными установками, производственными коллективами, отраслями экономики и пр. Оказывается, что у таких разнообразных вариантов управления есть общие законы. Они были открыты в науке кибернетике. С основными понятиями этой науки вам предстоит познакомиться.

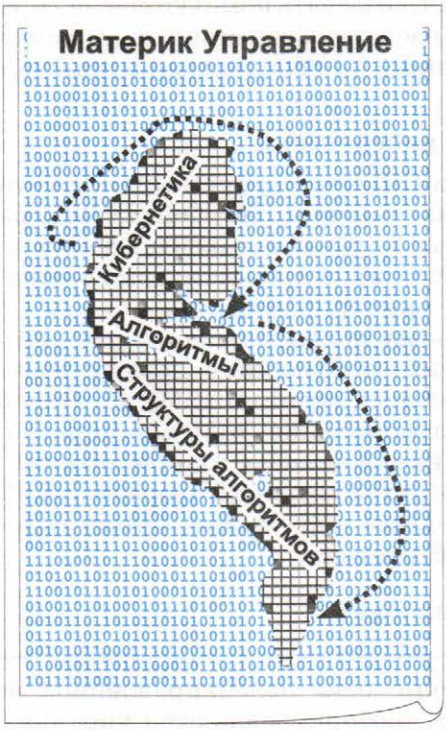
Всякое управление происходит по определенным правилам, которые называются алгоритмом управления. Алгоритм, записанный на языке программирования, «понятном» компьютеру, называется компьютерной программой. В этой части курса вы научитесь строить алгоритмы и писать несложные программы на языке программирования Паскаль.

Изобретения различных средств и инструментов для хранения, передачи и обработки информации совершались с древних времен и до наших дней. С историей таких изобретений вы познакомитесь в последней главе учебника. История компьютеров и информационно-коммуникационных технологий (ИКТ) берет начало с середины XX века. С этого времени начинается процесс информатизации различных областей деятельности людей, происходит формирование информационно-ориентированного общества. Проблемы информационного общества изучает раздел информатики, который называется социальной информатикой. О некоторых вопросах социальной информатики вы узнаете в конце нашего курса.

Посмотрите на карту Океана Информатики, в которой осталось всего три белых пятна — три материка знаний, которые вам предстоит посетить и освоить в этом году вместе с нашим проводником, отважным путешественником Точкой-Ру. Счастливого вам путешествия и отличных результатов!


Океан Информатики
ЭТ
БаД
КоС
МуМ
КоГ
ТоТ
Моделирование
Информация
Компьютер

Глава I
Управление и алгоритмы



Здесь вы узнаете:

- что изучает наука кибернетика
- что такое алгоритм управления
- какие бывают алгоритмы и как они описываются

§ 1

Управление и кибернетика

Основные темы параграфа:

- *возникновение кибернетики;*
- *что такое управление;*
- *алгоритм управления.*

Вы уже знакомы с некоторыми областями использования компьютеров. Знаете, что с помощью компьютера можно печатать книги, выполнять чертежи и рисунки; быстро передавать информацию на большие расстояния, создавать компьютерные справочники на любую тему; производить расчеты. Существует еще одно важное приложение компьютерной техники — использование компьютеров для управления.

Возникновение кибернетики

В 1948 году в США и Европе вышла книга американского математика ***Норберта Винера*** «Кибернетика, или Управление и связь в животном и машине». Эта книга провозгласила рождение новой науки — кибернетики.

Норберт Винер

Не случайно время появления этого научного направления совпало с созданием первых электронно-вычислительных машин (ЭВМ). Н. Винер предвидел, что использование ЭВМ для управления станет одним из важнейших их приложений, а для этого потребуется глубокий теоретический анализ самого процесса управления. Этому и посвящена наука кибернетика.

Что такое управление

Управление есть целенаправленное воздействие одних объектов, которые являются управляющими, на другие объекты — управляемые.

Простейшая ситуация — два объекта; один — управляющий, второй — управляемый. Например: человек и телевизор, хозяин и собака, светофор и автомобиль. В первом приближении взаимодействие между такими объектами можно описать схемой, изображенной на рис. 1.1.

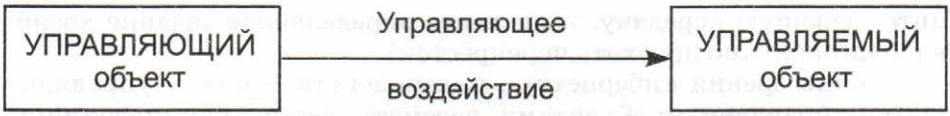


Рис. 1.1. Схема системы управления без обратной связи

В приведенных примерах управляющее воздействие производится в разных формах: человек нажимает кнопку управления телевизором; хозяин голосом подает команду собаке; светофор разными цветами управляет движением автомобилей и пешеходов на перекрестке.

С кибернетической точки зрения все варианты управляющих воздействий следует рассматривать *как управляющую информацию, передаваемую в форме команд.*

В примере с телевизором через пульт управления передаются команды следующего типа: «включить/выключить», «переключить канал», «увеличить/уменьшить громкость». Хозяин передает собаке команды голосом: «Сидеть!», «Лежать!», «Взять!». Световые сигналы светофора шофер воспринимает как команды: красный — «стоять», желтый — «приготовиться», зеленый — «ехать».

Алгоритм управления

В данном выше определении сказано, что управление есть целенаправленный процесс, т. е. команды отдаются не случайным образом, а с вполне определенной целью. В простейшем случае цель может быть достигнута после выполнения одной команды. Для достижения более сложной цели бывает необходимо выполнить последовательность (серию) команд.

Последовательность команд по управлению объектом, выполнение которой приводит к достижению заранее поставленной цели, называется **алгоритмом управления**.

В таком случае объект управления можно назвать **исполнителем управляющего алгоритма**. Значит, в приведенных выше примерах телевизор, собака, автомобиль являются исполнителями управляющих алгоритмов, направленных на вполне конкретные цели (найти

интересующую передачу, выполнить определенное задание хозяина, благополучно проехать перекресток).

С точки зрения кибернетики взаимодействие между управляющим и управляемым объектами рассматривается как *информационный процесс*. С этой позиции оказалось, что самые разнообразные процессы управления в природе, технике, обществе происходят сходным образом, подчиняются одним и тем же принципам.

Коротко о главном

Кибернетика — наука об общих свойствах процессов управления в живых и неживых системах.

Управление — это целенаправленное воздействие управляющего объекта на объект управления.

С точки зрения кибернетики управление происходит путем информационного взаимодействия между объектом управления и управляющим объектом.

Последовательность управляющих команд определяется алгоритмом управления, а исполнителем этого алгоритма является объект управления.

Вопросы и задания

1. Кто был основателем кибернетики? В каком году вышла первая книга по кибернетике?
2. Что такое управление?
3. Что представляет собой управляющее воздействие с точки зрения кибернетики?
4. Что такое алгоритм управления?
5. Определите, кто играет роль управляющего и кто (или что) играет роль объекта управления в следующих системах: школа, класс, самолет, стая волков, стадо коров. Подготовьте сообщение.
6. Для систем управления, выявленных в предыдущей задаче, назовите некоторые команды управления и скажите, в какой форме они отдаются.

www ЕК ЦОР: часть 2, глава 5, § 25. ЦОР № 4.

§ 2
Управление с обратной связью

Основные темы параграфа:

- *линейный алгоритм;*
- *обратная связь;*
- *модель управления с обратной связью;*
- *циклы и ветвления в алгоритмах;*
- *системы с программным управлением.*

Линейный алгоритм

Если внимательно обдумать рассмотренные в предыдущем параграфе примеры, то можно прийти к выводу, что строго в соответствии со схемой на рис. 1.1 работает только система «светофор — автомобили». Светофор, «не глядя», управляет движением машин, не обращая внимания на обстановку на перекрестке. Вот алгоритм работы светофора:

КРАСНЫЙ–ЗЕЛЕНЫЙ–ЖЕЛТЫЙ–КРАСНЫЙ–
ЗЕЛЕНЫЙ–ЖЕЛТЫЙ–КРАСНЫЙ и т. д.

Такой алгоритм называется ***линейным*** или ***последовательным***.

Обратная связь

Совсем иначе протекает процесс управления телевизором или собакой. Прежде чем отдать очередную команду, человек смотрит на состояние объекта управления, на результат выполнения предыдущей команды. Если он не нашел нужную передачу на данном канале, то он переключит телевизор на следующий канал; если собака не выполнила команду «Лежать!», хозяин повторит эту команду.

Из этих примеров можно сделать вывод, что управление происходит эффективнее, если управляющий не только отдает команды, т. е. работает **прямая связь**, но и принимает информацию от объекта управления о его состоянии. Этот процесс называется **обратной связью**.

Обратная связь — это процесс передачи информации о состоянии объекта управления управляющему объекту.

Модель управления с обратной связью

Управлению с обратной связью соответствует схема, изображенная на рис. 1.2.

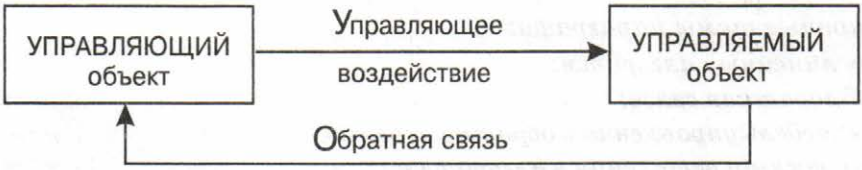


Рис. 1.2. Схема системы управления с обратной связью

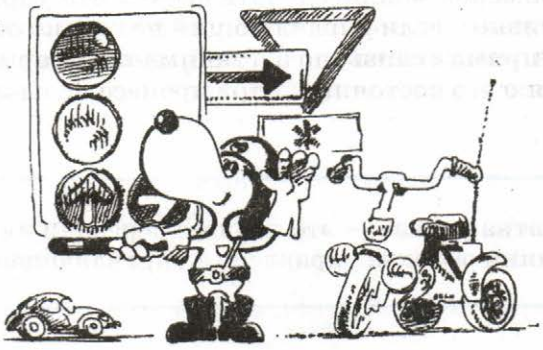
Циклы и ветвления в алгоритмах

Вот как можно записать алгоритм поиска нужной передачи по телевизору:

ВКЛЮЧИТЬ ТЕЛЕВИЗОР НА 1-М КАНАЛЕ
ПОКА НЕ БУДЕТ НАЙДЕНА ИСКОМАЯ ПЕРЕДАЧА,
ПОВТОРЯТЬ:
    ПЕРЕКЛЮЧИТЬ ТЕЛЕВИЗОР НА СЛЕДУЮЩИЙ КАНАЛ

В этом алгоритме содержится указание на повторение одних и тех же действий (переключить канал) по некоторому условию (пока не найдем передачу). Такой алгоритм называется **циклическим**.

Если вместо светофора на перекрестке дорог работает милиционер-регулировщик, то управление движением станет более рациональным. Регулировщик следит за скоплением машин на пересекающихся дорогах и дает «зеленую улицу» в том направлении, в котором в данный момент это нужнее. Нередко из-за «безмозглого» управления светофора на дорогах возникают «пробки». И тут на помощь может прийти регулировщик.



Назовем пересекающиеся дороги Дорога-1 и Дорога-2. Логика управления движением описывается следующим алгоритмом:

ЕСЛИ НА ДОРОГЕ-1 СКОПИЛОСЬ БОЛЬШЕ МАШИН
ТО ОТКРЫТЬ ДВИЖЕНИЕ ПО ДОРОГЕ-1
ИНАЧЕ ОТКРЫТЬ ДВИЖЕНИЕ ПО ДОРОГЕ-2

Здесь по определенному условию происходит выбор одного из двух действий. Такой алгоритм называется ***ветвящимся***. Проверка выполнения условия и в первом, и во втором примере стала возможна благодаря обратной связи: телезритель наблюдает за состоянием телевизора, милиционер наблюдает за состоянием движения на дорогах.

Итак, в варианте управления *без обратной связи* алгоритм может представлять собой только *однозначную (линейную) последовательность команд*. При наличии обратной связи и «интеллектуального» управляющего объекта алгоритмы управления могут иметь сложную структуру, содержащую альтернативные команды (ветвления) и повторяющиеся команды (циклы).

При наличии обратной связи алгоритм может быть более гибким, допускающим проверку условий, ветвления и циклы.

Принцип управления с обратной связью и есть основной закон, открытый наукой кибернетикой. Он действует в системах самой разной природы: технических, биологических, социальных.

Системы с программным управлением

Системы, в которых роль управляющего объекта поручается компьютеру, называются **автоматическими системами с программным управлением.**

Для функционирования такой системы, во-первых, между компьютером и объектом управления должна быть обеспечена прямая и обратная связь, во-вторых, в память компьютера должна быть заложена программа управления (алгоритм, записанный на языке программирования). Поэтому такой способ управления называют **программным управлением.**

Программное управление широко используется в технических системах: автопилот в самолете, автоматическая линия на заводе, ускоритель элементарных частиц в физической лаборатории, атомный реактор на электростанции и пр.

Коротко о главном

Управляющая информация передается по линии прямой связи в виде команд управления; по линии обратной связи передается информация о состоянии объекта управления.

Без учета обратной связи алгоритм управления может быть только линейным, при наличии обратной связи алгоритм может иметь сложную структуру, содержащую ветвления и циклы.

Системы, в которых роль управляющего объекта выполняет компьютер, называются автоматическими системами с программным управлением.

Вопросы и задания

1. Что такое обратная связь в процессе управления?
2. Какую структуру имеет управляющий алгоритм в системе без обратной связи?
3. Какую структуру может иметь управляющий алгоритм при наличии обратной связи?
4. Что такое система с программным управлением?
5. Проанализируйте систему «учитель—класс» как систему управления. Кто здесь управляющий объект, кто — объект управления? Какие действуют механизмы прямой и обратной связи?
6. Придумайте ситуации на уроке, когда учитель использует ветвление или цикл, принимая управляющие решения. Подготовьте сообщение.
7. Назовите систему, в которой учитель является объектом управления. Проанализируйте ее.
8. Опишите систему обучения, в которой роль учителя выполняет компьютер. Какие механизмы прямой и обратной связи действуют в такой системе? В чем преимущества и в чем недостатки компьютерного обучения по сравнению с традиционным?

www ЕК ЦОР: часть 2, § 26. ЦОР № 1.

§ 3
Определение и свойства алгоритма

Основные темы параграфа:

- *происхождение понятия «алгоритм»;*
- *исполнитель алгоритма;*
- *алгоритмический язык;*
- *свойства алгоритма;*
- *определение алгоритма;*
- *формальное исполнение алгоритма;*
- *что такое программа.*

Понятие алгоритма так же фундаментально для информатики, как и понятие информации. Поэтому в нем очень важно как следует разобраться.

Происхождение понятия «алгоритм»

Само слово «алгоритм» происходит от имени выдающегося математика средневекового Востока Мухаммеда ибн Мусы аль-Хорезми (787–850 гг.). Им были предложены приемы выполнения арифметических вычислений с многозначными числами (вам они хорошо знакомы из школьной математики). Позже в Европе эти приемы назвали алгоритмами, от *Algorithmi* — латинского написания имени аль-Хорезми. В наше время алгоритм понимается шире, не ограничивается только арифметическими вычислениями.

Исполнитель алгоритма

Из предыдущего параграфа вы узнали, что алгоритм — это последовательность команд управления каким-либо объектом. Мы назвали такой объект объектом управления или **исполнителем алгоритма**. Им может быть как техническое устройство, так и живое существо.

Рассмотрим исполнителя-человека. Для него можно сформулировать множество алгоритмов, например алгоритмы арифметических вычислений. С таким же успехом можно назвать алгоритмами множество различных инструкций, предписывающих последовательность действий человека для выполнения какой-либо работы. Например, кулинарный рецепт — это алгоритм работы повара с целью приготовления блюда; инструкция по сборке машинки из деталей детского конструктора — алгоритм для ребенка; инструкция по использованию кухонного комбайна — алгоритм для домохозяйки.

Вы, наверное, никогда не задумывались над тем, какое количество алгоритмов вам известно. Жизненный опыт человека растет с увеличением числа освоенных им алгоритмов. Например, чтобы ребенок научился покупать в магазине хлеб, ему нужно сначала рассказать (а лучше показать), как это делается. Освоив «алгоритм покупки хлеба», он в дальнейшем будет успешно выполнять эту работу.

Поиск выигрышной тактики, а следовательно, и алгоритма несложной игры — интересная и полезная задача. Рассмотрим одну из таких игр, которая называется игрой Баше.

Играют двое. Перед ними 21 предмет, допустим, камни (также может быть 11, 16, 26 и т. д.). Игроки берут камни по очереди. За один ход можно взять 1, 2, 3, 4 камня. Проигрывает тот, кто забирает последний камень.

Имеется выигрышная тактика для игрока, берущего камни вторым. Она заключается в том, чтобы брать такое количество камней, которое дополняет число камней, взятых соперником на предыдущем ходе, до пяти. Этот алгоритм можно описать в виде последовательности команд:

алг Игра Баше
нач
1. Предоставить ход сопернику.
2. Взять столько камней, чтобы в сумме с предыдущим ходом соперника получилось 5.
3. Если остался один камень, то объявить о своем выигрыше, иначе вернуться к выполнению команды 1.

кон

Игрок, строго следующий этому алгоритму, будет всегда выигрывать, даже если он не понимает, почему так происходит.

Алгоритмический язык

В приведенном примере записи алгоритма используется символика учебного **Алгоритмического языка (АЯ).**

Из примера видно, что при записи алгоритма на АЯ вначале пишется заголовок, начинающийся со **служебного слова алг** (сокращенное слово «алгоритм»). Затем указывается название алгоритма, которое составитель алгоритма придумывает сам. Следующая часть называется телом алгоритма. Она начинается со служебного слова **нач** (начало) и заканчивается словом **кон** (конец). Тело алгоритма представляет собой последовательность команд для исполнителя.

Здесь и в дальнейшем служебные слова в алгоритмах на АЯ будут записываться жирным шрифтом. В языках программирования (как и в АЯ) служебными называются слова, которые всегда употребляются в одном и том же смысле.

Свойства алгоритма

Процесс решения задачи должен быть разбит на последовательность отдельно выполняемых шагов.

Это свойство алгоритма называется **дискретностью.**

Всякий алгоритм составляется в расчете на конкретного исполнителя с учетом его возможностей. Для того чтобы алгоритм был выполним, нельзя включать в него команды, которые исполнитель не в состоянии выполнить. Нельзя повару поручать работу токаря, какая бы подробная инструкция ему ни давалась. У каждого исполнителя имеется свой перечень команд, которые он может исполнить. Такой перечень называется **системой команд исполнителя алгоритмов (СКИ).**

Алгоритм, составленный для конкретного исполнителя, должен включать только те команды, которые входят в систему команд исполнителя.

Это свойство алгоритма называется **понятностью.**

Алгоритм не должен быть рассчитан на принятие исполнителем каких-либо самостоятельных решений, не предусмотренных составителем алгоритма.

Каждая команда алгоритма должна определять однозначное действие исполнителя.

Это свойство алгоритма называется **точностью.**

Исполнение алгоритма должно завершиться за конечное число шагов.

Это свойство алгоритма называется **конечностью.**

Для успешного выполнения любой работы мало иметь ее алгоритм. Всегда требуются еще какие-то **исходные данные**, с которыми будет

работать исполнитель (продукты для приготовления блюда, детали для сбора технического устройства и т. п.). Исполнителю, решающему математическую задачу, требуется исходная числовая информация. Задача всегда формулируется так: *дана исходная информация, требуется получить какой-то результат.* В математике вы привыкли в таком виде записывать условия задач. Например:

Дано: катеты прямоугольного треугольника $a = 3$ см; $b = 4$ см.
Найти: гипотенузу c.

Алгоритм решения этой задачи можно представить в таком виде:

алг Гипотенуза
нач
1. Возвести a в квадрат.
2. Возвести b в квадрат.
3. Сложить результаты действий 1 и 2.
4. Вычислить квадратный корень результата действия 3 и принять его за значение c.

кон

Каждую из этих команд может выполнить любой человек, знающий основы математики, следовательно, они входят в его систему команд.

Еще пример: для поиска номера телефона нужного вам человека исходными данными являются: фамилия, инициалы человека и телефонная книга (точнее, информация, заключенная в телефонную книгу). Однако этого может оказаться недостаточно. Например, вы ищете номер телефона Смирнова А. И. и обнаруживаете, что в книге пять строк с фамилией «Смирнов А. И». Ваши исходные данные оказались *неполными* для точного решения задачи (вместо одного номера телефона вы получили пять). Оказалось, что нужно знать еще домашний адрес.

Набор: «фамилия — инициалы — телефонный справочник — адрес» является *полным набором данных* в этой ситуации.

Только имея полный набор данных, можно точно решить задачу.

Если исходные данные неполные, то задачу либо совсем нельзя решить (ничего нельзя узнать про гипотенузу по одному катету), либо получается неоднозначное решение (пять номеров телефонов).

В задачах управления физическими объектами (автомобиль, самолет, станок и т. п.) исходными данными является информация о состоянии объекта управления, об обстановке, его окружающей.

Определение алгоритма

Обобщая все сказанное, сформулируем определение алгоритма.

Алгоритм — понятное и точное предписание исполнителю выполнить конечную последовательность команд, приводящую от исходных данных к искомому результату.

Формальное исполнение алгоритма

Если алгоритм обладает перечисленными выше свойствами, то работа по нему будет производиться исполнителем ***формально*** (т. е. без всяких элементов творчества со стороны исполнителя). На этом основана работа программно управляемых исполнителей-автоматов, например промышленных роботов. Робот-манипулятор может выполнять работу токаря, если он умеет выполнять все операции токаря (включать станок, закреплять резец, перемещать резец, замерять изделие). От исполнителя не требуется понимания сущности алгоритма, он должен лишь точно выполнять команды, не нарушая их последовательности.

Что такое программа

А что такое программа? Отличается ли чем-то программа от алгоритма?

Программа — это алгоритм, записанный на языке исполнителя.

Иначе можно сказать так: алгоритм и программа не отличаются по содержанию, но могут отличаться по форме.

Для алгоритма строго не определяется форма его представления. Алгоритм можно изобразить графически, можно — словесно, можно — какими-нибудь специальными значками, понятными только его автору. Но программа должна быть записана на языке исполнителя.

Коротко о главном

Слово «алгоритм» происходит от имени Мухаммеда ибн Мусы аль-Хорезми, первым предложившего приемы выполнения арифметических операций с многозначными числами.

Исполнитель алгоритма — это тот объект, для управления которым составлен алгоритм.

Процесс решения задачи должен быть разбит на последовательность отдельных шагов (свойство дискретности алгоритма).

Система команд исполнителя (СКИ) — это вся совокупность команд, которые исполнитель умеет выполнять (понимает). Алгоритм можно строить только из команд, входящих в СКИ исполнителя (свойство понятности алгоритма).

Каждая команда алгоритма управления должна определять однозначное действие исполнителя (свойство точности алгоритма).

Выполнение алгоритма должно приводить к результату за конечное число шагов (свойство конечности алгоритма).

Для успешного выполнения работы, решения задачи необходимо сообщить (передать) исполнителю полный набор исходных данных.

Выполнение алгоритма исполнителем производится формально.

Программа от алгоритма может отличаться по форме, но не по содержанию. Программа — это алгоритм, представленный на языке исполнителя.

Вопросы и задания

1. Что такое алгоритм? Откуда произошло это слово?
2. Что такое исполнитель алгоритма?
3. Каковы основные свойства алгоритма?
4. Назовите исполнителей следующих видов работы: уборки мусора во дворе; перевозки пассажиров; выдачи заработной платы; приема экзаменов; сдачи экзаменов; обучения детей в школе. Попробуйте создать СКИ для каждого из этих исполнителей.
5. Определите полный набор данных для решения следующих задач обработки информации:
   - вычисления стоимости покупок в магазине;
   - вычисления суммы сдачи от данных вами продавцу денег;
   - определения времени показа по телевизору интересующего вас фильма;
   - вычисления площади треугольника;
   - определения времени падения кирпича с крыши дома;

- определения месячной платы за расход электроэнергии;
- перевода русского текста на итальянский язык;
- перевода итальянского текста на русский язык.

6. Попробуйте сформулировать алгоритмы обработки информации для заданий из п. 5, если исполнителем являетесь вы сами. Какие команды при этом вы должны уметь выполнять? Подготовьте сообщение.

ЕК ЦОР: часть 2, глава 5, § 27. ЦОР № 1.

§ 4

Графический учебный исполнитель

Основные темы параграфа:

- *назначение и возможности графического исполнителя (ГРИС);*
- *простые команды ГРИС;*
- *работа в программном режиме;*
- *линейные программы для ГРИС.*

Назначение и возможности графического исполнителя (ГРИС)

Учебные исполнители используются для обучения составлению управляющих алгоритмов.

Есть много учебных исполнителей, придуманных для занятий по информатике. У них разные, часто забавные названия: Черепашка, Робот, Чертежник, Кенгуренок, Пылесосик, Муравей, Кукарача и др. Одни исполнители создают рисунки на экране компьютера, другие складывают слова из кубиков с буквами, третьи перетаскивают предметы из одного места в другое. Все эти исполнители управляются программным путем. Любому из них свойственна определенная **среда деятельности, система команд управления, режимы работы.**

В предыдущих главах мы избегали детальных описаний работы с конкретными вариантами программ (редакторов, СУБД и пр.). И в этой главе мы не будем детально описывать работу с каким-то реальным учебным исполнителем из вышеперечисленных (в компьютерных классах разных школ может быть разное программное обеспечение). Мы опишем условного исполнителя, который очень похож на некоторых из существующих в главном: системой команд, языком и приемами программирования.

Многие из учебных исполнителей занимаются рисованием на экране компьютера. Из названных выше это Черепашка, Кенгуренок, Чертежник. Эту группу можно назвать графическими исполнителями. Пусть наш гипотетический (придуманный) исполнитель тоже будет из этой «компании». Назовем его ГРИС, что значит «ГРафический ИСполнитель».

Что умеет делать ГРИС? Он может перемещаться по полю и своим хвостом рисовать на этом поле (предположим, что у него есть хвост, к которому привязан кусочек мела).

Обстановка, в которой действует исполнитель, называется **средой исполнителя**. Среда графического исполнителя показана на рис. 1.3. Это лист (страница экрана) для рисования. ГРИС может перемещаться в горизонтальном и вертикальном направлениях с постоянным шагом. На рисунке 1.3 пунктиром показана сетка с шагом, равным шагу исполнителя. Исполнитель может двигаться только по линиям этой сетки. ГРИС не может выходить за границы поля.

Состояние исполнителя на поле определяется, во-первых, его местоположением (в какой точке поля он находится), во-вторых, направлением (куда он смотрит). Направление будем определять, как на географической карте: вверх — на север, вниз — на юг, влево — на запад, вправо — на восток. ГРИС может шагать или прыгать по линиям сетки, а также поворачиваться. Поворачиваться он умеет только против часовой стрелки.

Графический исполнитель — это объект управления. А управлять им будем мы с вами. Целью управления является получение определенного рисунка. Понятно, что этот рисунок может состоять только из горизонтальных и вертикальных отрезков, в других направлениях ГРИС двигаться не умеет.

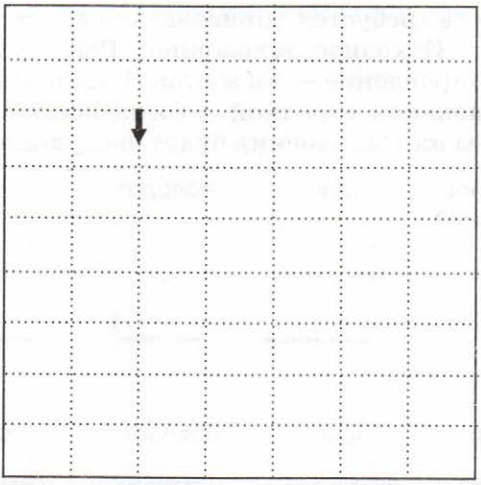


Рис. 1.3. Среда графического исполнителя. Стрелка указывает состояние исполнителя (местоположение и направление)

Задача обычно ставится так: исполнитель находится в данной точке поля, смотрит в данном направлении. Требуется получить определенный рисунок. Например: ГРИС находится в середине поля и смотрит на восток. Надо нарисовать букву «Т» с длиной каждой линии, равной четырем шагам.

Первоначально исполнителю придается исходное состояние. Это делается в специальном **режиме установки**.

Теперь перейдем к управлению графическим исполнителем. Здесь возможны два режима: **режим прямого управления** и **режим программного управления**.

Простые команды ГРИС

Работа в режиме прямого управления происходит так: человек отдает команду, ГРИС ее выполняет; затем отдается следующая команда и т. д. (как в примере с хозяином и собакой).

В режиме прямого управления система команд исполнителя следующая:

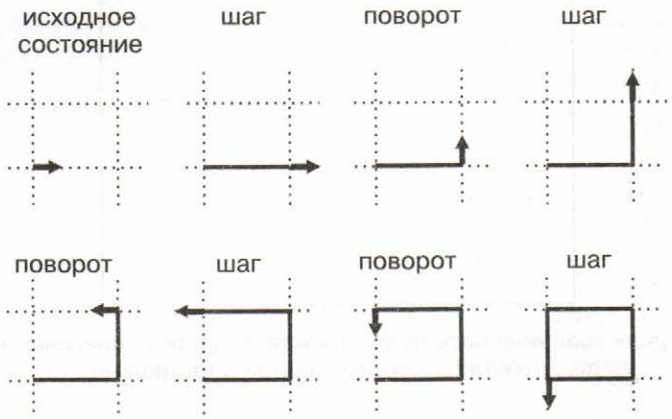
шаг — перемещение ГРИС на один шаг вперед с рисованием линии;

поворот — поворот на 90° против часовой стрелки;

прыжок — перемещение на один шаг вперед без рисования линии.

Эти команды будем называть **простыми командами**.

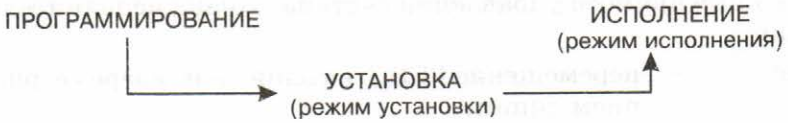
Например, пусть требуется нарисовать квадрат со стороной, равной одному шагу. Исходное положение ГРИС — в левом нижнем углу квадрата, направление — на восток. Будем отмечать состояние исполнителя маленькой стрелкой. Тогда последовательность команд и результаты их выполнения будут следующими:



Работа в программном режиме

Работа в программном режиме имитирует автоматическое управление исполнителем. Управляющая система (компьютер) обладает памятью, в которую заносится программа. *Человек составляет программу и вводит ее в память.* Затем ГРИС переводится в режим установки, и человек вручную (с помощью определенных клавиш) устанавливает исходное состояние исполнителя. После этого производится переход в **режим исполнения**, и ГРИС начинает работать по программе. Если возникает ситуация, при которой он не может выполнить очередную команду (выход за границу поля), то выполнение программы завершается аварийно. Если аварии не происходит, то работа исполнителя заканчивается на последней команде.

Таким образом, программное управление графическим исполнителем проходит этап подготовки (программирование и установка исходного состояния) и этап исполнения программы.



В режиме программного управления по-прежнему используются команды **шаг**, **поворот**, **прыжок**. Однако в этом режиме есть еще и другие команды. С ними вы познакомитесь позже.

Язык программирования для графического исполнителя — это учебный Алгоритмический язык (АЯ). Поэтому алгоритмы управления ГРИСом, записанные на АЯ, являются для него одновременно и программами.

Линейные программы для ГРИС

Будем осваивать программирование на примерах решения конкретных задач. С новыми командами СКИ будем знакомиться по мере появления потребности в них.

Задача 1. Составим и выполним программу, по которой ГРИС нарисует на поле букву «Т». Пусть длина вертикального и горизонтального отрезков должна быть равна четырем шагам.

Исходное состояние — чистый лист. Исполнитель находится в точке, где будет находиться левый конец горизонтального отрезка, направление — на восток.

Результат выполнения программы показан на рис. 1.4.

Структура такой программы (алгоритма) называется линейной. Команды выполняются одна за другой, каждая только один раз.

Для решения этой задачи оказалось достаточно той части СКИ, которая используется в режиме прямого управления.

Программа:

```
программа Буква Т
 нач
     шаг
     шаг
     шаг
     шаг
     поворот
     поворот
     прыжок
     прыжок
     поворот
     шаг
     шаг
     шаг
     шаг
кон
```

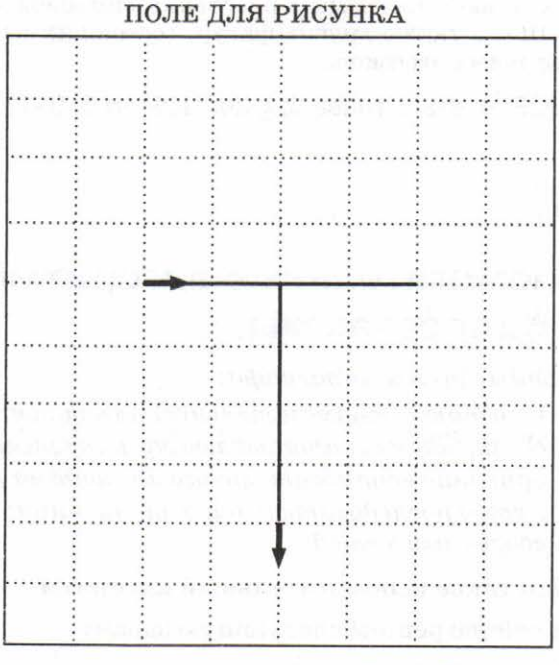


Рис. 1.4. Результат выполнения программы «Буква Т». Стрелками указаны начальное и конечное состояния ГРИС

Коротко о главном

ГРИС — это графический исполнитель, назначение которого — получение чертежей, рисунков на экране дисплея.

Управление ГРИС может происходить в режиме прямого управления или в режиме программного управления.

С помощью команд **шаг, поворот, прыжок** в пределах рабочего поля можно построить любой рисунок, состоящий из вертикальных и горизонтальных отрезков. Структура управляющего алгоритма при этом будет линейной.

Вопросы и задания

1. Какую работу может выполнять ГРИС?
2. Что представляет собой среда ГРИС?
3. В чем разница между управлением в прямом и программном режимах?
4. Какие простые команды входят в СКИ ГРИС; как они выполняются?
5. В какой последовательности происходит выполнение команд в линейном алгоритме?
6. Может ли данный ГРИС нарисовать: прямоугольник, треугольник, пятиконечную звезду, буквы «Н», «Х», «Р», «М»?
7. Составьте программы рисования символов «Е», «П», «Б», «Ч», «Ц», «Ш», а также других фигур, состоящих из горизонтальных и вертикальных отрезков.

www ЕК ЦОР: часть 2, глава 5, § 28. ЦОР № 5, 9–11, 13–15.

§ 5

Вспомогательные алгоритмы и подпрограммы

Основные темы параграфа:

- *что такое вспомогательный алгоритм;*
- *обращение к вспомогательному алгоритму (процедуре);*
- *описание вспомогательного алгоритма (процедуры);*
- *метод последовательной детализации;*
- *сборочный метод.*

Что такое вспомогательный алгоритм

А сейчас решим следующую задачу.

Задача 2. Пусть требуется составить программу, по которой ГРИС напишет на экране четырехзначное число 1919 (рис. 1.5).

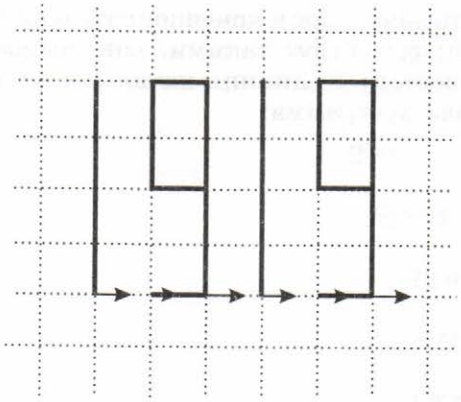


Рис. 1.5. Рисование числа 1919.
Стрелками указаны начальное и конечное состояния рисования каждой цифры

Конечно, можно поступить так, как в предыдущей задаче, написав одну длинную программу, по которой исполнитель шаг за шагом нарисует эти цифры. Но с очевидностью возникает другая идея: поскольку здесь дважды повторяются цифры 1 и 9, нельзя ли сократить работу, написав программы рисования той и другой цифры только один раз? Это действительно можно сделать.

Алгоритм, по которому решается некоторая подзадача из основной задачи и который, как правило, выполняется многократно, называется **вспомогательным алгоритмом**.

Вспомогательный алгоритм, записанный на языке программирования, называется **подпрограммой** или **процедурой**.

Обращение к вспомогательному алгоритму (процедуре)

В таком случае программа решения поставленной задачи разделяется на основную программу (основной алгоритм) и процедуры (вспомогательные алгоритмы). Каждая процедура должна иметь свое уникальное имя. Для рассматриваемой задачи имена процедур выберем следующими: ЕДИНИЦА и ДЕВЯТЬ. Тогда в основной программе команды обращения к этим процедурам будут такими:

сделай ЕДИНИЦА
сделай ДЕВЯТЬ

По этим командам управление передается соответствующим процедурам, и после их выполнения управление вернется к следующей команде основной программы.

Договоримся, что начальное и конечное состояния ГРИС при вычерчивании каждой цифры будут такими, как показано стрелками на рис. 1.6 (внизу, на восток). У единицы начальное и конечное состояния совпадают. Основная программа:

```
программа Число 1919
нач
    сделай ЕДИНИЦА
    прыжок
    сделай ДЕВЯТЬ
    прыжок
    сделай ЕДИНИЦА
    прыжок
    сделай ДЕВЯТЬ
кон
```

Данный пример познакомил вас с новой командой из СКИ графического исполнителя — командой ***обращения к процедуре***. Ее формат, т. е. общий вид, следующий:

сделай <имя процедуры>

Описание вспомогательного алгоритма (процедуры)

Вот и все! Так просто! Но теперь надо «объяснить» исполнителю, что такое ЕДИНИЦА и что такое ДЕВЯТЬ. Это делается в **описаниях процедур** (здесь порядок выполнения — по столбцам):

```
процедура ЕДИНИЦА
нач
    поворот
    шаг
    шаг
    шаг
    шаг
    поворот
    поворот
    прыжок
    прыжок
    прыжок
    прыжок
    поворот
кон
```

```
процедура ДЕВЯТЬ
нач
    шаг
    поворот
    шаг
    шаг
    шаг
    шаг
    поворот
    шаг
    поворот
кон
```

```
поворот
поворот
поворот
прыжок
прыжок
поворот
```

Формат описания процедуры:

процедура <имя процедуры>
нач
  <тело процедуры>
кон

Имя в описании и имя в обращении должны точно совпадать (никаких склонений по падежам!). Описание процедур располагается после основной программы.

Добавив к программе описание процедуры, мы тем самым расширили систему команд исполнителя. В данной программе стало возможным использование команды обращения к этой процедуре.

Метод последовательной детализации

Использованный нами подход облегчает программирование сложных задач. Задача разбивается на более простые подзадачи. Решение каждой оформляется в виде вспомогательного алгоритма, а основной алгоритм организует связку между ними.

Метод программирования, при котором сначала пишется основная программа, в ней записываются обращения к пока еще не составленным подпрограммам, а потом описываются эти подпрограммы, называется **методом последовательной (пошаговой) детализации**. Причем количество шагов детализации может быть гораздо бóльшим, чем в нашем примере, поскольку сами подпрограммы могут содержать внутри себя обращения к другим подпрограммам.

Сборочный метод

Возможен и другой подход к построению сложных программ: первоначально составляется множество подпрограмм, которые могут понадобиться при решении задачи, а затем пишется основная программа, содержащая обращения к ним. Подпрограммы могут быть объединены в **библиотеку подпрограмм** и сохранены в долговременной памяти компьютера. Такую библиотеку можно постепенно пополнять новыми подпрограммами.

Например, если для управления графическим исполнителем создать библиотеку процедур рисования всех букв и цифр, то программа получения любого текста будет состоять из команд обращения к библиотечным процедурам.

Описанный метод называется **сборочным программированием**. Часто в литературе по программированию используется такая терминология: метод последовательной детализации называют **программированием сверху вниз**, а сборочный метод — **программированием снизу вверх**.

Коротко о главном

Для упрощения программирования сложных задач используются вспомогательные алгоритмы.

Вспомогательный алгоритм — это алгоритм решения некоторой подзадачи исходной (основной) задачи.

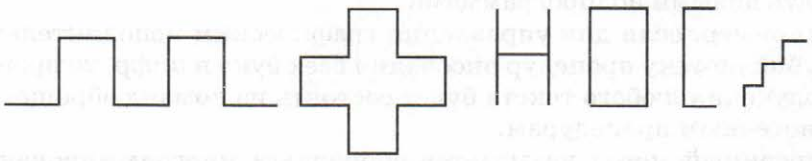
Вспомогательный алгоритм, записанный на языке программирования, называется процедурой.

Вспомогательный алгоритм должен быть описан. После этого в основном алгоритме можно использовать команду обращения к этому вспомогательному алгоритму.

Метод программирования, при котором сначала записывается основной алгоритм, а затем описываются использованные в нем вспомогательные алгоритмы, называется методом последовательной детализации, или программированием сверху вниз. Обратный порядок программирования называется программированием снизу вверх.

Вопросы и задания

1. Что такое основной алгоритм; вспомогательный алгоритм?
2. Чем отличается описание вспомогательного алгоритма от обращения к вспомогательному алгоритму?
3. Каковы правила описания вспомогательных алгоритмов (процедур) для исполнителя ГРИС?
4. Как записывается команда обращения к процедуре в языке исполнителя ГРИС?
5. В чем суть метода последовательной детализации?
6. Что такое программирование снизу вверх; сверху вниз?
7. Используя вспомогательные алгоритмы, запрограммируйте рисование следующих фигур.



ЕК ЦОР: часть 2, глава 5, § 29. ЦОР № 6, 9–12, 14, 15.

§ 6

Циклические алгоритмы

Основные темы параграфа:

- *команда цикла;*
- *цикл в процедуре;*
- *блок-схемы алгоритмов;*
- *цикл с предусловием.*

Команда цикла

Обсудим решение следующей задачи.

Задача 3. Исходное положение: ГРИС — у левого края поля, направление — на восток. Требуется нарисовать горизонтальную линию через весь экран.

Задачу можно решить, написав 15 раз команду **шаг** (если поперек поля рисунка 15 шагов). Но есть и более короткий вариант программы. Вот он:

```
пока впереди не край повторять
нц
  шаг
кц
```

Здесь использована команда, которая называется **циклом**. Формат команды цикла следующий:

```
пока <условие> повторять
нц
  <тело цикла>
кц
```

Служебное слово **нц** обозначает начало цикла, **кц** — конец цикла.

Это первая команда из СКИ, которая реализует обратную связь между графическим исполнителем и управляющим им компьютером. Она заключается в том, что проверяется, не вышел ли ГРИС на край поля, не грозит ли ему следующий шаг или прыжок в этом направлении аварией. Проверяемые условия звучат так: «*впереди край?*» или «*впереди не край?*». На что машина получает от исполнителя ответ «*да*» или «*нет*».

В приведенном примере проверяется условие «впереди не край?». Если «да», то делается шаг (т. е. выполняется <тело цикла>). Затем происходит возврат на проверку условия, и всё повторяется. Если проверка условия дает отрицательный результат (т. е. впереди край), то

выполнение цикла завершается и исполняется следующая после цикла команда программы.

При программировании цикла важно думать о том, чтобы цикл был конечным. Цикл, записанный выше, конечный. Двигаясь в одном направлении, исполнитель обязательно достигнет края, и на этом выполнение цикла закончится.

Ситуация, при которой выполнение цикла никогда не заканчивается, называется **зацикливанием**. Пусть ГРИС находится в середине поля. Исполнение следующего цикла:

```
пока впереди не край повторять
нц
    шаг
    поворот
кц
```

никогда не закончится. ГРИС будет бесконечно рисовать квадратик, так как проверка условия «впереди не край?» всегда будет давать положительный ответ.

Цикл в процедуре

Задача 4. Теперь составим программу, по которой графический исполнитель нарисует прямоугольную рамку по краю поля (рис. 1.6). Исходное положение: ГРИС находится в левом верхнем углу, смотрит на юг.

Рамка состоит из четырех линий, поэтому разумно воспользоваться процедурой, проводящей линию от края до края поля. Опять будем действовать методом последовательной детализации. Напишем сначала основную программу.

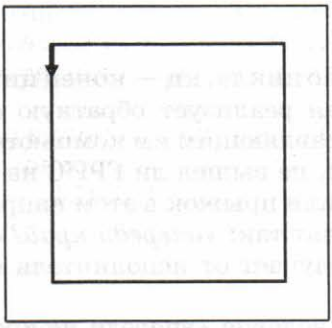


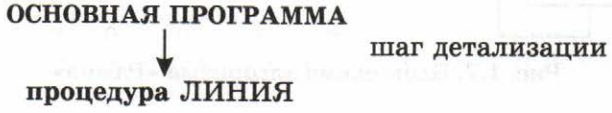
Рис. 1.6. Результат выполнения программы «Рамка». Стрелкой указано начальное состояние

```
программа Рамка
нач
    сделай ЛИНИЯ
    поворот
    сделай ЛИНИЯ
    поворот
    сделай ЛИНИЯ
    поворот
    сделай ЛИНИЯ
кон
```

Программа проведения линии нами уже рассматривалась. Осталось оформить ее в виде процедуры.

```
процедура ЛИНИЯ
нач
   пока впереди не край повторять
   нц
      шаг
   кц
кон
```

При составлении этой программы использовалась одношаговая детализация в такой последовательности:



Блок-схемы алгоритмов

Начиная с 50-х годов прошлого века, т. е. еще с эпохи ЭВМ первого поколения, программисты стали использовать графические схемы, изображающие алгоритмы, которые получили название блок-схем.

Блок-схема состоит из фигур (блоков), обозначающих действия исполнителя, и стрелок, соединяющих эти блоки и указывающих на последовательность их выполнения. Внутри каждого блока записывается выполняемое действие. Форма блока подсказывает характер действия, которое он обозначает. Для придания наглядности и единообразия схемам алгоритмов все графические элементы стандартизированы.

Посмотрите на рис. 1.7, где показана блок-схема алгоритма рисования рамки. Она состоит из двух частей: блок-схемы основного алгоритма и блок-схемы вспомогательного алгоритма ЛИНИЯ.

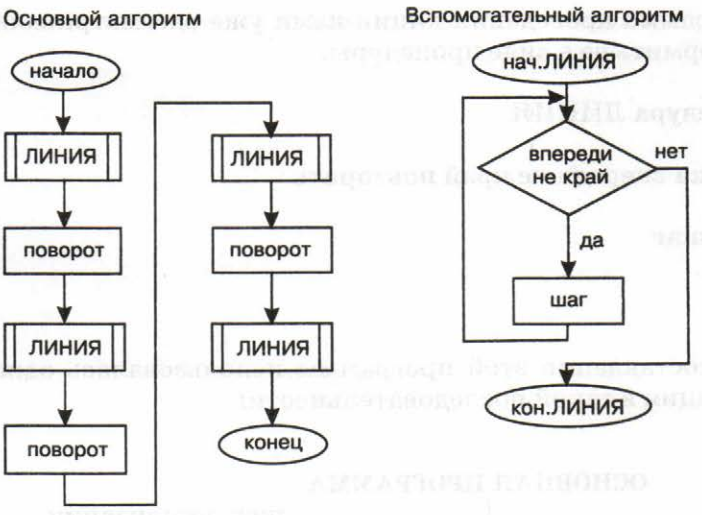


Рис. 1.7. Блок-схема алгоритма «Рамка»

Из этих схем понятно назначение блоков различной формы (рис. 1.8).

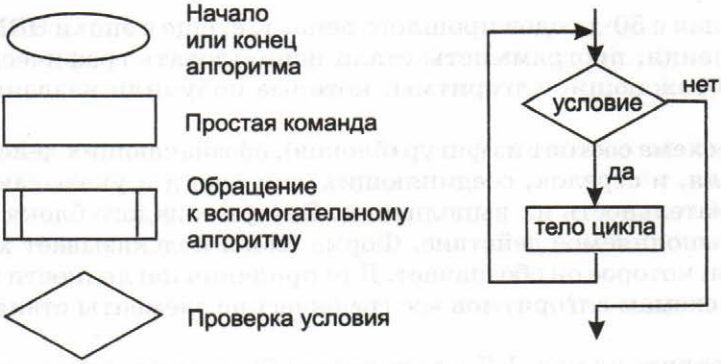


Рис. 1.8. Элементы блок-схем и структура «цикл»

Цикл с предусловием

Команда цикла изображается не отдельным блоком, а целой структурой, показанной на рис. 1.8. Такую структуру называют **циклом с предусловием** (так как условие предшествует телу цикла). Есть и другой вариант названия: **цикл-пока** (пока условие истинно, повторяется выполнение тела цикла).

При решении следующей задачи снова будем использовать метод последовательной детализации.

Задача 5. Требуется расчертить поле горизонтальными линиями (рис. 1.9). Исходное состояние исполнителя: верхний левый угол, направление — на юг.

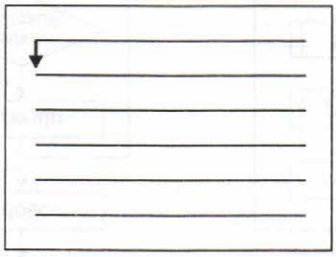


Рис. 1.9. Результат выполнения программы «Разлиновка». Стрелкой указано начальное состояние

В программе для решения этой задачи используется та же процедура ЛИНИЯ. Другая процедура — ВОЗВРАТ — возвращает ГРИС к левому краю поля для рисования следующей линии.

```
программа Разлиновка
нач
  пока впереди не край
  повторять
  нц
    поворот
    сделай ЛИНИЯ
    сделай ВОЗВРАТ
    прыжок
  кц
  поворот
  сделай ЛИНИЯ
кон
```

```
процедура ВОЗВРАТ
нач
  поворот
  поворот
  пока впереди не край
  повторять
  нц
    прыжок
  кц
  поворот
кон
```

Блок-схемы основного и вспомогательного алгоритмов представлены на рис. 1.10.

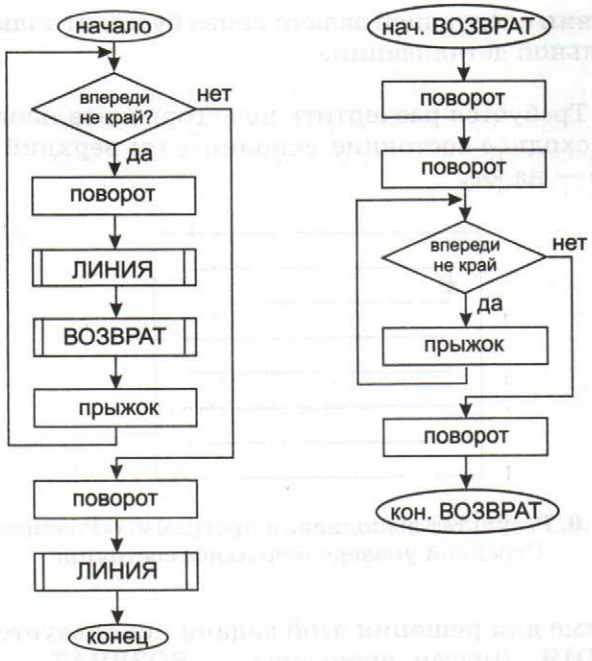


Рис. 1.10. Блок-схема алгоритма «Разлиновка»

Коротко о главном

Для программирования повторяющихся действий применяется команда цикла, которая имеет следующую структуру:

пока <условие> **повторять**
нц
<тело цикла>
кц

Команда цикла реализует обратную связь между объектом управления и управляющей системой. Проверка условия дает информацию управляющей системе о состоянии объекта управления.

В цикле с предусловием, если проверяемое условие выполняется (истинно), то выполняются команды, составляющие тело цикла. Если условие ложно, то происходит выход из цикла.

При программировании цикла необходимо следить за тем, чтобы не допускалось зацикливание.

Блок-схема — это графический способ описания алгоритма. Блоки обозначают действия исполнителя, а соединяющие их стрелки указывают на последовательность выполнения действий.

Вопросы и задания

1. Что такое цикл? Как записывается команда цикла?
2. Что такое условие цикла? Что такое тело цикла?
3. В каком случае происходит зацикливание алгоритма?
4. Что такое блок-схема?
5. Из каких блоков составляются блок-схемы (как они изображаются и что обозначают)?
6. Что обозначают стрелки на блок-схемах?
7. Составьте программу, переводящую ГРИС в угол поля из любого исходного состояния.
8. Составьте программу рисования прямоугольной рамки вдоль края поля при любом начальном состоянии исполнителя.

ЕК ЦОР: часть 2, глава 5, § 30. ЦОР № 5, 10–13, 16–18.

§ 7

Ветвление и последовательная детализация алгоритма

Основные темы параграфа:

- *команда ветвления;*
- *неполная форма команды ветвления;*
- *пример задачи с двухшаговой детализацией.*

Команда ветвления

Познакомимся еще с одной командой ГРИС. Она называется **командой ветвления**. Формат команды ветвления такой:

если <условие>
  то <серия 1>
  иначе <серия 2>
кв

Служебное слово **кв** обозначает конец ветвления.

По-прежнему ГРИС может проверять только два условия: *«впереди край?»* или *«впереди не край?»*. <серия> — это одна или несколько следующих друг за другом команд. Если <условие> справедливо,

то выполняется <серия 1>, в противном случае — <серия 2>. Такое ветвление называется ***полным***. Пример показан на рис. 1.11.

Рис. 1.11. Блок-схема полного ветвления

Неполная форма команды ветвления

В некоторых случаях используется ***неполная форма*** команды ветвления (рис. 1.12). Например:

если впереди край
    то поворот
кв

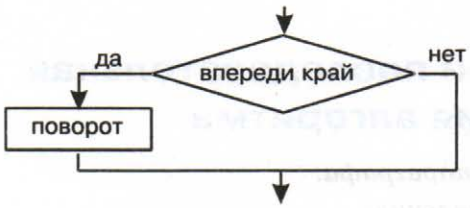


Рис. 1.12. Блок-схема неполного ветвления

Неполная форма команды ветвления:

если <условие>
    то <серия>
кв

Здесь <серия> выполняется, если <условие> справедливо, в противном случае происходит переход к следующей после ветвления команде алгоритма.

Составим последнюю, сравнительно сложную программу для ГРИС. На этом примере вы увидите, что применение метода последовательной детализации облегчает решение некоторых «головоломных» задач.

Пример задачи с двухшаговой детализацией

Задача 6. Построить орнамент, состоящий из квадратов, расположенных по краям поля. Исходное положение ГРИС — в верхнем левом углу, направление — на юг (рис. 1.13).

Процедуру, рисующую цепочку квадратов в одном направлении, назовем РЯД. Процедуру, рисующую один квадрат, назовем КВАДРАТ. Сначала напишем основную программу:

```
программа Орнамент
нач
    сделай РЯД
    поворот
    сделай РЯД
    поворот
    сделай РЯД
    поворот
    сделай РЯД
кон
```

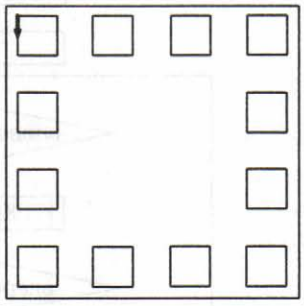


Рис. 1.13. Результат выполнения программы «Орнамент». Стрелкой указано начальное состояние

Теперь напишем процедуры РЯД и КВАДРАТ:

```
процедура РЯД
нач
   прыжок
   прыжок
   пока впереди не край повторять
   нц
     сделай КВАДРАТ
     если впереди не край
          то прыжок
     кв
   кц
кон
```

```
процедура КВАДРАТ
нач
   шаг
   поворот
   шаг
   поворот
   шаг
   поворот
   шаг
   поворот
   прыжок
кон
```

В процедуре РЯД в теле цикла содержится неполное ветвление. Структуру такого алгоритма можно назвать так: ***цикл с вложенным ветвлением.***

На рисунке 1.14 приведена блок-схема процедуры РЯД.

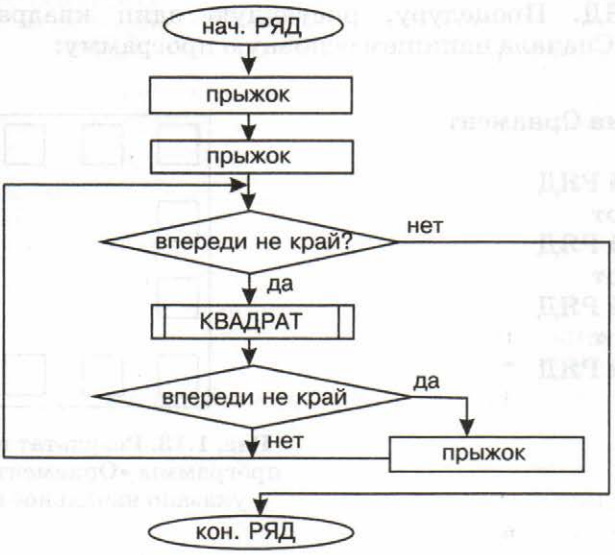


Рис. 1.14. Блок-схема процедуры РЯД

Составление этой программы потребовало двух шагов детализации алгоритма, которые выполнялись в такой последовательности:

Теперь вам известны все команды управления графическим исполнителем. Их можно разделить на три группы: простые команды; команда обращения к процедуре; структурные команды. К третьей группе относятся команды цикла и ветвления.

СКИ графического исполнителя

Простые команды

шаг
поворот
прыжок

Обращение к процедуре

сделай <имя процедуры>

Структурные команды

пока <условие> **повторять**
нц
<тело цикла>
кц

если <условие>
то <серия 1>
иначе <серия 2>
кв

Коротко о главном

Команда ветвления имеет следующий формат:

если <условие>
то <серия 1>
иначе <серия 2>
кв

Если <условие> истинно, то выполняются команды, составляющие <серию 1>, если ложно — <серию 2>.

Неполная форма команды ветвления:

если <условие>
то <серия >
кв

Если условие истинно, то выполняется <серия>, если ложно, то сразу происходит переход к следующей за ветвлением команде алгоритма.

Сложные алгоритмы удобно строить путем пошаговой детализации.

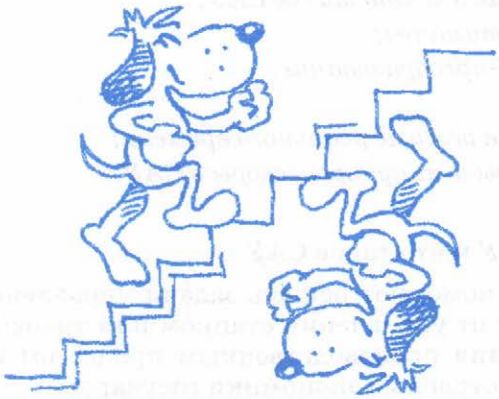
Вопросы и задания

1. Что такое пошаговая детализация?
2. Из каких команд могут состоять вспомогательные алгоритмы последнего уровня детализации?
3. Какой формат имеет команда ветвления? Какие действия исполнителя она определяет?
4. Чем отличается полное ветвление от неполного?
5. Путем пошаговой детализации составьте программы управления графическим исполнителем для решения следующих задач:
   - расчертить все поле горизонтальными пунктирными линиями;
   - нарисовать квадраты во всех углах поля;
   - расчертить все поле в клетку со стороной, равной шагу.

www ЕК ЦОР: часть 2, глава 5, § 31. ЦОР № 5, 9, 12, 15, 16.

Чему вы должны научиться, изучив главу I

- Освоить программное управление одним из учебных графических исполнителей.
- Составлять линейные программы.
- Составлять циклические программы.
- Составлять программы, содержащие ветвления.
- Описывать и использовать вспомогательные алгоритмы (подпрограммы).
- Применять метод последовательной детализации.



Дополнение к главе I

1.1. Автоматизированные и автоматические системы управления

Основные темы параграфа:

- *что такое АСУ и что такое САУ;*
- *простые автоматы;*
- *ЦАП–АЦП-преобразование;*
- *схема САУ;*
- *управление в режиме реального времени;*
- *контроллеры и микропроцессоры в САУ.*

Что такое АСУ и что такое САУ

Компьютеры помогают решать задачи управления в самых разных масштабах: от управления станком или транспортным средством до управления производственным процессом на предприятии или даже целой отраслью экономики государства.

Конечно, поручать компьютеру полностью, без участия человека, руководить предприятием или отраслью экономики сложно, да и небезопасно. Для управления в таком масштабе создаются компьютерные системы, которые называются **автоматизированными системами управления (АСУ)**. Такие системы работают вместе с человеком.

АСУ помогает руководителю получить необходимую информацию для принятия управляющего решения, а также может предложить наиболее оптимальные варианты таких решений. Однако окончательное решение принимает человек.

В АСУ используются самые современные средства информационных технологий: базы данных и экспертные системы, методы математического моделирования, машинная графика и пр.

С распространением персональных компьютеров технической основой АСУ стали компьютерные сети. В рамках одного предприятия это локальные компьютерные сети. Автоматизированные системы управления, работающие в масштабах отрасли, в государственных масштабах, используют глобальные компьютерные сети.

Другим вариантом применения компьютеров в управлении являются **системы автоматического управления (САУ)**. Объектами управления в этом случае чаще всего выступают технические устройства (станок, ракета, химический реактор, ускоритель элементарных частиц).

В **САУ** все операции, связанные с процессами управления (сбор и обработка информации, формирование управляющих команд, воздействие на управляемый объект), происходят автоматически, без непосредственного участия человека.

Простые автоматы

Устройства автоматического управления стали создаваться задолго до появления первых ЭВМ. Как правило, они основаны на использовании каких-либо физических явлений. Например, автоматический регулятор уровня воды в баке основан на выталкивающем действии воды на поплавок регулятора; автоматические предохранители в электрических сетях основаны на тепловом действии электрического тока; система автоматического регулирования освещенности в помещении использует явление фотоэффекта. Существуют и более сложные примеры бескомпьютерного автоматического управления.

Преимущество компьютерных систем автоматического управления перед такими устройствами — в их большей «интеллектуальности», в возможности осуществлять более сложное управление, чем простые автоматы.

ЦАП–АЦП-преобразование

Рассмотрим ситуацию, в которой объектом управления является техническое устройство (лабораторная установка, бытовая техника, транспортное средство или промышленное оборудование), а управляющим объектом — система автоматического управления.

Компьютер работает с двоичной информацией, помещенной в его память. Управляющая команда, выработанная программой, в компьютере имеет форму двоичного кода. Чтобы она превратилась в физическое воздействие на управляемый объект, необходимо преобразование этого кода в электрический сигнал, который приведет в движение «рычаги» управления объектом. Такое преобразование из двоичного кода в электрический сигнал называют цифро-аналоговым преобразованием. Выполняющий такое преобразование прибор называется **ЦАП (цифро-аналоговый преобразователь)**.

Приборы, которые дают информацию о состоянии объекта управления, называются **датчиками**. Они могут показывать, например, температуру, давление, деформации, напряженности полей и пр. Эти данные необходимо передать компьютеру по линиям обратной связи. Если показания датчиков имеют аналоговую форму (электрический ток или потенциал), то они должны быть преобразованы в двоичную цифровую форму. Такое преобразование называется аналого-цифровым, а прибор, его выполняющий, — **АЦП (аналого-цифровой преобразователь)***.

Схема САУ

Все сказанное отражается в схеме, приведенной на рис. 1.15. Такая система работает автоматически, без участия человека.

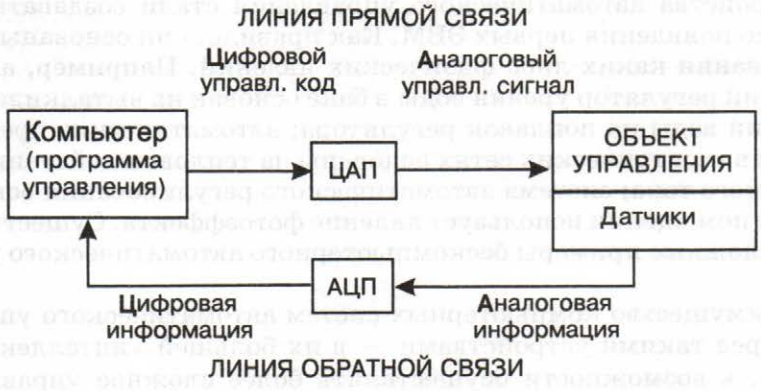


Рис. 1.15. Схема системы автоматического управления

* О ЦАП- и АЦП-преобразованиях речь уже шла в учебнике для 7 класса.

Управление в режиме реального времени

Системы автоматического управления работают в **режиме реального времени.** Легко понять, что всякая управляющая команда должна быть отдана вовремя. Любой процесс происходит с какой-то скоростью, в каких-то временны́х рамках.

Режим, при котором управляющая система работает синхронно с объектом управления, называется **режимом реального времени.**

При составлении программ управления в реальном времени программистам приходится решать вопрос не только о том, в каком порядке отдавать команды, но и в какие моменты времени это делать. Значит, система управления должна взаимодействовать с прибором, отмеряющим время: часами, таймером.

Напомним, что в составе персонального компьютера есть устройство, называемое **генератором тактовой частоты.** Работа всех узлов компьютера синхронизируется по тактовой частоте. Вот на эти «часы» и ориентируется программа управления в режиме реального времени.

Контроллеры и микропроцессоры в САУ

Не следует думать, что в системах автоматического управления всегда используется универсальный компьютер с полным комплектом всех устройств (клавиатура, монитор и пр.). Конечно, бывает и такое, но очень часто для этих целей применяются специализированные устройства — **контроллеры.** В их состав обязательно входят процессор, память и необходимые средства связи с объектом управления. Если управляющая система все время должна работать по одной и той же программе, то эта программа хранится в постоянной памяти (ПЗУ).

В простейших случаях для автоматического управления используются микропроцессоры, встроенные в управляемое устройство. Например, очень часто микропроцессоры применяются в транспортных средствах: автомобилях, самолетах, поездах. Каждый микропроцессор выполняет свою отдельную функцию, управляет работой определенного узла. Например, в автомобилях используется микропроцессор, управляющий работой карбюратора — устройства, регулирующего подачу топлива в двигатель. Такое автоматическое управление снижает расход горючего, повышает КПД (коэффициент полезного действия) двигателя.

Современные самолеты «нашпигованы» многочисленной электроникой: от микропроцессоров, управляющих отдельными приборами, до бортовых компьютеров, прокладывающих маршрут полета, т. е. выполняющих функции штурмана.

Коротко о главном

Автоматизированные системы управления (АСУ) помогают человеку в сборе информации и принятии управляющих решений.

В системах автоматического управления (САУ) все операции, связанные с процессами управления, происходят автоматически, без непосредственного участия человека, по заранее составленной программе.

В САУ на линии прямой связи для преобразования двоичной информации в аналоговый сигнал используется прибор ЦАП (цифроаналоговый преобразователь); на линии обратной связи для преобразования аналогового сигнала в двоичный код используется прибор АЦП (аналого-цифровой преобразователь).

Управление в САУ происходит в режиме реального времени.

Вопросы и задания

1. В чем различие между автоматизированными системами управления (АСУ) и системами автоматического управления (САУ)?
2. Какие аппаратные компоненты входят в систему управления техническим устройством с помощью компьютера?
3. Для чего нужны устройства ЦАП и АЦП?
4. Что такое управление в режиме реального времени?
5. Приведите примеры использования встроенных в оборудование микропроцессоров.

1.2. Использование рекурсивных процедур

Вернемся к вопросу об использовании процедур при составлении программ управления исполнителями алгоритмов (см. § 5 нашего учебника). Но это будет особый вид процедур, которые называются рекурсивными процедурами.

Рекурсивной называется **процедура**, в которой имеется обращение к самой себе.

Не все учебные исполнители алгоритмов допускают использование рекурсивных процедур (рекурсии). Такая возможность имеется в учебной программе «Стрелочка», реализующей один из вариантов графического исполнителя алгоритмов (ГРИС)*. При программировании некоторых задач рекурсия может служить альтернативой циклу.

Приведем пример использования в программе для ГРИС «Стрелочка» рекурсивной процедуры вместо цикла.

Пусть начальное положения «Стрелочки» — произвольная точка в первой строке рабочего поля, направление «вправо» (рис. 1.16). Требуется построить линию, идущую из этой точки до правой границы области.

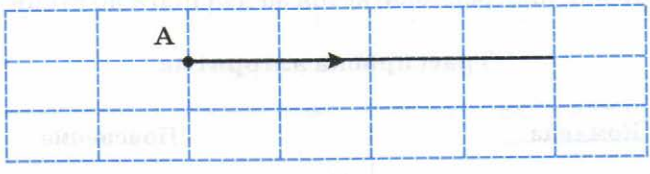


Рис. 1.16

Сначала приведем на языке «Стрелочки» программу, в которой используется цикл.

```
алгоритм ПУТЬ_1_0
  Дано: исполнитель в точке А
  Надо: воспроизвести образец
нач
  пока впереди НЕ стена
  нц
    шаг
  кц
кон
```

* Единая коллекция цифровых образовательных ресурсов (ЕК ЦОР): http://school-collection.edu.ru/

А теперь воспользуемся рекурсией, для чего опишем процедуру ЛИНИЯ_1.

```
алгоритм ПУТЬ_1_1
  Дано: Исполнитель в точке А
  Надо: Воспроизвести образец
нач
  делай ЛИНИЯ_1
кон

процедура ЛИНИЯ_1
    шаг
    делай ЛИНИЯ_1
конец процедуры
```

В теле этой процедуры присутствует команда обращения к самой себе: **делай ЛИНИЯ_1**.

Пошаговое исполнение (трассировка) такой программы иллюстрируется следующей трассировочной таблицей в случае, если «Стрелочка» вначале располагается за два шага до стенки.

Трассировка алгоритма

Команда	Пояснение
начало **делай ЛИНИЯ_1**	Вызов процедуры в основной программе
шаг **делай ЛИНИЯ_1**	1-й рекурсивный вызов процедуры
шаг **делай ЛИНИЯ_1**	2-й рекурсивный вызов процедуры
	Далее возникнет ситуация НЕ МОГУ, так как исполнитель дошел до стены. Выполнение программы завершится аварийно

В этом случае завершение исполнения алгоритма произойдет аварийным образом (ситуация НЕ МОГУ).

Дело в том, что в данном алгоритме мы имеем дело с бесконечно повторяющимся рекурсивным обращением к процедуре, что недопустимо. Количество обращений процедуры к самой себе при ее исполнении называется *глубиной рекурсии*. Глубина рекурсии должна быть конечной. Для исполнителя «Стрелочка» естественным огра-

ничением количества обращений процедуры самой к себе может служить только достижение стены. Следовательно, рекурсивный вызов в процедуре надо поместить в команду ветвления, проверяющую условие достижения стены. Вот новый вариант алгоритма.

```
алгоритм ПУТЬ_1_2
  Дано: Исполнитель в точке А
  Надо: Воспроизвести образец
нач
  делай ЛИНИЯ_2
кон
процедура ЛИНИЯ_2
  если впереди НЕ стена
  то
    шаг
    делай ЛИНИЯ_2
  все
конец процедуры
```

Трассировка этого алгоритма представлена в следующей таблице:

Трассировка алгоритма

Команда	Пояснение
начало	
делай ЛИНИЯ_2	Первый вызов процедуры
если впереди НЕ стена	Да
то	
шаг	
делай ЛИНИЯ_2	Рекурсивный вызов
если впереди НЕ стена	Да
то	
шаг	
делай ЛИНИЯ_2	Рекурсивный вызов
если впереди НЕ стена	Нет
все	
все	
все	Программа выполнилась, рисунок получен
конец	

Поскольку в описании алгоритма присутствует «точка остановки» (условие окончания исполнения алгоритма), исполнитель «Стрелочка», дойдя до стены, остановится.

Возникает вопрос: а стоит ли пользоваться рекурсией, если задача решается с помощью цикла? Немного усложним исходную задачу: *требуется построить линию, идущую из данной точки, до правой границы области, после чего исполнитель должен вернуться в исходную точку А.*

При использовании рекурсии решение этой задачи запишется достаточно просто, в то время как циклический алгоритм для такой задачи построить невозможно.

```
алгоритм ПУТЬ_2
  Дано: Исполнитель в точке А
  Надо: Воспроизвести образец
нач
  делай ЛИНИЯ_3
кон

процедура ЛИНИЯ_3
  если впереди НЕ стена
  то
    шаг
    делай ЛИНИЯ_3
    прыжок
  иначе
    поворот
    поворот
  все
конец процедуры
```

Трассировка алгоритма ПУТЬ_2 для исходного положения исполнителя за два шага до стенки показана в следующей таблице.

Трассировка алгоритма

Команда	Пояснение
начало	
делай ЛИНИЯ_3	Первый вызов процедуры
если впереди НЕ стена	Да
то	
шаг	
делай ЛИНИЯ_3	Рекурсивный вызов
если впереди НЕ стена	Да
то	
шаг	
делай ЛИНИЯ_3	Рекурсивный вызов
если впереди НЕ стена	Нет
иначе	
поворот	
поворот	Разворот на 180 градусов
все	
прыжок	Возврат в 2 прыжка в исходную точку
все	
прыжок	
все	
конец	Программа выполнилась

Если при выполнении процедуры происходит рекурсивное обращение к ней самой, то это значит, что не произошел выход из этой процедуры при предыдущем обращении. Этот выход откладывается на будущее. Затем, после окончания рекурсивных обращений, все отложенные выходы отрабатываются в порядке, обратном порядку рекурсивных обращений. В приведенной программе отработка отложенных выходов начинается с выполнения команды **поворот**.

Таким образом, существуют задачи, для которых использование рекурсии позволяет получить достаточно компактную запись алгоритма, в то время как его «циклическая версия» может оказаться либо очень сложной, либо вообще не реализуемой.

Коротко о главном

Рекурсивной называется процедура, в которой имеется обращение к самой себе.

Использование рекурсии может быть эквивалентом использованию цикла.

Существуют задачи, для которых рекурсивное решение является наиболее оптимальным или единственно возможным.

Вопросы и задания

1. Что такое рекурсивная процедура?
2. В каком порядке происходит выход из последовательности рекурсивных обращений?
3. Попробуйте разобраться, какую задачу решает следующая программа, содержащая рекурсивную процедуру. Реализуйте ее в среде исполнителя «Стрелочка».

```
алгоритм ФИГУРА-1
  Дано: Исполнитель в точке А
  Надо: Воспроизвести образец
нач
  делай ЛИНИЯ
  поворот
  поворот
  поворот
  шаг
  шаг
кон

процедура ЛИНИЯ
 если впереди НЕ стена
 то
   шаг
   делай ЛИНИЯ
   шаг
 иначе
   поворот
   поворот
   поворот
```

```
    шаг
    шаг
    поворот
    поворот
    поворот
  все
конец процедуры
```

4. Составьте программу с рекурсивной процедурой, по которой исполнитель, находящийся в произвольной точке поля, дойдет до стенки, затем повернется на 90 градусов по часовой стрелке и дойдет до конца этой стенки вдоль нее. В результате будет нарисован угол.

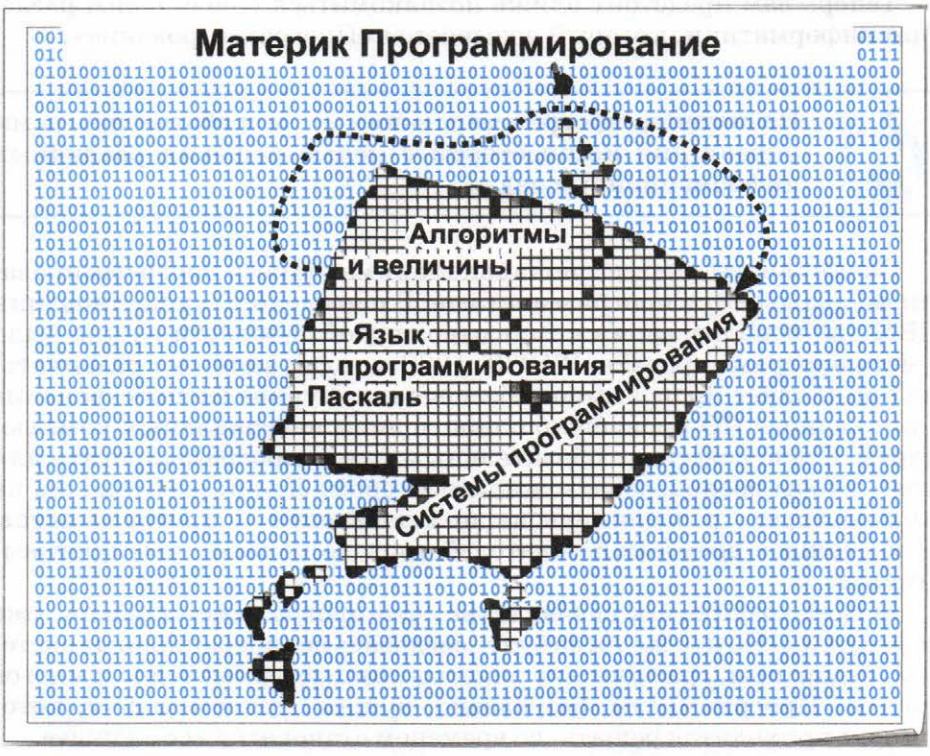
УПРАВЛЕНИЕ И

Кибернетическая модель управления

Управляющий объект	Объект, осуществляющий управление
Объект управления	Объект, выполняющий команды управления, — *исполнитель алгоритма управления*
Прямая связь	Канал передачи команд управления
Обратная связь	Канал передачи информации о состоянии объекта управления
Алгоритм управления	Последовательность команд управления
Автоматические системы с программным управлением	Технические системы, в которых функцию управляющего объекта выполняет компьютер

Система основных понятий главы I

Глава II
Введение в программирование



Здесь вы узнаете:

- что такое программирование
- как строятся вычислительные алгоритмы
- как составляются программы на языке Паскаль

§ 8

Что такое программирование

Основные темы параграфа:

- *кто такие программисты;*
- *что такое язык программирования;*
- *что такое система программирования.*

Кто такие программисты

Теперь вам предстоит ближе познакомиться еще с одним разделом информатики, который называется «Программирование».

Назначение **программирования** — разработка программ управления компьютером с целью решения различных информационных задач.

Специалисты, профессионально занимающиеся программированием, называются **программистами**. В первые годы существования ЭВМ для использования компьютера в любой области нужно было уметь программировать. В 1970–1980-х годах начинает развиваться прикладное программное обеспечение. Бурное распространение прикладного ПО произошло с появлением персональных компьютеров. Стало совсем не обязательным уметь программировать для того, чтобы воспользоваться компьютером. Люди, работающие на компьютерах, разделились на **пользователей** и **программистов**. В настоящее время пользователей гораздо больше, чем программистов.

Может возникнуть впечатление, что программисты теперь уже и не нужны! Но кто же тогда будет создавать все операционные системы, редакторы, графические пакеты, компьютерные игры и многое другое? Программисты, безусловно, нужны, причем задачи, которые им приходится решать, со временем становятся все сложнее.

Программирование принято разделять на системное и прикладное. **Системные программисты** занимаются разработкой системного программного обеспечения: операционных систем, утилит и пр., а также систем программирования. **Прикладные программисты** создают прикладные программы: редакторы, табличные процессоры, игры, обучающие программы и др. Спрос на высококвалифицированных программистов, как системных, так и прикладных, очень большой.

В данной главе вы познакомитесь с простейшими правилами и приемами программирования, заглянете в эту актуальную и престижную профессиональную область.

Что такое язык программирования

Для составления программ существуют разнообразные **языки программирования**.

Язык программирования — это фиксированная система обозначений для описания алгоритмов и структур данных.

За годы существования ЭВМ было создано много языков программирования. Наиболее известные среди них: Фортран, Паскаль, Бейсик, С (Си) и др. Распространенными языками программирования сегодня являются C++, Java, Pascal, Basic, Python.

Что такое система программирования

Для создания и исполнения на компьютере программы, написанной на языке программирования, используются **системы программирования**.

Система программирования — это программное обеспечение компьютера, предназначенное для разработки, отладки и исполнения программ, записанных на определенном языке программирования.

Существуют системы программирования на Паскале, Бейсике и других языках.

В данной главе речь будет идти о средствах и способах универсального программирования — не ориентированного на какую-то узкую прикладную область. Примером узкоспециализированного программирования является Web-программирование, ориентированное на создание Web-сайтов. Для этих целей, например, используется язык JavaScript. Языки Паскаль, Бейсик, Си относятся к числу **универсальных языков программирования**.

Разработка любой программы начинается с построения алгоритма решения задачи. Ниже мы обсудим особенности алгоритмов решения задач обработки информации на компьютере.

Коротко о главном

Программирование — область информатики, посвященная разработке программ управления компьютером с целью решения различных информационных задач.

Программирование бывает системным и прикладным.

Паскаль, Бейсик, Си, Фортран — это универсальные языки программирования.

Система программирования — это программное обеспечение компьютера, предназначенное для разработки, отладки и исполнения программ, записанных на определенном языке программирования.

Вопросы и задания

1. Что такое программирование?
2. Какие задачи решают системные и прикладные программисты?
3. Назовите наиболее распространенные языки программирования.
4. В чем состоит назначение систем программирования?

www ЕК ЦОР: часть 2, глава 6, § 32. ЦОР № 2, 5.

§ 9

Алгоритмы работы с величинами

Основные темы параграфа:

- *компьютер как исполнитель алгоритмов;*
- *величины: константы и переменные;*
- *система команд;*
- *команда присваивания;*
- *команда ввода;*
- *команда вывода.*

Компьютер как исполнитель алгоритмов

Вам уже известно, что всякий алгоритм составляется для конкретного исполнителя. *Теперь в качестве исполнителя мы будем рассматривать компьютер, оснащенный системой программирования на определенном языке.*

Компьютер-исполнитель работает с определенными ***данными*** по определенной ***программе***. Данные — это множество величин.

Величины: константы и переменные

Компьютер работает с информацией, хранящейся в его памяти. Отдельный информационный объект (число, символ, строка, таблица и пр.) называется величиной.

Всякая обрабатываемая программой величина занимает свое место (поле) в памяти компьютера. Значение величины — это информация, хранимая в этом поле памяти.

Существуют *три основных типа величин*, с которыми работает компьютер: **числовой**, **символьный** и **логический**. Изучая базы данных и электронные таблицы, вы уже встречались с этими типами. В данной главе мы будем строить алгоритмы, работающие с числовыми величинами.

Числовые величины в программировании, так же как и математические величины, делятся на переменные и константы (постоянные). Например, в формуле $(a^2 - 2ab + b^2)$ a, b — переменные, 2 — константа.

Константы записываются в алгоритмах своими десятичными значениями, например: 23, 3.5, 34. Значение константы хранится в выделенной под нее ячейке памяти и остается неизменным в течение работы программы.

Переменные в программировании, как и в математике, обозначаются символическими именами. Эти имена называют **идентификаторами** (от глагола «идентифицировать», что значит «обозначать», «символизировать»). Идентификатор может быть одной буквой, множеством букв, сочетанием букв и цифр и т. д. Примеры идентификаторов: *A*, *X*, *B*3, *prim*, *r*25 и т. п.

Система команд

Вам известно, что всякий алгоритм строится исходя из системы команд исполнителя, для которого он предназначен. Любой алгоритм работы с величинами может быть составлен из следующих команд:

- присваивание;
- ввод;
- вывод;
- обращение к вспомогательному алгоритму;
- цикл;
- ветвление.

Эти команды существуют во всех языках, поддерживающих структурное программирование: в Паскале, Си и др.

Команда присваивания

Команда присваивания — одна из основных команд в алгоритмах работы с величинами. Записывать ее мы будем так:

<переменная>:=<выражение>

Значок «:=» читается «присвоить». Например:

$$Z:=X+Y$$

Компьютер сначала вычисляет выражение, затем результат присваивает переменной, стоящей слева от знака «:=».

Если до выполнения этой команды содержимое ячеек, соответствующих переменным X, Y, Z, было таким:

X	2	Y	5	Z	—

то после выполнения команды оно станет следующим:

X	2	Y	5	Z	7

Прочерк в ячейке Z обозначает, что начальное число в ней может быть любым. Оно не имеет значения для результата данной команды.

Если слева от знака присваивания стоит числовая переменная, а справа — выражение, определяющее порядок вычисления числовой величины, то такую команду называют **арифметической командой присваивания**, а выражение — **арифметическим выражением**.

В частном случае арифметическое выражение в правой части оператора присваивания может быть представлено одной переменной или одной константой. Например:

$$X:=5$$
$$Y:=X$$

Команда ввода

Значения переменных, являющихся исходными данными решаемой задачи, как правило, задаются **вводом**.

Команда ввода в описаниях алгоритмов выглядит так:

ввод <список переменных>.

Например:

ввод *A*, *B*, *C*

Пользователю удобно, если ввод данных организован в режиме диалога, когда по команде ввода компьютер прерывает выполнение программы и ждет действий пользователя. Пользователь должен набрать на клавиатуре вводимые значения переменных и нажать клавишу <ВВОД>. Введенные значения присвоятся соответствующим переменным из списка ввода, и выполнение программы продолжится.

Вот схема выполнения приведенной выше команды.

1. Память до выполнения команды:

A	—	*B*	—	*C*	—

2. Процессор компьютера получил команду *ввод A, B, C*, прервал свою работу и ждет действий пользователя.
3. Пользователь набирает на клавиатуре:

1 3 5

и нажимает клавишу <ВВОД> (<Enter>).

4. Память после выполнения команды:

A	1	*B*	3	*C*	5

5. Процессор переходит к выполнению следующей команды программы.

При выполнении пункта 3 вводимые числа должны быть отделены друг от друга какими-нибудь разделителями. Обычно это пробелы.

Из сказанного выше можно сделать вывод:

Переменные величины получают конкретные значения в результате выполнения команды присваивания или команды ввода.

Если переменной величине не присвоено никакого значения (или не введено), то она является неопределенной. Иначе говоря, ничего нельзя сказать о том, какое значение имеет эта переменная.

Команда вывода

Результаты решения задачи сообщаются компьютером пользователю путем выполнения **команды вывода.**

Команда вывода в алгоритмах записывается так:

вывод <список вывода>

Например:

вывод $X1$, $X2$

По этой команде значения переменных $X1$ и $X2$ будут вынесены на устройство вывода (чаще всего это экран).

О других командах, применяемых в алгоритмах работы с величинами, вы узнаете позже.

Коротко о главном

Любой алгоритм работы с величинами может быть составлен из следующих команд: присваивание; ввод; вывод; обращение к вспомогательному алгоритму; цикл; ветвление.

Программа для компьютера — это алгоритм, записанный на языке программирования.

Язык программирования — это фиксированная система обозначений для описания алгоритмов и структур данных.

Всякая обрабатываемая программой величина занимает определенное поле в памяти компьютера. Значение величины — это информация, хранимая в этом поле.

Переменная величина получает значение в результате выполнения команды присваивания или команды ввода.

Формат команды присваивания:

<переменная>:=<выражение>

Сначала вычисляется выражение, затем полученное значение присваивается переменной.

Ввод — это занесение данных с внешних устройств в оперативную память компьютера. Исходные данные для решения задачи обычно задаются вводом.

Результаты решения задачи выносятся на устройства вывода (монитор, принтер) по команде вывода.

Вопросы и задания

1. Что такое величина? Чем отличаются переменные и постоянные величины?
2. Чем определяется значение величины?
3. Какие существуют основные типы величин в программировании?

4. Как записывается команда присваивания?
5. Что такое ввод? Как записывается команда ввода?
6. Что такое вывод? Как записывается команда вывода?
7. В схематическом виде (как это сделано в параграфе) отразите изменения значений в ячейках, соответствующих переменным A и B, в ходе последовательного выполнения команд присваивания:

1)	2)	3)
$A:=1$	$A:=1$	$A:=1$
$B:=2$	$B:=2$	$B:=2$
$A:=A+B$	$C:=A$	$A:=A+B$
$B:=2*A$	$A:=B$	$B:=A-B$
	$B:=C$	$A:=A-B$

8. Вместо многоточия впишите в алгоритм несколько команд присваивания, в результате чего должен получиться алгоритм возведения в четвертую степень введенного числа (дополнительные переменные не использовать):

ввод A ... вывод A

ЕК ЦОР: часть 2, глава 6, § 33. ЦОР № 2, 7.

§ 10
Линейные вычислительные алгоритмы

Основные темы параграфа:

- *присваивание; свойства присваивания;*
- *обмен значениями двух переменных;*
- *описание линейного вычислительного алгоритма.*

Присваивание. Свойства присваивания

Поскольку присваивание является важнейшей операцией в алгоритмах, работающих с величинами, поговорим о ней более подробно.

Переменная величина получает значение в результате присваивания.

Присваивание производится компьютером при выполнении одной из двух команд из представленной выше системы команд: команды присваивания или команды ввода.

Рассмотрим последовательность выполнения четырех команд присваивания, в которых участвуют две переменные: a и b. В приведенной ниже таблице против каждой команды указываются значения переменных, которые устанавливаются после ее выполнения. Такая таблица называется **трассировочной таблицей**, а процесс ее заполнения называется **трассировкой** алгоритма.

Команда	a	b
$a:=1$	1	—
$b:=2*a$	1	2
$a:=b$	2	2
$b:=a+b$	2	4

Прочерк в таблице означает неопределенное значение переменной. Конечные значения, которые получают переменные a и b, соответственно равны 2 и 4.

Этот пример иллюстрирует три основных свойства присваивания. Вот эти свойства:

1) пока переменной не присвоено значение, она остается неопределенной;
2) значение, присвоенное переменной, сохраняется вплоть до выполнения следующего присваивания этой переменной нового значения;
3) новое значение, присвоенное переменной, заменяет ее предыдущее значение.

Обмен значениями двух переменных

Рассмотрим еще один очень полезный алгоритм, с которым при программировании часто приходится встречаться. Даны две переменные величины: X и Y. Требуется произвести между ними обмен значения-

ми. Например, если первоначально было: $X = 1$; $Y = 2$, то после обмена должно стать: $X = 2$, $Y = 1$.

Хорошим аналогом для решения такой задачи является следующая: даны два стакана, в первом — молоко, во втором — вода; требуется произвести обмен их содержимым. Всякому ясно, что в этом случае нужен дополнительный, третий, пустой стакан. Последовательность действий будет следующей:

1) перелить из 1-го стакана в 3-й;
2) перелить из 2-го стакана в 1-й;
3) перелить из 3-го стакана во 2-й.

Цель достигнута!

По аналогии для обмена значениями двух переменных нужна третья дополнительная переменная. Назовем ее Z. Тогда задача решается последовательным выполнением трех операторов присваивания (пусть начальные значения 1 и 2 для переменных X и Y задаются вводом):

Команда	X	Y	Z
ввод X, Y	1	2	–
$Z:=X$	1	2	**1**
$X:=Y$	**2**	2	1
$Y:=Z$	2	**1**	1
вывод X, Y	**2**	**1**	1

Действительно, в итоге переменные X и Y поменялись значениями. На экран будут выведены значения X и Y: 2, 1. В трассировочной таблице выводимые значения выделены жирным шрифтом.

Аналогия со стаканами не совсем точна в том смысле, что при переливании из одного стакана в другой первый становится пустым. В результате же присваивания ($X:=Y$) переменная, стоящая справа (Y), сохраняет свое значение.

Описание линейного вычислительного алгоритма

Алгоритмы, результатами выполнения которых являются числовые величины, будем называть вычислительными алгоритмами. Рассмотрим пример решения следующей математической задачи:

даны две простые дроби; получить дробь, являющуюся результатом деления одной на другую.

В школьном учебнике математики правила деления обыкновенных дробей описаны так:

1. Числитель первой дроби умножить на знаменатель второй.
2. Знаменатель первой дроби умножить на числитель второй.
3. Записать дробь, числителем которой является результат выполнения пункта 1, а знаменателем — результат выполнения пункта 2.

В алгебраической форме это выглядит следующим образом:

$$\frac{a}{b} : \frac{c}{d} = \frac{a \cdot d}{b \cdot c} = \frac{m}{n}.$$

Теперь построим алгоритм деления дробей для компьютера. В этом алгоритме сохраним те же обозначения для переменных, которые использованы в записанной выше формуле. Исходными данными являются целочисленные переменные a, b, c, d. Результатом — также целые величины m и n.

Ниже алгоритм представлен в двух формах: в виде блок-схемы и на Алгоритмическом языке (АЯ).

Раньше прямоугольник в схемах алгоритмов управления мы называли блоком простой команды. Для вычислительных алгоритмов такой простой командой является команда присваивания. Прямо- угольник *будем называть блоком присваивания, или вычислительным блоком.* В форме параллелограмма рисуется блок ввода/вывода. Полученный алгоритм имеет линейную структуру (рис. 2.1).

```
алг  Деление дробей
цел  a, b, c, d, m, n
нач
   ввод a, b, c, d
   m:=a×d
   n:=b×c
   вывод m, n
кон
```

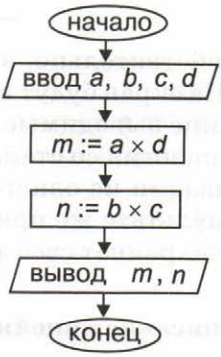


Рис. 2.1. Блок-схема алгоритма деления дробей

В алгоритме на АЯ строка, стоящая после заголовка алгоритма, называется ***описанием переменных***. Служебное слово **цел** означает целый тип. Величины этого типа могут иметь только целочисленные значения.

Описание переменных имеет вид:

<тип переменных> <список переменных>

Список переменных включает все переменные величины данного типа, обрабатываемые в алгоритме.

В блок-схемах типы переменных не указываются, но подразумеваются. Запись алгоритма на АЯ ближе по форме к языкам программирования, чем блок-схемы.

Коротко о главном

Основные свойства присваивания:

- значение переменной не определено, если ей не присвоено никакого значения;
- новое значение, присваиваемое переменной, заменяет ее старое значение;
- присвоенное переменной значение сохраняется в ней вплоть до нового присваивания.

Обмен значениями двух переменных можно производить через третью дополнительную переменную.

Трассировочная таблица используется для «ручного» исполнения алгоритма с целью его проверки.

В алгоритмах на АЯ указываются типы всех переменных. Такое указание называется описанием переменных.

Числовые величины, принимающие только целочисленные значения, описываются с помощью служебного слова **цел** (целый).

Вопросы и задания

1. Из каких команд составляется линейный вычислительный алгоритм?
2. Что такое трассировка? Как она производится?
3. В каком случае значение переменной считается неопределенным?
4. Что происходит с предыдущим значением переменной после присваивания ей нового значения?
5. Как вы думаете, можно ли использовать в выражении оператора присваивания неопределенную переменную? К каким последствиям это может привести?
6. Напишите на АЯ алгоритм сложения двух простых дробей (без сокращения дроби).

7. Напишите на АЯ алгоритм вычисления y по формуле

$$y = (1 - x^2 + 5x^4)^2,$$

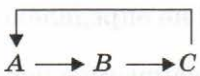
где x — заданное целое число. Учтите следующие ограничения: 1) в арифметических выражениях можно использовать только операции сложения, вычитания и умножения; 2) каждое выражение может содержать только одну арифметическую операцию. Выполните трассировку алгоритма при $x = 2$.

8. Пользуясь ограничениями предыдущей задачи, напишите наиболее короткие алгоритмы вычисления выражений:

$$y = x^8;\ y = x^{10};\ y = x^{15};\ y = x^{19}.$$

Постарайтесь использовать минимальное количество дополнительных переменных. Выполните трассировку алгоритмов.

9. Запишите алгоритм циклического обмена значениями трех переменных A, B, C. Схема циклического обмена:



Например, если до обмена было: $A = 1$, $B = 2$, $C = 3$, то после обмена должно стать: $A = 3$, $B = 1$, $C = 2$. Выполните трассировку.

www ЕК ЦОР: часть 2, глава 6, § 34. ЦОР № 9, 10.

§ 11

Знакомство с языком Паскаль

Основные темы параграфа:

- *возникновение и назначение Паскаля;*
- *структура программы на Паскале;*
- *операторы ввода, вывода, присваивания;*
- *правила записи арифметических выражений;*
- *пунктуация Паскаля.*

Возникновение и назначение Паскаля

После того как построен алгоритм решения задачи, составляется программа на определенном языке программирования.

Среди современных языков программирования одним из самых популярных является язык **Паскаль**. Этот язык разработан в 1971 году и назван в честь Блеза Паскаля — французского ученого, изобретателя механической вычислительной машины. Автор языка Паскаль — швейцарский профессор Никлаус Вирт.

Паскаль — это универсальный язык программирования, позволяющий решать самые разнообразные задачи обработки информации.

Команду алгоритма, записанную на языке программирования, принято называть **оператором**.

Программа на Паскале близка по своему виду к описанию алгоритма на АЯ. Сравните алгоритм решения уже знакомой вам задачи деления простых дробей с соответствующей программой на Паскале:

алг Деление дробей
цел a, b, c, d, m, n
нач
  ввод a, b, c, d
  $m := a \times d$
  $n := b \times c$
  вывод m, n
кон
	`Program Division;`
	`var a, b, c, d, m, n: integer;`
	`begin`
	`  readln(a, b, c, d);  {Ввод}`
	`  m:= a*d;  {Числитель}`
	`  n:= b*c;  {Знаменатель}`
	`  write(m, n)  {Вывод}`
	`end.`

Структура программы на Паскале

Даже не заглядывая в учебник по Паскалю, в этой программе можно все понять (особенно помогает знание английского языка).

Заголовок программы начинается со слова **Program** (программа), за которым следует произвольное имя, придуманное программистом:

Program <имя программы>;

Раздел описания переменных начинается со слова **Var** (variables — переменные), за которым идет список имен переменных через запятую. Тип указывается после двоеточия. В стандарте языка Паскаль существуют два типа числовых величин: **вещественный** и **целый**. Слово `integer` обозначает целый тип (является идентификатором целого типа). Вещественный тип обозначается словом `real`. Например, раздел описания переменных может быть таким:

```
var a, b: integer; c, d: real;
```

Идентификаторы переменных составляются из латинских букв и цифр; первым символом обязательно должна быть буква.

Раздел операторов — основная часть программы. Начало и конец раздела операторов программы отмечаются служебными словами **begin** (начало) и **end** (конец). В самом конце программы ставится точка:

```
begin
    <операторы>
end.
```

Операторы ввода, вывода, присваивания

Ввод исходных данных с клавиатуры происходит по оператору `read` (read — читать) или `readln` (read line — читать строку):

```
read(<список переменных>)
или readln(<список переменных>)
```

При выполнении команды ввода компьютер ожидает действий пользователя. Пользователь набирает на клавиатуре значения переменных в том порядке, в каком переменные указаны в списке, отделяя их друг от друга пробелами. Одновременно с набором данных на клавиатуре они появляются на экране. В конце нажимается клавиша <ВВОД> (<Enter>). Разница в выполнении операторов `readln` и `read` состоит в том, что после выполнения ввода по оператору `readln` экранный курсор перемещается в начало новой строки, а по оператору `read` этого не происходит.

Вывод результатов происходит по оператору `write` (write — писать) или `writeln` (write line — писать в строку):

```
write(<список вывода>)
или writeln(<список вывода>)
```

Результаты выводятся на экран компьютера в порядке их перечисления в списке. Элементами списка вывода могут быть константы, переменные, выражения.

Разница в выполнении операторов `writeln` и `write` состоит в том, что после выполнения вывода по оператору `writeln` экранный курсор перемещается в начало новой строки, а по оператору `write` этого не происходит.

Арифметический оператор присваивания на Паскале имеет следующий формат:

<числовая переменная>:=<арифметическое выражение>

Арифметическое выражение может содержать числовые константы и переменные, знаки арифметических операций, круглые скобки. Кроме того, в арифметических выражениях могут присутствовать функции.

Знаки основных арифметических операций записываются так:

\+ сложение,
\- вычитание,
* умножение,
/ деление.

Правила записи арифметических выражений

Запись арифметических выражений на Паскале похожа на обычную математическую запись. В отличие от математики, где часто пропускается знак умножения (например, пишут $2A$), в Паскале этот знак пишется обязательно: `2*A`. Например, математическое выражение

$$A^2 + B^2 - 12C$$

на Паскале записывается так:

```
A*A + B*B - 12*C
```

Это же выражение можно записать иначе:

```
SQR(A) + SQR(B) - 12*C
```

Здесь использована функция возведения в квадрат — SQR. Аргументы функций всегда пишутся в круглых скобках.

Последовательность выполнения операций определяется по их **приоритетам** (старшинству). К старшим операциям относятся умножение (*) и деление (/). Операции сложения и вычитания — младшие. В первую очередь выполняются старшие операции. Несколько операций одинакового старшинства, записанные подряд, выполняются в порядке их записи слева направо. Приведенное выше арифметическое выражение будет вычисляться в следующем порядке (порядок вычислений указан цифрами сверху):

```
  1   4   2   5   3
A * A + B * B - 12 * C
```

Круглые скобки в арифметических выражениях влияют на порядок выполнения операций. Как и в математике, в первую очередь выполняются операции в скобках. Если имеется несколько пар вложен-

ных скобок, то сначала выполняются операции в самых внутренних скобках. Например:

```
      6     1     3     2     4     5
A + ((C - D) / (2 + K) - 1)*B
```

Пунктуация Паскаля

Необходимо строгое соблюдение правописания (синтаксиса) программы. В частности, в Паскале однозначно определено назначение знаков пунктуации.

Точка с запятой (;) ставится в конце заголовка программы, в конце раздела описания переменных, является разделителем описания переменных в разделе переменных и разделителем операторов. Перед словом **end** точку с запятой можно не ставить.

Запятая (,) является разделителем элементов во всевозможных списках: списке переменных в разделе описания, списках вводимых и выводимых величин.

Текст программы заканчивается точкой.

Строгий синтаксис в языке программирования необходим потому, что *компьютер является формальным исполнителем программы*. Если, допустим, разделителем в списке переменных должна быть запятая, то любой другой знак будет восприниматься как ошибка. Если точка с запятой является разделителем операторов, то в качестве оператора компьютер воспринимает всю часть текста программы от одной точки с запятой до другой. Если программист забыл поставить «;» между какими-то двумя операторами, то компьютер будет принимать их за один с неизбежной ошибкой.

В программу на Паскале можно вставлять комментарии. Комментарий — это пояснение к программе, которое записывается в фигурных скобках. В комментариях можно использовать русские буквы. На исполнение программы комментарий никак не влияет.

Заметим, что в Паскале нет различия между строчными и прописными буквами. Например, для Паскаля тождественны следующие варианты записи: **begin**, **Begin**, **BEGIN**, **BeGiN**. Использование строчных или прописных букв — дело вкуса программиста.

Коротко о главном

Паскаль — универсальный язык программирования.

Программа на Паскале состоит из заголовка, описаний и операторов.

Заголовок программы:

Program <имя программы>;

Описание переменных:

```
var <список однотипных переменных>: <тип>; ...
```

Раздел операторов:

```
begin
    <операторы>
end.
```

Операторы ввода данных с клавиатуры:

```
read(<список ввода>),  readln(<список ввода>)
```

Операторы вывода на экран:

```
write(<список вывода>, writeln(<список вывода>
```

Арифметический оператор присваивания:

```
<переменная>:=<арифметическое выражение>
```

Арифметическое выражение может содержать любое количество арифметических операций и функций.

Последовательность выполнения операций определяется расстановкой скобок и старшинством операций (приоритетами). Старшие операции: *, /; младшие операции: +, – .

Точка с запятой ставится в конце заголовка программы, в конце раздела описания переменных, является разделителем переменных в разделе переменных и разделителем операторов. Текст программы заканчивается точкой.

Вопросы и задания

1. Когда появился язык Паскаль и кто его автор?
2. Как записывается заголовок программы на Паскале?
3. Как записывается раздел описания переменных?
4. С какими типами числовых величин работает Паскаль?
5. Как записываются операторы ввода и вывода в Паскале?
6. Что такое оператор присваивания?
7. Как записываются арифметические выражения?
8. По каким правилам определяется порядок выполнения операций в арифметическом выражении?
9. Какая задача решается по следующей программе?

```
Program Test;
var A, B, C: integer;
begin
  readln(A,B);
  C:=(A+B)*(B-A);
  writeln(C)
end.
```

Какой результат будет получен, если в качестве исходных значений A и B ввести, соответственно, 7 и 8?

10. Составьте программы на Паскале для решения задач № 6–9 из заданий к § 10. При этом отмените ограничения на количество операций в арифметическом выражении, сформулированные в условиях задач.

ЕК ЦОР: часть 2, глава 6, § 35. ЦОР № 2, 7.

§ 12

Алгоритмы с ветвящейся структурой

Основные темы параграфа:

- *представление ветвлений на АЯ. Трассировка ветвящихся алгоритмов;*
- *сложные ветвящиеся алгоритмы.*

Представление ветвлений на АЯ. Трассировка ветвящихся алгоритмов

Рассмотрим несколько задач, решение которых на компьютере получается с помощью ветвящихся алгоритмов.

Первая задача: **даны два числа; выбрать большее из них.**

Пусть исходными данными являются переменные A и B. Их значения будут задаваться вводом. Значение большего из них должно быть присвоено переменной C и выведено на экран компьютера. Например, если $A = 5$, $B = 8$, то должно получиться: $C = 8$.

Блок-схема алгоритма решения этой задачи изображена на рис. 2.2.

Нетрудно понять смысл этого алгоритма. Если значение переменной A больше, чем B, то переменной C присвоится значение A. В противном случае, когда $A \le B$, переменной C присвоится значение B.

Условием, по которому разветвляется алгоритм, является *отношение неравенства* $A > B$. Изучая базы данных и электронные таблицы, вы узнали, что такое отношение является *логическим выражением*. Если оно справедливо, то результатом будет логическая величина «истина» и выполнение алгоритма продолжится по ветви «да»; в противном случае логическое выражение примет значение «ложь» и выполнение алгоритма пойдет по ветви «нет».

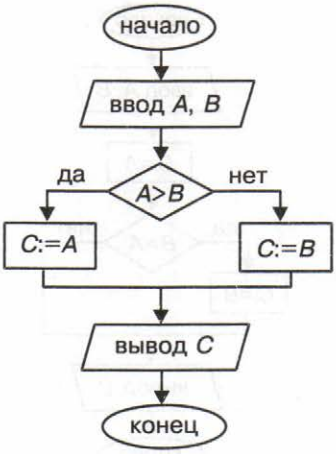


Рис. 2.2. Алгоритм выбора большего из двух чисел (с полным ветвлением)

До выполнения на компьютере правильность алгоритма можно проверить путем заполнения трассировочной таблицы. Вот как будет выглядеть трассировка нашего алгоритма для исходных значений $A = 5$, $B = 8$.

Шаг	Операция	*A*	*B*	*C*	Проверка условия
1	ввод *A*, *B*	5	8	–	
2	*A*>B	5	8	–	5 > 8, нет (ложь)
3	*C*:=*B*	5	8	8	
4	вывод *C*	5	8	8	

Ветвление является **структурной командой**. Его исполнение происходит в несколько шагов: проверка условия (выполнения логического выражения) и выполнение команд на одной из ветвей «да» или «нет». Поэтому в трассировочной таблице записываются не команды алгоритма, а отдельные операции, выполняемые компьютером на каждом шаге.

В алгоритме на рис. 2.2 используется **полное ветвление**. Эту же самую задачу можно решить, применяя структурную команду **неполного ветвления**. Блок-схема такого алгоритма изображена на рис. 2.3.

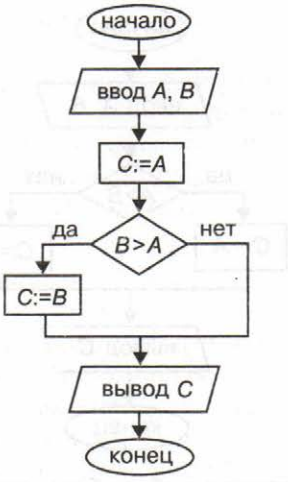


Рис. 2.3. Алгоритм выбора большего из двух значений (с неполным ветвлением)

Выполните самостоятельно трассировку этого алгоритма для вариантов 1) $A = 0{,}2$, $B = 0{,}3$; 2) $A = 7$, $B = 4$; 3) $A = 5$, $B = 5$. Если вы всё проделаете правильно, то убедитесь, что алгоритм верный.

А теперь запишем рассмотренные алгоритмы на АЯ. Во-первых, нужно решить вопрос о том, как описать переменные в этом алгоритме. Для всех переменных в алгоритме на АЯ необходимо указать их тип.

Переменные A, B, C — числовые величины. В этой задаче они могут принимать любые значения. В программировании числовые величины, которые могут иметь любые значения — целые, дробные, называются вещественными. Им ставится в соответствие **вещественный тип**. На АЯ этот тип указывается служебным словом **вещ**.

Как выглядит команда ветвления, вы уже знаете. Вот два алгоритма на АЯ, соответствующие блок-схемам на рис. 2.2 и 2.3.

```
алг БИД1
вещ A, B, C
нач
    ввод A, B
    если A>B
        то C:=A
        иначе C:=B
    кв
    вывод C
кон
```

```
алг БИД2
вещ A, B, C
нач
    ввод A, B
    C:=A
    если B>A
        то C:=B
    кв
    вывод C
кон
```

Под сокращенным названием алгоритмов БИД подразумевается «Большее из двух».

Для программирования характерно то, что одна и та же задача может быть решена с помощью разных алгоритмов. И чем сложнее задача, тем больше можно придумать различных алгоритмов ее решения. Для больших задач (производственных, научных) практически невозможно точное совпадение алгоритмов, составленных разными программистами.

Следующая задача: **упорядочить значения двух переменных X и Y по возрастанию.** Смысл этой задачи следующий: если для исходных значений переменных справедливо отношение $X \le Y$ (например, $X = 1$, $Y = 2$), то оставить их без изменения; если же $X > Y$ (например, $X = 2$, $Y = 1$), то выполнить обмен значениями.

Алгоритм обмена значениями двух переменных был рассмотрен в предыдущем параграфе. Вспомним, что для обмена мы использовали третью, вспомогательную переменную.

В алгоритме решения данной задачи используется неполное ветвление. Приведем блок-схему (рис. 2.4) и алгоритм на АЯ.

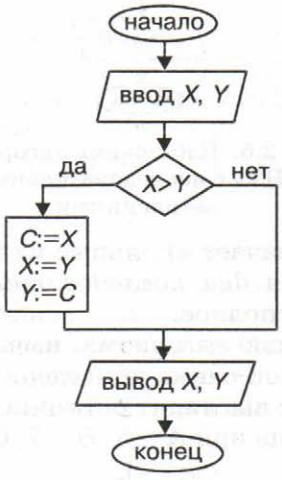


Рис. 2.4. Блок-схема алгоритма упорядочения двух величин

```
алг СОРТИРОВКА
вещ X, Y, C
нач
    ввод X, Y
    если X>Y
          то C:=X
             X:=Y
             Y:=C
    кв
    вывод X, Y
кон
```

Здесь роль вспомогательной переменной для обмена выполняет C.

Сложные ветвящиеся алгоритмы

Получим алгоритм решения еще одной задачи: **найти наибольшее значение среди трех величин: A, B, C.**

Естественно, возникает следующая идея этого алгоритма: сначала нужно найти большее из значений A и B и присвоить его какой-то дополнительной переменной, например D; затем найти большее среди D и C. Это значение можно присвоить той же переменной D.

Решение задачи сводится к двукратному применению уже знакомого алгоритма нахождения большего из двух значений. Блок-схема алгоритма — на рис. 2.5.

```
алг БИТ1
вещ A, B, C, D
нач
    ввод A, B, C
    если A>B
        то D:=A
        иначе D:=B
    кв
    если C>D
        то D:=C
    кв
    вывод D
кон
```

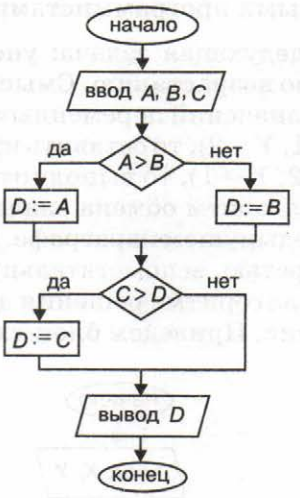


Рис. 2.5. Блок-схема алгоритма «БИТ» с последовательными ветвлениями

Нетрудно догадаться, что «БИТ» обозначает «Большее из трех». В структуре этого алгоритма содержатся *два последовательных ветвления*: первое — полное, второе — неполное.

Эту же задачу можно решить с помощью алгоритма, имеющего структуру *вложенных ветвлений*. Его блок-схема приведена в следующем параграфе на рис. 2.6. А вот как выглядят описание этого алгоритма на АЯ и трассировочная таблица при $A = 5$, $B = 7$, $C = 2$.

```
алг БИТ2
вещ A, B, C, D
нач
    ввод A, B, C
    если A>B
        то если A>C то D:=A иначе D:=C кв
        иначе если B>C то D:=B иначе D:=C кв
    кв
    вывод D
кон
```

Шаг	Операция	*A*	*B*	*C*	*D*	Проверка условия
1	ввод *A*, *B*, *C*	5	7	2	–	
2	*A*>*B*	5	7	2	–	5 > 7, нет
3	*B*>*C*	5	7	2	–	7 > 2, да
4	*D*:=*B*	5	7	2	7	
5	вывод *D*	5	7	2	7	

Коротко о главном

В команде ветвления в качестве условия может использоваться отношение неравенства между величинами.

Числовые величины, которые могут принимать целые и дробные значения, имеют вещественный тип.

Для решения одной и той же задачи можно построить несколько вариантов алгоритмов.

Несколько ветвлений в одном алгоритме могут быть последовательными и вложенными.

Вопросы и задания

1. Какую структуру имеет алгоритм нахождения большего из двух значений?
2. Почему отношение неравенства можно назвать логическим выражением?
3. В каком случае для числовой переменной следует указывать целый тип, в каком — вещественный?
4. Составьте алгоритм (в виде блок-схемы и на АЯ) нахождения меньшего из двух значений.
5. Составьте алгоритм нахождения наименьшего из трех значений.
6. Для вывода на экран произвольной символьной строки нужно в команде вывода записать эту строку в кавычках. Например, по команде

   вывод "ОТВЕТ"

   на экран выведется слово ОТВЕТ.

   Определите, какая задача решается по следующему алгоритму.

```
алг Задача-6
вещ X
нач
    ввод X
    если X<0
        то вывод "отрицательное число"
        иначе вывод "положительное число"
    кв
кон
```

7. Составьте алгоритм, по которому на компьютере будет происходить следующее: в переменную S вводится возраст Саши, в переменную M вводится возраст Маши. В качестве результата на экран выводится фраза «Саша старше Маши» или «Маша старше Саши» (предполагаем, что кто-нибудь из них обязательно старше).
8. Решите предыдущую задачу, учитывая возможность одинакового возраста Саши и Маши. В таком случае может быть получен ответ: «Саша и Маша — ровесники».
9. Составьте алгоритм упорядочения значений трех переменных по возрастанию, т. е. при любых исходных значениях A, B, C отсортируйте их так, чтобы стало: $A \le B \le C$. Проверьте алгоритм трассировкой при разных вариантах значений исходных данных.

www ЕК ЦОР: часть 2, глава 6, § 36. ЦОР № 6, 12–14.

§ 13

Программирование ветвлений на Паскале

Основные темы параграфа:

- *оператор ветвления на Паскале;*
- *программирование полного и неполного ветвления;*
- *программирование вложенных ветвлений;*
- *логические операции;*
- *сложные логические выражения.*

Оператор ветвления на Паскале

В языке Паскаль имеется **оператор ветвления**. Другое его название — условный оператор.

Формат полного оператора ветвления следующий:

```
if <логическое выражение> then <оператор1>
                          else <оператор2>
```

Здесь `if` — «если», `then` — «то», `else` — «иначе».

Программирование полного и неполного ветвления

Сравните запись алгоритма БИД1 из предыдущего параграфа с соответствующей программой.

```
алг БИД1
вещ A, B, C
нач
    ввод A, B
    если A>B
        то C:=A
        иначе C:=B
    кв
    вывод C
кон
```

```
Program BID1;
var A, B, C: real;
begin
  readln(A, B);
  if A>B
  then C:=A
  else C:=B;

  writeln(C)
end.
```

Очень похоже на перевод с русского языка на английский. Обратите внимание на следующее отличие: в программе нет специального служебного слова, обозначающего конец ветвления. Здесь признаком конца оператора ветвления является точка с запятой. (Разумеется, оставлять в программе пустую строку совсем не обязательно. Здесь это сделано только ради наглядности.)

Простой формой логического выражения является **операция отношения**. Как и в АЯ, в Паскале допускаются все виды отношений (ниже указаны их знаки):

< (меньше);	<= (больше или равно);
> (больше);	= (равно);
<= (меньше или равно);	<> (не равно).

А теперь запрограммируем на Паскале алгоритм БИД2, в котором использовано неполное ветвление.

```
алг БИД2
вещ A, B, C
нач
    ввод A, B
    C:=A
    если B>A
        то C:=B
    кв
    вывод C
кон
```

```
Program BID2;
var  A, B, C : real;
begin
  readln(A, B);
  C:=A;
  if  B>A
  then C:=B;

  write(C)
end.
```

Опять всё очень похоже. Ветвь **else** в операторе ветвления может отсутствовать.

Программирование вложенных ветвлений

Запишем на Паскале программу определения большего из трех чисел, блок-схема которой показана на рис. 2.6. Структура этого алгоритма — вложенные ветвления. Алгоритм на АЯ (БИТ2) приведен в предыдущем параграфе.

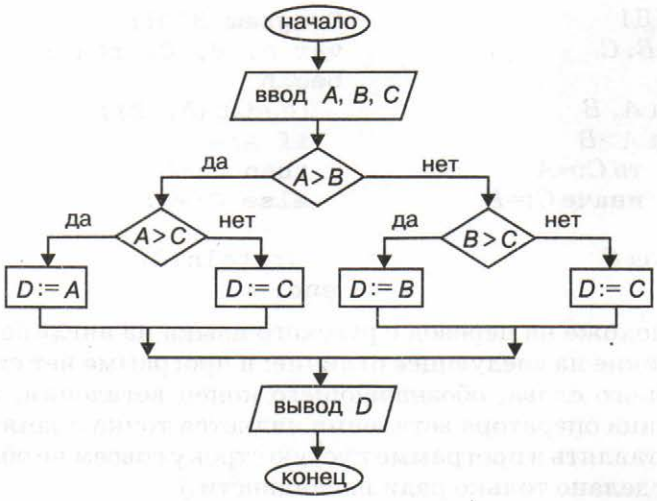


Рис. 2.6. Блок-схема алгоритма «БИТ» с вложенными ветвлениями

```
Program BIT2;
var A, B, C, D: real;
begin
    readln(A, B, C);
    if A>B
       then if A>C then D:=A else D:=B
       else if B>C then D:=B else D:=C;
    writeln(D)
end.
```

Обратите внимание на то, что перед **else** символ «;» не ставится, так как этот символ является разделителем операторов. Вся ветвящаяся часть структуры алгоритма заканчивается на точке с запятой после оператора `D:=C`.

Составим программу упорядочения значений двух переменных.

```
алг СОРТИРОВКА
вещ X, Y, C
нач
   ввод X, Y
   если X>Y
        то C:=X
           X:=Y
           Y:=C
   кв
   вывод X, Y
кон
```

```
Program SORTING;
var X, Y, C: real;
begin
  readln(X, Y);
  if  X>Y
  then begin C:=X;
             X:=Y;
             Y:=C
       end;
write(X,Y)
end.
```

Этот пример иллюстрирует следующее правило Паскаля: если на какой-то из ветвей оператора ветвления находится несколько последовательных операторов, то их нужно записывать между служебными словами **begin** и **end**. Конструкция такого вида:

begin <последовательность операторов> **end**

называется **составным оператором**. Следовательно, в описанной выше общей форме ветвления <оператор1> и <оператор2> могут быть простыми и составными операторами.

Логические операции

Наконец, составим еще один, третий вариант программы определения наибольшего числа из трех.

```
Program BIT3;
var A, B, C, D: real;
begin
      readln(A,B,C);
      if (A>=B) and (A>=C) then D:=A;
      if (B>=A) and (B>=C) then D:=B;
      if (C>=A) and (C>=B) then D:=C;
      writeln(D)
end.
```

Нетрудно понять смысл этой программы. Здесь использованы три последовательных неполных ветвления. А условия ветвлений представляют собой **сложные логические выражения**, включающие **логическую операцию and (И)**. С логическими операциями вы встречались, работая с базами данных и с электронными таблицами.

Напомним, что операция **and** называется логическим умножением или конъюнкцией. Ее результат — «истина», если значения обоих операндов — «истина». Очевидно, что если $A \geq B$ и $A \geq C$, то A имеет наибольшее значение и т. д. В Паскале присутствуют все три основные логические операции:

and — И (конъюнкция),
or — ИЛИ (дизъюнкция),
not — НЕ (отрицание).

Замечание. Мы рассмотрели три варианта решения задачи о поиске максимального числа из трех заданных (БИТ1, БИТ2, БИТ3). Предложенные варианты решения, помимо всего прочего, отличаются еще и эффективностью: эффективность тем выше, чем меньше исполняется операций при выполнении программы. С этой точки зрения, самым неэффективным является алгоритм БИТ3. В любом случае в нем будет выполняться 6 операций отношения. При исполнении алгоритмов БИТ1 и БИТ2 выполняются 2 операции отношения. Понятно, что на практике необходимо использовать наиболее эффективные алгоритмы.

Сложные логические выражения

Обратите внимание на то, что отношения, связываемые логическими операциями, заключаются в скобки. Так надо делать всегда! Например, требуется определить, есть ли среди чисел *A*, *B*, *C* хотя бы одно отрицательное. Эту задачу решает следующий оператор ветвления:

```
if (A<0) or (B<0) or (C<0)
   then write('YES') else write('NO')
```

Выражение, истинное для отрицательного числа, может быть записано еще и так:

```
not (A>=0)
```

Коротко о главном

Формат оператора ветвления (условного оператора) Паскаля:

```
if <логическое выражение>
  then <оператор1> else <оператор2>
```

На ветвях условного оператора могут находиться простые или составные операторы. Составной оператор — это последовательность операторов, заключенная между служебными словами **begin** и **end**.

В сложных логических выражениях используются логические операции: **and**, **or**, **not**.

Вопросы и задания

1. Как программируется на Паскале полное и неполное ветвление?
2. Что такое составной оператор? В каких случаях составной оператор используется в операторе ветвления?
3. Выполните на компьютере все программы, приведенные в данном параграфе.
4. Составьте не менее трех вариантов программы определения наименьшего из трех данных чисел.
5. Составьте программу сортировки по возрастанию значений трех переменных: *A*, *B*, *C*.
6. Составьте программу вычисления корней квадратного уравнения по данным значениям его коэффициентов.

www ЕК ЦОР: часть 2, глава 6, § 37. ЦОР № 6.

§ 14
Программирование диалога с компьютером

Основные темы параграфа:

- *что такое диалог с компьютером;*
- *пример программирования диалога.*

Что такое диалог с компьютером

Если вы исполняли рассмотренные выше программы на компьютере, то почувствовали определенное неудобство при работе с машиной. Во-первых, непонятно, когда машина начинает ожидать ввода данных, какие данные и в каком порядке нужно вводить (это ведь можно и забыть). Во-вторых, результаты получаются в виде чисел на экране, без всяких пояснений их смысла. Ясно, что люди между собой так не общаются.

Любую программу составлять нужно так, чтобы ее исполнение реализовывало диалог между компьютером и пользователем в понятной для человека форме.

Прежде чем начать составление программы, нужно продумать **сценарий** такого диалога.

Например, составим сценарий работы программы, вычисляющей сумму двух целых чисел. На экране компьютера последовательно должны появляться следующие строки (для примера предположим, что будем вводить числа 237 и 658):

Введите первое слагаемое: A = **237**
Введите второе слагаемое: B = **658**

A + B = 895
Пока!

Здесь курсивом записаны символы, которые выводит компьютер по программе, а прямым жирным шрифтом — символы, вводимые пользователем.

Любой вывод на экран происходит по оператору вывода, записанному в программе.

Следовательно, с помощью оператора вывода на экран выносятся не только результаты решения задачи, но и все элементы диалога со стороны компьютера.

Вот программа, которая реализует наш сценарий:

```
Program Summa;
var A, B: integer;
begin
        write ('Введите первое слагаемое: A =');
        readln(A);
        write('Введите второе слагаемое: B =');
        readln(B);
        writeln;
        writeln('A + B = ',  A+B);
        writeln('Пока!')
end.
```

В этой программе используется возможность включать в список вывода символьные строки (они заключаются в апострофы) и арифметические выражения. Выражение `A+B` сначала вычисляется, а потом полученное число выводится на экран. Конечно, для вычисления суммы можно было написать отдельный оператор присваивания, но можно сделать и так, как в этом примере.

Еще обратите внимание на оператор `writeln` без списка вывода. Он обеспечивает пропуск строки на экране.

Пример программирования диалога

Компьютерная программа совсем не обязательно должна иметь математическое содержание. Вот пример сценария, судя по которому, компьютер выполняет роль электронной няньки, заботящейся о здоровье школьника. Приводятся два варианта развития сценария, в зависимости от ответа ребенка.

Вариант 1:

Ты вчера был болен. Измерь-ка температуру!
Сообщи, какая у тебя температура: **36.5**
Ты здоров, дружок! Можешь идти в школу.
Желаю успехов!

Вариант 2:

Ты вчера был болен. Измерь-ка температуру!
Сообщи, какая у тебя температура: **37.3**
Ты еще болен! Раздевайся и ложись в постель.
Поправляйся, дружок!

Алгоритм этой программы содержит ветвление. Идея алгоритма состоит в том, что значение температуры ребенка сравнивается с величиной нормальной температуры человека: 36,6 °С. И если у ребен-

ка температура выше, то он нездоров. Вот соответствующий алгоритм на АЯ:

```
алг НЯНЬКА
вещ T
нач
   вывод "Ты вчера был болен. Измерь-ка температуру!"
   вывод "Сообщи, какая у тебя температура: "
   ввод (T)
   если T> 36.6
   то вывод "Ты еще болен! Раздевайся и ложись в постель."
      вывод "Поправляйся, дружок!"
   иначе вывод "Ты здоров, дружок! Можешь идти в школу."
        вывод "Желаю успехов!"
   кв
кон
```

По этому алгоритму получается следующая программа на Паскале:

```
Program  NANNY;
Var T: real;
begin
  writeln('Ты вчера был болен. Измерь-ка
           температуру!');
  write('Сообщи, какая у тебя температура:');
  readln(T);
  if T>36.6
     then begin
             writeln('Ты еще болен! Раздевайся
                      и ложись в постель.');
             writeln('Поправляйся, дружок!')
          end
     else begin
             writeln('Ты здоров, дружок!
                      Можешь идти в школу.');
             writeln('Желаю успехов!')
          end
end.
```

Обратите внимание на два момента: во-первых, *перед словом* **else** *ни в коем случае нельзя ставить точку с запятой*; во-вторых, в записи и при вводе вещественных чисел *целая и дробная части числа отделяются десятичной точкой.*

Составляя подобную программу, вы сами организуете интерфейс компьютера с пользователем вашей программы. Этот *интерфейс* обязательно *должен быть дружественным.* Содержание диалога должно быть понятным и удобным.

Коротко о главном

Сценарий работы программы — это описание ее общения с пользователем (пользовательского интерфейса). Интерфейс обязательно должен быть дружественным.

Любой символьный вывод на экран программируется с помощью оператора `write` или `writeln`.

Вопросы и задания

1. Что означает понятие «диалоговый характер программы»?
2. Какими средствами программируется диалог между пользователем и компьютером?
3. Что означает понятие «дружественный интерфейс»?
4. Выполните на компьютере все программы, приведенные в данном параграфе.
5. Постройте алгоритм и составьте программу, по которой будет реализован следующий сценарий: компьютер запрашивает номер дня недели, после ввода компьютер сообщает название этого дня. Например, если ввели 1, то выведется фраза «Это понедельник» и т. д.

www ЕК ЦОР: часть 2, глава 6, § 38. ЦОР № 2, 6–8.

§ 15

Программирование циклов

Основные темы параграфа:

- *этапы решения расчетной задачи на компьютере;*
- *задача о перестановке букв. Программирование цикла на Паскале;*
- *что такое отладка и тестирование программы.*

Вы научились составлять линейные и ветвящиеся программы на Паскале. Теперь нужно освоить программирование циклов. Снова будем учиться на примере конкретной задачи. Но, в отличие от предыдущих примеров, подход к ее решению будет несколько другим.

Этапы решения расчетной задачи на компьютере

Часто задача, которую требуется решить, сформулирована не на математическом языке. Для решения на компьютере ее сначала нужно привести к форме математической задачи, а потом уже программировать.

Работа по решению таких задач с использованием компьютера проходит через следующие этапы:

1. Постановка задачи.
2. Математическая формализация.
3. Построение алгоритма.
4. Составление программы на языке программирования.
5. Отладка и тестирование программы.
6. Проведение расчетов и анализ полученных результатов.

Эту последовательность называют **технологией решения задачи на компьютере**.

В чистом виде программированием, т. е. разработкой алгоритма и программы, здесь являются лишь 3-й, 4-й и 5-й этапы.

На этапе постановки задачи должно быть четко определено, **что дано и что требуется найти**.

Второй этап — математическая формализация. Здесь задача переводится на язык математических формул, уравнений, отношений. Далеко не всегда эти формулы очевидны. Нередко их приходится выводить самому или отыскивать в специальной литературе. Если решение задачи требует математического описания какого-то реального объекта, явления или процесса, то формализация равносильна получению соответствующей **математической модели**.

Третий этап — построение алгоритма. Вы знаете два способа описания алгоритмов: блок-схемы и АЯ.

Первые три этапа — это работа без компьютера. Дальше следует собственно программирование на определенном языке в определенной системе программирования. Последний (шестой) этап — это использование уже разработанной программы в практических целях.

Задача о перестановке букв. Программирование цикла на Паскале

Проследим все этапы технологии решения задачи на компьютере на примере конкретной задачи.

1. Постановка задачи. Дано N кубиков, на которых написаны разные буквы. Сколько различных N-буквенных слов можно составить из этих кубиков (слова не обязательно должны иметь смысл)?

Искомую целочисленную величину обозначим буквой F. Тогда постановка задачи выглядит так:

Дано: N.

Найти: F.

2. Математическая формализация. Получим расчетную формулу. Сначала рассмотрим несколько конкретных примеров. Имеются два кубика с буквами «И» и «К». Ясно, что из них можно составить два слова:

ИК КИ.

Добавим к ним *третью* букву, «С». Теперь число разных слов будет в *три раза* больше предыдущего, т. е. равно 6:

ИКС КИС ИСК КСИ СКИ СИК.

Если добавить *четвертую* букву, например «А», то число слов возрастет *в четыре раза* и станет равным 24:

КИСА КИАС КСИА КСАИ КАИС КАСИ ИКСА ИКАС ИСКА ИСАК ИАКС ИАСК СКИА СКАИ СИКА СИАК САКИ САИК АКИС АКСИ АИКС АИСК АСКИ АСИК.

Попробуйте записать все варианты слов из пяти букв: И, К, С, А, У. Сделать это непросто. Ясно лишь, что количество таких слов будет *в пять раз больше* 24, т. е. равно 120. Из шести букв можно составить 720 различных слов. С ростом числа букв число слов быстро растет. Например, для 10 букв получается 3 628 800 слов.

Подобные задачи решает раздел математики, который называется **комбинаторикой**.

Количество различных комбинаций из N предметов, получаемых изменением их порядка, называется числом **перестановок**. *Это число выражается функцией от N, которая называется* **факториалом** *и записывается так:*

$N!$

Читается: **«эн факториал»**. Для любого натурального N значение $N!$ вычисляется как произведение последовательности натуральных чисел от 1 до N. Например:

$1! = 1$;
$2! = 1 \cdot 2 = 2$;
$3! = 1 \cdot 2 \cdot 3 = 6$;
$4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$;
$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$
и т. д.

Теперь вернемся к формулировке задачи. Если N обозначает количество букв, а F — количество слов из этих букв, то расчетная формула такова:

$$F = N! = 1 \cdot 2 \cdot \ldots \cdot N.$$

3. Построение алгоритма. Поскольку алгоритм должен быть независимым от данного значения N, то его нельзя сделать линейным. Дело в том, что для разных N надо выполнить разное число умножений. В таком случае с изменением N линейная программа должна была бы менять длину.

Алгоритм решения данной задачи будет **циклическим**. С циклическими алгоритмами вы уже познакомились, работая с графическим исполнителем.

Цикл — это команда исполнителю многократно повторить указанную последовательность команд.

Рассмотрим блок-схему на рис. 2.7 и алгоритм на АЯ.

```
алг СЛОВА
цел F, N, R
нач
    ввод N
    F:=1
    R:=1
    пока R<=N повторять
    нц
      F:=F*R
      R:=R+1
    кц
    вывод F
кон
```

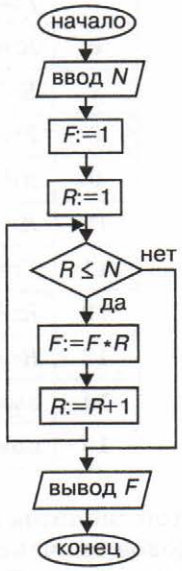


Рис. 2.7. Блок-схема алгоритма вычисления факториала

Здесь применена знакомая вам алгоритмическая структура **«цикл с предусловием»**. Выполняется она так: *пока истинно условие цикла, повторяется выполнение тела цикла.*

Тело цикла составляют две команды присваивания, заключенные между служебными словами **нц** и **кц**. Условие цикла — это отношение $R<=N$ (R меньше или равно N).

В данном алгоритме переменная R выполняет роль множителя, значение которого меняется от 1 до N через единицу. Произведение накапливается в переменной F, начальное значение которой равно 1. Цикл заканчивается, когда R становится равным $N + 1$. Это значение в произведение уже не попадет.

Для проверки правильности алгоритма построим трассировочную таблицу (для случая $N = 3$):

Шаг	Операция	N	F	R	Условие
1	ввод N	3	–	–	
2	F:=1		1	–	
3	R:=1			1	
4	$R<=N$				1<=3, да
5	$F:=F*R$		1		
6	$R:=R+1$			2	
7	$R<=N$				2<=3, да
8	$F:=F*R$		2		
9	$R:=R+1$			3	
10	$R<=N$				3<=3, да
11	$F:=F*R$		6		
12	$R:=R+1$			4	
13	$R<=N$				4<=3, нет
14	вывод F		6		
15	конец				

Из этой таблицы хорошо видно, как менялись значения переменных. Новое значение, присвоенное переменной, стирает ее старое значение (в данной таблице не повторяется запись значения переменной, если оно не изменяется; в таком виде таблица менее загромождена числами). Последнее значение F равно 6. Оно выводится в качестве результата. Очевидно, что результат верный: $3! = 6$.

4. Составление программы. Чтобы составить программу решения нашей задачи, нужно научиться программировать циклы на Паскале. Основной циклической структурой является **цикл с предусловием (цикл-пока)**. С помощью этой структуры можно построить любой циклический алгоритм. Оператор цикла с предусловием в Паскале имеет следующий формат:

while <выражение> **do** <оператор>

Служебное слово **while** означает «пока», **do** — «делать», «выполнять».

Оператор, стоящий после слова **do**, называется **телом цикла**. Тело цикла может быть простым или составным оператором, т. е. последовательностью операторов между служебными словами **begin** и **end**.

А теперь запрограммируем на Паскале алгоритм решения нашей задачи (добавив к нему организацию диалога).

```
Program Words;
var F, N, R: integer;
begin
    write('Введите число букв');
    readln(N);
    F:=1;
    R:=1;
    while R<=N  do
    begin
        F:=F*R;
        R:=R+1
    end;
    write('Из ',N,' букв можно составить ',
        F,' слов')
end.
```

Снова бросается в глаза схожесть алгоритма на АЯ и программы на Паскале. Обратите внимание на то, что в Паскале нет специальных служебных слов для обозначения конца цикла (так же как и конца ветвления). Во всех случаях, где это необходимо, используются слова **begin** и **end**.

Что такое отладка и тестирование программы

5. Отладка и тестирование. *Под отладкой программы понимается процесс испытания работы программы и исправления обнаруженных при этом ошибок.* Обнаружить ошибки, связанные с нарушением правил записи программы на Паскале (синтаксические и семантические ошибки), помогает используемая *система програм-*

мирования. Пользователь получает сообщение об ошибке, исправляет ее и снова повторяет попытку исполнить программу.

Поиск алгоритмических ошибок в программе производится с помощью тестирования. ***Тест** — это конкретный вариант значений исходных данных, для которого известен ожидаемый результат.* Прохождение теста — необходимое условие правильности программы. На тестах проверяется правильность реализации программой запланированного сценария.

Нашу программу, например, можно протестировать на значении $N = 6$. На экране должно получиться:

Введите число букв: **6**
Из 6 букв можно составить 720 слов.

6. Проведение расчетов и анализ полученных результатов — этот этап технологической цепочки реализуется при разработке практически полезных (не учебных) программ. Например, программы «Расчет прогноза погоды». Ясно, что ею будут пользоваться длительное время, и правильность ее работы очень важна для практики. А поэтому в процессе эксплуатации эта программа может дорабатываться и совершенствоваться.

Коротко о главном

Последовательность этапов работы программиста при решении задачи на компьютере называется технологией решения задачи на компьютере. Таких этапов шесть: 1) постановка задачи; 2) математическая формализация; 3) построение алгоритма; 4) составление программы на языке программирования; 5) отладка и тестирование программы; 6) проведение расчетов и анализ полученных результатов.

Количество различных комбинаций из N предметов, получаемых изменением их порядка, называется числом перестановок. Число перестановок равно $N!$ (N факториал):

$$N! = 1 \cdot 2 \cdot \ldots \cdot N.$$

Любой циклический алгоритм может быть построен с помощью команды «цикл-пока» (цикл с предусловием).

Формат оператора цикла с предусловием в Паскале:

while <выражение> **do** <оператор>

Оператор, составляющий тело цикла, может быть простым или составным.

Вопросы и задания

1. Как блок-схемой и на алгоритмическом языке представляется команда цикла с предусловием?
2. Как программируется цикл с предусловием на Паскале?
3. Почему алгоритм вычисления $N!$ должен быть циклическим?
4. Из каких этапов состоит работа программиста по решению задачи на компьютере?
5. Что такое математическая формализация задачи?
6. Что такое отладка программы? Что называется тестом?
7. Составьте алгоритм вычисления суммы всех натуральных чисел, не превышающих заданного натурального числа N. Проверьте алгоритм трассировкой. Напишите программу на Паскале.
8. Дано целое число X и натуральное N. Составьте алгоритм вычисления X^N. Проверьте алгоритм трассировкой. Напишите программу на Паскале.

ЕК ЦОР: часть 2, глава 6, § 39. ЦОР № 7, 13–16.

§ 16
Алгоритм Евклида

Основные темы параграфа:

- *наибольший общий делитель;*
- *идея алгоритма Евклида;*
- *описание алгоритма Евклида блок-схемой;*
- *алгоритм на АЯ и программа на Паскале.*

Наибольший общий делитель

Рассмотрим следующую задачу: требуется составить программу определения наибольшего общего делителя (НОД) двух натуральных чисел.

Вспомним математику. Наибольший общий делитель двух натуральных чисел — это самое большое натуральное число, на которое они оба делятся нацело. Например, у чисел 12 и 18 имеются общие делители: 2, 3, 6. Наибольшим общим делителем является число 6. Это записывается так:

$$\text{НОД}(12, 18) = 6.$$

Обозначим исходные данные как M и N. Постановка задачи выглядит следующим образом:

Дано: M, N.
Найти: НОД(M, N).

В данном случае какой-то дополнительной математической формализации не требуется. Сама постановка задачи носит формальный математический характер. Не существует формулы для вычисления НОД(M, N) по значениям M и N. Но зато достаточно давно, задолго до появления ЭВМ, был известен алгоритмический способ решения этой задачи. Называется он ***алгоритмом Евклида***.

Идея алгоритма Евклида

Идея этого алгоритма основана на том свойстве, что если $M>N$, то

$$\text{НОД}(M, N) = \text{НОД}(M - N, N).$$

Иначе говоря, НОД двух натуральных чисел равен НОД их положительной разности (модуля их разности) и меньшего числа.

Легко доказать это свойство. Пусть K — общий делитель M и N ($M > N$). Это значит, что $M = mK$, $N = nK$, где m, n — натуральные числа, причем $m > n$. Тогда $M - N = K(m - n)$, откуда следует, что K — делитель числа $M - N$. Значит, все общие делители чисел M и N являются делителями их разности $M - N$, в том числе и наибольший общий делитель.

Второе очевидное свойство:

$$\text{НОД}(M, M) = M.$$

Для «ручного» счета алгоритм Евклида выглядит так:

1) если числа равны, то взять любое из них в качестве ответа, в противном случае продолжить выполнение алгоритма;
2) заменить большее число разностью большего и меньшего из чисел;
3) вернуться к выполнению п. 1.

Рассмотрим этот алгоритм на примере M=32, N=24:

M	32	8	8	8
N	24	24	16	8

Получили: НОД(32, 24) = НОД(8, 8) = 8, что верно.

Описание алгоритма Евклида блок-схемой

На рисунке 2.8 приведена блок-схема алгоритма Евклида.

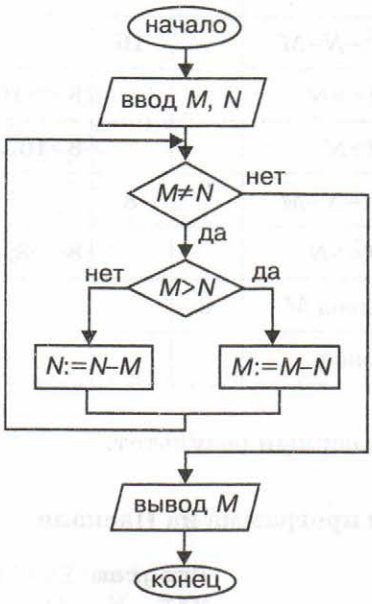


Рис. 2.8. Блок-схема алгоритма Евклида

Структура алгоритма — цикл-пока с вложенным ветвлением. Цикл повторяется, пока значения M и N не равны друг другу. В ветвлении большее из двух значений заменяется на их разность.

А теперь посмотрите на трассировочную таблицу алгоритма для исходных значений $M = 32$, $N = 24$:

Шаг	Операция	M	N	Условие
1	ввод M	32		
2	ввод N		24	
3	$M<>N$			32<>24, да
4	$M>N$			32>24, да
5	$M:=M–N$	8		
6	$M<>N$			8<>24, да

Шаг	Операция	*M*	*N*	Условие
7	*M*>*N*			8>24, нет
8	*N*:=*N*–*M*		16	
9	*M*<>*N*			8 <>16, да
10	*M*>*N*			8>16, нет
11	*N*:=*N*–*M*		8	
12	*M*<>*N*			8<>8, нет
13	вывод *M*	8		
14	конец			

В итоге получился верный результат.

Алгоритм на АЯ и программа на Паскале

```
алг Евклид
цел M, N
нач
  вывод "Введите M и N"
  ввод M, N
  пока M ≠ N повторять
  нц
    если M>N
       то M:=M–N
       иначе N:=N–M
    кв
  кц
  вывод "НОД=", M
кон
```

```
Program Evklid;
var  M, N: integer;
begin
  writeln('Введите M и N');
  readln(M, N);
  while M<>N do
    begin
      if M>N
        then  M:=M-N
        else  N:=N-M
    end;
  write('НОД=', M)
end.
```

Коротко о главном

Алгоритм Евклида предназначен для получения наибольшего общего делителя двух натуральных чисел. Структура алгоритма Евклида — цикл с вложенным ветвлением.

Ручная трассировка может использоваться для проверки правильности лишь сравнительно простых алгоритмов. Поиск алгоритмических ошибок в программе производится с помощью тестирования.

Вопросы и задания

1. Выполните на компьютере программу `Evklid` (из параграфа). Протестируйте ее на значениях $M = 32$, $N = 24$; $M = 696$, $N = 234$.
2. Составьте программу нахождения наибольшего общего делителя трех чисел, используя следующую формулу:

$$НОД(A, B, C) = НОД(НОД(A, B), C).$$

3. Составьте программу нахождения наименьшего общего кратного (НОК) двух чисел, используя формулу:

$$A \cdot B = НОД(A, B) \cdot НОК(A, B).$$

ЕК ЦОР: часть 2, глава 6, § 40. ЦОР № 8, 9.

§ 17
Таблицы и массивы

Основные темы параграфа:

- *что такое массив;*
- *описание и ввод значений в массив в Алгоритмическом языке;*
- *цикл с параметром в Алгоритмическом языке;*
- *расчет среднего значения элементов массива.*

Изучая базы данных, электронные таблицы, вы познакомились с табличным способом организации данных. Вы уже знаете, что большие наборы данных удобно представлять в табличном виде. В таблицах могут храниться данные разных типов. На практике чаще всего приходится встречаться с таблицами, содержащими числовые и символьные (текстовые) данные.

Что такое массив

Для представления таблиц в языках программирования можно использовать **массивы**. Вот, например, таблица, содержащая среднемесячные значения температуры в Перми в 2000 году:

Месяц	1	2	3	4	5	6	7	8	9	10	11	12
Температура	–21	–18	–7,5	5,6	10	18	22,2	24	17	5,4	–7	–18

Такую таблицу называют **линейной**. Она представляет собой последовательность пронумерованных чисел. Для обозначения этих чисел используют **индексированные имена**. Например, через $T[1]$ обозначается температура в январе (первом месяце года), $T[5]$ — температура в мае и т. д.

В программировании линейная таблица называется **одномерным массивом**. В нашем примере T — это имя массива. Элементы массива пронумерованы. Порядковый номер элемента называется его **индексом**. Каждый элемент массива обозначается индексированным именем в следующей форме:

<имя массива> [<индекс>]

Индекс записывается в квадратных скобках: $T[2]$, $T[10]$, $T[12]$. Индексы могут быть представлены не только в виде констант, но и в виде целых переменных и даже выражений целого типа: $T[i]$, $T[k]$, $T[i+k]$, $T[2*k]$. Важно следить, чтобы значения индексов не выходили за допустимые границы. В примере с температурами они должны лежать в диапазоне от 1 до 12.

Все элементы массива должны иметь одинаковый тип. Если массив состоит только из целых чисел, то тип массива — целый. В нашем примере значения температур могут быть дробными, поэтому тип массива — вещественный.

Массив — ограниченная последовательность однотипных величин.

Количество величин определяется при описании массива. Все элементы массива могут выбираться произвольно и являются одинаково доступными.

Решение задач по обработке массива связано, как правило, с перебором элементов массива. Такой перебор происходит в цикле, в котором изменяется значение индекса от начальной до конечной величины. Для того чтобы организовать ввод исходных данных в массив, нужно также использовать цикл.

Описание и ввод значений в массив в Алгоритмическом языке

Запишем алгоритм ввода значений в массив температур. Сначала посмотрим, как это делается в АЯ. Рассмотрим два варианта алгоритмов на АЯ, использующих разные способы организации цикла.

```
алг Ввод массива, вариант 1
вещ таб T[1:12]
цел I
нач
      I:=1
      пока I<=12 повторять
      нц
         вывод "T[", I , "]="
         ввод T[I]
         I:=I+1
      кц
кон
```

```
алг Ввод массива, вариант 2
вещ таб T[1:12]
цел I
нач
   для I от 1 до 12 шаг 1 повторять
   нц
      вывод "T[", I , "]="
      ввод T[I]
   кц
кон
```

Обратите внимание на вторую строку алгоритмов. В ней присутствует описание массива температур. В АЯ массив называется таблицей. Запись

вещ таб $T[1:12]$

описывает таблицу (массив) вещественного типа, имя которого T и элементы пронумерованы от 1 до 12.

Цикл с параметром в Алгоритмическом языке

В первом варианте алгоритма используется уже знакомая вам алгоритмическая структура *цикла с предусловием*. Переменная I играет роль параметра цикла, изменяющегося от 1 до 12 с шагом 1. Внутри цикла она используется в качестве индекса в обозначении элементов таблицы: $T[I]$.

Ввод организован в режиме диалога. Вы уже знаете, что это обязательное условие дружественности интерфейса программы. Перед вводом каждого очередного элемента таблицы на экран будет выводиться его имя. Это результат выполнения команды вывод "T[", I , "]=". После этого программист должен ввести с клавиатуры соответствующее число (команда ввод $T[I]$). На экране получим:

$T[1]= -21$
$T[2]= -18$
$T[3]= -7.5$ и т. д.

Во втором варианте используется алгоритмическая структура, которая называется **цикл с параметром**. Ее общая форма такая:

```
для <параметр цикла> от <начальное значение
    параметра> до <конечное значение параметра>
    шаг <величина приращения параметра> повторять
нц
  <тело цикла>
кц
```

Параметром цикла должна быть переменная целого типа. В нашем примере это переменная *I*. Выполнение тела цикла повторяется для всех последовательных значений параметра от начального до конечного значения включительно с изменением его значения при каждом повторении на величину шага. Следовательно, по второму варианту алгоритма будут выполняться те же самые действия, что и по первому.

Расчет среднего значения элементов массива

	A	B
1	*Месяц*	*Температура*
2	1	–21
3	2	–18
4	3	–7,5
5	4	5,6
6	5	10
7	6	18
8	7	22,2
9	8	24
10	9	17
11	10	5,4
12	11	–7
13	12	–18
14	Среднее:	2,56

Рис. 2.9. Таблица температур

Теперь сформулируем задачу обработки массива температур, которую будем решать дальше. Вычислим среднегодовую температуру. Для этого нужно сложить все 12 значений таблицы и разделить сумму на 12. Полученную величину выведем в качестве результата.

Эту задачу легко решить с помощью электронных таблиц. На рисунке 2.9 показана такая таблица. В ячейки B2:B13 заносятся значения температур. В ячейку B14 помещается формула: =СРЗНАЧ(B2:B13). Результат (в режиме отображения значений) в ячейке B14 — 2,56.

Табличный процессор — это программа, составленная программистами на некотором языке программирования. Вот мы и разберемся, как программируется вычисление среднего значения числового массива, реализованное в функции СРЗНАЧ.

Запишем алгоритм в полном виде (с вводом, вычислениями и выводом), используя в нем для организации циклов структуру цикла с параметром.

```
алг Средняя температура
вещ таб T[1:12]
цел I, вещ Tsred
нач
    {Цикл ввода}
    для I от 1 до 12 шаг 1 повторять
    нц
        вывод "T[", I , "]="
        ввод T[I]
    кц
    {Цикл суммирования}
    Tsred:=0
    для I от 1 до 12 шаг 1 повторять
    нц
        Tsred:=Tsred+T[I]
    кц
    {Вычисление среднего}
    Tsred:=Tsred/12
    вывод "Среднегодовая температура = ", Tsred
кон
```

Обратим внимание на следующие особенности алгоритма. Появилась новая переменная *Tsred*, в которой вычисляется среднее значение:

$$Tsred=(T[1]+T[2]+T[3]+\ldots.+T[12])/12.$$

Переменная *Tsred* имеет вещественный тип. Перед циклом суммирования этой переменной присваивается нулевое значение. Так всегда следует поступать с переменной, в которой накапливается сумма какой-то последовательности слагаемых. При каждом повторении цикла к значению переменной *Tsred* добавляется очередное слагаемое. После окончания цикла полученная сумма делится на 12. Это искомый результат, который выводится на экран.

Коротко о главном

Массив — ограниченная последовательность однотипных величин.

Линейная таблица в программировании называется одномерным массивом.

В описании массива указывается его тип, имя, границы индексов.

В алгоритмах, связанных с перебором элементов массива, удобно использовать структуру «цикл с параметром».

Вопросы и задания

1. Что такое массив?
2. Самостоятельно придумайте примеры данных, которые можно организовать в виде массива. В каждом примере отметьте: каким именем можно обозначить массив, как пронумеровать его элементы, какой тип будет иметь массив? Опишите массивы по правилам АЯ.
3. Для тех же исходных данных, что рассматриваются в параграфе, составьте алгоритм, в котором вычисляются четыре величины: средние температуры зимних, весенних, летних, осенних месяцев.
4. Вы посетили магазин и купили 10 видов товара. В таблицу T[1:10] вы записали количество купленного товара каждого вида. В таблицу C[1:10] записали цены единиц каждого вида товара соответственно. Составьте алгоритм вычисления общей стоимости всех покупок.

www ЕК ЦОР: часть 2, глава 6, § 41. ЦОР № 8, 9.

§ 18
Массивы в Паскале

Основные темы параграфа:

- *описание массива в Паскале;*
- *цикл с параметром на Паскале;*
- *форматы вывода;*
- *программа с двумя массивами.*

А теперь посмотрим, как можно на Паскале запрограммировать алгоритм вычисления среднегодовой температуры. Для этого сначала познакомимся с правилами описания массивов. Заметим, что в этом учебнике мы ограничиваемся работой только с одномерными массивами (линейными таблицами).

Описание массива в Паскале

Формат описания одномерного массива на Паскале такой:

```
var <имя массива>: array [<нижняя граница индекса .. верхняя граница индекса>] of <тип компоненты>
```

Слово «array» буквально переводится как «массив». Границы индекса могут быть любыми целыми числами. Важно, чтобы нижняя граница была меньше верхней границы. Описание массива температур будет следующим:

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

Цикл с параметром на Паскале

Рассмотрим полный текст программы на Паскале.

```
Program Temperature;
var T: array[1..12] of real;
I: integer;  Tsred: real;
begin
     {Цикл ввода}
     for I:=1 to 12 do
     begin
          write('T[',I:2,'] = ');
          readln(T[I])
     end;
     {Цикл суммирования}
     Tsred:=0;
     for I:=1 to 12 do
        Tsred:=Tsred+T[I];
     {Вычисление среднего}
     Tsred:=Tsred/12;
     writeln('Среднегодовая температура = ',
             Tsred:6:2,' градусов')
end.
```

В этой программе дважды использован оператор цикла с параметром. В простейшем случае формат этого оператора следующий:

```
for <параметр цикла>:=<начальное значение
параметра> to <конечное значение параметра>
do <тело цикла>
```

Значение параметра цикла увеличивается на единицу при каждом повторении тела цикла. Существует другой вариант этого оператора, в котором вместо слова **to** записывается **downto**. В этом случае значение параметра цикла убывает на единицу, следовательно, начальное значение должно быть больше конечного.

Так же как и для оператора цикла **while**, здесь *тело цикла может быть либо простым оператором, либо составным*. В первом случае тело цикла заканчивается на ближайшей точке с запятой. В нашем примере — это цикл суммирования. Во втором случае тело цикла заключается между словами **begin** и **end** (цикл ввода).

Форматы вывода

В программе присутствует еще один новый для вас элемент Паскаля: формат вывода. Это числа с двоеточиями, стоящие после переменных в операторе вывода `write`:

```
write('T[',I:2,'] = ')
```

В этой записи `I:2` обозначает, что значение переменной `I` выводится как целое число в две символьные позиции на экране. Для однозначного числа в первой позиции будет помещен пробел, например: `_5`.

В операторе вывода результата также используется формат вывода: `Tsred:6:2`. Значение переменной `Tsred` выводится как смешанное число в 6 позиций, две последние из которых занимает дробная часть. В третьей справа позиции — точка. Лишние позиции для целой части занимаются пробелами. Например: `_34.25`.

Результат выполнения программы `Temperature` будет выведен на экран в следующем виде:

```
Среднегодовая температура =   2.56 градусов
```

Программа с двумя массивами

А теперь расширим условие задачи. Требуется для каждого месяца определить отклонение его средней температуры от среднегодовой величины.

Вернемся к электронной таблице на рис. 2.9. Добавим к ней еще один столбец С, в котором будут вычисляться искомые отклонения. В ячейку С2 занесем формулу =B2–B14. По этой формуле вычислится отклонение январской температуры от среднегодовой. Скопировав эту формулу в ячейки С3:С13, получим все остальные величины. Смысл «замораживания» адреса B14 вам должен быть понятен. Результаты приведены в таблице на рис. 2.10.

Реализуем вычисление отклонений в программе на Паскале. Очевидно, в программе должен появиться еще один массив для размещения в нем таблицы отклонений. Дадим этому массиву имя `Dt`. Как и массив температур, он состоит из 12 чисел: `Dt[1]`, `Dt[2]`, `Dt[3]`, ..., `Dt[12]`.

К предыдущей программе надо добавить описание массива `Dt` в следующем виде:

```
var Dt: array[1..12] of real;
```

Значение каждого элемента массива вычисляется как разность между средней температурой в соответствующем месяце и среднегодовой температурой. Например, для января `Dt[1]:=T[1]-Tsred`.

	A	B	C
1	*Месяц*	*Температура*	*Отклонения*
2	1	–21	–23,56
3	2	–18	–20,56
4	3	–7,5	–10,06
5	4	5,6	3,04
6	5	10	7,44
7	6	18	15,44
8	7	22,2	19,64
9	8	24	21,44
10	9	17	14,44
11	10	5,4	2,84
12	11	–7	–9,56
13	12	–18	–20,56
14	Среднее:	2,56	

Рис. 2.10. Температуры и отклонения от среднего

Такие вычисления повторяются в цикле 12 раз. Значения массива Dt выводятся на экран.

Запишем на Паскале фрагмент, который надо вставить в конец предыдущей программы, чтобы решить поставленную задачу.

```
for I:=1 to 12 do
begin
     Dt[I]:=T[I]-Tsred;
     writeln('Dt[',I:2,'] = ',Dt[I]:6:2)
end
```

Здесь вычисление значений массива Dt и вывод их на экран совмещены в одном цикле. Результат работы программы будет следующим:

```
Dt[ 1]= -23.56
Dt[ 2]= -20.56
Dt[ 3]= -10.06
...
Dt[12]= -20.56
```

Как и следовало ожидать, это те же самые числа, что получены в электронной таблице.

Коротко о главном

Простейший формат описания одномерного массива:

```
var <имя массива>: array [<нижняя граница индекса
  .. верхняя граница индекса>] of <тип массива>
```

В простейшем случае оператор цикла с параметром записывается так:

```
for <параметр цикла>:=<начальное значение
    параметра> to <конечное значение параметра>
    do <тело цикла>
```

В формате вывода указывается количество позиций на экране для вывода значения. Для вещественного числа указывается также количество цифр в дробной части.

В программе на Паскале должен быть описан каждый используемый в ней массив.

Вопросы и задания

1. Как можно описать на Паскале массив, в котором будут храниться значения численности населения Москвы к концу каждого года XX века?
2. Вы приобрели котенка. Каждый вечер вы определяете его вес с помощью весов. Как можно описать на Паскале массив, в котором будут храниться значения веса котенка в течение месяца (например, мая)?
3. Напишите фрагмент программы на Паскале ввода исходных данных для массивов, определенных в заданиях 1 и 2.
4. Введите в компьютер программу `Temperature`, добавив к ней обработку массива `Dt`. Выполните программу, получите результаты. Сравните их с приведенными в параграфе.
5. Составьте программы на Паскале по алгоритмам из заданий 3, 4 предыдущего параграфа. Выполните эти программы на компьютере.

www ЕК ЦОР: часть 2, глава 6, § 42. ЦОР № 3, 8.

§ 19
Одна задача обработки массива

Основные темы параграфа:

- *что такое случайные числа;*
- *датчик случайных чисел на Паскале;*
- *алгоритм поиска числа в массиве;*
- *программа поиска числа в массиве.*

Решим следующую задачу. Массив заполняется случайным набором целых чисел. Нужно определить, сколько раз данное целое число входит в этот массив.

Что такое случайные числа

Сначала несколько слов о случайных числах. Все себе представляют игральный кубик, имеющий шесть граней. При каждом бросании кубика выпадение какого-то числа есть случайное событие. С равной вероятностью может выпасть любое число от 1 до 6. Результат бросания кубика — это **случайное число**. А теперь представьте себе кубик с 10 гранями. Правда, кубиком его можно назвать только условно. Это десятигранник, на гранях которого нанесены числа от 1 до 10. Результат бросания такого «кубика» — случайное число в диапазоне от 1 до 10. Еще один пример. При розыгрыше лотереи из вращающегося барабана достают пронумерованные шары. Выпавший номер шара — случайное число.

Датчик случайных чисел на Паскале

В языках программирования, как правило, имеется аналог подобного «кубика» или лототрона, позволяющий получать случайные числа. Он называется **датчиком случайных чисел**. Это стандартная функция. В Паскале она записывается так: `random(X)`. Здесь `X` — целое число. При выполнении функции ее результатом становится целое число в диапазоне от 0 до `X`. Например, если `X = 50`, то в результате можем получить любое целое число от 0 до 50.

Приведем программу, которая демонстрирует работу датчика случайных чисел на Паскале:

```
Program Example;
var I: integer;
begin
    for I:=1 to 10 do
        write(random(50):4)
end.
```

По этой программе на экран выводится десять случайных чисел из диапазона от 0 до 50. Вот результат тестового выполнения этой программы:

```
0  3  17  20  27  7  31  16  37  42
```

А теперь вернемся к условию задачи. Получающиеся с помощью датчика случайные числа «раскладываются» по элементам массива. Назовем массив `Rand`, а число элементов в нем пусть будет равно 20. Число, которое нужно искать в массиве, будет вводиться в переменную `X`.

Алгоритм поиска числа в массиве

На рисунке 2.11 приведена блок-схема алгоритма поиска в массиве *Rand* величины *X* с подсчетом числа ее вхождений в массив в переменной *NumberX*.

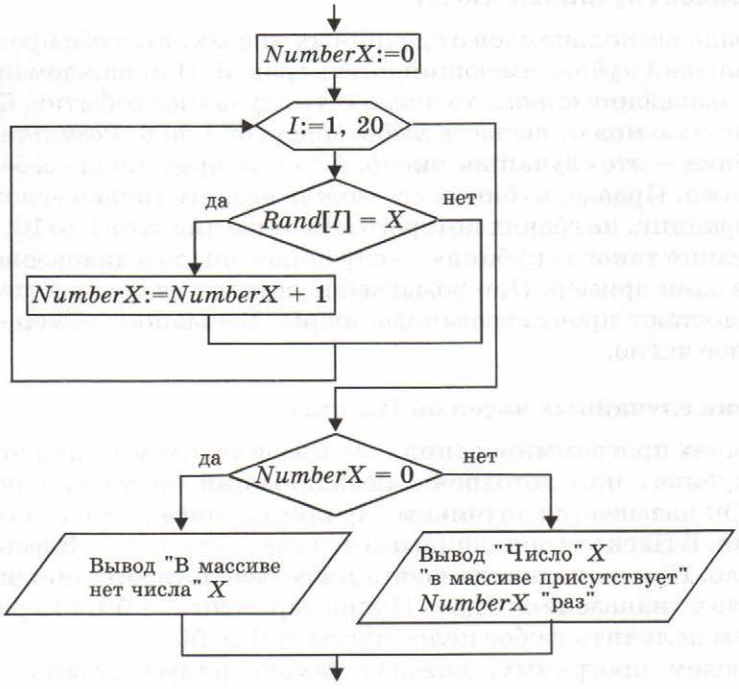


Рис. 2.11. Подсчет вхождений числа в массив

Обратите внимание на блок, отображающий цикл с параметром. Он имеет форму вытянутого шестиугольника. В блоке записывается параметр цикла (переменная *I*), начальное и конечное значения параметра через запятую (:=1, 20).

Переменная *NumberX* играет роль счетчика. Вначале ей присваивается ноль. Затем в цикле происходит перебор всех элементов массива, и при каждом выполнении условия равенства к счетчику добавляется единица. Так всегда организуются счетчики в программах! В результате выполнения программы на экран будет выведен один из двух вариантов ответа: либо сообщение, что в массиве нет данного числа, либо сообщение о том, сколько раз это число присутствует в массиве, если оно там обнаружено.

Программа поиска числа в массиве

Напишем программу на Паскале, содержащую как заполнение массива случайными числами, так и алгоритм, описанный в блок-схеме на рис. 2.11.

```
Program Example2;
var Rand: array[1..20] of integer;
    I, X, NumberX : integer,
begin
    {Установка датчика случайных чисел}
    Randomize;
    {Заполнение массива случайными числами
    и вывод их на экран}
    writeln('Массив случайных чисел:')
    for I:=1 to 20 do
    begin
         Rand[I]:=random(50); Write(Rand[I]:4)
    end;
    writeln;
    {Ввод X}
    write('Введите X:'); Readln(X);
    {Подсчет числа вхождений X в массив}
    NumberX:=0;
    for I:=1 to 20 do
       if Rand(I)=X then NumberX:=NumberX+1;
    {Анализ и вывод результата}
    if NumberX=0
       then writeln('В массиве нет числа ',X)
       else writeln('Число ',X,
        ' в массиве присутствует ',NumberX,' раз')
end.
```

В этой программе присутствует еще один новый для нас оператор: `Randomize`. Это стандартная процедура Паскаля, которая производит *установку начального состояния датчика случайных чисел*. Дело в том, что без этого оператора функция `random` при многократном повторении выполнения программы всегда будет выдавать одну и ту же последовательность чисел. Процедура `Randomize` *случайным образом* устанавливает начальное состояние датчика. Поэтому при каждом выполнении программы будут получаться разные наборы случайных чисел.

Посмотрите на результаты выполнения этой программы. Первое выполнение:

```
Массив случайных чисел:
5 44  0 41 29 12  1 39 32 25 17 31 44  5  5 16 29 10 13 34
Введите X: 5
Число 5 в массиве присутствует 3 раз
```

Жирным шрифтом обозначено вводимое с клавиатуры значение. Все остальные символы выводятся на экран по программе. Второе выполнение программы:

```
Массив случайных чисел:
41 33 26 26 33 11 19 38 29  4 31 20  6 32 39 21 28 40  7 19
Введите X: 2
В массиве нет числа 2
```

Коротко о главном

Случайные числа — результаты случайного выбора из конечного множества значений (игровой кубик, жребий, лотерея).

Функция `random(X)` — датчик случайных чисел в диапазоне от 0 до `X` на Паскале.

Для подсчета количества величин используется переменная-счетчик.

Вопросы и задания

1. Какие значения может принимать целая переменная `Y`, если в программе записано: `Y:=10+random(5)`?
2. Введите в компьютер программу `Example2`. Выполните программу, получите результаты.
3. Используя функцию `random(X)`, получите числа в диапазонах: от 1 до 10, от −10 до +10, от 50 до 100.

4. Составьте программу заполнения массива из 100 чисел случайными значениями из диапазона от –20 до 20. Подсчитайте в этом массиве количество положительных и количество отрицательных значений.
5. Заполните случайными числами в диапазоне от 1 до 5 два массива: $A[1:20]$ и $B[1:20]$. Найдите и выведите на экран только те элементы этих массивов, значения которых совпадают. Например, если $A[2] = B[2] = 4$, то на экран надо вывести:
   Номер: 2 значение: 4
   Если таких совпадений нет, то выведите на экран сообщение об этом.
6. Придумайте свои способы получения случайных чисел.

ЕК ЦОР: часть 2, глава 6, § 43. ЦОР № 10–11.

§ 20

Поиск наибольшего и наименьшего элементов массива

Основные темы параграфа:

- *поиск максимума и минимума в электронной таблице;*
- *блок-схемы алгоритмов поиска максимума и минимума в массиве;*
- *программа на Паскале поиска максимума и минимума в массиве.*

Поиск максимума и минимума в электронной таблице

Одной из типовых задач обработки массивов является поиск наибольшего или наименьшего значения среди значений его элементов. Построим алгоритм решения этой задачи и составим программу на Паскале. Для примера возьмем итоговые данные чемпионата России по футболу в премьер-лиге за 2003 год.

На рисунке 2.12 показана электронная таблица с итогами чемпионата. В столбце A расположены названия команд, в столбце B — количество очков, набранных каждой командой. Команды перечислены в алфавитном порядке. Победителем является команда, набравшая наибольшее количество очков. Команда, набравшая очков меньше всех других, в следующем сезоне покидает премьер-лигу.

Для определения максимального значения в электронной таблице существует функция МАКС( ), а для нахождения минимального значения — функция МИН( ). В ячейке B17 записана формула МАКС(B1:B16), а в ячейке B18 — формула МИН(B1:B16). Результа-

	A	B
1	ДИНАМО	46
2	ЗЕНИТ	56
3	КРЫЛЬЯ СОВЕТОВ	42
4	ЛОКОМОТИВ	52
5	РОСТОВ	34
6	РОТОР	32
7	РУБИН	53
8	САТУРН	45
9	СПАРТАК	36
10	СПАРТАК-АЛАНИЯ	31
11	ТОРПЕДО	43
12	ТОРПЕДО-МЕТАЛЛУРГ	29
13	УРАЛАН	28
14	ЦСКА	59
15	ЧЕРНОМОРЕЦ	24
16	ШИННИК	47
17	Максим. кол-во очков	59
18	Миним. кол-во очков	24

Рис. 2.12. Итоги чемпионата

ты вы видите в таблице. Отсюда делаем вывод: чемпионом России стала команда ЦСКА, а на последнем месте — «Черноморец».

Блок-схемы алгоритмов поиска максимума и минимума в массиве

Разберемся, как же программируется определение максимального и минимального значений в числовом массиве. Начнем с поиска максимума. Опишем алгоритм на Алгоритмическом языке.

Пусть в целочисленный массив $B[1:16]$ заносятся очки команд в том порядке, в каком они расположены в таблице на рис. 2.12. Максимальное количество очков получим в переменной $MaxB$. Кроме того, найдем еще и номер команды, занявшей первое место. Для этого будем использовать переменную $Nmax$.

Рассмотрим алгоритм решения задачи. Алгоритм будет составлен исходя из упрощающего предположения о том, что максимальное количество баллов набрала только одна команда. Именно ее номер и

будет выведен в качестве результата. Более общий вариант предлагается для рассмотрения в задании к данному параграфу. Ниже приведен полный алгоритм на Алгоритмическом языке, а на рис. 2.13 — фрагмент блок-схемы, относящийся только к выбору максимального элемента (без ввода и вывода).

```
алг Премьер-лига
цел таб B[1:16]
цел I, MaxB, Nmax
нач
    {Цикл ввода}
    для I от 1 до 16 повторять
    нц
        Вывод "B[", I , "]="
        Ввод B[I]
    кц
    {Поиск наибольшего значения и его номера}
    MaxB:=B[1]; Nmax:=1
    для I от 2 до 16 повторять
    нц
        если B[I]>MaxB
           то MaxB:=B[I];
             Nmax:=I
        кв
    кц
    Вывод  "Максимальное число очков:", MaxB
    Вывод "Номер команды-победителя:", Nmax
кон
```

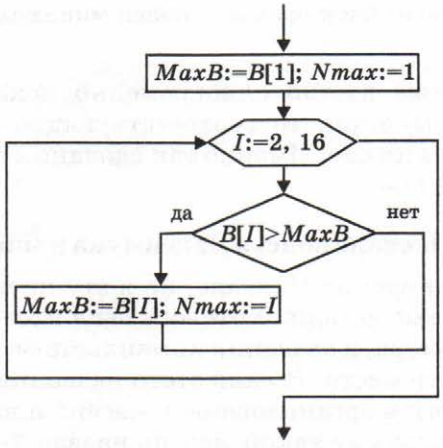


Рис. 2.13. Фрагмент блок-схемы — поиск максимального элемента

Идея алгоритма состоит в следующем. В переменную *MaxB* заносится значение первого элемента массива, в переменную *Nmax* — единица, т. е. номер первого элемента. Затем в цикле последовательно с *MaxB* сравниваются все остальные элементы массива *B*. Если очередной элемент оказывается больше текущего значения *MaxB*, то его значение заносится в *MaxB*, а его номер — в *Nmax*. Когда закончится цикл, в *MaxB* останется наибольшее число из всего массива, а в *Nmax* — его номер.

Теперь нетрудно догадаться, как искать минимальное значение в массиве и его номер. Если для этого использовать в программе переменные *MinB* и *Nmin* и в условии ветвления заменить знак отношения «больше» на «меньше», то получим нужный алгоритм. Его фрагмент показан блок-схемой на рис. 2.14.

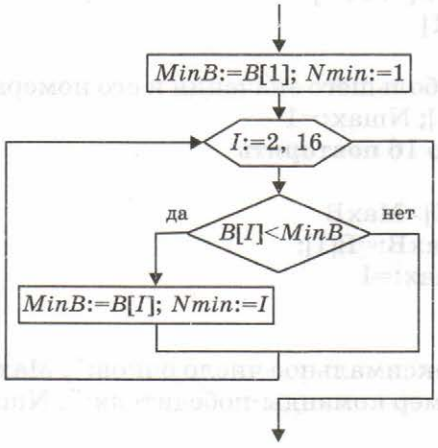


Рис. 2.14. Фрагмент блок-схемы — поиск минимального элемента

Если в алгоритме нужно одновременно искать максимальное и минимальное значения, то соответствующие ветвления можно объединить в одном цикле. Именно так сделано в приведенной ниже программе на Паскале.

Программа на Паскале поиска максимума и минимума в массиве

Составим программу на Паскале, но в эту программу мы внесем еще некоторые новые детали. Хотелось бы в итоге работы программы получить не номера, а названия команды-победителя и команды, занявшей последнее место. Но для этого названия всех команд чемпионата должны быть организованы в массив и введены как исходные данные. В программе такой массив назван `Team[1..16]` и тип его элементов объявлен как `string`.

String — *это строковый тип данных Паскаля*. Величина такого типа может принимать значение, представляющее собой произвольную символьную последовательность (в том числе и из русских букв), длина которой не должна превышать 255. Для названий команд это вполне подходящие условия.

```
Program Premier_liga;
var B: array[1..16] of integer;
    Team: array[1..16] of string;
    MaxB, MinB, Nmax, Nmin, I: integer;
begin
{Ввод названий команд и набранных ими очков}
writeln('Введите названия команд и полученные
        ими очки');
for I:=1 to 16 do
  begin
      write(I,' Название: '); Readln(Team[I]);
      write('Очки: '); Readln(B[I])
  end;
{Поиск наибольшего и наименьшего значений и их номеров}
MaxB:=B[1]; Nmax:=1; MinB:=B[1]; Nmin:=1;
for I:=2 to 16 do
begin
  {Выбор максимума}
  if B[I]>MaxB then
  begin
      MaxB:=B[I];
      Nmax:=I
  end;
  {Выбор минимума}
  if B[I]<MinB then
  begin
      MinB:=B[I];
      Nmin:=I
  end
end;
{Вывод результатов}
writeln('Команда-победитель чемпионата ',
        Team[Nmax], 'набрала ', MaxB, 'очков');
writeln('На последнем месте',Team[Nmin],
        ' с ',MinB,'очками')
end.
```

Обратите внимание на то, как определяется название команды-победителя и команды, занявшей последнее место. Это делается по

значениям индексов максимального и минимального элементов массива B: Nmax и Nmin. В переменной Team[Nmax] находится название чемпиона, а в переменной Team[Nmin] — название последней команды в чемпионате.

При выполнении программы на экране будет отражено следующее:

```
Введите названия команд и полученные ими очки
1 Название: ДИНАМО
Очки:  46
2 Название: ЗЕНИТ
Очки: 56
3 Название: КРЫЛЬЯ СОВЕТОВ
Очки:  42
......
16 Название : ШИННИК
Очки: 47
Команда-победитель чемпионата ЦСКА набрала 59 очков
На последнем месте ЧЕРНОМОРЕЦ с 24 очками
```

Коротко о главном

Алгоритм выбора максимального (минимального) значения в массиве имеет структуру цикла с вложенным неполным ветвлением.

Для обработки последовательностей символов в Паскале имеется строковый тип данных: string.

Вопросы и задания

1. Придумайте собственные примеры данных, которые можно было бы представить в виде строкового массива. Подготовьте сообщение.
2. Представьте себе, что две команды набрали по 59 очков. Например, ЦСКА и ЗЕНИТ. Номер какой команды был бы выведен в качестве результата?
3. При условии из предыдущего задания определите, какие будут выведены результаты, если в операторе ветвления, где отбирается максимальное значение, заменить знак отношения «>» на «>=»?
4. Введите в компьютер программу Premier_liga. Выполните ее, получите результаты. Сравните с результатами, приведенными в параграфе.
5. По условиям чемпионата 2003 года из премьер-лиги выбывают две последние в турнирной таблице команды. Составьте программу, определяющую обе команды, выбывающие из премьер-лиги.

www ЕК ЦОР: часть 2, заключение, § 6.1. ЦОР № 6, 7.

§ 21

Сортировка массива

Основные темы параграфа:

- ***алгоритм сортировки методом пузырька;***
- ***программа на Паскале сортировки методом пузырька.***

Известно, что данные в электронной таблице можно сортировать по возрастанию или убыванию значений в столбцах. Для задачи с таблицей футбольного чемпионата естественным действием была бы сортировка по убыванию значений набранных очков. Тогда вверху таблицы останется победитель чемпионата, а в нижней строчке — команда, занявшая последнее место. На рисунке 2.15 показана такая отсортированная таблица. Из нее мы получаем исчерпывающую информацию об итогах чемпионата: кто какое место занял.

	A	B
1	ЦСКА	59
2	ЗЕНИТ	56
3	РУБИН	53
4	ЛОКОМОТИВ	52
5	ШИННИК	47
6	ДИНАМО	46
7	САТУРН	45
8	ТОРПЕДО	43
9	КРЫЛЬЯ СОВЕТОВ	42
10	СПАРТАК	36
11	РОСТОВ	34
12	РОТОР	32
13	СПАРТАК-АЛАНИЯ	31
14	ТОРПЕДО-МЕТАЛЛУРГ	29
15	УРАЛАН	28
16	ЧЕРНОМОРЕЦ	24

Рис. 2.15. Итоги чемпионата

Алгоритм сортировки методом пузырька

Рассмотрим, как программируется сортировка массива. Для решения этой задачи существует целый класс алгоритмов. Мы рассмотрим здесь только один из них, известный под названием «метод пузырька». Откуда такое название, станет ясно немного позже.

Проиллюстрируем идею метода пузырька на маленьком массиве из пяти чисел. Пусть это будет массив $B[1:5]$, исходные значения в котором распределены случайным образом (рис. 2.16). Требуется отсортировать числа по убыванию.

Первый этап (первый проход). Последовательно сравниваются пары соседних чисел и упорядочиваются по убыванию. Сначала сравниваются $B[1]$ и $B[2]$. Поскольку на втором месте должно стоять меньшее число, то числа меняются местами. $B[1]$ становится равным 3, $B[2]$ — равным 1. Затем упорядочиваются $B[2]$ и $B[3]$. Их значения тоже переставляются: $B[2]=2$, $B[3]=1$. Затем упорядочиваются $B[3]$ и $B[4]$. И наконец, $B[4]$ и $B[5]$. В результате первого прохода минимальное число попадает на свое место: $B[5]=1$.

Алгоритм первого прохода можно описать так:

```
для I от 1 до 4 повторять
нц
    если B[I]<B[I+1] то
        X:=B[I]; B[I]:=B[I+1]; B[I+1]:=X
    кв
кц
```

Обмен значениями $B[I]$ и $B[I+1]$ происходит через третью переменную X.

Вот теперь можно понять смысл образа пузырька! В результате прохода минимальное значение «всплывает» в конец массива, как воздушный пузырек в воде всплывает на поверхность, подталкиваемый архимедовой силой.

Далее нужно повторять такие проходы еще три раза. После второго прохода на своем месте окажется $B[4]$, после третьего прохода — $B[3]$. После четвертого прохода упорядочатся $B[2]$ и $B[1]$. Нетрудно понять, что при втором проходе не надо трогать $B[5]$, т. е. цикл должен повторяться для I от 1 до 3. При третьем проходе — от 1 до 2. И наконец, на четвертом проходе — от 1 до 1, т. е. всего 1 раз.

Следовательно, циклы, реализующие проходы, сами должны циклически повторяться. Причем при каждом следующем повторении длина цикла должна уменьшаться на единицу. Отсюда вывод: *структура алгоритма должна представлять собой два вложенных цикла.*

	B[1]	*B*[2]	*B*[3]	*B*[4]	*B*[5]
Исходные значения	1	3	2	4	5
1-й проход	3	2	4	5	1
2-й проход	3	4	5	2	1
3-й проход	4	5	3	2	1
4-й проход	5	4	3	2	1

Рис. 2.16. Иллюстрация метода пузырька

Вот полный алгоритм сортировки массива *B*[1:16]:

```
алг Сортировка методом пузырька
цел таб B[1:16]
цел I, K, X
нач
    {Цикл ввода}
    для I от 1 до 16 повторять
    нц
        Вывод "B[", I , "]="
        Ввод B[I]
    кц
    {Циклы сортировки}
    для K от 1 до 15 повторять
    нц
        для I от 1 до 16-K повторять
        нц
          если B[I]<B[I+1]
            то X:=B[I]; B[I]:=B[I+1]; B[I+1]:=X
        кц
    кц
    {Вывод отсортированного массива}
    для I от 1 до 16 повторять
    нц
        Вывод "B[", I , "]=", B[I]
    кц
кон
```

Здесь переменная *K* играет роль номера прохода. Для массива длиной 16 такие проходы требуется повторить 15 раз. Длина каждого *K*-го прохода равна 16–*K*.

Программа на Паскале сортировки методом пузырька

Теперь запишем программу на Паскале. Но мы ее немного усложним по сравнению с построенным алгоритмом. По условию исходной задачи нам нужно получить список команд в порядке занятых ими мест и число очков, полученных каждой командой. Следовательно, сортировать нужно не только массив B, но и массив Team. Делается это очень просто: в массиве Team параллельно с массивом B производятся те же самые перестановки. В конце работы программы на экран выводятся одновременно элементы обоих отсортированных массивов.

```
Program Premier_liga_2;
var B: array[1..16] of integer;
    Team: array[1..16] of string;
    I, K, X: integer;
    St: string;
begin
   {Ввод названий команд и набранных ими очков}
   writeln('Введите названия команд и полученные
            ими очки');
    for I:=1 to 16 do
    begin
       write(I,' Название: '); Readln(Team[I]);
       write('Очки: '); Readln(B[I])
    end;
    {Сортировка}
    for K:=1 to 15 do
    for I:=1 to 16-K do
    if (B[I] < B[I+1]) then
      begin
           X:=B[I]; B[I]:=B[I+1]; B[I+1]:=X;
           St:=Team[I]; Team[I]:=Team[I+1];
           Team[I+1]:=St
      end;
    {Вывод отсортированной таблицы}
    for I:=1 to 16 do
    begin
       {Присоединение пробелов к названиям команд}
       for K:=1 to 18-length(Team[I]) do
           Team[I]:=Team[I]+' ';
       {Вывод: место, команда, очки}
       writeln(I:2,' ',Team[I]:18,B[I]:2)
    end
end.
```

Поясним новые средства Паскаля, которые применены в этой программе. Обмен значениями между элементами строкового массива Team должен происходить через переменную строкового типа. Для этого в программе используется переменная St.

Вывод результатов на экран организован так, чтобы на экране номера мест, занятых командами, названия команд и набранные очки выводились в три ровных столбца. Названия разных команд имеют разную длину. Самое длинное название у команды ТОРПЕДО-МЕТАЛЛУРГ состоит из 17 символов. Для выравнивания длин строк каждое название дополняется пробелами до 18 символов. Число добавляемых пробелов вычисляется так:

```
18-length(Team[I])
```

Здесь length( ) — это стандартная функция, вычисляющая длину строки (число символов), указанной в скобках. Например, для ЦСКА длина строки равна 4, а для ТОРПЕДО-МЕТАЛЛУРГ длина равна 17. Значит, к ЦСКА добавится 14 пробелов, а к ТОРПЕДО-МЕТАЛЛУРГ — 1 пробел.

В операторе Team[I]:= Team[I]+' '; используется операция «+» присоединения символов. В данном случае присоединяется пробел. К строке Team[I] добавится столько пробелов, сколько раз повторится присоединение. После этого по команде

```
Writeln(I:2,' ',Team[I]:18, B[I]:2)
```

в ровные колонки выведутся места, названия команд и очки. Результаты будут иметь на экране следующий вид:

```
 1 ЦСКА                59
 2 ЗЕНИТ               56
 3 РУБИН               53
. . . . . . . . . . . . .
14 ТОРПЕДО-МЕТАЛЛУРГ   29
15 УРАЛАН              28
16 ЧЕРНОМОРЕЦ          24
```

Коротко о главном

Метод пузырька — алгоритм сортировки числового массива.

Структура алгоритма метода пузырька — два вложенных цикла с переменной длиной внутреннего цикла.

length() — функция определения длины строковой переменной.

В Паскале существует операция присоединения строк. Ее знак — «+».

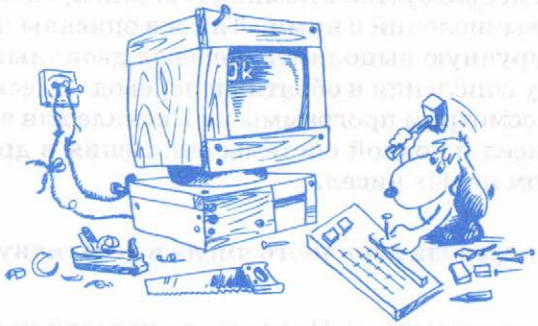
Вопросы и задания

1. Как пояснить название метода сортировки массива — «метод пузырька»?
2. Сколько проходов с перестановками элементов потребуется при сортировке массива из 100 чисел?
3. Введите в компьютер программу `Premier_liga_2`.
   а) Выполните ее, получите результаты. Сравните с результатами, приведенными в параграфе.
   б) Внесите изменения в программу для того, чтобы получить список в обратном порядке (по возрастанию очков). Выполните программу.
   в) Возможно, что массив окажется отсортированным до завершения всех проходов. В таком случае число повторений внешнего цикла можно сократить, и программа будет выполняться быстрее. Попробуйте усовершенствовать приведенную программу с учетом этого факта. Проверьте результат на тестах.
4. Если несколько команд набрали одинаковое количество очков, то места между ними распределяются по разнице забитых и пропущенных мячей: чем разница больше, тем место выше. Попробуйте усовершенствовать программу, учитывая это правило. Для этого в программу надо добавить массив с разницами мячей. Придумайте тест, на котором можно проверить работу программы.
5. Условие то же, что и в предыдущем задании. Но в качестве исходных данных вводится еще два массива: с числом забитых и пропущенных мячей каждой командой.

www ЕК ЦОР: часть 2, заключение, § 6.2. ЦОР № 6.

Чему вы должны научиться, изучив главу II

- Строить несложные вычислительные алгоритмы с использованием блок-схем и Алгоритмического языка.
- Выполнять трассировку алгоритмов.
- Составлять программу на Паскале по данному алгоритму.
- Работать с системой программирования на Паскале: набирать текст программы; сохранять программу на диске и вызывать ее с диска; компилировать и исполнять программу; исправлять ошибки в программе.



Дополнение к главе II

2.1. Программирование перевода чисел из одной системы счисления в другую

Основные темы параграфа:

- *перевод двоичного числа в десятичную систему счисления;*
- *перевод десятичного числа в двоичную систему счисления.*

В § 18 учебника для 8 класса рассказано о двоичной системе счисления, которая используется в компьютере для представления чисел и выполнения вычислений с ними. Там же описаны правила, по которым можно вручную выполнить перевод двоичных чисел в десятичную систему счисления и обратный перевод — десятичных чисел в двоичные. Рассмотрим программы на Паскале для автоматического перевода чисел из одной системы счисления в другую. Ограничимся переводом целых чисел.

Программа перевода двоичного числа в десятичную систему счисления

Рассмотрим программу на Паскале, по которой происходит перевод целого двоичного числа в десятичную систему.

```
Program Numbers_2_10;
var N10,N2,k: longint;
begin
  write('N2='); readln(N2);
                  {ввод исходного двоичного числа}
  k:=1; N10:=0;
  while (N2<>0) do
          {цикл выполняется, пока N2 не равно нулю}
    begin
    N10:=N10 + (N2 mod 10)*k;
               {суммирование развернутой формы}
```

```
        k:=k*2;
           {вычисление степеней двойки: 2, 2^2, 2^3 ...}
        N2:=N2 div 10
              {отбрасывание младшей цифры}
    end;
    writeln('N10=', N10) {вывод десятичного числа}
end.
```

В программе использованы следующие переменные:
N2 — целое двоичное число — исходное данное;
N10 — десятичное число — результат;

Тип `longint` — длинный целый тип. Значения величин этого типа лежат в диапазоне от –2 147 483 648 до 2 147 483 647. Значит, данная программа может работать с числами, не более чем 9-значными.

В этой программе используются две незнакомые вам операции с целыми числами. Операция **div** — целочисленное деление. Делимое и делитель являются целыми числами, а результат — целая часть частного. Например: `7 div 2 = 3`. Для отбрасывания младшего разряда целого числа используется целочисленное деление на 10. Например: `1234 div 10 = 123` — отбрасывается младший разряд.

Операция **mod** дает остаток от целочисленного деления. Например: `7 mod 2 = 1`. Для получения младшего разряда целого числа вычисляется остаток от целочисленного деления на 10. Например: `1234 mod 10 = 4` — выделяется разряд единиц.

Пример. При переводе по данной программе двоичного числа 1101_2 в десятичную систему на экране увидим:

```
N2=1101
N10=13
```

Следовательно, в итоге получили: $1101_2=13$.

Для лучшего понимания работы программы внимательно изучите приведенную далее трассировочную таблицу. Она отражает изменения значений переменных на каждом шаге выполнения алгоритма, реализованного в программе.

Шаг алгоритма	Команды алгоритма	N2	k	N10	Проверка условия
1	Ввод N2, k:=1, N10:=0	1101	1	0	
2	N2<>0				1101≠0, да
3	N10:=N10 + (N2 mod 10)*k			1	
4	k:=k*2		2		
5	N2:=N2 div 10	110			
6	N2<>0				110≠0, да
7	N10:=N10 + (N2 mod 10)*k			1	
8	k:=k*2		4		
9	N2:=N2 div 10	11			
10	N2<>0				11≠0, да
11	N10:=N10 + (N2 mod 10)*k			5	
12	k:=k*2		8		
13	N2:=N2 div 10	1			
14	N2<>0				1≠0, да
15	N10:=N10 + (N2 mod 10)*k			13	
16	k:=k*2		16		
17	N2:=N2 div 10	0			
18	N2<>0				0≠0, нет
19	Вывод N10			13	

Программа перевода десятичного числа в двоичную систему счисления

Теперь познакомьтесь с программой перевода целого десятичного числа в двоичную систему счисления.

```
Program Numbers_10_2;
var N10, N2, k: longint;
begin
  write('N10='); readln(N10);
       {Ввод исходного десятичного числа}
  k:=1; N2:=0;
  repeat
    N2:=N2 + (N10 mod 2)*k;
       {Суммирование развернутой формы}
    k:=k*10;  {Вычисление базиса: 10, 100, 1000, …}
    N10:=N10 div 2 {Целочисленное деление на 2}
  until (N10=0);
         {Цикл заканчивает выполнение при N10=0}
  writeln('N2=', N2)       {Вывод двоичного числа}
end.
```

Здесь использованы те же обозначения, что и в предыдущей программе. Исходными данными являются: `N10` — десятичное число. Результат получается в переменной `N2` — число в системе с основанием 2.

В алгоритме используется цикл с постусловием (**repeat ... until**). Цикл повторяется до выполнения условия: `N10 = 0`.

Пример использования программы. Переведем число 25 в двоичную систему счисления. Работа программы на экране компьютера отразится следующим образом:

```
N10=25
N2=11001
```

Следовательно, в результате получили: $25 = 11001_2$.

Для лучшего понимания работы программы рекомендуем построить трассировочную таблицу, наподобие предыдущей.

Коротко о главном

Программирование перевода $10 \to 2$ и $2 \to 10$ основано на использовании операций над целыми числами: **div** — целочисленное деление, **mod** — остаток от целочисленного деления.

Вопросы и задания

1. Введите в компьютер и отладьте программу `Numbers_2_10`. Переведите с ее помощью в десятичную систему счисления следующие двоичные числа: 111110; 1111111; 100000. Проверьте правильность результатов.
2. Введите в компьютер и отладьте программу `Numbers_10_2`. Переведите с ее помощью в двоичную систему счисления следующие десятичные числа: 255; 512; 1023. Проверьте правильность результатов.

2.2. Сложность алгоритмов

Основные темы параграфа:

- *объемная сложность алгоритма;*
- *временна́я сложность алгоритма;*
- *алгоритмы перебора;*
- *сложность алгоритмов перебора.*

Традиционно принято оценивать степень сложности алгоритма по объему используемых им основных ресурсов компьютера: процессорного времени и оперативной памяти. В связи с этим вводятся такие понятия, как временна́я сложность и объемная сложность алгоритма.

Объемная сложность связана с количеством данных, которые при обработке нужно хранить в оперативной памяти. Проблемы могут возникнуть при обработке больших массивов данных (числовых или символьных). Если весь объем обрабатываемой информации не помещается одновременно в оперативную память, то эти данные приходится хранить на устройствах внешней памяти (дисках) и в процессе обработки перемещать частями из внешней памяти в оперативную память и обратно. Поскольку время чтения и записи данных на устройствах внешней памяти намного больше времени обмена процессора с оперативной памятью, то в целом время выполнения программы существенно возрастает.

Временна́я сложность связана с количеством операций, выполняемых процессором в течение работы программы. Наибольшая часть процессорного времени тратится на выполнение циклов. Поэтому оценка временно́й сложности производится по количеству повторений циклов. Нетрудно понять, что при обработке массива данных количество повторений циклов связано с размером массива. Например, пусть вычисляется сумма элементов массива X, состоящего из N чисел:

```
S:=0;   for i:=1 to N do S:=S + X[i];
```

В теле цикла выполняется две операции: сложение и присваивание. Число повторений цикла равно N. Следовательно, суммарное

число выполняемых операций равно $N \cdot 2$. Значит, время выполнения всего цикла будет пропорционально $N \cdot 2$: $T \sim N \cdot 2$. В таком случае говорят, что *временна́я сложность алгоритма зависит линейно от объема данных*. Во сколько раз возрастет N, во столько же раз возрастет время выполнения программы.

Если вернуться к алгоритму поиска наибольшего и наименьшего значений массива, то в нем также имеется один цикл, хотя тело цикла содержит большее число операций. Но с увеличением размера массива (N) время выполнения программы будет также увеличиваться линейно, т. е. пропорционально N. Следовательно, временна́я сложность алгоритмов суммирования массива и поиска в массиве максимального (минимального) элемента одинаковая — линейная.

Теперь оценим временну́ю сложность алгоритма сортировки массива методом пузырька. По-прежнему обозначим через N размер массива. Алгоритм содержит два вложенных цикла. Внешний цикл имеет длину $N - 1$. Внутренний цикл с каждым повторением изменяет свою длину по убыванию: $N - 1$, $N - 2$, $N - 3$, ..., 2, 1. Суммарное число повторений цикла можно посчитать так:

- вычислим среднюю длину внутреннего цикла: $[(N - 1) + 1]/2 = N/2$;
- умножим эту величину на число повторений внешнего цикла: $(N - 1) \cdot N/2 = (N^2 - N)/2$.

Временна́я сложность алгоритма определяется слагаемым с наибольшей степенью: $T \sim N^2$. В таком случае говорят, что *временна́я сложность алгоритма сортировки методом пузырька имеет второй порядок по объему данных*, т. е. пропорциональна квадрату N. Например, если размер массива увеличить в 10 раз, то время сортировки возрастет в 100 раз.

Сложность алгоритмов перебора

Большим классом задач, которые часто решаются на компьютере, являются задачи поиска. Одну из таких задач мы уже рассматривали: поиск максимального элемента в массиве. Общий смысл задач поиска сводится к следующему: из множества данных, хранящихся в памяти компьютера, требуется выбрать нужные данные, удовлетворяющие определенным условиям (критериям). Такой поиск осуществляется перебором всех элементов структуры данных и их проверкой на удовлетворение условию поиска. Перебор, при котором просматриваются все элементы структуры, называется *полным перебором*.

На примере алгоритма поиска максимального элемента мы знаем, что полный перебор элементов одномерного массива производится с помощью одного цикла, число повторений которого пропорционально размеру массива.

Рассмотрим другой пример. В одномерном массиве X заданы координаты N точек, произвольно расположенных на вещественной числовой оси. Точки пронумерованы. Их номера соответствуют номерам элементов в массиве X. Нужно определить пару точек, расстояние между которыми наибольшее.

Применяя метод перебора, эту задачу можно решать так: перебрать все пары точек из N данных и определить номера тех точек, расстояние между которыми наибольшее (наибольший модуль разности координат). Такой полный перебор реализуется через два вложенных цикла:

```
Number1:=1;
Number2:=2;
for i:=1 to N do
  for j:=1 to N do
    if Abs(X[i] - X[j]) > Abs(X[Number1] - X[Number2])
    then
    begin
      Number1:=i;
      Number2:=j
    end;
```

Но очевидно, что такое решение задачи нерационально. Здесь *каждая пара точек будет просматриваться дважды*, например при $i = 1$, $j = 2$ и $i = 2$, $j = 1$. Для случая $N = 100$ циклы повторят выполнение $100 \cdot 100 = 10\,000$ раз.

Выполнение программы ускорится, если исключить повторения. Исключить также следует и совпадения значений i и j. Для исключения повторений нужно в предыдущей программе изменить начало внутреннего цикла с 1 на $i + 1$. Программа примет вид:

```
Number1:=1;
Number2:=2;
for i:=1 to N do
  for j:=i + 1 to N do
    if Abs(X[i] - X[j]) > Abs(X[Number1] - X[Number2])
    then
    begin
      Number1:=i;
      Number2:=j
    end;
```

В таком алгоритме число повторений цикла будет равно $N(N-1)/2$. При $N = 100$ получается 4950. Рассмотренный вариант алгоритма назовем **перебором без повторений.**

Замечание. Конечно, эту задачу можно было решить и другим способом, но в данном случае нас интересовал именно переборный алгоритм. В случае же точек, расположенных не на прямой, а на плоскости или в пространстве, поиск альтернативы переборному алгоритму становится весьма проблематичным.

В следующей задаче требуется выбрать из массива X без повторений все тройки чисел, сумма которых равна десяти. В этом случае алгоритм будет строиться из трех вложенных циклов. Внутренние циклы имеют переменную длину.

```
for i := 1 to N do
  for j := i + 1 to N do
    for k := j + 1 to N do
      if X[i] + X[j] + X[k] = 10
      then writeln(X[i], X[j], X[k]);
```

А теперь представьте, что из массива X требуется выбрать всевозможные группы чисел, сумма которых равна десяти. Количество чисел в группах может быть любым — от 1 до N. В этом случае количество вариантов перебора резко возрастает, а сам алгоритм становится нетривиальным.

Казалось бы, ну и что? Компьютер работает быстро! И все же посчитаем. Число различных групп из N объектов (включая пустую) составляет 2^N. При $N = 100$ это будет $2^{100} \approx 10^{30}$. Компьютер, работающий со скоростью миллиард операций в секунду, будет осуществлять такой перебор приблизительно 10 лет. Даже исключение перестановочных повторений не сделает такой переборный алгоритм практически осуществимым.

Путь практической разрешимости подобных задач состоит в нахождении способов исключения из перебора бесперспективных с точки зрения условия задачи вариантов. Для некоторых задач это удается сделать. К таким задачам относится задача поиска выхода из лабиринта, задача о восьми ферзях (расставить на шахматной доске восемь ферзей так, чтобы они не угрожали друг другу). Если бы шахматные программы составлялись методом полного перебора всевозможных ходов, то ни один суперкомпьютер не смог бы в реальном времени играть в шахматы. Очевидно, что в алгоритме программы заложены знания стратегии и тактики игры в шахматы, которыми владеют сильнейшие шахматисты, что существенно сокращает перебор возможных ходов.

Впечатляющим примером решения фундаментальной математической проблемы методом поиска является Проект GIMPS (Great Internet Mersenne Prime Search), направленный на поиск простых чисел Мерсенна — последовательности чисел, подчиняющихся за-

кону 2^p-1, где p — простое число. В ноябре 2001 года в рамках данного проекта было найдено число Мерсенна, содержащее в своей десятичной записи более 4 млн цифр. Десятки тысяч компьютеров по всему миру, отдавая часть своих вычислительных ресурсов, работали над этой задачей два с половиной года.

Коротко о главном

В программировании используются два критерия сложности алгоритма: объемная сложность и временна́я сложность.

Объемная сложность связана с количеством данных, которые при обработке нужно хранить в оперативной памяти.

Временна́я сложность связана с количеством операций, выполняемых процессором в течение работы программы; количество операций пропорционально времени выполнения программы.

Временна́я сложность оценивается как функция зависимости числа операций от объема данных и может быть линейной, квадратичной и пр.

Задача перебора: из множества данных, хранящихся в памяти компьютера, требуется выбрать нужные данные, удовлетворяющие определенным условиям (критериям).

Временна́я сложность полного перебора может привести к превышению разумного времени выполнения программы на компьютере.

Оптимизация алгоритма перебора состоит в исключении анализа бесперспективных вариантов.

Вопросы и задания

1. Почему временна́я сложность алгоритма зависит от его объемной сложности?
2. Составьте алгоритм поиска для следующей задачи: на координатной плоскости заданы своими координатами N точек. Найти две самые удаленные друг от друга точки. Оцените временну́ю сложность алгоритма. Рассмотрите два варианта алгоритма: с полным и с неполным перебором и сравните их.
3. Составьте алгоритм для решения задачи, аналогичной предыдущей, с учетом того что точки расположены в трехмерном пространстве.

2.3. О языках программирования и трансляторах

Основные темы параграфа:

- *системы программирования;*
- *уровни языков программирования;*
- *трансляция и трансляторы;*
- *о двух способах трансляции;*
- *работа компилятора;*
- *работа интерпретатора.*

Системы программирования

«Родным» языком ЭВМ является язык машинных команд (ЯМК). Самые первые ламповые ЭВМ понимали только этот язык. В программах на ЯМК данные обозначаются их адресами в памяти машины, выполняемые операции — числовыми кодами. Программист сам должен заботиться о расположении в памяти ЭВМ команд программы и данных.

Современные программисты так не работают. Для программирования на современных компьютерах применяются **системы программирования**. В учебнике 7 класса (глава 2) говорилось о том, что программное обеспечение компьютера делится на три части:

- системное ПО;
- прикладное ПО;
- системы программирования.

С первыми двумя видами программного обеспечения вы уже знакомы. Системное ПО — это прежде всего операционные системы, сервисные программы. Прикладное ПО — это многочисленные редакторы, электронные таблицы, информационные системы, математические пакеты, экспертные системы и многое другое, с чем работает абсолютное большинство пользователей.

Системы программирования предназначены для создания компьютерных программ.

Системы программирования (СП) позволяют разрабатывать и исполнять на компьютере программы, написанные на языках более высокого уровня, чем язык машинных команд.

Уровни языков программирования

Что понимается под уровнем языка программирования? Понятие уровня языка связано со степенью его удаленности от языка процессора компьютера и приближенности к естественному человеческому языку, к формальному языку предметной области (чаще всего — математики). *Чем выше уровень, тем дальше язык от компьютера и ближе к человеку.* Этот принцип схематически отражен на рис. 2.17.

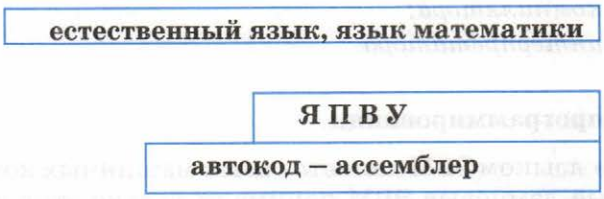


Рис. 2.17. Уровни языков программирования

Язык машинных команд — это язык самого низкого уровня. Первые языки программирования, отличные от ЯМК, появились на машинах первого поколения, и назывались они ***автокодами***.

Автокод — это машинно-ориентированный язык символического программирования.

Одна команда на автокоде соответствует одной машинной команде. Работая на автокоде, программист освобожден от необходимости распределять память под программу и величины; ему не приходится работать с адресами ячеек. Переменные величины и числовые константы обозначаются так же, как в математике, коды операций — мнемонически буквами.

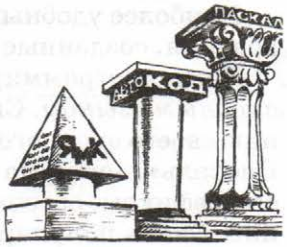
Начиная с машин третьего поколения, языки такого типа стали называть ассемблерами. В наше время на ассемблере программируют довольно редко. Это, как правило, делают системные программисты.

Аббревиатура ЯПВУ расшифровывается так: языки программирования высокого уровня. Сегодня большинство программистов работают именно на этих языках. Примеры языков высокого уровня: Паскаль, Бейсик, Си (С), Фортран.

Вот пример записи одной и той же команды сложения двух чисел на трех языках разного уровня: ЯМК, автокоде и Паскале:

```
C:=A+B          Паскаль
ADD A, B, C     автокод
01 24 28 2C     ЯМК
```

Видно, как с повышением уровня языка повышается «понятность» команды (по-английски слово «ADD» означает «сложить»). Однако чем понятнее язык для человека, тем он непонятнее для процессора компьютера. Процессор понимает только ЯМК. Человеку же легче писать программы на языках более высокого уровня. Как же быть?



Трансляция и трансляторы

Как сделать так, чтобы человек мог писать программы на автокоде или Паскале, а компьютер мог исполнять эти программы? Ответ на поставленный вопрос такой же, как на вопрос «Как мне общаться с японцем, если я не знаю японского языка?». Нужен переводчик! По-английски «переводчик» — «translator».

Программы-переводчики с автокода, Паскаля, Фортрана и других языков на язык машинных команд называются **трансляторами**.

Таким образом, компьютер сам производит перевод под управлением программы-транслятора. Процесс перевода программы на язык машинных команд называется ***трансляцией***. Прежде чем выполнить, например, программу на Паскале, ее нужно *оттранслировать*. Трансляцию можно представить как спуск с верхней ступеньки языка на самую первую ступеньку — ЯМК (рис. 2.18).

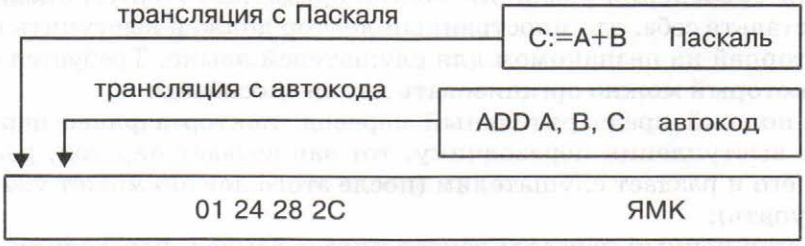


Рис. 2.18. Трансляция с автокода и Паскаля на ЯМК

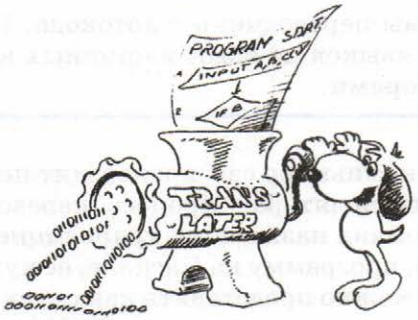
Транслятор является обязательным элементом любой системы программирования. Первые СП включали в себя только транслятор. Затем к транслятору стали добавляться различные сервисные средства: текстовые редакторы, отладчики, системы обслуживания программных библиотек, средства организации дружественного интерфейса с пользователем.

Наиболее удобными для пользователя стали системы программирования, созданные на персональных компьютерах.

Язык программирования, с которым работает СП, называется ее *входным языком*. Системы программирования именуются по названию своего входного языка. Например: «система Бейсик», «система Паскаль», «система Фортран». Иногда в название систем включаются префиксы, обозначающие, например, ее фирменное происхождение. Очень популярны системы с приставкой «Турбо»: Турбо Паскаль, Турбо С и др. Это системы программирования, разработанные фирмой Borland.

О двух способах трансляции

Реализовать тот или иной язык программирования на компьютере — это значит создать транслятор с этого языка для данного компьютера. Существуют два принципиально различных метода трансляции. Они называются **компиляция** и **интерпретация**.



Для объяснения различия можно предложить такую аналогию: представьте себе, что иностранный лектор должен выступить перед аудиторией на незнакомом для слушателей языке. Требуется перевод, который можно организовать двумя способами:

1) полный предварительный перевод: лектор заранее передает текст выступления переводчику, тот записывает перевод, размножает его и раздает слушателям (после этого лектор может уже и не выступать);

2) синхронный перевод: лектор читает доклад, переводчик одновременно с ним, слово за словом, переводит выступление.

Компиляция является аналогом полного предварительного перевода; интерпретация — аналог синхронного перевода. Транслятор, работающий по принципу компиляции, называется **компилятором.** Транслятор, работающий методом интерпретации, называется **интерпретатором.**

Работа компилятора

При компиляции в память компьютера загружается программа-компилятор. Она воспринимает текст программы на ЯПВУ как исходную информацию. Компилятор производит синтаксический контроль программы и при обнаружении ошибок выводит диагностические сообщения. Если ошибок нет, то результатом компиляции является программа на языке машинных команд.

Затем компилятор удаляется из оперативной памяти. В памяти остается только программа на ЯМК, которая выполняется для получения результатов.

На рисунке 2.19 схематически показан процесс выполнения программы на ЯПВУ с использованием компиляции. Прямоугольниками изображены программы в машинных кодах, овалами — обрабатываемая и конечная информация.

Рис. 2.19. Выполнение программы на ЯПВУ с использованием компилятора

Конечно, компиляция с автокода (ассемблера) много проще, чем с языков высокого уровня. Для этой процедуры часто применяют специальный термин — *ассемблирование.* А под словом «ассемблер» понимается не только язык программирования, но и транслятор с него.

Работа интерпретатора

Интерпретатор в течение всего времени работы программы находится во внутренней памяти (иногда для этого используется ПЗУ). В ОЗУ помещается программа на ЯПВУ. Интерпретатор «читает» ее первый оператор, переводит его в машинные команды и тут же организует выполнение этих команд. Затем переходит к переводу и выполнению следующего оператора, и так до конца программы. При

этом результаты предыдущих переводов в памяти не сохраняются. При повторном выполнении одного и того же оператора в цикле он снова будет транслироваться. Перед трансляцией каждого оператора происходит его синтаксический анализ.

На рисунке 2.20 схематически показан процесс выполнения программы на ЯПВУ с использованием интерпретатора.

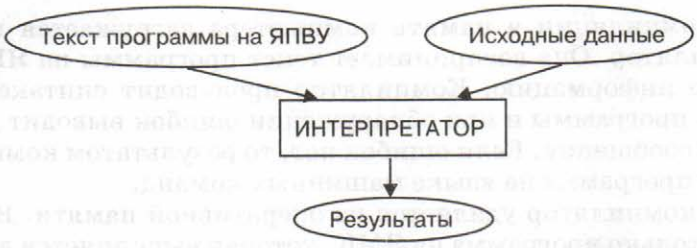


Рис. 2.20. Выполнение программы на ЯПВУ с использованием интерпретатора

Таким образом, при компиляции трансляция и исполнение программы идут последовательно друг за другом. При интерпретации — параллельно.

Один раз откомпилированная программа может быть сохранена во внешней памяти и затем многократно выполнена. На компиляцию машинное время тратиться больше не будет. Программа на интерпретируемом языке при каждом выполнении подвергается повторной трансляции. Кроме того, интерпретатор может занимать значительное место в оперативной памяти.

Из-за указанных причин использование компиляторов удобнее для больших программ, требующих быстрого счета и большого объема памяти. Программы на Паскале, Си, Фортране всегда компилируются. Язык Бейсик часто реализуется через интерпретатор.

Коротко о главном

Для разработки программ программисты используют системы программирования (СП).

Язык программирования, с которым позволяет работать данная СП, называется ее входным языком.

Язык процессора компьютера — это язык машинных команд — ЯМК.

Уровень языка программирования определяется степенью его удаленности от ЯМК (чем дальше, тем выше уровень).

Автокод (ассемблер) — это машинно-ориентированный язык символического программирования.

Наиболее удобным средством программирования являются языки высокого уровня (ЯПВУ). Сегодня с ними работает большинство программистов.

Трансляция — это процесс перевода текста программы на язык машинных команд. Программа-переводчик называется транслятором.

Существуют два способа трансляции: компиляция и интерпретация. При компиляции сначала весь текст программы переводится на ЯМК, затем производится ее исполнение. При интерпретации перевод и исполнение происходят параллельно.

Вопросы и задания

1. Что такое язык программирования?
2. Что обозначает понятие «уровень языка программирования»?
3. К какому уровню относятся языки типа автокод (ассемблер)?
4. Почему языки программирования высокого уровня называют машинно-независимыми языками?
5. Какие из языков программирования высокого уровня вы знаете?
6. Что такое трансляция? Что такое транслятор?
7. В чем различие между компиляцией и интерпретацией?

2.4. История языков программирования

Основные темы параграфа:

- *первые шаги автоматизации программирования;*
- *первые языки высокого уровня: Кобол и Фортран;*
- *языки процедурного программирования;*
- *языки искусственного интеллекта;*
- *современные языки объектно-ориентированного и визуального программирования;*
- *программный продукт и его жизненный цикл.*

Первые шаги автоматизации программирования

Программы для первых ЭВМ программисты писали на языках машинных команд. Это очень трудоемкий и длительный процесс. Проходило значительное время между началом составления программы и началом ее использования. Решить эту проблему можно было лишь путем создания средств автоматизации программирования.

Первыми «инструментами», которые экономили труд программистов, стали подпрограммы. В августе 1944 года для релейной машины «Марк-I» под руководством Грейс Хоппер (женщина-программист, морской офицер ВМФ США) была написана первая подпрограмма для вычисления $\sin x$.

Не одну Грейс Хоппер волновала проблема облегчения труда программистов. В 1949 году Джон Моучли (один из создателей ЭВМ ENIAC) разработал систему Short Code, которую можно считать предшественницей языков программирования высокого уровня. Программист записывал решаемую задачу в виде математических формул, преобразовывал формулы в двухбуквенные коды. В дальнейшем специальная программа переводила эти коды в двоичный машинный код. Таким образом, Дж. Моучли разработал один из первых примитивных интерпретаторов. А в 1951 году Г. Хоппер создала первый компилятор А-0. Она же впервые ввела этот термин.

Первые языки высокого уровня: Кобол и Фортран

В 50-е годы прошлого века группа под руководством Г. Хоппер приступила к разработке нового языка и компилятора В-0. Новый язык позволил бы программировать на языке, близком к обычному английскому. Разработчики языка выбрали около 30 английских слов, для распознавания которых Г. Хоппер придумала способ, сохранившийся в операторах будущих языков программирования: каждое слово содержит неповторимую комбинацию из первой и третьей букв. Благодаря этому компилятор при создании машинного кода программы может игнорировать все остальные буквы в слове.

Необходимость появления такой системы, язык которой приближен к разговорному, Г. Хоппер связывала с тем, что область применения ЭВМ будет расширяться, в связи с чем будет расти и круг пользователей. По словам Г. Хоппер, следует оставить попытки «превратить их всех в математиков».

В 1958 году система В-0 получила название FLOW–MATIC и была ориентирована на обработку коммерческих данных. В 1959 году был разработан язык COBOL (Common Business Oriented Language, Кобол) — машинно-независимый язык программирования высокого уровня для решения задач бизнеса. Одна и та же программа, написанная на машинно-независимом языке, может быть выполнена на ЭВМ разных типов, оснащенных соответствующим транслятором с этого языка. Консультантом при создании языка COBOL вновь выступила Г. Хоппер.

В 1954 году публикуется сообщение о создании языка FORTRAN (FORmula TRANslation, Фортран). Местом рождения языка стала штаб-квартира фирмы IBM в Нью-Йорке. Одним из главных разра-

ботчиков является Джон Бэкус. Он же стал автором НФБ (нормальная форма Бэкуса), которая используется для описания синтаксиса многих языков программирования.

Одним из основных достоинств первой версии Фортрана являлось то, что язык давал возможность автоматизировать организацию циклов. Этот язык стал незаменимым для разработки научных и инженерных приложений. Следует отметить долговечность языка Фортран. За полстолетия он сильно изменился. В настоящее время Фортран реализован на персональных компьютерах, суперкомпьютерах и по-прежнему широко используется в научных исследованиях.

В тот же период в европейских странах и в СССР популярным становится язык ALGOL (Алгол). Как и FORTRAN, он ориентировался на математические задачи. В нем была реализована передовая для того времени технология программирования — структурное программирование.

Большое количество новых языков появилось в 60–70-е годы прошлого столетия, но не все из них выдержали испытание временем. К языкам-долгожителям следует отнести язык Basic (Бейсик), разработанный в Дартмутском университете в 1964 году под руководством Джона Кемени и Томаса Курца. По замыслу авторов, это простой язык, легко изучаемый, предназначенный для программирования несложных расчетных задач. Наибольшее распространение Basic получил на микроЭВМ и персональных компьютерах. Однако первоначально этот язык был неструктурным и с этой точки зрения плохо подходил для обучения качественному программированию. В 1985 году была создана версия языка True Basic, которая, по мнению разработчиков, была совершеннее, чем Pascal. В 1991 году появилась первая версия языка Visual Basic.

Языки процедурного программирования

Для первых языков программирования характерной чертой была *предметная ориентация*. Это значит, что каждый язык предназначался для решения какого-то определенного класса задач. COBOL был ориентирован на решение задач бизнеса, FORTRAN — на проведение инженерных и научных расчетов. В эпоху ЭВМ третьего поколения большое распространение получил язык PL/1 (Program Language/1), разработанный фирмой IBM. Это был первый язык, претендовавший на универсальность, т. е. на возможность решать любые задачи: вычислительные, обработки текстов, накопления и поиска информации. PL/1 оказался слишком сложным языком, транслятор с него недостаточно оптимальным, содержащим ряд невыявленных ошибок. По этой причине этот язык не получил распространения. Однако линия на универсализацию языков была про-

должена. Старые языки были модернизированы в универсальные варианты. Примером тому стал FORTRAN 77.

Значительным событием в истории языков программирования стало создание в 1971 году языка Pascal (Паскаль). Его автором является Никлаус Вирт, профессор из Швейцарии. Вирт назвал этот язык в честь французского математика и физика Блеза Паскаля, который в 1642 году сконструировал вычислительный механизм. Первоначально Pascal создавался как язык для обучения. В нем ярко выражена структурная линия программирования. Широкое практическое применение язык получил с появлением персональных компьютеров в версии Turbo Pascal.

Язык программирования С (Си) был задуман как инструментальный язык для разработки операционных систем (ОС). Он создавался одновременно с операционной системой UNIX. Авторами этого языка и ОС UNIX являются американские программисты Деннис Ричи и Кеннет Томпсон. Первоначально К. Томпсон начал писать ОС UNIX на языке FORTRAN. В дальнейшем был создан язык С, и в 1973 году ядро операционной системы вместе с программами-утилитами было переписано на С. Этот язык является структурным языком высокого уровня. В настоящее время он применяется для разработки не только операционных систем, но и трансляторов, системных и прикладных программ.

Языки искусственного интеллекта

В 90-х годах прошлого столетия планировалось появление компьютеров пятого поколения, называемых машинами «искусственного интеллекта». В качестве основных языков программирования в этом, пока неосуществленном, проекте предполагались языки искусственного интеллекта LISP и PROLOG.

Создателем языка LISP (1956–1959 гг.) является Джон Маккарти, которого называют отцом искусственного интеллекта. Именно он первым ввел термин «искусственный интеллект». Основным в языке LISP является понятие рекурсивно определенных функций. Доказано, что любой алгоритм может быть описан с помощью некоторого набора рекурсивных функций. Основные идеи этого языка были позже использованы в языке программирования для детей LOGO, разработанном в 70-е годы XX века в Массачусетском технологическом институте под руководством Сэймура Пейперта. Подмножество языка LOGO, включающее команды для Черепашки, применяется при раннем обучении программированию.

Язык PROLOG разработан во Франции в 1972 году также для решения проблем искусственного интеллекта. PROLOG позволяет в формальном виде описывать различные утверждения, логику рассуждений, заставляет компьютер давать ответы на заданные вопросы.

Современные языки объектно-ориентированного и визуального программирования

В последнее время одним из основных направлений в развитии программного обеспечения компьютера стал **объектно-ориентированный подход**. Под словом «объект» понимается структура, объединяющая в единое целое данные и программы их обработки. Стали популярны объектно-ориентированные операционные системы (например, Windows), прикладные программы, а также **системы объектно-ориентированного программирования** (ООП).

Первым языком с элементами ООП был язык Симула-67. В Turbo Pascal, начиная с версии 5.5, появились средства ООП. Итогом развития Turbo Pascal в этом направлении стало создание фирмой Borland системы программирования Delphi (Делфи). Использование этой системы, в частности, дает возможность легко и быстро программировать сложный графический интерфейс. В 1991 году появилась первая версия языка Visual Basic. Начиная с пятой версии (1997 г.) язык стал полностью объектно-ориентированным. По данным на конец 90-х годов прошлого столетия, количество программистов, использующих для своих разработок Visual Basic, не уступает числу сторонников Visual C++ и Delphi.

В 1985 году лаборатория Bell Labs (США) сообщила о создании языка программирования C++ (Си++). Этот язык является сегодня наиболее популярным среди языков объектно-ориентированного программирования. С его помощью возможно создание программных приложений, ориентированных на любые машины — от персональных до суперкомпьютеров. Создатель языка — Бьорн Страуструп.

Представителем языков объектно-ориентированного программирования является и язык JAVA, созданный в 1995 году под руководством Джеймса Гослинга группой инженеров компании Sun Microsystems. При его разработке была поставлена цель — создать простой язык, не требующий специального изучения. Язык JAVA был разработан так, чтобы быть максимально похожим на C++. JAVA является идеальным инструментом при создании приложений для Интернета.

Программный продукт и его жизненный цикл

За последние несколько десятилетий программирование стало индустрией. Как и в каждой индустрии, здесь есть производители продукции и ее потребители. Чаще всего производители — это коллективы программистов, объединенные в фирмы, компании (пример: Microsoft, Adobe, C1, «Кирилл и Мефодий»). Пользователи приобретают у производителей программные продукты. **Программным продуктом (ПП)** *называется сама программа в совокупнос-*

ти с пользовательской документацией. Поэтому программный продукт можно эксплуатировать без участия его разработчика.

Жизненным циклом программного продукта называют следующую последовательность этапов.

Анализ — изучение потребностей пользователя (например, функционирование системы, которую нужно автоматизировать) и определение перечня требований к будущей программе. Эту работу выполняет системный аналитик.

Проектирование — определение общей структуры (архитектуры) ПП. Результат — спецификация ПП. Эту работу выполняет системный программист.

Реализация — написание программного кода отдельных модулей, их тестирование, составление документации.

Сборка, тестирование, испытание — сборка в единый комплекс всего, что сделано разными программистами, тестирование всего программного комплекса. Альфа-тестированием называют тестирование ПП людьми из той же организации, не участвовавшими в разработке программ. В результате должна быть получена работоспособная программа.

Внедрение (выпуск) — на этом этапе разработчик работает на одного заказчика (пользователя). Программный продукт доводится до состояния, которое полностью удовлетворяет потребности пользователя. Происходит обучение пользователя, после чего он может работать с ПП без участия автора.

Выпуск ПП на рынок начинается с этапа бета-тестирования. Соответствующая версия программы называется бета-версией. Изготавливается несколько экземпляров ПП, которые передаются потенциальным пользователям. Цель — еще раз проверить работоспособность ПП. После бета-тестирования происходит выпуск коммерческой версии.

Сопровождение — устранение обнаруживаемых в ходе эксплуатации ошибок. Внесение непринципиальных усовершенствований. Накопление предложений для разработки следующей версии ПП.

Коротко о главном

Первые средства автоматизации программирования: Short Code (1949); компилятор A-0 (1951).

Первыми распространенными полноценными языками высокого уровня были: FORTRAN (Фортран) (1954), ориентированный на математические вычисления, COBOL (Кобол), ориентированный на задачи бизнеса.

Языки, распространившиеся в 60–70-х годах XX века: ALGOL, Basic, Pascal, PL/1; C (Си) — первый язык высокого уровня, применяемый в системном программировании; языки искусственного интеллекта: PROLOG, LISP.

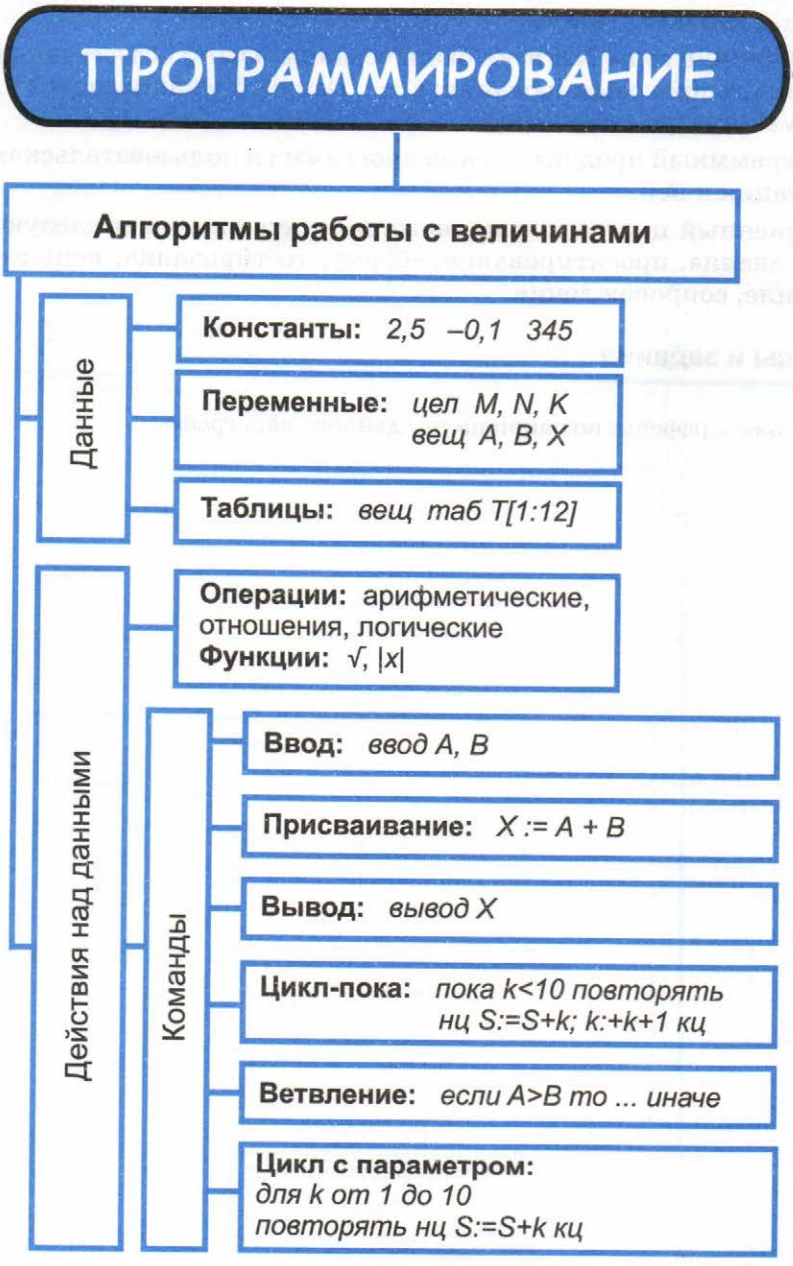
В 1980–90-е годы XX века были созданы языки объектно-ориентированного программирования: C++, Delphi, Visual Basic; JAVA — язык Web-программирования.

Программный продукт — сама программа и пользовательская документация к ней.

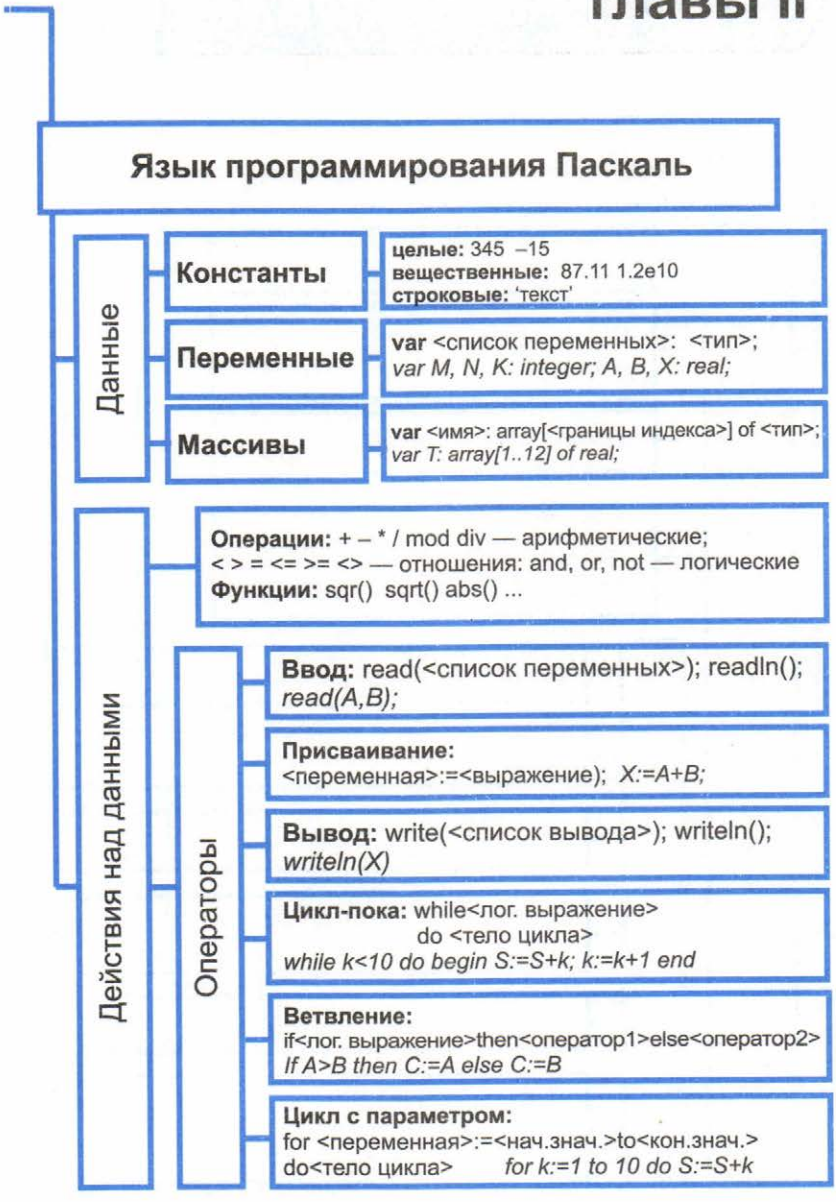
Жизненный цикл программного продукта включает следующие этапы: анализ, проектирование, сборку, тестирование, испытание, внедрение, сопровождение.

Вопросы и задания

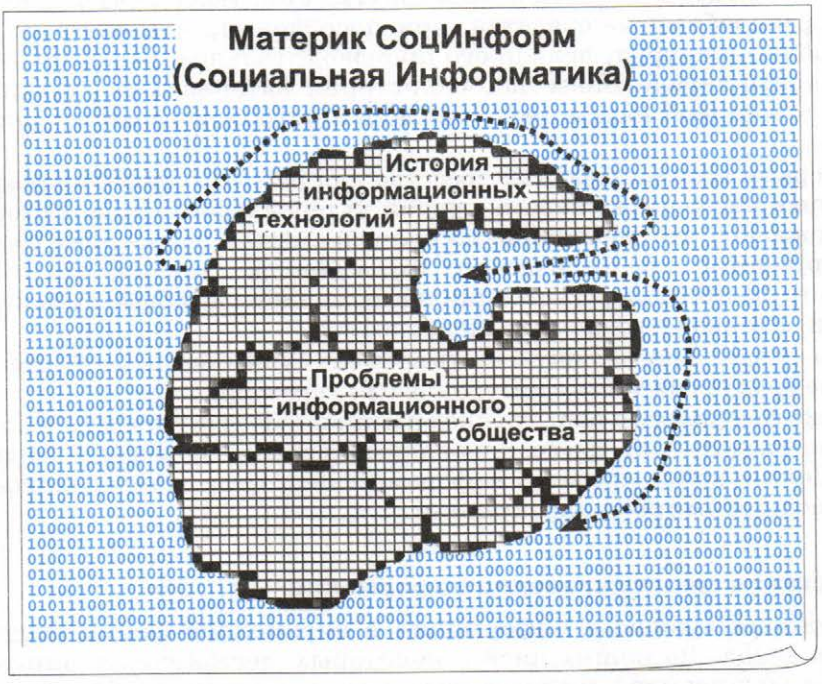
Подготовьте реферат по одной из тем данного параграфа.


ПРОГРАММИРОВАНИЕ
Алгоритмы работы с величинами
Данные
Константы: 2,5 –0,1 345
Переменные: цел M, N, K
вещ A, B, X
Таблицы: вещ таб T[1:12]
Действия над данными
Операции: арифметические, отношения, логические
Функции: √, |x|
Команды
Ввод: ввод A, B
Присваивание: X := A + B
Вывод: вывод X
Цикл-пока: пока k<10 повторять
нц S:=S+k; k:+k+1 кц
Ветвление: если A>B то ... иначе
Цикл с параметром:
для k от 1 до 10
повторять нц S:=S+k кц

Система основных понятий главы II



Глава III
Информационные технологии и общество



Здесь вы узнаете

- историю информационных технологий и компьютерной техники
- что такое информационное общество
- что такое информационные ресурсы общества
- как решаются проблемы информационной безопасности

§ 22

Предыстория информатики

Основные темы параграфа:

- *история средств хранения информации;*
- *история средств передачи информации;*
- *история средств обработки информации;*
- *Аналитическая машина Бэббиджа — предшественница ЭВМ.*

В любой деятельности человек всегда придумывал и создавал самые разнообразные средства, приспособления, орудия труда. Все это облегчало труд, делало его производительнее, расширяло возможности людей. Известно, что история материального производства и мировой науки тесно связана с историей развития орудий труда.

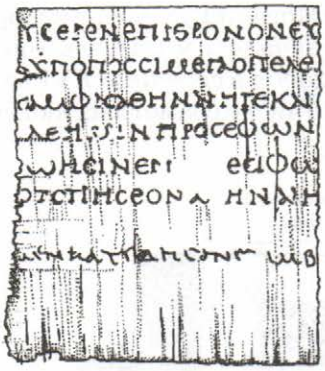
Первые вспомогательные средства для работы с информацией появились много позже первых орудий материального труда. Историки утверждают, что расстояние во времени между появлением первых инструментов для физического труда (топор, ловушка для охоты) и инструментов для регистрации информационных образов (на камне, кости) составляет около миллиона лет!

Следовательно, бóльшую часть времени существования человека на Земле труд носил только материальный характер.

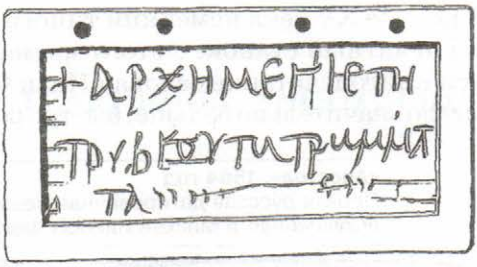
Уже говорилось о том, что информационную деятельность человека можно разделить на три составляющие: **хранение**, **передачу** и **обработку**. Долгое время средства информационного труда развивались отдельно по этим трем направлениям.

История средств хранения информации

История хранения информации в письменной форме уходит в глубь веков. До наших дней в некоторых местах сохранились наскальные письмена древнего человека, выполненные 25–20 тысяч лет назад; лунный календарь, выгравированный на кости 20 тысяч лет назад. Для письма также использовались дерево, глина. Многие века письменные документы составлялись на пергаментных свитках. Это было «очень дорогим удовольствием». Пергамент делался из кожи животных. Ее растягивали, чтобы получить тонкие листы. Когда на востоке научились ткать шелк, его стали использовать не только для одежды, но и для письма.



Папирус. I–II века.
Отрывок из поэмы «Илиада» древнегреческого поэта Гомера



Школьная восковая табличка
Сохранившиеся таблички происходят в основном из Египта, Дакии, Помпеи и Геркуланума. Деревянные таблички покрывались цветным воском, надписи на нем процарапывались стилом – острым предметом

Глиняный диск. XVII век до н. э.
Найден в городе Фест на Крите.
По обеим сторонам диска спиралью «бегут» так и не расшифрованные письмена

Письменный прибор древнеегипетского писца
состоял из заостренных палочек для письма с расщепленными кончиками, которые складывались в пенал, тушницы для туши красного и черного цветов и мешочка с песком

Во II веке нашей эры в Китае изобрели **бумагу**. Однако до Европы она дошла только в XI веке. Вплоть до XV века письма, документы, книги писались вручную. В качестве инструмента для письма использовались кисточки, перья птиц, позже — металлические перья; были изобретены краски, чернила. Книг было очень мало, они считались предметами роскоши.

В середине XV века немецкий типограф Иоганн Гутенберг изобрел первый **печатный станок**. С этого времени началось книгопечатание. На Руси книгопечатание основал Иван Федоров в середине XVI века. Книг стало значительно больше, быстро росло число грамотных людей.

«Апостол». 1564 год
Первая русская датированная печатная книга, выпущенная в Москве Иваном Федоровым

До сегодняшнего дня лист бумаги остается основным носителем информации. Но у него появились серьезные «конкуренты».

В XIX веке была изобретена **фотография**. Носителями видеоинформации стали фотопленка и фотобумага.

В 1895 году французы братья Люмьер продемонстрировали в Париже первый в мире кинофильм, используя аппарат собственного изобретения. Этот год считается годом рождения **кино**.

История технических средств хранения и воспроизведения звука начинается с 1877 года. В этом году в США Томас Эдисон создал **фонограф**. Звук механическим способом — с помощью записывающей иглы — наносился на поверхность вращающегося барабана, покрытого воском. Немного позднее был создан механический **граммофон**, а затем его портативный вариант — **патефон**, воспроизводящие звук, записанный на целлулоидной грампластинке. Электрический аналог патефона — **электрофон** был изобретен в XX веке. В XX веке был изобретен **магнитофон**. И совсем недавно на магнитную ленту научились записывать не только звук, но и изображение: появился **видеомагнитофон**.

История средств передачи информации

Первоначально люди пользовались лишь средствами ближней связи: речью, слухом, зрением. Затем появилось первое средство дальней связи — **почта**, чему немало способствовало развитие письменности.

Почтовый курьер инков
Такой гонец пробегал около полутора километров и передавал устное послание. Чтобы оно не задерживалось, он еще издали оповещал о своем приближении, подавая звуковые сигналы голосом или трубя в раковину

Для быстрой передачи каких-то важных сведений часто использовались оригинальные идеи. Известно, например, применение на Кавказе **костровой связи**. Два костровых сигнальщика находились на расстоянии прямой видимости на возвышенных местах или башнях. Когда приближалась опасность (нападали враги), сигнальщики, зажигая цепочку костров, предупреждали об этом население. В XVIII веке возник семафорный телеграф — это тоже световая связь, но технически более совершенная.

Факельный телеграф. В Древней Греции пользовались оптической сигнальной связью: днем — дымом, ночью — огнем

Очень богатым на открытия в области связи был XIX век. В этом веке люди овладели электричеством, которое породило множество

изобретений. Сначала П. Л. Шиллинг в России в 1832 году изобрел **электрический телеграф**. А в 1837 году американец С. Морзе создал **электромагнитный телеграфный аппарат** и придумал специальный телеграфный код — азбуку, которая носит его имя.

В 1876 году американец А. Белл изобрел **телефон**. И наконец, в 1895 году русский изобретатель А. С. Попов открыл эпоху **радиосвязи**.

Самым замечательным изобретением XX века в области связи можно назвать **телевидение**. Освоение космоса привело к созданию **спутниковой связи**.

История средств обработки информации

Теперь познакомимся со средствами обработки информации. Важнейшим видом такой обработки являются вычисления. Появление и развитие счетных инструментов стимулировали развитие земледелия, торговли, мореплавания, астрономии и многих других областей практической и научной деятельности людей.

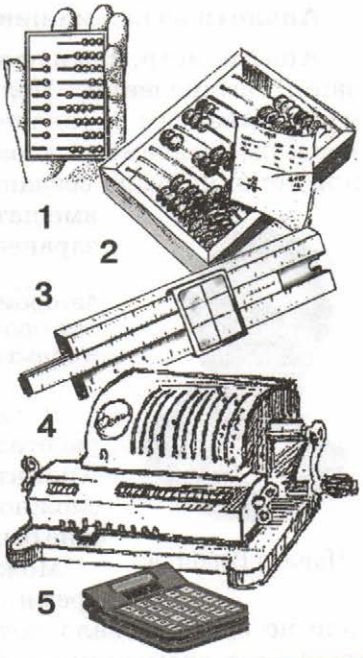
Нетрудно догадаться, что первым счетным средством для человека были его **пальцы**. Этот инструмент всегда «под рукой»! Кто из вас им не пользовался?

Вот как описывает пальцевой счет туземцев Новой Гвинеи знаменитый русский путешественник Н. Н. Миклухо-Маклай: «...папуас загибает один за другим пальцы руки, причем издает определенный звук, например "бе, бе, бе"... Досчитав до пяти, он говорит "ибон-бе" (рука). Затем он загибает пальцы другой руки, снова повторяет "бе, бе"... пока не дойдет до "ибон али" (две руки). Затем он идет дальше, приговаривая "бе, бе"... пока не дойдет до "самба-бе" и "самба-али" (одна нога, две ноги). Если нужно считать дальше, папуас пользуется пальцами рук и ног кого-нибудь другого».

В V веке до нашей эры в Греции и Египте получил распространение **абак**. «Абак» — это греческое слово, которое переводится как «счетная доска». Вычисления на абаке производились перемещением камешков по желобам на мраморной доске.

Подобные счетные инструменты распространялись и развивались по всему миру. Например, китайский вариант абака назывался суан-пан. «Потомком» абака можно назвать и русские счеты. В России они появились на рубеже XVI–XVII веков. До недавнего времени они активно использовались, преимущественно в торговле.

В начале XVII века шотландский математик Джон Непер ввел понятие логарифма, опубликовал таблицы логарифмов. Затем в течение двух веков развивались вычислительные инструменты, основанные на использовании этой математической функции. Логарифмы позволяют свести трудоемкие арифметические операции — умножение и деление — к более простым — сложению и вычитанию. В результате появилась **логарифмическая линейка**. Этот инструмент до недавнего времени был вычислительным средством инженеров. И лишь ближе к концу XX столетия его вытеснили электронные калькуляторы.



1 — греческий абак;
2 — счеты;
3 — логарифмическая линейка;
4 — арифмометр;
5 — калькулятор

В 1645 году французский математик Блез Паскаль создал первую счетную машину. **Машина Паскаля** позволяла быстро выполнять сложение многозначных чисел.

Немецкий ученый Лейбниц, развив идею Паскаля, создал **механический арифмометр**, на котором можно было выполнять все четыре арифметические операции с многозначными числами. Позднее арифмометр многократно совершенствовался, в том числе и русскими изобретателями П. Л. Чебышевым и В. Т. Однером.

Арифмометр был предшественником современного **калькулятора** — маленького электронно-вычислительного устройства. Сейчас практически у каждого школьника есть калькулятор, который помещается в кармане. Любому академику начала XX века такое устройство показалось бы фантастическим.

Аналитическая машина Бэббиджа — предшественница ЭВМ

Арифмометр, как и простой калькулятор, — это средство механизации вычислений. Человек, производя вычисления на таком устройстве, сам управляет его работой, определяет последовательность выполняемых операций. Мечтой изобретателей вычислительной техники было создание считающего автомата, который бы без вмешательства человека производил расчеты по заранее составленной программе.

Чарльз Бэббидж

Автором первого проекта вычислительного автомата был профессор Кембриджского университета **Чарльз Бэббидж**

В период между 1820 и 1856 годами Бэббидж работал над созданием программно управляемой **Аналитической машины**. Это было настолько сложное механическое устройство, что проект так и не был реализован.

Можно сказать, что Бэббидж опередил свое время. Для осуществления его проекта в ту пору еще не существовало подходящей технической базы. Некоторым ученым современникам Бэббиджа его труд казался бесплодным. Однако пророчески звучат сейчас слова самого Чарльза Бэббиджа: «Природа научных знаний такова, что малопонятные и совершенно бесполезные приобретения сегодняшнего дня становятся популярной пищей для будущих поколений».

Основные идеи, заложенные в проекте Аналитической машины, в нашем веке были использованы конструкторами ЭВМ. Все главные компоненты современного компьютера присутствовали в конструкции Аналитической машины: это *склад* (в современной терминологии — память), где хранятся исходные числа и промежуточные результаты; *мельница* (арифметическое устройство), в которой осуществляются операции над числами, взятыми из склада; *контора* (устройство управления), производящая управление последовательностью операций над числами соответственно заданной программе; *блоки ввода* исходных данных и *печати результатов*.

Для программного управления Аналитической машиной использовались перфокарты — картонные карточки с пробитыми в них отверстиями (перфорацией). Перфокарты были изобретены в начале XIX века во Франции Жозефом М. Жаккардом для управления работой автоматического ткацкого станка.

Интересным историческим фактом является то, что первую программу для машины Бэббиджа в 1846 году написала Ада Лавлейс — дочь великого английского поэта Джорджа Байрона.

Аналитическая машина Бэббиджа — это уже универсальное средство, объединяющее обработку информации, хранение информации и обмен исходными данными и результатами с человеком.

Коротко о главном

1. Важнейшие этапы развития средств хранения информации (до изобретения компьютера):
   - изобретение бумаги в Китае — II век;
   - изобретение печатного станка — XV век, И. Гутенберг;
   - изобретение фотографии, кино, фонографа — XIX век;
   - изобретение магнитных носителей информации — XX век.
2. Важнейшие изобретения технических средств передачи информации (до изобретения компьютера):
   - телеграф — первая половина XIX века, П. Л. Шиллинг, С. Морзе;
   - телефон — вторая половина XIX века, А. Белл;
   - радио — конец XIX века: А. С. Попов, Г. Маркони;
   - телевидение — XX век.
3. Важнейшие изобретения технических средств обработки информации (до изобретения компьютера):
   - машина Паскаля, механический калькулятор Лейбница — XVII век;
   - Аналитическая машина — начало XIX века, Ч. Беббидж;
   - механический арифмометр — конец XIX века, В. Т. Однер;
   - электрический калькулятор — XX век.

Вопросы и задания

1. Какие средства хранения информации были первыми?
2. Когда появилось книгопечатание, кто его изобретатель?
3. Какие средства хранения информации изобретены в XIX–XX вв.?
4. Назовите основные технические средства передачи информации в хронологической последовательности их изобретения.
5. Перечислите основные вычислительные средства в хронологической последовательности их изобретения.
6. Кто, когда и где разработал первый проект автоматической вычислительной машины?
7. Какое влияние проект Аналитической машины оказал на дальнейшее развитие вычислительной техники?
8. Подготовьте презентацию по одной из тем заданий 1–7.

ЕК ЦОР: часть 2, глава 7, § 44. ЦОР № 8.

§ 23
История ЭВМ

Основные темы параграфа:

- *счетно-перфорационные и релейные машины;*
- *начало эпохи ЭВМ;*
- *четыре поколения ЭВМ;*
- *перспективы пятого поколения.*

В § 22 рассказывалось о проекте Чарльза Бэббиджа — первой в истории попытке создания программно управляемого вычислительного автомата. Бэббиджу так и не удалось построить свою Аналитическую машину, используя техническую базу середины XIX столетия. К концу XIX — началу XX века с развитием электротехники появилась возможность снова вернуться к этой проблеме.

Счетно-перфорационные и релейные машины

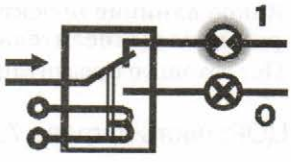
В конце XIX века Герман Холлерит в Америке изобрел *счетно-перфорационные машины*. В них, так же как и в Аналитической машине Бэббиджа, использовались перфокарты, но только не для представления программы, а для хранения числовой информации. Каждая такая машина могла выполнять только одну определенную программу, манипулируя с числами, пробитыми на перфокартах. Счетно-перфорационные машины осуществляли перфорацию, сортировку, суммирование, вывод на печать числовых таблиц. На этих машинах удавалось решать многие типовые задачи статистической обработки, бухгалтерского учета и др.

Г. Холлерит основал фирму по выпуску счетно-перфорационных машин, которая затем была преобразована в фирму IBM, которая ныне является самым известным в мире производителем компьютеров.

Непосредственными предшественниками ЭВМ были **релейные вычислительные машины**. К 30-м годам XX века получила большое развитие релейная автоматика.

В процессе работы релейной машины происходят переключения тысяч реле из одного состояния в другое.

Электромеханическое реле — это двухпозиционный переключатель, который имеет два состояния: включено и выключено. Это свойство позволяет использовать реле для кодирования информации в двоичном виде



Релейная машина «Марк-2», изготовленная в 1947 году, содержала около 13 000 реле. Одной из наиболее совершенных релейных машин была машина советского конструктора Н. И. Бессонова — РВМ-1. Она была построена в 1956 году и проработала почти 10 лет, конкурируя с существовавшими уже в то время ЭВМ. Поскольку реле — это механическое устройство, то его инерционность (задержка при переключении) достаточно велика, что сильно ограничивало скорость работы таких машин. Скорость РВМ-1 составляла 50 сложений или 20 умножений в секунду. Практически это был предел скорости для машин этого типа.

Начало эпохи ЭВМ

В первой половине XX века бурно развивалась радиотехника. Основным элементом радиоприемников и радиопередатчиков в то время были электронно-вакуумные лампы. Электронные лампы стали технической основой для первых **электронно-вычислительных машин (ЭВМ)**.

Первая ЭВМ — универсальная машина на электронных лампах — была построена в США в 1945 году.

Эта машина называлась ENIAC (расшифровывается так: электронный цифровой интегратор и вычислитель). Конструкторами ENIAC были Дж. Моучли и Дж. Эккерт. Скорость счета этой машины превосходила скорость релейных машин того времени в тысячу раз.

Первый электронный компьютер ENIAC программировался с помощью штекерно-коммутационного способа, т. е. программа строилась путем соединения проводниками отдельных блоков машины на коммутационной доске. Эта сложная и утомительная процедура подготовки машины к работе делала ее неудобной в эксплуатации.

Основные идеи, по которым долгие годы развивалась вычислительная техника, были разработаны крупнейшим американским математиком **Джоном фон Нейманом**

Джон фон Нейман

В 1946 году в журнале Nature вышла статья Дж. фон Неймана, Г. Голдстайна и А. Беркса «Предварительное рассмотрение логической конструкции электронного вычислительного устройства». В этой статье были изложены принципы устройства и работы ЭВМ. Главный из них — **принцип хранимой в памяти программы**, согласно которому данные и программа помещаются в общую память машины.

Принципиальное описание устройства и работы компьютера принято называть **архитектурой ЭВМ**. Идеи, изложенные в упомянутой выше статье, получили название «*архитектура ЭВМ Дж. фон Неймана*».

В 1949 году была построена первая ЭВМ с архитектурой Неймана — английская машина EDSAC. Годом позже появилась американская ЭВМ EDVAC. Названные машины существовали в единственных экземплярах. Серийное производство ЭВМ началось в развитых странах мира в 50-х годах XX века.

С. А. Лебедев

В нашей стране первая ЭВМ была создана в 1951 году. Называлась она МЭСМ — малая электронная счетная машина. Конструктором МЭСМ был **Сергей Алексеевич Лебедев**

Велика роль академика С. А. Лебедева в создании отечественных компьютеров. Под его руководством в 1950-х годах были построены серийные ламповые ЭВМ БЭСМ-1 (Большая электронная счетная машина), БЭСМ-2, М-20. В то время эти машины были одними из лучших в мире.

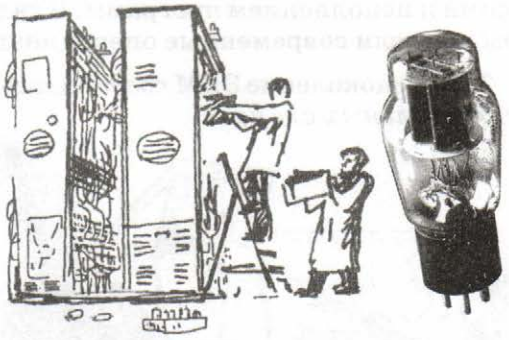
В 60-х годах XX века С. А. Лебедев руководил разработкой полупроводниковых ЭВМ БЭСМ-3М, БЭСМ-4, М-220, М-222. Выдающимся достижением того периода была машина БЭСМ-6. Это первая отечественная и одна из первых в мире ЭВМ с быстродействием один миллион операций в секунду. Последующие идеи и разработки С. А. Лебедева способствовали созданию более совершенных машин следующих поколений.

Четыре поколения ЭВМ

Электронно-вычислительную технику принято делить на поколения. Смены поколений чаще всего были связаны со сменой элементной базы ЭВМ, с прогрессом электронной техники. Это всегда приводило к росту вычислительной мощности ЭВМ, т. е. быстродействия и объема памяти. Но это не единственный признак смены поколений. При таких переходах, как правило, происходили существенные изменения в архитектуре ЭВМ, расширялся круг задач, решаемых на ЭВМ, менялся способ взаимодействия между пользователем и компьютером.

Первое поколение ЭВМ — *ламповые машины 50-х годов XX века*. Скорость счета самых быстрых машин первого поколения доходила до 20 тысяч операций в секунду (ЭВМ М-20). Для ввода программ и данных использовались перфоленты и перфокарты. Поскольку внутренняя память этих машин была невелика (могла вместить

в себя несколько тысяч чисел и команд программы), то они, главным образом, использовались для инженерных и научных расчетов, не связанных с переработкой больших объемов данных.



Это были довольно громоздкие сооружения, содержавшие в себе тысячи ламп, занимавшие иногда сотни квадратных метров, потреблявшие электроэнергию в сотни киловатт

Программы для таких машин составлялись на языках машинных команд. Это довольно сложная и трудоемкая работа. Программирование в те времена было доступно немногим.

В 1949 году в США был создан первый полупроводниковый прибор, заменяющий электронную лампу. Он получил название «транзистор». Транзисторы быстро внедрялись в радиотехнику.

В 60-х годах XX века транзисторы стали элементной базой для **ЭВМ второго поколения.**

Быстродействие большинства машин достигло десятков и сотен тысяч операций в секунду. Объем внутренней памяти возрос в сотни раз по сравнению с ЭВМ первого поколения.

Переход на полупроводниковые элементы улучшил качество ЭВМ по всем параметрам: они стали компактнее, надежнее, менее энергоемкими

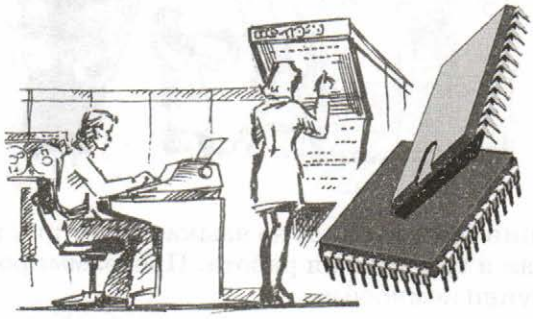
Большое развитие получили устройства внешней (магнитной) памяти: магнитные барабаны, накопители на магнитных лентах (НМЛ). Благодаря этому появилась возможность создавать на ЭВМ информационно-справочные, поисковые системы, позволявшие длительно хранить на магнитных носителях большие объемы информации.

Во времена второго поколения активно стали развиваться языки программирования высокого уровня. Первыми из них были ФОРТРАН, АЛГОЛ, КОБОЛ. Составление программы перестало зависеть от модели машины, сделалось проще, понятнее, доступнее.

Программирование как элемент грамотности стало широко распространяться в системе высшего образования.

Появились мониторные системы, управляющие режимом трансляции и исполнением программ. В дальнейшем из мониторных систем выросли современные операционные системы.

Третье поколение ЭВМ *создавалось на новой элементной базе — интегральных схемах.*



С помощью очень сложной технологии специалисты научились монтировать на маленькой пластине из полупроводникового материала, площадью менее 1 см^2, достаточно сложные электронные схемы. Их назвали интегральными схемами

Первые интегральные схемы (ИС) содержали в себе десятки, затем сотни элементов (транзисторов, сопротивлений и др.). Когда степень интеграции (количество элементов) приблизилась к тысяче, их стали называть большими интегральными схемами — БИС; затем появились сверхбольшие интегральные схемы — СБИС.

ЭВМ третьего поколения начали производиться во второй половине 60-х годов прошлого века. Тогда американская фирма IBM приступила к выпуску системы машин IBM-360. Это были машины на ИС. Немного позднее стали выпускаться машины серии IBM-370, построенные на БИС. В Советском Союзе в 70-х годах XX века начался выпуск машин серии ЕС ЭВМ (Единая система ЭВМ) по образцу IBM-360/370.

Переход к третьему поколению связан с существенными изменениями архитектуры ЭВМ. Появилась возможность выполнять одновременно несколько программ на одной машине. Такой режим работы называется мультипрограммным (многопрограммным) режимом.

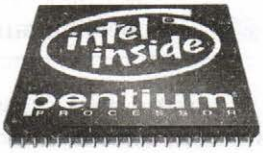
Скорость работы наиболее мощных моделей ЭВМ достигла нескольких миллионов операций в секунду. На машинах третьего поколения появился новый тип внешних запоминающих устройств — магнитные диски. Как и на магнитных лентах, на дисках можно хранить большое количество информации. Вместе с тем накопители на магнитных дисках (НМД) работают гораздо быстрее, чем НМЛ. Широко стали использоваться новые типы устройств ввода/вывода: дисплеи, графопостроители.

В этот период существенно расширились области применения ЭВМ. Стали создаваться базы данных, первые системы искусственного интеллекта, системы автоматизированного проектирования (САПР) и управления (АСУ).

В 70-е годы XX века получила мощное развитие линия малых (мини) ЭВМ. Своеобразным эталоном здесь стали машины американской фирмы DEC серии PDP-11. В нашей стране по этому образцу создавалась серия машин СМ ЭВМ (Система малых ЭВМ). Они меньше, дешевле, надежнее больших машин. Машины этого типа хорошо приспособлены для целей управления различными техническими объектами: производственными установками, лабораторным оборудованием, транспортными средствами. По этой причине их называют управляющими машинами. Во второй половине 70-х годов XX века производство мини-ЭВМ превысило производство больших машин.

Очередное революционное событие в электронике произошло в 1971 году, когда американская фирма Intel объявила о создании микропроцессора.

Микропроцессор — это сверхбольшая интегральная схема, способная выполнять функции основного блока компьютера — процессора



Микропроцессор — это миниатюрный «мозг», работающий по программе. Первоначально микропроцессоры стали встраивать в различные технические устройства: станки, автомобили, самолеты. Такие микропроцессоры осуществляют автоматическое управление работой этой техники.

Соединив микропроцессор с устройствами ввода/вывода, оперативной и внешней памяти, получили новый тип компьютера: микроЭВМ

МикроЭВМ относятся к **ЭВМ четвертого поколения**. Существенным отличием микроЭВМ от своих предшественников являются их малые габариты (размеры бытового телевизора) и сравнительная дешевизна. Это первый тип компьютеров, который появился в розничной продаже.

Самой популярной разновидностью ЭВМ сегодня являются персональные компьютеры

Появление феномена персональных компьютеров (ПК) связано с именами двух американских специалистов: Стива Джобса и Стива Возняка. В 1976 году на свет появился их первый серийный ПК Apple-1, а в 1977 году — Apple-2.

Сущность того, что такое ПК, кратко можно сформулировать так:

Персональный компьютер — это микроЭВМ с дружественным к пользователю аппаратным и программным обеспечением.

В аппаратном комплекте ПК используется цветной графический дисплей, манипуляторы типа «мышь», «джойстик», удобная клавиатура, удобные для пользователя компактные диски (оптические). Программное обеспечение позволяет человеку легко общаться с машиной, быстро усваивать основные приемы работы с ней, получать пользу от компьютера, не прибегая к программированию. Общение человека и ПК может принимать форму игры с красочными картинками на экране, звуковым сопровождением.

Неудивительно, что машины с такими свойствами быстро приобрели популярность, причем не только среди специалистов. Персональный компьютер становится такой же привычной бытовой техникой, как радиоприемник или телевизор. Их выпускают огромными тиражами, продают в магазинах.

С 1980 года «законодателем мод» на рынке ПК становится американская фирма IBM. Ее конструкторам удалось создать такую архитектуру, которая стала фактически международным стандартом на профессиональные ПК. Машины этой серии получили название IBM PC (Personal Computer).

В конце 80-х — начале 90-х годов XX века большую популярность приобрели машины фирмы Apple Corporation марки Macintosh. В США они широко используются в системе образования.

Появление и распространение ПК по своему значению для общественного развития сопоставимо с появлением книгопечатания. Именно ПК сделали компьютерную грамотность массовым явлением. Развитие этого типа машин вызвало появление понятия «информационные технологии», без которых уже становится невозможным обойтись в большинстве областей деятельности человека.

Есть и другая линия в развитии ЭВМ четвертого поколения. Это **суперкомпьютеры**. Машины этого класса имеют быстродействие в сотни миллионов и миллиарды операций в секунду. Только суперкомпьютеры могут справиться с обработкой больших объемов информации, например статистическими данными по переписи населения, результатами метеорологических наблюдений, финансовой информацией. Иногда скорость обработки информации имеет решающее значение. Примером может служить составление прогноза погоды, моделирование климатических изменений, позволяющее предсказать стихийное бедствие (цунами, тайфун, землетрясение и т. д.).

Суперкомпьютер — это многопроцессорный вычислительный комплекс. Высокое быстродействие достигается благодаря тому, что множество процессоров, его составляющих, осуществляют параллельную (одновременную) обработку данных.

Суперкомпьютеры являются дорогими машинами, стоимость которых может достигать десятков миллионов долларов. Поэтому возникает проблема доступности таких дорогих вычислительных ресурсов. Решение этой проблемы связано с созданием многопользовательских суперкомпьютерных центров.

В качестве альтернативы суперкомпьютерам создаются так называемые кластерные системы. Кластерная система — это сеть из множества рабочих станций на базе ПК. Чтобы рабочие станции функционировали как многопроцессорная вычислительная система, в такой сети используется специальное программное обеспечение. Оказалось, что можно построить многопроцессорный комплекс — кластер, который лишь в 2–3 раза уступает по быстродействию суперкомпьютеру, но дешевле его в сотни раз. В крупных российских университетах и научных центрах установлены и активно используются кластерные системы.

Перспективы пятого поколения

ЭВМ пятого поколения — это машины недалекого будущего. Основным их качеством должен быть высокий интеллектуальный уровень. *Машины пятого поколения — это реализованный искусственный интеллект.* В них будет возможен ввод с голоса, голосовое общение, машинное «зрение», машинное «осязание». Многое уже практически сделано в этом направлении.

Коротко о главном

Появлению ЭВМ предшествовали счетно-перфорационные и релейные машины.

ЭВМ первого поколения — ламповые машины (50-е годы XX века).

ЭВМ второго поколения — полупроводниковые машины (60-е годы XX века).

ЭВМ третьего поколения — машины на интегральных схемах (вторая половина 60-х годов).

Персональные компьютеры и суперкомпьютеры (многопроцессорные вычислительные комплексы) относятся к машинам четвертого поколения (70-е годы XX века).

ЭВМ пятого поколения — машины, основанные на искусственном интеллекте.

Вопросы и задания

1. Когда и кем были изобретены счетно-перфорационные машины? Какие задачи на них решались?
2. Что такое электромеханическое реле? Когда создавались релейные вычислительные машины? Каким быстродействием они обладали?
3. Где и когда была построена первая ЭВМ? Как она называлась?
4. Какова роль Джона фон Неймана в создании ЭВМ?
5. Кто был конструктором первых отечественных ЭВМ?
6. На какой элементной базе создавались машины первого поколения? Каковы были их основные характеристики?
7. На какой элементной базе создавались машины второго поколения? В чем их преимущества по сравнению с первым поколением ЭВМ?
8. Что такое интегральная схема? Когда были созданы первые ЭВМ на интегральных схемах? Как они назывались?
9. Какие новые области применения ЭВМ возникли с появлением машин третьего поколения?
10. Что такое микропроцессор? Когда и где был создан первый микропроцессор?
11. Что такое микроЭВМ и персональный компьютер?
12. Какие типы ПК наиболее распространены в мире?
13. Что такое суперкомпьютер?
14. Что такое кластерные системы ПК?
15. В чем особенность компьютеров пятого поколения?
16. Подготовьте сообщения по темам заданий 1–15 для школьного сайта.

www ЕК ЦОР: часть 2, глава 7, § 46. ЦОР № 5.

§ 24

История программного обеспечения и ИКТ

Основные темы параграфа:

- *структура программного обеспечения;*
- *история систем программирования;*
- *история системного ПО;*
- *история прикладного ПО;*
- *ИКТ и их приложения.*

Вы уже хорошо знаете, что возможности современных информационных технологий определяются не только техническими характеристиками компьютеров, но и составом их программного обеспечения (ПО).

Структура программного обеспечения

Структура ПО современных персональных компьютеров схематически изображена на рис. 3.1.

Именно благодаря такому разнообразию программных средств персональный компьютер оказывается полезным и школьнику, и ученому, и домохозяйке. Представление об ИКТ — информационно-коммуникационных технологиях — связано с широким распространением всего этого множества программных продуктов.

Рис. 3.1. Структура программного обеспечения компьютера

Познакомимся с историей возникновения и развития программного обеспечения ЭВМ. Появление каждого нового типа программ связано с появлением новых областей приложения компьютеров, расширением круга пользователей.

История систем программирования

Первые ЭВМ были доступны исключительно программистам. Поэтому исторически первым типом ПО стали системы программирования.

На машинах первого поколения языков программирования (в современном понимании) не существовало. Программисты работали на языке машинных кодов, что было весьма сложно. ЭВМ первого и второго поколений были приспособлены, прежде всего, для выполнения математических расчетов. А в таких расчетах часто приходится вычислять математические функции: квадратные корни, синусы, логарифмы и пр. Для вычисления этих функций программисты создавали стандартные программы, к которым производили обращения из своих расчетных программ. Стандартные программы хранились все вместе на внешнем носителе (тогда это преимущественно были магнитные ленты). Такое хранилище называлось библиотекой стандартных программ. **Библиотеки стандартных программ** (БСП) — первый вид программного обеспечения ЭВМ.

Затем в БСП стали включать стандартные программы решения типовых математических задач: вычисления корней уравнений, решения систем линейных уравнений и пр. Поскольку все эти программы носили математический характер, то в тот период чаще употреблялся термин **«математическое обеспечение ЭВМ»**. Библиотеки стандартных программ используются и в современных системах программирования (см. рис. 3.1).

В эпоху второго поколения ЭВМ распространяются языки программирования высокого уровня (ЯПВУ). Об этом уже говорилось в предыдущем параграфе. ЯПВУ сделали программирование доступным не только для профессиональных программистов. Программировать стали многие научные работники, инженеры, студенты различных специальностей и даже школьники, проходящие специальную подготовку по программированию.

В программное обеспечение ЭВМ включаются **трансляторы с ЯПВУ**. Подробнее о языках программирования и трансляторах читайте в разделе 2.3 материала для дополнительного обучения. Понятие **системы программирования** в современном виде возникло в период третьего поколения ЭВМ, когда программисты для разработки программ стали пользоваться терминальным вводом (клавиатурой и дисплеем). В состав систем программирования были включены **текстовые редакторы** для ввода и редактирования программы и **отлад-**

чики, позволяющие программисту исправлять ошибки в программе в интерактивном режиме.

История системного ПО

Операционные системы (ОС). Первые версии ОС появились еще на ЭВМ второго поколения, но массовое распространение операционные системы получают, начиная с машин третьего поколения.

Основная проблема, которую решали разработчики ОС, — повышение эффективности работы компьютера. На первых ЭВМ процессор — основное вычислительное устройство — нередко больше простаивал, чем работал во время выполнения программы. Такое происходило, если выполняемая программа часто обращалась к внешним устройствам: ввода, вывода, внешней памяти. Дело в том, что эти устройства работают в тысячи раз медленнее процессора.

Операционная система позволяет реализовать многопрограммный режим работы компьютера, при котором в состоянии выполнения находятся одновременно несколько программ. Когда одна программа обращается к внешнему устройству, процессор прерывает работу с ней (внешнее устройство продолжает работу без участия процессора) и переходит к обработке другой программы. Затем процессор может прервать работу со второй программой и продолжить выполнение первой. Таким образом, несколько программ «выстраивается в очередь» к процессору, а ОС управляет обслуживанием этой очереди. Точно так же ОС управляет обслуживанием очереди к внешним устройствам, например к принтеру. Управляют ОС и очередью к средствам ПО: трансляторам, библиотекам, прокладным программам и пр. *Управление ресурсами ЭВМ — это первая функция операционных систем.*

С появлением систем коллективного пользования ЭВМ операционные системы стали поддерживать многопользовательский режим работы. В таких системах с одной ЭВМ одновременно работают множество людей через терминальные устройства: клавиатуру и дисплей. *ОС обеспечивает режим диалога с пользователями — интерактивный режим общения.* При этом у каждого пользователя (программиста) создается впечатление, что он работает с компьютером один на один.

Еще одной важной функцией ОС стала организация работы с файлами. На ЭВМ третьего поколения появились магнитные диски, на которых информация хранится в файловой форме. *Файловая система — это компонент ОС, работающий с файлами.*

Операционные системы современных ПК также выполняют все эти функции. Особенностью, отличающей их от первых ОС, является дружественный графический интерфейс. А в последнее время — поддержка сетевого режима работы как в локальных, так и в глобальных сетях.

Сервисные программы. Этот тип ПО возник и начал развиваться в эпоху персональных компьютеров. Сюда входят разнообразные утилиты, антивирусные программы, программы-архиваторы.

Утилита — это небольшая программа, выполняющая действия, направленные на улучшение работы компьютера. Например, программа восстановления ошибочно удаленных файлов, программа обслуживания жесткого диска: лечения, дефрагментации и т. д.

Компьютерным вирусом является программа, способная внедряться в другие программы. Программы-вирусы выполняют нежелательные и даже опасные для работы компьютера действия: разрушают файловые структуры, «засоряют» диски и даже выводят из строя устройства компьютера. Для защиты от вирусов используются специализированные **антивирусные программы** (антивирус Касперского AVP, Norton Antivirus и т. д.).

Потребность в **программах-архиваторах** первоначально возникла в 80–90-х годах XX века в связи с небольшими информационными объемами устройств внешней памяти — магнитных дисков. Программа-архиватор (WinRaR, ZipMagic и др.) позволяет сократить объем файла в несколько раз без потери содержащейся в нем информации. В последнее время большое значение приобрело использование архивированных файлов в сетевых технологиях: электронной почте, файловых архивах — FTP-службе Интернета.

История прикладного ПО

Именно благодаря этому типу ПО персональные компьютеры получили широкое распространение в большинстве областей деятельности человека: медицине, экономике, образовании, делопроизводстве, торговле и даже в быту.

Самым массовым спросом среди прикладных программ пользуются, конечно, **текстовые редакторы** и **текстовые процессоры** (например, Microsoft Word). Ушли в прошлое пишущие машинки. Персональный компьютер, оснащенный текстовым редактором, и принтер стали основными инструментами для создания любых текстовых документов.

В 1979 году был создан первый **табличный процессор** — электронная таблица VisiCalc, ставшая самой популярной программой в среде предпринимателей, менеджеров и бухгалтеров. Идея электронной таблицы принадлежала Дэниелу Бриклину — студенту Гарвардской школы бизнеса. Начиная с 80-х годов прошлого века табличные процессоры входят в число лидирующих категорий программного обеспечения.

В конце 70-х — начале 80-х годов XX века появились первые коммерческие **системы управления базами данных (СУБД)** — программное обеспечение, которое позволяет пользователям создавать и об-

служивать компьютерную базу данных, а также управлять доступом к ней. В зависимости от области применения различают:

- настольные СУБД (Access, FoxPro, Paradox и т. д.), предназначенные для работы с небольшими базами данных, хранящимися на локальных дисках ПК или в небольших локальных сетях;
- СУБД серверного типа (Oracle, SQL Server, Informix и т. д.), ориентированные на работу с большими базами данных, расположенными на компьютерах-серверах.

В настоящее время все чаще приходится обрабатывать информацию (видео, звук, анимацию), которую невозможно хранить в традиционных базах данных. Jasmine является первой в мире СУБД, ориентированной на разработку баз данных, хранящих мультимедийную информацию.

Электронный офис — в последнее время часто используемое понятие. Обычно под этим понимают такой метод ведения делопроизводства, при котором всю циркулирующую информацию обрабатывают электронным способом с помощью определенных технических средств и программного обеспечения. Таким программным обеспечением являются **интегрированные пакеты**, включающие набор приложений, каждое из которых ориентировано на выполнение определенных функций, создание документов определенного типа (текстовых документов, электронных таблиц и т. д.). В процессе работы может происходить обмен информацией между документами, могут создаваться составные документы, включающие в себя объекты разных типов (текст, рисунки, электронные таблицы).

Широко используемым сегодня интегрированным пакетом является офисная система Microsoft Office, базовыми компонентами которой принято считать текстовый редактор Microsoft Word и табличный процессор Microsoft Excel. В состав пакета также включены СУБД Microsoft Access, система подготовки презентаций Microsoft PowerPoint и некоторые другие программы. Все большей популярностью в учебных заведениях пользуются программы, входящие в свободно распространяемый пакет OpenOffice.org. Важным его достоинством является отсутствие лицензионной платы за использование. Пакет включает в себя текстовый процессор Writer, табличный процессор Calc, систему подготовки презентаций Impress, СУБД реляционного типа Base. Существуют версии OpenOffice.org, работающие в средах ОС Windows и ОС Linux.

В 90-е годы XX века появляется термин **мультимедиа**: в дополнение к традиционным тексту и графике появилась возможность работать с такими видами информации, как видео и звук. Для хранения мультимедиафайлов требуются большие объемы внешней памяти

ПК, для обработки — большие процессорные мощности. Создание объемного реалистического изображения обеспечивается современными видеокартами, обработка звука — звуковой картой. Появляются программы редактирования и монтажа звука и видео, предназначенные для профессионалов в области музыки и видео. Наряду с этим создаются **программы-проигрыватели мультимедиа файлов** (Windows Media Player, Real Media Player и др.), ориентированные на широкий круг пользователей.

В 1991 году сотрудник Женевской лаборатории практической физики Тим Бернерс-Ли разрабатывает систему гипертекстовых страниц Интернета, получившую название World Wide Web (WWW) — Всемирная паутина. Создание собственной Web-страницы и опубликование ее в сети под силу многим пользователям благодаря специальным **программам-конструкторам Web-страниц.** Наиболее популярными сегодня являются Microsoft FrontPage, входящий в состав пакета Microsoft Office, и Macromedia DreamWeaver. Этими программами пользуются не только любители, но и профессионалы Web-дизайна. В ОС Linux популярна программа OpenOffice.org Write/Web.

Прикладное ПО специального назначения. Данный тип программного обеспечения служит информатизации различных профессиональных областей деятельности людей. Трудно дать исчерпывающий обзор для этой области. Сейчас практически в любой профессии, связанной с обработкой информации, существует свое специализированное ПО, свои средства информационных технологий.

Информационная технология — совокупность массовых способов и приемов накопления, передачи и обработки информации с использованием современных технических и программных средств.

ИКТ и их приложения

В последнее время в употребление вошел термин **информационно-коммуникационные технологии — ИКТ**. Рассмотрим лишь некоторые примеры профессионального использования ИКТ.

Технологии подготовки документов. Любая деловая сфера связана с подготовкой различной документации: отчетной, научной, справочной, сопроводительной, финансовой и т. д. Сегодня подготовка документа любой сложности немыслима без применения компьютера.

Для подготовки текстовых документов используются текстовые процессоры, которые прошли путь развития от простейших **редак-**

торов, не дающих возможность даже форматировать текст, до **текстовых процессоров**, позволяющих создавать документы, включающие в себя не только текст, но и таблицы, рисунки. Информационные технологии, связанные с созданием текстовых документов, широко используются в полиграфической промышленности. Там получили распространение **издательские системы** (например, Page Maker), позволяющие создавать макеты печатных изданий (газет, журналов, книг).

Большую роль в автоматизации подготовки финансовых документов сыграли электронные таблицы. С электронными таблицами работают программы — **табличные процессоры**. Пример такой программы — Microsoft Excel.

В настоящее время в финансовой сфере все больше используются **бухгалтерские системы** (1С-бухгалтерия и др.). Их широкое применение объясняется тем, что с помощью такой системы можно не только произвести финансовые расчеты, но и получить бумажные и электронные копии таких документов, как финансовый отчет, расчет заработной платы и пр. Электронные копии могут быть отправлены с помощью сетевых технологий в проверяющую организацию, например в налоговую инспекцию.

Для подготовки научных документов, содержащих математические расчеты, используются **математические пакеты программ** (MathCAD, Maple и пр.). Современные математические пакеты позволяют создавать документы, совмещающие текст с математическими расчетами и чертежами. С помощью такого документа можно получить результаты расчетов для разных исходных данных, изменяя их непосредственно в тексте документа. Большинство математических систем, используемых сегодня, было создано еще в середине 80-х годов прошлого столетия, т. е. вместе с появлением персональных компьютеров. Новые версии этих систем включают в себя новые возможности, например использование сетевых технологий: организацию доступа к ресурсам сети Интернет во время работы в среде математического пакета.

ИКТ в управлении предприятием. Эффективность работы компании (производственной, торговой, финансовой и пр.) зависит от того, как организованы хранение, сбор, обмен, обработка и защита информации. Для решения этих проблем уже более двадцати лет назад стали внедряться **автоматизированные системы управления** (АСУ).

В настоящее время в этой области произошли большие перемены. Классическая АСУ включает в себя систему сбора информации, базу данных, систему обработки и анализа информации, систему формирования выходной информации. Блок обработки и анализа информации является центральным. Его работа основана на экономико-математической модели предприятия. Он решает задачи прогнозирования деятельности компании на основе финансово-бухгалтерских расче-

тов, реагирования на непредвиденные ситуации, т. е. оказывает помощь в принятии управленческих решений.

Как правило, *АСУ работают на базе локальной сети предприятия*, что обеспечивает оперативность и гибкость в принятии решений. С развитием глобальных сетей появилась **коммуникационная технология Intranet**, которую называют корпоративной паутиной. Intranet обеспечивает информационное взаимодействие между отдельными сотрудниками и подразделениями компании, а также ее отдаленными внешними партнерами. Intranet помогает поддерживать оперативную связь центрального офиса с коммерческими представительствами компании, которые обычно располагаются далеко друг от друга.

ИКТ в проектной деятельности. Информатизация произвела на свет еще одну важную технологию — **системы автоматизированного проектирования (САПР).**

Проектирование включает в себя создание эскизов, чертежей, выполнение экономических и технических расчетов, работу с документацией.

Существуют САПРы двух видов: чертежные и специализированные. Чертежные САПРы универсальны и позволяют выполнить сложные чертежи в любой сфере технического проектирования (AutoCad). Специализированная САПР, например на проектирование жилых зданий, содержит в базе данных все необходимые сведения о строительных материалах, о стандартных строительных конструкциях, фундаментах. Инженер-проектировщик создает чертежи, производит технико-экономические расчеты с использованием таких систем. При этом повышается производительность труда конструктора, качество чертежей и расчетных работ.

Геоинформационные системы. Геоинформационные системы (ГИС) хранят данные, привязанные к географической карте местности (района, города, страны). Например, муниципальная ГИС содержит в своих базах данных информацию, необходимую для всех служб, поддерживающих жизнедеятельность города: городских властей, энергетиков, связистов, медицинских служб, милиции, пожарной службы и пр. Вся эта разнородная информация привязана к карте города. Использование ГИС помогает соответствующим службам оперативно реагировать на чрезвычайные ситуации: стихийные бедствия, экологические катастрофы, технологические аварии и пр.

ИКТ в образовании. В наше время от уровня образованности людей существенно зависит уровень развития страны, качество жизни ее населения. Требования к качеству образования постоянно растут. Старые, традиционные методы обучения уже не успевают за этими требованиями. Возникает очевидное противоречие. Использование ИКТ в образовании может помочь в разрешении этого противоречия.

Технологии обучения мало изменились за последние 100 лет. Пока в основном действует метод коллективного обучения. Не всегда такой способ обучения дает высокие результаты. Причина заключается в разном уровне способностей разных учеников. Учителя хорошо понимают, что необходим индивидуальный подход в работе с учащимися. Решению этой проблемы может помочь использование в процессе обучения специальных программ (обучающих, контролирующих, тренажерных и т. д.), входящих в состав **электронного учебника.**

Обучение — это процесс получения знаний. Традиционный источник знаний — учебник ограничен в своих информационных возможностях. Обучающимся на любой ступени образования всегда требовались дополнительные источники информации: библиотеки, музеи, архивы и пр. В этом отношении жители крупных городов находятся в более благоприятных условиях, чем сельские жители. Здесь можно говорить о существовании *информационного неравенства*. Решить эту проблему поможет широкое использование в обучении **информационных ресурсов Интернета.** В частности, *специализированных порталов учебной информации.*

Примером такого портала является портал **Единой коллекции цифровых образовательных ресурсов (ЕК ЦОР).** Арес в Интернете: http://school-collection.edu.ru. Ресурсы этого портала охватывают цифровые средства обучения по всем предметам во всех классах средней общеобразовательной школы. Сюда входят иллюстративные материалы к теории, практикумы по решению задач, ЦОРы для автоматического контроля знаний, виртуальные лаборатории и другие типы учебных программ. Особо отметим назначение **виртуальных лабораторий.** Это интерактивные программы, позволяющие воспроизводить на компьютере учебные эксперименты, чаще всего по естественнонаучным дисциплинам: физике, химии, биологии. Например, ученик может собрать на экране компьютера электрическое устройство (схему) и провести его испытание, используя виртуальные измерительные приборы: вольтметры, амперметры и др.

Еще одна проблема системы образования связана с неравными возможностями получения качественного образования из-за географической отдаленности от образовательных центров. Например, для жителя Якутии проблематично получить диплом престижного московского вуза. В решении этой проблемы на помощь приходит новая форма обучения — **дистанционное образование**, реализация которого стала возможна благодаря развитию компьютерных сетей.

Дистанционное образование приходит на смену старой форме заочного образования, при которой весь информационный обмен происходил в письменном виде через почтовую связь. Сетевое дистанционное образование позволяет вести обучение в режиме реального времени. Обучаемые могут не только читать учебный материал,

но и видеть и слышать лекции крупных ученых, сдавать экзамены в прямом контакте с экзаменатором.

Коротко о главном

Программное обеспечение компьютера включает в себя системное ПО, прикладное ПО и системы программирования.

Исторически первым видом ПО стали системы программирования.

Ядро системного ПО — операционные системы, зародились в период второго поколения ЭВМ, но распространение получили, начиная с третьего поколения.

Сервисные программы (утилиты, архиваторы, антивирусные программы) получили распространение на персональных компьютерах.

Прикладное программное обеспечение общего назначения развивалось от внедрения отдельных программ (текстовых редакторов, табличных процессоров, СУБД и пр.) до интегрированных систем — офисных пакетов.

Информационная технология — совокупность массовых способов и приемов накопления, передачи и обработки информации с использованием современных технических и программных средств.

Информационно-коммуникационные технологии (ИКТ) в настоящее время используются в большинстве профессиональных областей, связанных с обработкой информации, в том числе все шире применяются в образовании.

Вопросы и задания

1. Какова структура программного обеспечения современного компьютера?
2. Почему первыми пользователями ЭВМ стали программисты?
3. Когда началось распространение операционных систем? С чем это связано?
4. Какие виды программ кроме ОС относятся к системному ПО?
5. Как классифицируется прикладное ПО? Подготовьте сообщение.
6. Перечислите основные виды прикладных программ общего назначения и назовите информационные задачи, которые решаются с их помощью.
7. Приведите примеры профессионального использования прикладных программ.
8. Опишите формы использования ИКТ, с которыми вам приходилось иметь дело в школе. Какой эффект от их использования вы можете отметить? Подготовьте сообщение.

www ЕК ЦОР: часть 2, глава 7, § 47. ЦОР № 1, 10.

§ 25

Информационные ресурсы современного общества

Основные темы параграфа:

- *понятие информационных ресурсов;*
- *национальные информационные ресурсы;*
- *виды национальных информационных ресурсов.*

Понятие информационных ресурсов

Ресурс — это запас или источник некоторых средств. Всякое общество имеет определенные ресурсы, необходимые для его жизнедеятельности. Традиционными видами общественных ресурсов являются материальные, сырьевые (природные), энергетические, трудовые, финансовые ресурсы. В дополнение к этому, одним из важнейших видов ресурсов современного общества являются **информационные ресурсы**. К информационным ресурсам уместно отнести все научно-технические знания, произведения литературы и искусства.

Информационные ресурсы — это знания, идеи человечества и указания по их реализации, зафиксированные в любой форме, на любом носителе информации.

Между информационными ресурсами и всякими иными существует важное различие: *всякий ресурс, кроме информационного, после его использования исчезает* (например, иссякают нефтяные месторождения, расходуются финансовые ресурсы). Информационным ресурсом можно пользоваться многократно. Кроме того, использование информационных ресурсов влечет за собой создание новых ресурсов, в том числе информационных. Информационные ресурсы тем быстрее растут, чем быстрее их расходуют.

Национальные информационные ресурсы

Применительно к отдельному государству говорят о **национальных информационных ресурсах**, включающих библиотечные и архивные ресурсы, научно-техническую информацию, отраслевую информацию, информацию государственных (властных) структур, информационные ресурсы социальной сферы и пр.

Виды национальных информационных ресурсов

Огромные информационные ресурсы скрыты в **библиотеках**. Библиотечная сеть России насчитывает 150 тысяч библиотек. Библиографическая регистрация и статистический учет всей выходящей в стране печатной продукции осуществляются Российской книжной палатой. Библиотечное дело в массовом порядке переводится на использование информационных технологий: создаются электронные каталоги, специализированные (библиографические) базы данных. В ряде центральных библиотек сформированы собственные электронные информационные ресурсы. Начался процесс публикации этих ресурсов через Интернет.

Архивы хранят материалы, связанные с историей и культурой страны. Объемы архивных материалов огромны и накапливаются зачастую быстрее, чем их удается обрабатывать. Архивный фонд Российской Федерации ежегодно пополняется на 1,6 млн документов. В государственных архивах создано и поддерживается более 400 баз данных.

Во всех развитых странах существуют специализированные **центры научно-технической информации**, включающие специализированные издания, патентные службы и пр. Примером такого центра в нашей стране является ВИНИТИ РАН — Всероссийский институт научной и технической информации Российской академии наук. Этот информационный центр генерирует 60–70% всей научно-технической информации в России. Информация такого рода часто является дорогостоящим товаром.

Свои **отраслевые информационные ресурсы** имеются у любой промышленной, социальной и иной сферы общества.

Информационные ресурсы социальной сферы связаны с образованием, медициной, системой пенсионного обеспечения, службами занятости, медицинского и социального страхования. Например, основу информационных ресурсов в области образования составляют библиотеки вузов — высших учебных заведений (университетов, институтов, академий), общий фонд этих библиотек составляет более 300 млн экземпляров. Создана и функционирует федеральная телекоммуникационная университетская сеть. В сети Интернет отражена информация о большинстве российских вузов.

Следует отметить, что сегодня информационные ресурсы по своей значимости сопоставимы с другими ресурсами (материальными, сырьевыми, энергетическими, финансовыми). Об этом свидетельствует тот факт, что они становятся товаром, стоимость которого сопоставима со стоимостью других ресурсов. Для развитых стран характерно увеличение объема потребления из-за рубежа природных

ресурсов (газ, нефть и пр.) в обмен на экспорт своих информационных ресурсов (научно-технических знаний, информационных технологий).

Цифровые информационные ресурсы. Практически все вышеперечисленные виды национальных информационных ресурсов всё в большей степени переводятся в компьютерную (цифровую) форму хранения и обработки. Поскольку не существует ограничения на цифровое представление для любых видов информации (текстов, изображений, живой речи, научно-технической документации, музыки, кино и пр.), то постепенно цифровая форма станет основной формой хранения информации. Использование же сетевых технологий делает доступ к этой информации быстрым и удобным. Основные проблемы, которые требуют решения в этих условиях: разграничение доступа, обеспечение безопасности конфиденциальной информации, соблюдение авторских прав, оплата коммерческих ресурсов.

Коротко о главном

Информационные ресурсы — это знания, идеи человечества и указания по их реализации, зафиксированные в любой форме, на любом носителе информации.

Всякий ресурс, кроме информационного, после его использования исчезает. Информационным ресурсом можно пользоваться многократно.

К национальным информационным ресурсам относятся: фонды библиотек и архивов, центры научно-технической информации (ЦНТИ), отраслевые информационные ресурсы, информационные ресурсы социальной сферы, в том числе сферы образования.

Вопросы и задания

1. В чем основное отличие информационных ресурсов от материальных?
2. Перечислите основные виды национальных информационных ресурсов.
3. Назовите, какими видами информационных ресурсов вам приходится (или приходилось) пользоваться на портале www.edu.ru.

ЕК ЦОР: часть 2, глава 7, § 48. ЦОР № 6.

§ 26
Проблемы формирования информационного общества

Основные темы параграфа:

- *что такое информационное общество;*
- *что такое информатизация;*
- *задачи информатизации.*

Что такое информационное общество

Человеческое общество вступает в период своего развития, который называют **информационным обществом**.

В **информационном обществе** преимущественным видом трудовой деятельности людей станет информационная деятельность. Информационные ресурсы становятся важнейшими из всех видов ресурсов, влияющими на общественный прогресс.

Средствами информационной деятельности людей выступает компьютерная техника, информационно-коммуникационные технологии (ИКТ).

Что такое информатизация

Обсудим еще один часто употребляемый термин: информатизация. В узком смысле **информатизация** — это процесс создания, развития и массового применения информационных средств и технологий. Этот процесс происходил в течение всей истории человеческого общества и начался отнюдь не с появлением ЭВМ. Мы уже обсуждали этот вопрос в данной главе. Именно способность работать с информацией, создавать орудия информационного труда окончательно выделила человека из мира животных.

Однако в наше время смысл и значение процесса информатизации существенно изменились. Как никогда раньше возросла роль информационных ресурсов (об этом уже говорилось ранее) для всего общества и для каждого отдельного человека.

Информатизация — организационный социально-экономический и научно-технический процесс создания оптимальных условий для удовлетворения информационных потребностей и реализации прав граждан, органов государственной власти, органов местного самоуправления, организаций, общественных объединений на основе формирования и использования информационных ресурсов*.

Задачи информатизации

1. *Информационное обеспечение всех видов человеческой деятельности.* В качестве примеров видов деятельности можно назвать науку, образование, промышленное производство, сельское хозяйство, здравоохранение и многое другое. Информационное обеспечение складывается из специализированных фондов (баз данных, архивов, библиотек) и совокупности методов и средств их организации и использования.

2. *Информационное обеспечение активного отдыха и досуга людей.* В первую очередь эта задача достигается путем обеспечения населению возможности теледоступа ко всей сокровищнице мировой культуры, а также создания индустрии телеразвлечений.

3. *Формирование и развитие информационных потребностей людей.* Эта задача носит определенный воспитательный характер и является одной из задач общего образования. Учебная или производственная деятельность людей должна стимулировать их к повышению своего уровня информированности. Без решения этой задачи нельзя рассчитывать на успех информатизации, так как ее результаты могут оказаться просто невостребованными.

4. *Формирование условий, обеспечивающих осуществление информатизации.* К таким условиям относятся различные виды обеспечения информатизации: экономические, организационные, научно-технические, правовые. Создание этих условий — функция государства, органов, управляющих процессами информатизации.

Коротко о главном

В информационном обществе основным предметом трудовой деятельности людей становится информация.

Информатизация — организационный социально-экономический и научно-технический процесс создания оптимальных условий

* Из Федерального закона «Об информации, информатизации и защите информации» от 20.02.1995 № 24.

для удовлетворения информационных потребностей и реализации прав граждан, органов государственной власти, органов местного самоуправления, организаций, общественных объединений на основе формирования и использования информационных ресурсов.

Вопросы и задания

1. По каким признакам можно судить о наступлении эпохи информационного общества? Подготовьте сообщение.
2. Попробуйте оценить, какая часть родителей ваших одноклассников трудится в сфере материального производства, а какая — в информационной сфере. Отсюда сделайте вывод, как далеко мы находимся от стадии информационного общества.
3. Что такое информатизация? Опишите основные цели информатизации. Какие из этих целей в большей или меньшей степени, по вашему мнению, достигнуты в нашей стране? Подготовьте реферат.

ЕК ЦОР: часть 2, глава 7, § 49. ЦОР № 7.

§ 27

Информационная безопасность

Основные темы параграфа:

- *информационные преступления и информационная безопасность;*
- *программно-технические способы защиты информации;*
- *правовая защита информации.*

Информационные преступления и информационная безопасность

Многие черты информационного общества уже присутствуют в современной жизни развитых стран. Компьютеры контролируют работу атомных реакторов, распределяют электроэнергию, управляют самолетами и космическими кораблями, определяют надежность систем обороны страны и банковских систем, т. е. используются в областях общественной жизни, обеспечивающих благополучие и даже жизнь множества людей.

Жизненно важной для общества становится проблема информационной безопасности действующих систем хранения, передачи и обработки информации.

О важности этой проблемы свидетельствуют многочисленные факты. Каждые двадцать секунд в США происходит преступление с использованием программных средств. Более 80% компьютерных преступлений осуществляется через глобальную сеть Интернет, которая обеспечивает широкие возможности злоумышленникам для нарушений в глобальном масштабе. Если компьютер, являющийся объектом атаки, подключен к Интернету, то для «взломщика» нет большой разницы, где он находится — в соседней комнате или на другом континенте. Потери от хищения или повреждения компьютерных данных составляют более 100 млн долларов в год. Во многих случаях организации даже не подозревают, что «вторжение» имело место, хищение информации происходит незаметно.

Перечислим некоторые виды компьютерных преступлений, когда компьютер является инструментом для совершения преступления, а объектом преступления является информация.

1. *Несанкционированный (неправомерный) доступ к информации.* Лицо получает доступ к секретной информации, например, путем подбора шифра (пароля). Подавляющее большинство разработок в области информационной безопасности посвящено предотвращению именно этого вида преступлений, что позволяет обеспечить охрану государственных и военных секретов.
2. *Нарушение работоспособности компьютерной системы.* В результате преднамеренных действий ресурсы вычислительной системы становятся недоступными или снижается ее работоспособность. Примером такого рода преступлений является создание и распространение компьютерных вирусов.
3. *Подделка (искажение или изменение), т. е. нарушение целостности компьютерной информации.* Эта деятельность является разновидностью неправомерного доступа к информации. К подобного рода действиям можно отнести подтасовку результатов голосования на выборах, референдумах и т. д. путем внесения изменений в итоговые протоколы.

Программно-технические способы защиты информации

Если речь идет о персональной информации отдельного пользователя ПК, то главной опасностью является потеря данных по непреднамеренным причинам, а также из-за случайного проникновения вредоносных вирусов. Основные правила безопасности, которые следует соблюдать, такие:

- периодически осуществлять **резервное копирование:** файлы с наиболее важными данными дублировать и сохранять на внешних носителях;

- регулярно осуществлять **антивирусную проверку** компьютера;
- подключать ПК к электросети через **блок бесперебойного питания** (ББП).

Блок бесперебойного питания физически защищает компьютер от последствий отключения электроэнергии или резких скачков напряжения в сети. Если компьютер от этого не защищен, то можно не только потерять данные, но и сам компьютер: какие-то его части могут выйти из строя.

Компьютерный вирус — вредоносная программа, распространяемая без ведома пользователя и наносящая тот или иной ущерб данным пользователя. Основным разносчиком вирусов является нелицензионное программное обеспечение, файлы, скопированные из случайных источников, а также службы Интернета: электронная почта, Всемирная паутина —WWW.

Каждый день в мире появляются сотни новых компьютерных вирусов. Борьбой с этим злом занимаются специалисты, создающие **антивирусные программы**. В составе программного обеспечения компьютера обязательно должны присутствовать такие программы.

Лицензионные антивирусные программы следует покупать у фирм-производителей. Однако антивирусную программу недостаточно лишь однажды установить на компьютер. После этого нужно регулярно обновлять ее базу — добавлять настройки на новые типы вирусов. Наиболее оперативно такое обновление производится через Интернет серверами фирм-производителей этих программ.

Наибольшим опасностям подвергаются пользователи глобальных сетей, Интернета. Для защиты компьютеров, подключенных к сети, от подозрительных объектов, «кочующих» по ней, используются защитные программы, которые называются **«брандмауэры»**, (английское название *firewall* — противопожарная стена, огнеупор). Существуют брандмауэры, защищающие сети, подключенные к другим сетям, они называются **межсетевыми экранами**. Межсетевые экраны осуществляют фильтрацию пакетов данных, передаваемых по каналам межсетевой связи в соответствии с действующими разрешениями и запретами. Для защиты отдельных компьютеров, подключенных к сети, используются **персональные брандмауэры**. Брандмауэры используются для защиты от так называемых хакерских атак. Во всех современных операционных системах есть программы, выполняющие функции брандмауэров.

Утечка информации может происходить путем перехвата ее в процессе передачи по каналам связи. Если от этого не удается защититься техническими средствами, то применяют системы шифрования. Методами шифрования занимается **криптография**.

Правовая защита информации

Наряду с программно-техническими средствами защиты информации действуют правовые, юридические меры защиты. В 1996 году в **Уголовный кодекс РФ** был впервые внесен раздел «Преступления в сфере компьютерной информации». Он определил меру наказания за некоторые виды преступлений, ставших, к сожалению, распространенными:

- неправомерный доступ к компьютерной информации;
- создание, использование и распространение вредоносных программ для ЭВМ;
- умышленное нарушение правил эксплуатации ЭВМ и их сетей.

К защите информации относится также и осуществление авторских и имущественных прав на интеллектуальную собственность, каковой является программное обеспечение.

Необходимой составляющей общей культуры современного человека становится информационная культура. Это понятие включает в себя не только умение использовать средства информационно-коммуникационных технологий, но и соблюдение правовых норм в своей информационной деятельности.

Опасности социальных сетей

Социальной сетью называют интерактивный многопользовательский Web-сайт, контент (содержание) которого наполняется самими участниками сети. Сайт предоставляет удобную автоматизированную социальную среду, позволяющую общаться группам пользователей, имеющих общие интересы. Среди социальных сетей популярными в нашей стране являются «Одноклассники», «В контакте», «Facebook», «Twitter» и др. Переписка в социальных сетях имеет не только положительную сторону, связанную с удобством и широким кругом общения, но и свои, весьма серьезные, опасности. Главный принцип, который должны усвоить пользователи социальных сетей, состоит в следующем: размещаемая в сети деловая или личная информация навсегда сохраняется в Интернете и становится общедоступной. Такая информация может быть найдена и использована кем угодно, в том числе и не обязательно с благими намерениями.

Всё большее число детей школьного возраста пользуется социальными сетями. Им следует помнить, что излишняя «болтливость» в Интернете может стать причиной бед лично для них и их близких. Не следует разглашать семейную информацию: о материальном достатке, о деятельности своих родителей, о времени и местах нахож-

дения близких людей, о семейных связях и пр., т. е. всё то, что может быть использовано в преступных целях.

Коротко о главном

Проблема информационной безопасности связана с широким использованием компьютеров в жизненно важных областях.

Основные формы компьютерных преступлений: несанкционированный (неправомерный) доступ к информации, нарушение работоспособности компьютерной системы, нарушение целостности компьютерной информации.

Основные меры по защите от компьютерных преступлений: технические и аппаратно-программные, административные, юридические.

Важнейшая научно-техническая задача в области информатики: разработка технологий создания защищенных автоматизированных систем обработки информации.

Программное обеспечение является интеллектуальной собственностью разработчиков. Его использование должно оплачиваться.

Не следует разглашать конфиденциальную информацию (личную, семейную, деловую) при общении в социальных сетях.

Вопросы и задания

1. Какие действия относятся к области информационных преступлений?
2. Приведите примеры, когда человек бессознательно совершает информационное правонарушение.
3. Какие существуют меры по защите от информационных преступлений? Какие меры вы могли бы предложить сами?
4. Почему использование «пиратских» копий программного обеспечения является преступлением?
5. Как вы думаете, что привлекает хакеров (взломщиков информационных систем) в их преступной деятельности? Какими эффективными средствами можно пресечь такую деятельность?
6. Что такое компьютерные вирусы и какие существуют средства борьбы с ними? Подготовьте реферат.
7. Обсудите в классе проблему безопасности при работе в социальных сетях.

www ЕК ЦОР: часть 2, глава 7, § 49. ЦОР № 2, 6.

Чему вы должны научиться, изучив главу III

- Соблюдать меры безопасности для своего компьютера:
  - — осуществлять резервное копирование важной информации;
  - — регулярно осуществлять анитивирусную проверку;
  - — использовать блок бесперебойного питания.
- Соблюдать правовые нормы поведения в информационном пространстве.

СОЦИАЛЬНАЯ

История развития средств информатизации

Предыстория средств информатизации

- Средства **хранения** информации:
  камень, глина, папирус, бумага, книги, фотопленка, кинопленка, магнитная лента
- Средства **передачи** информации:
  почта, телеграф, телефон, радио, телевидение
- Средства **обработки** информации (вычислений):
  системы счисления (непозиционные, позиционные);
  вычислительные механизмы: *абак, счеты, логарифмическая линейка, арифмометр, калькулятор;*
  прообраз компьютера: *Аналитическая машина Бэббиджа*

История компьютера

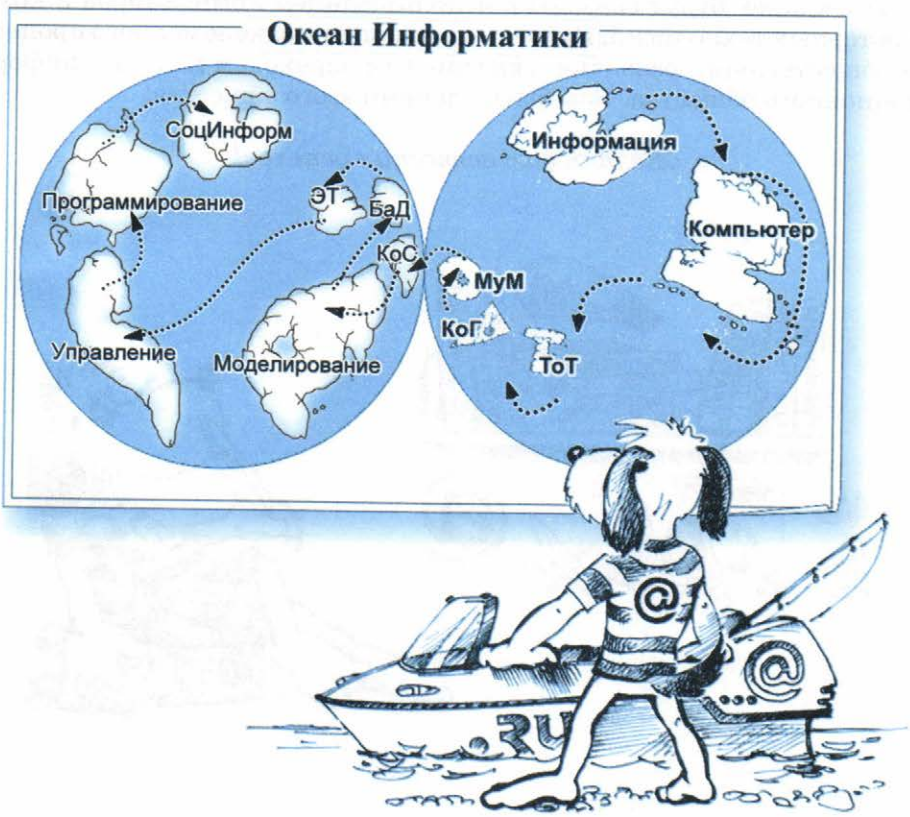
- 1-е поколение:
  ламповые машины 50-х годов XX века
- 2-е поколение:
  полупроводниковые машины 60-х годов XX века
- 3-е поколение:
  машины на интегральных схемах 70-х годов XX века
- 4-е поколение:
  современные ПК и суперкомпьютеры

История ПО компьютера

- 1-2-е поколения: библиотеки стандартных программ;
  языки программирования высокого уровня
- 3-е поколение: операционные системы,
  системы программирования, СУБД,
  пакеты прикладных программ
- 4-е поколение: интегрированные офисные пакеты,
  языки и системы web-программирования

Система основных понятий главы III

Путешествие завершено



Дорогие ученики!

Вот и завершилось ваше путешествие по Океану знаний под названием «ИНФОРМАТИКА»! Заполнены все белые пятна на карте. Теперь вы можете еще раз внимательно рассмотреть карту, вспомнить, какие материки и острова вы посетили, что нового для себя узнали, какие приобрели практические навыки.

Заключение

Знания и умения, полученные при изучении курса информатики, понадобятся вам в дальнейшей учебе. В современной школе все большее значение приобретает компьютерная техника и ИКТ в качестве средства обучения. Наряду с привычными бумажными учебниками используются электронные учебники. Неограниченным источником учебной информации становится Интернет. Все шире различные школьные предметы будут обращаться к компьютерному моделированию как средству обучения и развития школьников. Велика вероятность того, что после окончания школы ваша дальнейшая учеба и работа также будут связаны с использованием компьютеров и компьютерных технологий. Всё это — проявление закономерного процесса общественного развития, связанного с переходом к стадии информационного общества. Вам быть членами этого общества!

Желаем вам всяческих успехов!